TU MÜNCHEN

Fakultät für Informatik

CONNECTIONIST MODELS FOR LEARNING LOCAL IMAGE DESCRIPTORS: AN EMPIRICAL CASE STUDY

Christian Anton Osendorfer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr. Daniel Cremers

Prüfer der Dissertation:

1. Univ.-Prof. Dr. Patrick van der Smagt
2. Univ.-Prof. Dr. Jürgen Schmidhuber

Die Dissertation wurde am 07.12.2015 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 30.03.2016 angenommen.

ABSTRACT

Recently, several difficult visual object classification benchmarks have been tackled very successfully by deep learning-based systems. The equally important problem of learning (compact) descriptors for low-level image patches has been mostly neglected by this type of models, however.

This work demonstrates that deep supervised Convolutional Networks perform competitively on a challenging correspondence task for local image patches, even if the model is severely restricted. State-of-the-art results can be achieved if the learning objective and the optimization algorithm are finely adapted to the correspondence task.

Similarly, real-valued as well as binary representations learned in a completely unsupervised way perform comparably to engineered descriptors if the learning algorithm factors the latent representation.

Apart from these extensive empirical analyses three algorithmic contributions are described: *Explicit negative contrasting* shows how learning of a multi-view undirected graphical model can be improved by utilizing negative pairs of samples. *Hobbesian Networks* use differential equations to induce a deep Autoencoder architecture. And *vaeRIM* successfully combines a variational inference method for directed graphical models with an unsupervised clustering algorithm in a novel way.

ZUSAMMENFASSUNG

In jüngster Vergangenheit haben tiefe Lernarchitekturen im Bereich der visuellen Objekterkennung sehr gute Ergebnisse erzielt. Die gleichermaßen wichtige Aufgabe, kompakte Darstellungen für kleinteilige Bildausschnitte zu lernen, wurde von diesen Methoden jedoch vernachlässigt.
Diese Arbeit zeigt, dass tiefe überwachte Konvolutionsnetzwerke eine schwierige Ähnlichkeitsaufgabe, definiert mittels kleinteiliger Bildausschnitte, sehr gut lösen, auch wenn das Modell bewußt eingeschränkt wird. Die Ergebnisse können mit denen der besten alternativen Methoden konkurrieren, falls die Zielfunktion und der Optimierungsalgorithmus entsprechend abgestimmt sind.
Die Arbeit zeigt auch, dass unüberwachte Lernmethoden für dieses Problem ähnlich gut funktionieren wie von Menschen entworfene Repräsentationen. Das gilt sowohl für reellwertige als auch für binarisierte gelernte Darstellungen der Bildausschnitte, solange eine faktorisierte Repräsentation verwendet wird.
Neben diesen umfassenden empirischen Analysen werden drei algorithmische Neuerungen beschrieben. Durch *explicit negative contrasting* kann ein ungerichtetes graphisches Modell, definiert über einem zwei-modalen Eingaberaum, verbessert werden indem während des Lernens explizit negative Eingabepaare verwendet werden. Eine Reinterpretation numerischer Methoden zum Lösen von Differentialgleichungen erlaubt es diese zu verwenden, um tiefe Autoassoziationsstrukturen zu induzieren (*Hobbes'sche Netzwerke*). Und *vaeRIM* verwendet variationelle Inferenzmethoden eines gerichteten graphischen Modells auf neue Art und Weise, um einen unüberwachten Clusteringalgorithmus zu regularisieren.

PUBLICATIONS

PEER REVIEWED CONFERENCES / WORKSHOPS

- Saskia Golz, Christian Osendorfer, and Sami Haddadin. *Using tactile sensation for learning contact knowledge: Discriminate collision from physical interaction*. In *International Conference on Robotics and Automation (ICRA)*, 2015.
- Christian Osendorfer, Hubert Soyer, and Patrick van der Smagt. *Image super-resolution with fast approximate Convolutional Sparse Coding*. In *Neural Information Processing (ICONIP)*, 2014. **Best Paper Award**.
- Hubert Soyer and Christian Osendorfer. *Fast image super-resolution utilizing convolutional neural networks*. In *Forum Bildverarbeitung 2014*, 2014.
- Justin Bayer, Christian Osendorfer, Daniela Korhammer, Nutan Chen, Sebastian Urban, and Patrick van der Smagt. *On fast dropout and its applicability to recurrent networks*. In *Workshop International Conference for Learning Representations (ICLR)*, 2014.
- Nutan Chen, Sebastian Urban, Christian Osendorfer, Justin Bayer, and Patrick van der Smagt. *Estimating finger grip force from an image of the hand using Convolutional Neural Networks and Gaussian Processes*. In *International Conference on Robotics and Automation (ICRA)*, 2014.
- Rachel Hornung, Holger Urbanek, Julian Klodmann, Christian Osendorfer, and Patrick van der Smagt. *Model-free robot anomaly detection*. In *Intelligent Robots and Systems (IROS)*, 2014.
- Justin Bayer and Christian Osendorfer. *Learning stochastic recurrent networks*. In *NIPS 2014 Workshop on Advances in Variational Inference*, 2014.
- Justin Bayer, Christian Osendorfer, Sebastian Urban, and Patrick van der Smagt. *Training neural networks with implicit variance*. In *Neural Information Processing (ICONIP)*, 2013.
- Christian Osendorfer, Justin Bayer, Sebastian Urban, and Patrick van der Smagt. *Convolutional Neural Networks learn compact local image descriptors*. In *Neural Information Processing (ICONIP)*, 2013.
- Sebastian Urban, Justin Bayer, Christian Osendorfer, Göran Wesling, Benoni B. Edin, and Patrick van der Smagt. *Computing grip*

force and torque from finger nail images using Gaussian Processes.In *Intelligent Robots and Systems (IROS)*, 2013.

- Justin Bayer, Christian Osendorfer, and Patrick van der Smagt. *Learning sequence neigbourhood metrics*. In *International Conference on Artificial Neural Networks*, 2012.

- Justin Bayer, Christian Osendorfer, and Patrick van der Smagt. *Learning sequence neigbourhood metrics*. In *NIPS 2011 Workshop Beyond Mahalanobis: Supervised Large-Scale Learning of Similarity*, 2011.

- Christian Osendorfer, Jan Schlüter, Jürgen Schmidhuber, and Patrick van der Smagt. *Unsupervised learning of low-level audio features for music similarity estimation*. In *ICML Workshop on Learning Architectures, Representations, and Optimization for Speech and Visual Information Processing*, 2011.

- Thomas Rückstieß, Christian Osendorfer, and Patrick van der Smagt. *Sequential feature selection for classification*. In *Australasian Conference on Artificial Intelligence*, 2011.

- Jan Schlüter and Christian Osendorfer. *Music similarity estimation with the mean-covariance restricted boltzmann machine*. In *International Conference on Machine Learning and Applications (ICMLA)*, 2011.

- Frank Sehnke, Christian Osendorfer, Jan Sölter, Jürgen Schmidhuber, and Ulrich Rührmair. *Policy gradients for cryptanalysis*. In *International Conference on Artificial Neural Networks (ICANN)*, 2010.

- Frank Sehnke, Alex Graves, Christian Osendorfer, and Jürgen Schmidhuber. *Multimodal parameter-exploring policy gradients*. In *International Conference on Machine Learning and Applications (ICMLA)*, 2010.

- Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. *Policy gradients with parameter-based exploration for control*. In *International Conference on Artificial Neural Networks (ICANN)*, 2008.

PEER REVIEWED JOURNALS

- Thomas Rückstieß, Christian Osendorfer, and Patrick van der Smagt. *Minimizing data consumption with sequential online feature selection*. In *International Journal of Machine Learning and Cybernetics*, 4(3):235–243, 2013.

- Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. *Parameter-exploring policy gradients*. In *Neural Networks*, 23(2):551–559, 2010.

ACKNOWLEDGMENTS

First and foremost I want to thank my parents, Anton and Marianne[1]. I was always free to follow my interests, knowing that whatever I decide to do, they would support me the best they could. In addition to having great parents I also have the privilege of having two great siblings, Daniela and Markus. I am also incredibly lucky to have a wonderful wife and a wonderful son. Julia, thank you for all these years together, for your support, for your love and for the laughs together. Thanks for putting up with me every day. Levi, thank you for being our new source of joy and happiness and for being our sunshine all year long. It goes without saying that whatever I accomplish is to a very large part due to having a great family. Thank you so much.

I want to thank my two advisers, Prof. Patrick van der Smagt and Prof. Jürgen Schmidhuber, for guiding me through the Ph.D. experience. Jürgen accepted me, a novice to Machine Learning back then, as a Ph.D. student, introducing me to the wonderful world of Neural Networks. Thank you. Patrick helped me to continue working at TUM and without him this thesis would have never been finished. Thank you for the many valuable discussions over all these years.

I also want to thank Prof. Alois Knoll for hosting me so long at his chair. Dr. Gerhard Schrott and the three angels from the secretariat, Monika Knürr, Gisela Hibsch and Amy Bücherl, made the day-to-day operations for a research- and teaching-assistant bearable, not least with the right amount of humor. Prof. Darius Burschka claded me with many long and insightful discussions at the beginning of my Ph.D. career, while Prof. Sami Haddadin helped me to not lose track of the overall goal at the end of writing down the thesis. Thank you all.

I had the great luck of having a large number of awesome colleagues and collaborators during my time at TUM: Alex Graves, Agneta Gustus, Bastian Bischoff, Brian Jensen, Claus Lenz, Daniela Korhammer, Elmar Mair, Florian Trifterer, Frank Sehnke, Georg Stillfried, Hannes Höppner, Holger Urbanek, Hubert Soyer, Jörn Vogel, Juan Carlos Ramirez de la Cruz, Justin Bayer, Marc de Kamps, Markus Rickert, Martin Felder, Marvin Ludersdorfer, Maximilian Karl, Maximilian Sölch, Nutan Chen, Oliver Ruepp, Philipp Heise, Rachel Hornung, Sebastian Urban, Steffen Wittmeier, Thomas Rückstieß, Thorsten Röder, Tino Gomez and Zhongwen Song.

I am particularly grateful to the *Buam*, Alex, Flo, Frank, Justin, Martin and Thomas, for many many long, inspiring and insightful discussions but also for at least as many fun moments. Justin, special thanks for being an awesome idea-pusher as well as an awesome laugh-pusher for the last four years. And finally, a shout-out to George, for all motivational moments over the last decade.

1 Names are ordered lexicographically.

CONTENTS

Part I

MAIN WORK

1 INTRODUCTION

How does visual learning happen? The recent surge in massive performance improvements for object classification tasks achieved by deep Convolutional Networks [63, 59, 61, 209, 380, 350, 54] might indicate that it is enough to collect huge amounts of category-labeled image data and present these to a very powerful supervised learning system. But is this the way the human visual system is trained?

In some very rare cases, this is what seems to happen. When the visual task is of a very specific definition then humans are trained in this way, e.g. doctors or inspectors. And these are also very well emulated by deep learning systems, e.g. for steel defect detection [248] or in the medical image domain [62].

But generally this is not the case. So is it unsupervised learning, then? This presupposition might be tempting because most of the time we don't seem to have any explicit supervision signal. But at least one very powerful teacher, albeit mostly invisible, exists—physics. The images we constantly perceive do not form random sequences of pixel heaps but a very smooth movie, consistent in time and space. It helps that two views of a complicated three-dimensional environment are available and that evolution developed deterministic and non-deterministic control of the imaging system. The latter already proved to be crucial [61, 84] for supervised classification tasks: random saccades can be approximated by random geometric transformation that leave object-identity invariant [333]. However, rarely does this movie show well-centered and rectified objects, especially not in the beginning of the human learning adventure.

Most of the time the learning signals are therefore weak and only related to correspondences between local image regions (*patches*). But out of these pieces of correspondence information a powerful visual system can be built in a *bottom-up* manner, requiring relatively little detailed supervision in order to understand high-level concepts. This seems to be the exact opposite of the currently dominating top-down approach employed for supervised object recognition systems, requiring humongous amounts of labeled training data. Even more, learning low-level image correspondences has been completely neglected by deep learning approaches even though it is a central building block for many Computer Vision tasks beyond object classification.

A Computer Vision system built in a bottom-up manner would heavily rely on low-level patch correspondences, forming further levels of intermediate representations also through weak labels, representing similarities on larger contexts (in space or time, for example).

Different high-level tasks would then share most of these less abstract representations and, relying on a diverse set of tasks, only a small number of fine-grained supervised signals (hopefully) are necessary.

With the greater goal of building such a system in a bottom-up manner in mind, I conduct an in-depth empirical study of supervised and unsupervised algorithms for low-level image correspondence learning. More specifically, the class of algorithms I investigate is restricted to those employing distributed representations. I denote this class with the old, and somewhat vague term *connectionism* [318, 255]. *Connectionist learning systems* are adaptive methods that are based on distributed representations. Unsurprisingly, these are usually realized with Neural Network-like approaches (only because Neural Networks are currently the most powerful approximation technique which has a general and computationally tractable learning algorithm).

Describing Neural Networks as an expressive approximation algorithm is a central concern of Chapter 2. It covers all necessary mathematical foundations in order to understand Neural Networks from a technical point of view and how these can be applied as approximation functions. In particular it uses Matrix Calculus as the core mathematical tool to derive the important backpropagation algorithm [35, 333], allowing Neural Networks to be easily applied to very complex mathematical formulations (which are usually built out of matrix expressions). The chapter also introduces two novel unsupervised learning models: *Hobbesian Networks* are unsupervised Neural Networks with an architectural structure induced by differential equations of competitive submodules. Variational AutoEncoding Regularized Information Maximization (vaeRIM) uses an unsupervised generative model as a regularization method for an unsupervised clustering algorithm.

In Chapter 3 I introduce a large set of matching and non-matching low-level image patches, representing a low-level correspondence task. I also argue that this dataset (or more broadly, any kind of weak similarity task) is a good benchmark for purely unsupervised algorithms, considering that there seems to be no consistent evaluation method for this class of algorithms [385].

Chapter 4 empirically studies deep *supervised* Convolutional Networks for the previously presented dataset. Compared to typical variants of deep networks, the model is severely restricted in size (both with respect to the number of parameters as well as the dimensionality of the final representation for a patch). Nonetheless, using specifically adapted cost functions and optimization methods it is possible to perform competitively to state-of-the-art approaches on the correspondence task. Moreover, the learned representations (*descriptors*) are successfully validated on a challenging transfer problem.

Chapter 5 covers a similar empirical investigation, looking at unsupervised algorithms. While the results are by far not as impressive as those achieved with a supervised deep Convolutional Network it is still possible to reach competitive results with respect to hand-engineered image patch descriptors if semantically relevant representations are selected to form the final descriptor.

In Chapter 6 I collect a range of attempts to supervised as well as unsupervised learning that did not perform as hoped for on the matching task. Among others I describe how a multi-view undirected graphical model can be substantially improved by enhancing its learning algorithm with explicitly selected non-matching input pairs. Nevertheless, compared to overall results on the dataset the improved model still performs below the best (supervised) approaches. The most promising result in this chapter stems from representing descriptors as densities. It seems that for the correspondence problem the good results that were reported in parallel work [406] for NLP problems can be repeated.

Chapter 7 concludes this thesis, introducing a new unsupervised model for learning descriptors for local image patches, `patch2vec`. The model is heavily inspired by a successful model for learning distributed representations for words [261] and is suited very well to the overall goal of building a general Computer Vision system in a bottom-up manner.

2 FUNDAMENTALS

Data is recorded everywhere and at every moment. In order to allow algorithms to handle these streams of bits and bytes, to extract knowledge from it, Machine Learning methods need to support *scalable learning* with respect to the number of samples as well as with respect to the dimensionality of a single sample. Learning algorithms must provide (or at least enable) *fast approximate inference*[1] (whereby *approximate* is here defined loosely as *good enough*) and, for general applicability, must be *compositional*—depending on the task, it should be easy to reduce or enhance the model in complexity.

I think that the framework of Neural Networks is currently the approach that fulfills these very broad requirements for a general Machine-Learning method in the best way. This chapter provides a concise path to Neural Networks (Section 2.7) from an algorithmic and mathematical point of view. Because Machine Learning is a blend of various mathematical disciplines I also *review* the necessary mathematical foundations from Linear Algebra (Section 2.1), (Matrix) Calculus (Section 2.2) and Probability Theory (Section 2.3) and also look briefly at Graphical Models (Section 2.6) which encompass Neural Networks (Section 2.7) as a special kind. Some general concepts of Machine Learning are reviewed in Section 2.4. The goal of these reviews is simplicity and clarity, so whenever possible (and necessary) mathematical intricacies are left out.

Overall, this review covers slightly more than the bare necessities for Neural Networks—it enables the reader to put Neural Networks into the broader context of Machine Learning. A reader with a modest mathematical understanding and basic knowledge of what Machine Learning is about in general will be able to follow this chapter easily.

2.1 LINEAR ALGEBRA

NOTATION. An $m \times n$ matrix $\mathbf{A} \in \mathbf{R}^{m \times n}$ is a collection of scalar values arranged in a rectangle of m rows and n columns. The element of $\mathbf{A}$ at position (i, j) is written $[\mathbf{A}]_{ij}$, or more conveniently, $\mathbf{A}_{ij}$. If $\mathbf{A} \in \mathbf{R}^{m \times n}$ and $I \subset \{1, \ldots, m\}$ and $J \subset \{1, \ldots, n\}$ are (ordered) index sets then $\mathbf{A}_{I,J}$ denotes the $|I| \times |J|$ sub-matrix formed from $\mathbf{A}$ by selecting the corresponding entries $(i, j), i \in I, j \in J$. An $m \times n$ matrix is called square, when $m = n$. A square matrix $\mathbf{A}$ is said to be positive definite, denoted by $\mathbf{A} \succ 0$, if $\mathbf{x}^T \mathbf{A} \mathbf{x} > 0$ for all $\mathbf{x} \neq \mathbf{0}$. Hereby, $\mathbf{x}$ is a (column) vector $\mathbf{x} \in R^n$ which can be considered a $n \times 1$ matrix. Its elements

1 This is indispensable if learning relies on inference.

are denoted with $\mathbf{x}_i$. A matrix of the form $1 \times n$ is often referred to as a row vector and usually denoted by $\mathbf{x}^\top$. A vector whose elements are all ones is denoted by $\mathbf{1}$, or $\mathbf{1}_n$ if its dimensionality is known. A vector whose elements are all zeros apart from a one at position i is denoted by $\boldsymbol{e}_i$.

For many operations in Linear Algebra the dimensions of the involved matrices must fit. If the specific shapes are missing in this textf it is generally assumed that all matrices are *conformable* with respect to the operations.

OPERATORS. The *transpose* of an $m \times n$ matrix $\boldsymbol{A}$ is the $n \times m$ matrix $\boldsymbol{A}^\top$, with $\boldsymbol{A}^{\top\top} = \boldsymbol{A}$:

$$[\boldsymbol{A}^\top]_{ij} = [\boldsymbol{A}]_{ji} \tag{2.1}$$

The *conjugate transpose* of $\boldsymbol{A}$ is $\boldsymbol{A}^*$ and is defined as $\boldsymbol{A}^* = \bar{\boldsymbol{A}}^\top$ (i.e. $\boldsymbol{A}$ transposed and complex conjugated elementwise). A square matrix $\boldsymbol{A}$ is called *symmetric* if $\boldsymbol{A}^\top = \boldsymbol{A}$. Symmetric matrices are of particular interest, as they often arise in Machine Learning, for example as covariance matrices, kernel matrices or Hessian matrices.

The *trace operator* is only defined on square matrices. The trace of a matrix $\boldsymbol{A} \in \mathbf{R}^{n\times n}$ is the sum of the elements along the leading diagonal

$$\mathbf{tr}(\boldsymbol{A}) = \sum_{k=1}^{n} [\boldsymbol{A}]_{ii} \tag{2.2}$$

and, with $\boldsymbol{B} \in \mathbf{R}^{n\times n}$

$$\mathbf{tr}(\boldsymbol{A} + \boldsymbol{B}) = \mathbf{tr}(\boldsymbol{A}) + \mathbf{tr}(\boldsymbol{B}) \tag{2.3}$$

Clearly, the trace is invariant with respect to a scalar a:

$$\mathbf{tr}(a) = a \tag{2.4}$$

Likewise, the *determinant* $|\boldsymbol{A}|$ is also only defined for square matrices:

$$|\boldsymbol{A}| = \sum_{i} (-1)^{i+j} [\boldsymbol{A}]_{ij} \mathrm{cof}_{ij} \tag{2.5}$$

where cof_{ij} denotes the *Cofactor* of $\boldsymbol{A}$ at position (i, j). The (i, j)-th Cofactor of any squared matrix $\boldsymbol{A}$ is the determinant of a matrix obtained from $\boldsymbol{A}$ by deleting the row i and column j. For two matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ of the same size, *matrix addition* is defined element-wise. The $m \times n$ zero matrix $\mathbf{0}$, for which every entry is 0, is the neutral element of addition.

For an $m \times n$ matrix $\boldsymbol{A}$ and a $n \times p$ matrix $\boldsymbol{B}$, the *matrix product* $\boldsymbol{AB}$ is an $m \times p$ matrix with elements

$$[\boldsymbol{AB}]_{ij} = \sum_{k=1}^{n} [\boldsymbol{A}]_{ik} [\boldsymbol{B}]_{kj}, \quad i = 1 \dots m,\ j = 1 \dots p \tag{2.6}$$

The neutral element for matrix multiplication is the $n \times n$ *identity matrix* $\mathbf{I}_n$, usually denoted $\mathbf{I}$ when there is no ambiguity over dimensionality. The identity matrix has elements $[\mathbf{I}]_{ij} = \delta_{ij}$, where δ_{ij} is the Kronecker delta:

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{otherwise.} \end{cases} \tag{2.7}$$

The identity matrix is a special case of a *diagonal matrix* which has only non-zero elements on its diagonal. A diagonal matrix is sometimes denoted by $\text{diag}(d_1, d_2, \ldots, d_n)$. The matrix-vector product $\mathbf{D}\boldsymbol{x}$ between a diagonal matrix $\mathbf{D} \equiv \text{diag}(d_1, d_2, \ldots, d_n)$ and some vector $\boldsymbol{x} \equiv (x_1, x_2, \ldots, x_n)^\mathsf{T}$ is $(d_1 x_1, d_2 x_2, \ldots, d_n x_n)^\mathsf{T}$ and can be written compactly as the product between the vector $\mathbf{d}$ of the diagonal elements of $\mathbf{D}$ and $\boldsymbol{x}$, that is

$$\mathbf{D}\boldsymbol{x} = \mathbf{d} \odot \boldsymbol{x} \tag{2.8}$$

The identity matrix is also a special case of a *Toeplitz matrix* [166]. A Toeplitz matrix has constant values on each descending diagonal from left to right. An $n \times m$ matrix $\mathbf{A}$ is a Toeplitz matrix iff $\mathbf{A}_{ij} = \mathbf{A}_{i+1,j+1}$, i.e. it has the following form:

$$\begin{pmatrix} a_0 & a_{-1} & a_{-2} & \cdots & \cdots & a_{-m+1} \\ a_1 & a_0 & a_{-1} & \ddots & \ddots & a_{-m+2} \\ a_2 & a_1 & a_0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & \ddots & \ddots & a_1 & a_0 & a_{-1} \\ a_{n-1} & a_{n-2} & \cdots & a_2 & a_1 & a_0 \end{pmatrix} \tag{2.9}$$

The *Kronecker product* $\mathbf{A} \otimes \mathbf{B}$ between $\mathbf{A} \in \mathbf{R}^{m \times n}$ and $\mathbf{B} \in \mathbf{R}^{p \times q}$ is the $mp \times nq$ matrix defined by

$$\mathbf{A} \otimes \mathbf{B} := \begin{pmatrix} \mathbf{A}_{11}\mathbf{B} & \cdots & \mathbf{A}_{1n}\mathbf{B} \\ \vdots & & \vdots \\ \mathbf{A}_{m1}\mathbf{B} & & \mathbf{A}_{mn}\mathbf{B} \end{pmatrix} \tag{2.10}$$

The *inverse* of a square matrix $\mathbf{A}$, denoted $\mathbf{A}^{-1}$ satisfies $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. It is not always possible to find an inverse, in which case $\mathbf{A}$ is called *singular*. If $\mathbf{A}$ is singular, $|\mathbf{A}| = 0$. Note that the inverse (if it exists) of a symmetric matrix is also symmetric. There is a convoluted (but useful) way of describing the inverse of a matrix $\mathbf{A}$ using cofactors:

$$[\mathbf{A}^{-1}]_{ij} = \text{cof}_{ji}/|\mathbf{A}| \tag{2.11}$$

Computing inverses for matrices is usually expensive (it scales cubically with the number of rows). If a matrix has a special structure then the inverse can be computed more efficiently, using the *Sherman-Morrison-Woodbury* formula. Let $\mathbf{A} \in \mathbf{R}^{n \times n}$, $\mathbf{U} \in \mathbf{R}^{n \times k}$, $\mathbf{C} \in \mathbf{R}^{k \times k}$ and $\mathbf{V} \in \mathbf{R}^{k \times n}$ then the inverse of the *rank* k *correction of* $\mathbf{A}$ is

$$(\mathbf{A} + \mathbf{UCV})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{C}^{-1} + \mathbf{V}\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}\mathbf{A}^{-1} \quad (2.12)$$

This is cheap to compute if the inverse of $\mathbf{A}$ is already known or simple to evaluate (e.g. because $\mathbf{A}$ is diagonal). In this case only matrices of shape $k \times k$ need to inverted which eases the computational burden a lot if $k \ll n$.

A special from of matrix multiplication is the *scalar product* between a row vector $\mathbf{x}^T$ and a column vector $\mathbf{y}$

$$\mathbf{x}^T\mathbf{y} = \sum_{i=1}^{n} x_i y_i \quad (2.13)$$

The scalar product has a natural geometric interpretation as

$$\mathbf{x}^T\mathbf{y} = |\,\mathbf{x}\,|\,|\,\mathbf{y}\,|\cos(\alpha) \quad (2.14)$$

where $|\cdot|$ denotes the length of a vector and α the angle between the two vectors. The *squared length* of a vector $\mathbf{x}$ is defined as

$$|\mathbf{x}|^2 = \mathbf{x}^T\mathbf{x} = \sum_i x_i^2 \quad (2.15)$$

and $\mathbf{x}$ is denoted a *unit vector* if $\mathbf{x}^T\mathbf{x} = 1$. The length of a vector is a special case of the p-*norm* of a vector $\mathbf{x}$ for $p = 2$ (also called *Euclidean norm*). The p-norm of $\mathbf{x}$ is defined as

$$\|\mathbf{x}\|_p = \left(\sum_i x_i^p\right)^{1/p} \quad (2.16)$$

Sort of the opposite operation to the scalar product is the *outer product* $\mathbf{x}\mathbf{y}^T$ between two *arbitrary* vectors $\mathbf{x} \in \mathbf{R}^m$ and $\mathbf{y} \in \mathbf{R}^n$. It is an $m \times n$ *rank-1* matrix and is, according to the general matrix-matrix product (Eq. (2.6)), given by

$$[\mathbf{x}\mathbf{y}^T]_{ij} = x_i y_j \quad (2.17)$$

In the case of a sum over outer products, matrices can greatly compactify the expression (and also make it more amendable to the general tools of Linear Algebra). So let

$$\mathbf{M} = \sum_{i=1}^{n} \mathbf{x}_i \mathbf{y}_i{}^T \quad (2.18)$$

with $\mathbf{x}_i \in \mathbf{R}^n$ and $\mathbf{y}_i \in \mathbf{R}^m$ for $1 \leqslant i \leqslant n$. The element $\mathbf{M}_{kl}$ is given by

$$\mathbf{M}_{kl} = \sum_{i=1}^{n} \mathbf{x}_{ik}\mathbf{y}_{il} \tag{2.19}$$

which can be interpreted as the scalar product between the k-th row of a matrix $\mathbf{X}$ having the n vectors $\mathbf{x}_i$ as *columns* and the l-th column of a matrix $\mathbf{Y}$ having the n vectors $\mathbf{y}_i$ as *rows*. That is, with

$$\mathbf{X} = \begin{pmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \\ | & | & & | \end{pmatrix} \tag{2.20}$$

and

$$\mathbf{Y} = \begin{pmatrix} — & \mathbf{y}_1{}^T & — \\ — & \mathbf{y}_2{}^T & — \\ & \vdots & \\ — & \mathbf{y}_n{}^T & — \end{pmatrix} \tag{2.21}$$

Eq. (2.18) can be written as

$$\sum_{i=1}^{n} \mathbf{x}_i \mathbf{y}_i{}^T = \mathbf{X}\mathbf{Y} \tag{2.22}$$

For an $n \times n$ matrix $\mathbf{A}$, the *eigenvector equation* is defined by

$$\mathbf{A}\mathbf{u}_i = \lambda_i \mathbf{u}_i \tag{2.23}$$

for $i = 1 \ldots n$, where $\mathbf{u}_i$ is an *eigenvector* and λ_i the corresponding *eigenvalue*. In general, the eigenvalues are complex numbers (and hence, the eigenvectors are vectors with complex entries), but for symmetric matrices the eigenvalues are real. In this case Equation 2.23 is often denoted the *spectral composition*. It can be written more compactly as a matrix-matrix operation

$$\mathbf{A}\mathbf{U} = \mathbf{U}\mathbf{\Delta} \tag{2.24}$$

or

$$\mathbf{A} = \mathbf{U}\mathbf{\Delta}\mathbf{U}^T \tag{2.25}$$

where $\mathbf{\Delta}$ is the diagonal matrix of eigenvalues. The last equation can also be written as a weighted sum of outer products formed by the eigenvectors (see also Eq. (2.22)):

$$\mathbf{A} = \sum_{i=1}^{n} \lambda_i \mathbf{u}_i \mathbf{u}_i^T \tag{2.26}$$

While the spectral decomposition only exists for real valued, symmetric matrices, any $m \times n$ matrix $\mathbf{A}$ can be decomposed using the *Singular Value Decomposition* (SVD):

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{2.27}$$

where $\mathbf{U}$ is an $n \times n$ orthogonal matrix, $\mathbf{V}$ an $m \times m$ orthogonal matrix and $\mathbf{S}$ a diagonal matrix, with all three matrices having only real valued entries. The entries of $\mathbf{S}$ are called the *singular values* of $\mathbf{A}$ and are non-negative. The singular values are actually the roots of the eigenvalues of $\mathbf{A}\mathbf{A}^T$ (and also of $\mathbf{A}^T\mathbf{A}$, so there can be at most $\min(n, m)$ non-zero singular values).

The *norm* $\|\cdot\|$ of a matrix is a generalization of the concept of a norm of a vector [118]. Vector norms are used to define *operator norms* on a matrix $\mathbf{A}$ in the following way:

$$\|\mathbf{A}\| := \sup_{\mathbf{x}\neq 0} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|} \tag{2.28}$$

If the p-norm for vectors (Eq. (2.16)) is used as a basis, the special case of $p = 2$, the *spectral norm*, is defined as the square root of the largest eigenvalue of $\mathbf{A}^*\mathbf{A}$:

$$\|\mathbf{A}\|_2 = \sqrt{\lambda_{\max}(\mathbf{A}^*\mathbf{A})} = \sigma_{\max}(\mathbf{A}) \tag{2.29}$$

$\sigma_{\max}(\mathbf{A})$ is the largest singular value of $\mathbf{A}$. Another widely used norm is the Frobenius norm $\|\cdot\|_F$, defined as

$$\|\mathbf{A}\|_F = \sqrt{\sum_i \sum_j |A_{ij}|^2} \tag{2.30}$$

A matrix norm $\|\cdot\|$ is *consistent* with its inducing vector norm if

$$\|\mathbf{A}\mathbf{x}\| \leqslant \|\mathbf{A}\|\|\mathbf{x}\|. \tag{2.31}$$

In this case the matrix norm is also *submultiplicative*, that is

$$\|\mathbf{A}\mathbf{B}\| \leqslant \|\mathbf{A}\|\|\mathbf{B}\| \tag{2.32}$$

for $\mathbf{A}$ and $\mathbf{B}$ being conformable matrices. The norms defined above are both submultiplicative, the spectral norm is also consistent.

ALGEBRAIC PROPERTIES. The trace operator has a very useful cyclic property,

$$\mathbf{tr}(\mathbf{A}\mathbf{B}) = \sum_{i=1}^{m}\sum_{k=1}^{m} [\mathbf{A}]_{ik}[\mathbf{B}]_{ki} = \sum_{k=1}^{n}\sum_{i=1}^{m} [\mathbf{B}]_{ki}[\mathbf{A}]_{ik} = \mathbf{tr}(\mathbf{B}\mathbf{A}) \tag{2.33}$$

which clearly extends to the product of any number of matrices. Applying this to the eigenvalue decomposition of a symmetric matrix $\mathbf{A}$ one gets:

$$\mathbf{tr}(\mathbf{A}) = \mathbf{tr}(\mathbf{U}\mathbf{\Delta}\mathbf{U}^T) = \sum_i \lambda_i \tag{2.34}$$

The determinant of a product of two matrices is given by

$$|\boldsymbol{A}\boldsymbol{B}| = |\boldsymbol{A}||\boldsymbol{B}| \tag{2.35}$$

and thus

$$|\boldsymbol{A}|^{-1} = |\boldsymbol{A}|^{-1} \tag{2.36}$$

Applied to the eigenvalue decomposition of a symmetric matrix $\boldsymbol{A}$this means

$$|\boldsymbol{A}| = \prod_{i=1}^{n} \lambda_i \tag{2.37}$$

$\boldsymbol{A}$ therefore is positive definite if and only if all its eigenvalues are positive.

TENSORS. *Tensors* are a generalization of matrices to multidimensional arrays (like matrices are a generalization of vectors). Tensors are becoming more and more relevant in Machine Learning, as they encode valuable structural information e.g. [6, 277], though usually tensors are vectorized and then treated within the simpler Linear Algebra framework for matrices. It is possible to define generalizations of matrix products and other operators to tensors, but these are not used in this work and therefore left out. One operator is however briefly used and it is strongly related to a geometric view of tensors: The *point reflection* $\boldsymbol{A}^{\circ}$ of an $n_1 \times n_2 \times \cdots \times n_m$ tensor $\boldsymbol{A}$ is defined elementwise as

$$[\boldsymbol{A}^{\circ}]_{i_1,i_2,\cdots,i_m} = [\boldsymbol{A}]_{n_1-i_1,n_2-i_2,\cdots,n_m-i_m} \tag{2.38}$$

2.2 MULTIVARIATE AND MATRIX CALCULUS

Assuming familiarity with one dimensional calculus [205, 147], this section succinctly reiterates some important aspects of Multivariate Calculus and also briefly addresses some concepts of Matrix Calculus [243].

A general multivariate function $f(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m)$ maps points from $\mathbf{R}^m$ to $\mathbf{R}^n$, i.e. $f : \mathbf{R}^m \to \mathbf{R}^n$. An important special case for Machine Learning is given by $n = 1$ which represents the situation that a scalar loss is induced by an m-dimensional parameter setting on a given training set. In this case the partial derivative of $f(\boldsymbol{x})$ with respect to $\boldsymbol{x}_i$ is defined as the following limit (when it exists):

$$\frac{\partial f}{\partial \boldsymbol{x}_i} = \lim_{h \to 0} \frac{f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{i-1}, \boldsymbol{x}_i + h, \boldsymbol{x}_{i+1}, \ldots, \boldsymbol{x}_m) - f(\boldsymbol{x})}{h} \tag{2.39}$$

and induces the *gradient* on f, denoted ∇f:

$$\nabla f(\boldsymbol{x}) \equiv \begin{pmatrix} \frac{\partial f}{\partial \boldsymbol{x}_1} \\ \vdots \\ \frac{\partial f}{\partial \boldsymbol{x}_m} \end{pmatrix} \tag{2.40}$$

∇f is a vector in the input domain and points (locally) along the direction in which the function increases most rapidly. This can be easily seen by considering a first order *Taylor expansion* of f at some point $\mathbf{x}$:

$$\begin{aligned} f(\mathbf{x}+\boldsymbol{\delta}) &= f(\mathbf{x}) + \sum_i \delta_i \frac{\partial f_i}{\partial x_i} + O(\boldsymbol{\delta}^2) \\ &= f(\mathbf{x}) + (\nabla f)^T \boldsymbol{\delta} + O(\boldsymbol{\delta}^2)\,, \qquad |\boldsymbol{\delta}|^2 \ll 1 \end{aligned} \tag{2.41}$$

The scalar product $(\nabla f)^T \boldsymbol{\delta}$ (and therefore the local Taylor expansion) is maximized, if the angle between ∇f and $\boldsymbol{\delta}$ is 0. Hence, the direction along which the function changes the most rapidly is along $\nabla f(\mathbf{x})$. This implies a simple iterative optimization scheme (*steepest gradient ascent* [51, 278]): Maximizing $f(\mathbf{x})$ (i.e. find a $\mathbf{x}'$ such that $f(\mathbf{x}')$ is maximal, at least locally) can be done with repeatedly following the gradient until it vanishes:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \eta_n \nabla f(\mathbf{x}_n) \tag{2.42}$$

with $\mathbf{x}_0$ chosen arbitrarily. η_n is denoted the *step size at iteration* n and needs to be chosen small enough (otherwise the Taylor approximation does not hold).

Given the gradient $\nabla f(\mathbf{x})$ the derivative of $f(\mathbf{x})$, $f'(\mathbf{x})$, is defined as

$$f'(\mathbf{x}) = (\nabla f(\mathbf{x}))^T \tag{2.43}$$

For an arbitrary vector valued function $f : \mathbf{R}^m \to \mathbf{R}^n$ the derivative of f is given by stacking the n derivative row vectors on top of each other, resulting in the $n \times m$ *Jacobian* Df at $\mathbf{x}$:

$$Df(\mathbf{x}) = f'(\mathbf{x}) = \frac{\partial f}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{pmatrix} \tag{2.44}$$

As an example consider the following vector function $f : \mathbf{R}^n \to \mathbf{R}^n$ which simply applies scalar functions element-wise:

$$[f(\mathbf{x})]_i = g_i(x_i), \quad \text{with } g_i : \mathbf{R} \to \mathbf{R}, \quad \forall i \in \{1, \ldots, n\} \tag{2.45}$$

Then $f'(\mathbf{x})$ is an $n \times n$ diagonal matrix with the element-wise derivatives on the diagonal:

$$[f'(\mathbf{x})]_{ij} = \begin{cases} g_i'(x_i), & \text{if } i = j \\ 0, & \text{otherwise.} \end{cases} \tag{2.46}$$

On more formal grounds, the Jacobian is the $n \times m$ matrix $\mathbf{A}$ such that $f(\mathbf{x}+\mathbf{u})$, with $\|\mathbf{u}\| < \varepsilon$, can be written as

$$f(\mathbf{x}+\mathbf{u}) = f(\mathbf{x}) + \mathbf{A}\mathbf{u} + \mathbf{r}(\mathbf{u}) \tag{2.47}$$

with

$$\lim_{\mathbf{u}\to 0} \frac{\mathbf{r}(\mathbf{u})}{\|\mathbf{u}\|} = 0 \tag{2.48}$$

With this definition the summation rule and the chain rule generalize from one dimensional calculus to arbitrary vector valued functions (the product rule and quotient rule only generalize to scalar functions without any additional restrictions). So for the summation rule, let $f_1, f_2 : \mathbf{R}^m \to \mathbf{R}^n$, then

$$[f_1(\mathbf{x}) + f_2(\mathbf{x})]' = f_1'(\mathbf{x}) + f_2'(\mathbf{x}) \tag{2.49}$$

For the chain rule let $f : \mathbf{R}^n \to \mathbf{R}^r$ and $g : \mathbf{R}^m \to \mathbf{R}^n$, then

$$Df(g(\mathbf{x})) = (f(g(\mathbf{x})))' = (f'(g(\mathbf{x})))g'(\mathbf{x}), \tag{2.50}$$

an $m \times r$ dimensional Jacobian (assuming that $f'(\cdot)$ and $g'(\cdot)$ exist). Specifically, if $f : \mathbf{R}^n \to \mathbf{R}$ then

$$(f(g(\mathbf{x})))' = (\nabla f(g(\mathbf{x})))^T g'(\mathbf{x}) \tag{2.51}$$

An important implication of the chain rule is the method for integration by substitution. Let $f : \mathbf{R}^m \to \mathbf{R}^n$ and $g : \mathbf{R}^n \to \mathbf{R}^n$, an injective and continuous differentiable function with its determinant $|g'(\mathbf{x})|$ always positive or always negative. Then

$$\int f(\mathbf{x})d\mathbf{x} = \int f(g(\mathbf{x}))|g'(\mathbf{t})|d\mathbf{t} \tag{2.52}$$

This is also true if g is not injective over the integration volume or if its determinant is (only sometimes) 0 or changes signs. In the later case the integration must be split up over volumina where the sign is constant.

MATRIX CALCULUS. The concept of a vector function can be extended to a *matrix function* [243]. A matrix function F is a function that maps from $\mathbf{R}^{m\times n}$ to $\mathbf{R}^{q\times r}$ for arbitrary m, n, q, r. In order to define the (first) derivative of a matrix function, a new operator is necessary: $\text{vec}(\cdot)$. If $\mathbf{A}$ is an $m \times n$ matrix, with $\mathbf{a}_i$ its i-th *column*, then $\text{vec}(\mathbf{A})$ is the $mn \times 1$ vector obtained by *stacking* the columns of $\mathbf{A}$ underneath each other:

$$\text{vec}(\mathbf{A}) = \begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_n \end{pmatrix} \tag{2.53}$$

For a vector $\mathbf{a}$ the $\text{vec}(\cdot)$ operation is invariant:

$$\text{vec}(\mathbf{a}) = \text{vec}(\mathbf{a}^T) = \mathbf{a} \tag{2.54}$$

The inverse operator to $\text{vec}(\cdot)$ is denoted by $\text{cev}(\cdot, n, m)$, taking an nm-dimensional vector $\mathbf{a}$ and building an $n \times m$ matrix by aligning the n-dimensional columns:

$$\text{vec}(\text{cev}(\mathbf{a}, n, m)) = \mathbf{a} \tag{2.55}$$

A function $F : \mathbf{R}^{m \times n} \to \mathbf{R}^{p \times q}$ is differentiable at $\mathbf{C} \in \mathbb{R}^{m \times n}$ if, for some $\mathbf{U} \in \mathbb{R}^{m \times n}$ with $\|\mathbf{U}\| < \varepsilon$, there exists a real $qr \times mn$ matrix $\mathbf{A}(\mathbf{C})$ (i.e. depending on $\mathbf{C}$ but not on $\mathbf{U}$) such that

$$\text{vec}(F(\mathbf{C} + \mathbf{U})) = \text{vec}(F(\mathbf{C})) + \mathbf{A}(\mathbf{C})\text{vec}(\mathbf{U}) + \text{vec}(\mathbf{R}_{\mathbf{C}}(\mathbf{U})) \tag{2.56}$$

for all $\mathbf{U} \in \mathbf{R}^{m \times n}$ with $\|\mathbf{U}\| < \varepsilon$ and

$$\lim_{\mathbf{U} \to 0} \frac{\mathbf{R}_{\mathbf{C}}(\mathbf{U})}{\|\mathbf{U}\|} = 0 \tag{2.57}$$

The *first differential of* F *at* $\mathbf{C}$ *with increment* $\mathbf{U}$ is the $m \times p$ matrix $dF(\mathbf{C}; \mathbf{U})$ defined by

$$\text{vec}(dF(\mathbf{C}; \mathbf{U})) := \mathbf{A}(\mathbf{C})\text{vec}(\mathbf{U}). \tag{2.58}$$

Then, the *first derivative of* F *at* $\mathbf{C}$, $F'(\mathbf{X})$, is the $pq \times mn$ matrix $\mathbf{A}(\mathbf{C})$.

Instead of the matrix function F one can consider the equivalent vector function $f : \mathbf{R}^{mn} \to \mathbf{R}^{pq}$, defined by

$$f(\text{vec}(\mathbf{X})) = \text{vec}(F(\mathbf{X})) \tag{2.59}$$

With this mapping all calculus properties of vector functions can be transferred to corresponding properties of matrix functions. Following [243] the definition of the Jacobian of F at $\mathbf{X}$ is the $pq \times mn$ matrix

$$F'(\mathbf{X}) = DF(\mathbf{X}) = \frac{\partial \text{vec}(F(\mathbf{X}))}{\partial (\text{vec}(\mathbf{X})^T)} \tag{2.60}$$

Among other things this definition ensures that Eq. (2.48) is still correct if vector-valued functions are considered as special cases of matrix functions. It is often convenient to work with both Eq. (2.48) and Eq. (2.60) depending on the formulation of the problem.

Let $F : \mathbf{R}^{m \times n} \to \mathbf{R}^{p \times q}$ be differentiable at $\mathbf{C}$ and let $G : \mathbf{R}^{p \times q} \to \mathbf{R}^{r \times s}$ be differentiable at $\mathbf{B} = F(\mathbf{C})$. Then $H : \mathbf{R}^{m \times n} \to \mathbf{R}^{r \times s}$ defined by

$$H(\mathbf{X}) = G(F(\mathbf{X})) \tag{2.61}$$

is differentiable at $\mathbf{C}$ according to the chain rule for matrix functions and

$$H'(\mathbf{C}) = DH(\mathbf{C}) = G'(\mathbf{B})F'(\mathbf{C}) \tag{2.62}$$

One strategy to find the Jacobian of a matrix function is to evaluate each of its partial derivatives (akin to Eq. (2.48)). A more elegant, and

often more concise and easier way is to find the differential and then match terms according to Eq. (2.58).

As an example consider the following vector-valued linear matrix function $F : \mathbf{R}^{m\times n} \to \mathbf{R}^m$, with $\mathbf{a} \in \mathbf{R}^n$ a constant vector

$$f(\mathbf{X}) = \mathbf{X}\mathbf{a} \tag{2.63}$$

Following Eq. (2.56) it is straightforward to determine F':

$$\text{vec}((\mathbf{X}+\mathbf{U})\mathbf{a}) = (\mathbf{X}+\mathbf{U})\mathbf{a} = \underbrace{\mathbf{X}\mathbf{a}}_{\text{vec}(\mathbf{X}\mathbf{a})} + \underbrace{\mathbf{U}\mathbf{a}}_{F'(\mathbf{X})\text{vec}(\mathbf{U})} \tag{2.64}$$

$\text{vec}(\mathbf{U}) \in \mathbf{R}^{mn}$, so $F'(\mathbf{X})$, an $m \times mn$ matrix, must have the following form:

$$\begin{pmatrix} a_1 & 0 & \cdots & 0 & a_2 & 0 & \cdots & 0 & \cdots & a_n & 0 & \cdots & 0 \\ 0 & a_1 & 0 & 0 & 0 & a_2 & 0 & 0 & \cdots & 0 & a_n & 0 & 0 \\ \vdots & & & \vdots & \vdots & & & \vdots & \cdots & \vdots & & & \vdots \\ 0 & \cdots & & a_1 & 0 & \cdots & & a_2 & \cdots & 0 & \cdots & & a_n \end{pmatrix} \tag{2.65}$$

The Kronecker product (Eq. (2.10)) allows a more compact expression:

$$F'(\mathbf{X}) = \mathbf{a}^T \otimes \mathbf{I}_m \tag{2.66}$$

Note that for some vector $\mathbf{c} \in \mathbf{R}^m$ the expression $\mathbf{c}^T(\mathbf{a}^T \otimes \mathbf{I}_m)$, a $1 \times mn$ matrix, can be written as

$$\mathbf{c}^T(\mathbf{a}^T \otimes \mathbf{I}_m) = \text{vec}(\mathbf{c}\mathbf{a}^T)^T \tag{2.67}$$

On the other hand the vector-valued linear vector function $f : \mathbf{R}^n \to \mathbf{R}^m$, with $\mathbf{A} \in \mathbf{R}^{m\times n}$ a constant matrix,

$$f(\mathbf{x}) = \mathbf{A}\mathbf{x} \tag{2.68}$$

has a simple derivative f':

$$f'(\mathbf{x}) = \mathbf{A} \tag{2.69}$$

Two similarly simple examples are the following scalar-valued vector functions. First, consider $f(\mathbf{x}) = \mathbf{a}^T\mathbf{x}$, where $\mathbf{x} \in \mathbf{R}^n$ and $\mathbf{a} \in \mathbf{R}^n$, a constant. The differential is simply $\mathbf{a}^T\mathbf{u}$ (using $\mathbf{u}$ instead of $\mathbf{U}$), so, with Eq. (2.54),

$$f'(\mathbf{x}) = \mathbf{a}^T. \tag{2.70}$$

Second, consider the quadratic form $f(\mathbf{x}) = \mathbf{x}^T\mathbf{A}\mathbf{x}$. The differential is given by (relying on the fact that the transpose is invariant to scalars)

$$\mathbf{u}^T\mathbf{A}\mathbf{x} + \mathbf{x}^T\mathbf{A}\mathbf{u} = (\mathbf{u}^T\mathbf{A}\mathbf{x})^T + \mathbf{x}^T\mathbf{A}\mathbf{u} = \mathbf{x}^T(\mathbf{A}^T + \mathbf{A})\mathbf{u} \tag{2.71}$$

Again, with Eq. (2.54) it follows that

$$f'(\mathbf{x}) = \mathbf{x}^T(\mathbf{A}^T + \mathbf{A}). \tag{2.72}$$

If $\mathbf{A}$ is symmetric then

$$f'(\mathbf{x}) = 2\mathbf{x}^T\mathbf{A}. \tag{2.73}$$

The trace operator is an often-used scalar-valued matrix function. The most basic function to be considered first is

$$F(\mathbf{X}) = \mathbf{tr}(\mathbf{AX}) \tag{2.74}$$

with $\mathbf{X} \in \mathbf{R}^{n\times m}$ and a constant $\mathbf{A} \in \mathbf{R}^{m\times n}$, the first differential is simply $\mathrm{tr}(\mathbf{AU})$. $F'(\mathbf{X})$ must be a $1 \times nq$ row vector because $\mathrm{vec}(\mathbf{U}) \in \mathbf{R}^{nm}$. With

$$\mathrm{tr}(\mathbf{AU}) = \sum_i \sum_j \mathbf{A}_{ij}\mathbf{U}_{ji} \tag{2.75}$$

it follows that

$$F'(\mathbf{X}) = \mathrm{vec}(\mathbf{A}^T)^T \tag{2.76}$$

This derivative is very helpful in combination with the cyclic property of the trace operator (Eq. (2.33)). Let $F(\mathbf{X}) = \mathbf{a}^T\mathbf{X}\mathbf{a}$ with $\mathbf{a} \in \mathbf{R}^n$ being a constant vector and $\mathbf{X} \in \mathbf{R}^{n\times n}$. Using Eq. (2.4) and Eq. (2.33) rewrite $F(\mathbf{X})$ as

$$F(\mathbf{X}) = \mathbf{a}^T\mathbf{X}\mathbf{a} = \mathbf{tr}(\mathbf{a}^T\mathbf{X}\mathbf{a}) = \mathbf{tr}(\mathbf{a}\mathbf{a}^T\mathbf{X}) \tag{2.77}$$

With Eq. (2.76) one gets

$$F'(\mathbf{X}) = (\mathrm{vec}(\mathbf{a}\mathbf{a}^T))^T \tag{2.78}$$

A more complicated expression involving the trace operator is

$$F(\mathbf{X}) = \mathrm{tr}(\mathbf{XAX}^T\mathbf{B}) \tag{2.79}$$

with $\mathbf{X}, \mathbf{A}$ and $\mathbf{B}$ conformable matrices. The first differential in this case is

$$\mathrm{tr}(\mathbf{XAU}^T\mathbf{B}) + \mathrm{tr}(\mathbf{UAX}^T\mathbf{B}) \tag{2.80}$$

Relying on the cyclic property of $\mathrm{tr}(\cdot)$ and utilizing Eq. (2.76) it follows that

$$F'(\mathbf{X}) = \mathrm{vec}(\mathbf{AX}^T\mathbf{B} + \mathbf{A}^T\mathbf{X}^T\mathbf{B}^T)^T \tag{2.81}$$

Finally, an important derivative for a symmetric matrix $\mathbf{A}$ is $\frac{\partial|\mathbf{A}|}{\partial\mathbf{A}}$. For a symmetric matrix $\mathbf{A}$, $\mathrm{cof}_{ij} = \mathrm{cof}_{ji}$ and so

$$\frac{d|\mathbf{A}|}{d[\mathbf{A}]_{ij}} = \mathrm{cof}_{ij} = \mathrm{cof}_{ji} = |\mathbf{A}|[\mathbf{A}^{-1}]_{ij} \tag{2.82}$$

Combining these element-wise derivatives into a matrix, we get

$$\frac{\partial |\boldsymbol{A}|}{\partial \boldsymbol{A}} = |\boldsymbol{A}| \boldsymbol{A}^{-1} \tag{2.83}$$

and similarly

$$\frac{\partial |\boldsymbol{A}^{-1}|}{\partial \boldsymbol{A}} = -|\boldsymbol{A}^{-1}| \boldsymbol{A}^{-1} \tag{2.84}$$

Applying the chain rule, one derives

$$\frac{\partial \ln |\boldsymbol{A}|}{\partial \boldsymbol{A}} = \boldsymbol{A}^{-1} \tag{2.85}$$

and with $|\boldsymbol{A}^{-1}| = |\boldsymbol{A}|^{-1}$ it follows that

$$\frac{\partial \ln |\boldsymbol{A}|}{\partial \boldsymbol{A}^{-1}} = -\frac{\partial \ln |\boldsymbol{A}^{-1}|}{\partial \boldsymbol{A}^{-1}} = -\boldsymbol{A} \tag{2.86}$$

2.3 PROBABILITY THEORY

Our model of the world around us will always be limited by our observations and understanding. Having only imperfect knowledge of the world, we are forced to take uncertainty into account. Probability theory provides a consistent framework for the quantification and manipulation of uncertainty. In what follows, I only cover informally some basic notions of probability theory. For a thorough introduction, I suggest [136].

A *random variable* X is a function $X : \Omega \to \mathbf{R}$ with the property that $\{\omega \in \Omega : X(\omega) \leqslant x\} \in \mathcal{F}$ for each $x \in \mathbf{R}$. Ω denotes a *sample space* [136, p. 1] and $\mathcal{F}$ denotes an underlying sigma-field [136, p. 3]. A random variable X is uniquely determined by its *distribution function* $F : \mathbf{R} \to [0, 1]$, given by $F_X(x) = P(X \leqslant x)$. $P(\cdot)$ denotes a probability measure [136, p. 5] that is defined over Ω and $\mathcal{F}$. A random variable is not limited to mapping only to the real line. Instead, it can be extended easily to the idea of a *random vector*. The *joint distribution function* of a random vector $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ is the function $F_{\mathbf{X}} : \mathbf{R}^n \to [0, 1]$, given by $F_{\mathbf{X}}(\mathbf{x}) = P(\mathbf{X} \leqslant \mathbf{x})$.

A random vector (which is from now on also named random variable) $\mathbf{X}$ is called *discrete* if it takes values in some countable subset $\{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$ only. The discrete random variable $\mathbf{X}$ has the *joint (probability) mass function* $p_{\mathbf{X}} : \mathbf{R}^n \to [0, 1]$ given by $p_{\mathbf{X}}(\mathbf{x}) = P(\mathbf{X} = \mathbf{x})$. The *marginal* mass function p_{X_i} can be evaluated using the *sum rule*:

$$p_{X_i}(x_i) = \sum_{\mathbf{x}_{-i}} p_{\mathbf{X}}(x_i, \mathbf{x}_{-i}) \tag{2.87}$$

$\mathbf{x}_{-i}$ denotes the elements of the vector $\mathbf{x}$ except the i-th element. We denote n random variables $\mathbf{X}_1, \mathbf{X}_2, \ldots \mathbf{X}_n$ stochastically *independent* if and only if

$$p_{\mathbf{X}}(\mathbf{X}) = \prod_{i=1}^{n} p_{X_i}(x_i) \qquad \forall\, \mathbf{x}_i \in \mathbf{R} \tag{2.88}$$

The *conditional distribution function* of $\mathbf{Y}$ given $\mathbf{X} = \mathbf{x}$, written $F_{\mathbf{Y}|\mathbf{X}}(\cdot|\mathbf{x})$ is defined by

$$F_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) = P(\mathbf{Y} \leqslant \mathbf{y}|\mathbf{X} = \mathbf{x}) \tag{2.89}$$

for any $\mathbf{x}$ such that $P(\mathbf{X} = \mathbf{x}) > 0$. The *conditional (probability) mass function* of $\mathbf{Y}$ given $\mathbf{X} = \mathbf{x}$, written $p_{\mathbf{Y}|\mathbf{X}}(\cdot|\mathbf{x})$ is defined by

$$p_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) = P(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x}) \tag{2.90}$$

Using this definition the *product rule* is:

$$p_{\mathbf{X},\mathbf{Y}}(\mathbf{x}, \mathbf{y}) = p_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x})p_{\mathbf{X}}(\mathbf{x}) = p_{\mathbf{X}|\mathbf{Y}}(\mathbf{x}|\mathbf{y})p_{\mathbf{Y}}(\mathbf{y}) \tag{2.91}$$

The inherent symmetry of the product rule then induces the Bayes' rule:

$$p_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}) = \frac{p_{\mathbf{X}|\mathbf{Y}}(\mathbf{x}|\mathbf{y})p_{\mathbf{Y}}(\mathbf{y})}{p_{\mathbf{X}}(\mathbf{x})} \tag{2.92}$$

The *expectation* of a random variable $\mathbf{X}$ with joint probability mass function $p_{\mathbf{X}}$ under any $g : \mathbf{R}^n \to \mathbf{R}^m$ is

$$\mathbb{E}[g(\mathbf{X})] = \sum_{\mathbf{x}} g(\mathbf{x})p_{\mathbf{X}}(\mathbf{x}) \tag{2.93}$$

whenever this sum is absolutely convergent. The expectation operator $\mathbb{E}[\cdot]$ is a linear operator in the space of random variables:

- $\mathbb{E}[\mathbf{c}] = \mathbf{c}$ for any constant $\mathbf{c}$.
- If $\mathbf{A} \in \mathbf{R}^{m\times n}$, then $\mathbb{E}[\mathbf{AX}] = \mathbf{A}\mathbb{E}[\mathbf{X}]$ and $\mathbb{E}\left[\mathbf{XA}^T\right] = \mathbb{E}[\mathbf{X}]\mathbf{A}^T$.
- $\mathbb{E}[\mathbf{X} + \mathbf{Y}] = \mathbb{E}[\mathbf{X}] + \mathbb{E}[\mathbf{Y}]$.

If f is convex and $\mathbf{X}$ is a discrete random variable with finite mean then

$$\mathbb{E}[f(\mathbf{x})] \leqslant f(\mathbb{E}[\mathbf{x}]), \tag{2.94}$$

denoted *Jensen inequality*[185].

The *covariance matrix* $\mathbb{V}[\mathbf{X}]$ is defined as

$$\begin{aligned}\mathbb{V}[\mathbf{X}] &= \mathbb{E}\left[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^T\right] \\ &= \mathbb{E}\left[\mathbf{XX}^T\right] - \mathbb{E}[\mathbf{X}]\mathbb{E}[\mathbf{X}]^T\end{aligned} \tag{2.95}$$

The following simple properties are very useful to know:

- $\mathbb{V}[\mathbf{c}] = 0$ for any constant $\mathbf{c}$.
- $\mathbb{V}[c\mathbf{X}] = c^2\mathbb{V}[\mathbf{X}]$.

The *covariance matrix* $\mathbb{V}[\mathbf{X}, \mathbf{Y}]$ between two random variables $\mathbf{X}$ and $\mathbf{Y}$ is defined as

$$\begin{aligned}\mathbb{V}[\mathbf{X}, \mathbf{Y}] &= \mathbb{E}\left[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{Y} - \mathbb{E}[\mathbf{Y}])^T\right] \\ &= \mathbb{E}\left[\mathbf{X}\mathbf{Y}^T\right] - \mathbb{E}[\mathbf{X}]\,\mathbb{E}[\mathbf{Y}]^T\end{aligned} \tag{2.96}$$

For a set of n random variables $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n$ the variance of its sum is simple the summed variances, if the n random variables are i.i.d:

$$\mathbb{V}\left[\sum_{i=1}^{n} \mathbf{X}_i\right] = \sum_{i=1}^{n} \mathbb{V}[\mathbf{X}_i] \tag{2.97}$$

For a set of n data points $x_1, x_2, \ldots, x_n$, which are states of a random variable $\mathbf{X}$, the *empirical distribution* $\hat{p}_X$ has probability mass distributed evenly over the data points, and zero elsewhere:

$$\hat{p}_X(x) = \frac{1}{n} \sum_{i=1}^{n} \delta(x = x_n) \tag{2.98}$$

where $\delta(x)$ is the Dirac delta function. The Dirac delta function is a distribution function and defined functionally as

$$\int_{-\infty}^{\infty} \delta(x - x_0) f(x) dx = f(x_0) \tag{2.99}$$

The *Kullback-Leibler divergence* (KL divergence) measures the difference between two discrete distributions q_X and p_X:

$$\begin{aligned}\mathcal{KL}[p_X \| q_X] &= \mathbb{E}[\log p_X(x) - \log q_X(x)] \\ &= \sum_x p_X(x) \log \frac{p_X(x)}{q_X(x)}\end{aligned} \tag{2.100}$$

The Kullback-Leibler divergence is not a valid distance metric (it is not symmetric), however it can be used as a semi-distance metric, because it is non-negative:

$$\begin{aligned}\mathcal{KL}[p_X \| q_X] = \sum_x p_X(x) \log \frac{p_X(x)}{q_X(x)} &= -\sum_x p_X(x) \log \frac{q_X(x)}{p_X(x)} \\ &\geqslant -\log \sum_x p_X(x) \frac{q_X(x)}{p_X(x)} = -\log \sum_x q_X(x) \\ &= -\ln 1 = 0\end{aligned} \tag{2.101}$$

The proof uses Jensen's inequality (Eq. (2.94)) applied to the *concave* function log. The *discrete entropy* $\mathcal{H}[p_X]$ of a probability mass function p_X is defined as

$$\mathcal{H}[p_X] = -\sum_x p_X(x) \log p_X(x) \tag{2.102}$$

It is a measure of the uncertainty in a distribution p_X. One way to see this is that $\mathcal{H}[p_X] = \mathcal{KL}[p_X \| u_X] + c$, where u_X is the uniform

distribution over $\mathbf{X}$ and $c \geqslant 0$. Because $\mathcal{KL}[p||u] \geqslant 0$, the less similar to the uniform distribution $p_{\mathbf{X}}$ is, the smaller is its entropy.

A random variable $\mathbf{X}$ is called *continuous* if its cumulative distribution function can be expressed as

$$F_{\mathbf{X}}(\mathbf{x}) = \int_{-\infty}^{\mathbf{x}} f_{\mathbf{X}}(\mathbf{u}) d\mathbf{u}, \quad \mathbf{x} \in \mathbf{R}^n \tag{2.103}$$

for some integrable function $f_{\mathbf{X}} : \mathbf{R}^n \to [0, 1]$, the *(probability) density function (pdf)* of $\mathbf{X}$. For very small $\Delta\mathbf{x}$, one can approximately compute

$$P(\mathbf{x} \leqslant \mathbf{X} \leqslant \mathbf{x} + \Delta\mathbf{x}) \approx f_{\mathbf{X}}(\mathbf{x})\Delta\mathbf{x} \tag{2.104}$$

Even though a probability density function is a much more complex object than a probability mass function, all previous results for discrete distributions also hold for continuous distributions, in particular the sum rule, the product rule, Bayes' rule and expectation results (operationally, these results are generally obtained by simply substitute the summation sign by the integral sign). Because a continuous signal carries an infinite amount of information, the entropy must be defined differently. The *differential entropy* is defined as

$$\mathcal{H}[f_{\mathbf{X}}] = -\int f_{\mathbf{X}}(\mathbf{x}) \log f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} \tag{2.105}$$

Additionally, there is one aspect for continuous random variables that has no equivalent behavior for discrete random variables. If a continuous random variable $\mathbf{X} \in \mathbf{R}^m$ gets transformed by some monotonic function $g : \mathbf{R}^m \to \mathbf{R}^m$, then the probability density function $f_{\mathbf{Y}}$ of the resulting random variable $\mathbf{Y} = g(\mathbf{X})$ has to be adapted following the change of variable theorem (Eq. (2.52)):

$$f_{\mathbf{Y}}(\mathbf{y}) = f_{\mathbf{X}}(\mathbf{x})|\mathbf{J}| \tag{2.106}$$

where $\mathbf{J}$ is the Jacobian of the inverse function $g^{-1} : \mathbf{R}^m \to \mathbf{R}^m$. If g is not monotonic over the complete space then the above rule must be applied piecewise to subsets where g is monotonic. For example, let $\mathbf{X} \in \mathbf{R}^n$ be an arbitrary continuous random variable with pdf $f_{\mathbf{X}}(\mathbf{x})$ and $\mathbf{Y} = \mathbf{D}\mathbf{X}$, with $\mathbf{D} \in \mathbf{R}^{n \times n}$ a matrix of rank n. Then

$$f_{\mathbf{Y}}(\mathbf{y}) = f_{\mathbf{X}}(\mathbf{D}^{-1}\mathbf{y})/|\mathbf{D}| \tag{2.107}$$

From now on I simplify the notation of probabilistic expressions: Instead of writing $p_{\mathbf{X}}(\mathbf{x})$ and $f_{\mathbf{X}}(\mathbf{x})$ I will use $p(\mathbf{x})$ and $f(\mathbf{x})$. That is, the variable argument to p and f also denote the underlying random variable. So for example $p(\mathbf{x})$ is a different distribution from $p(\mathbf{y})$ (which would not be the case for $p_{\mathbf{X}}(\cdot)$, because $p_{\mathbf{X}}(\mathbf{x}) \equiv p_{\mathbf{X}}(\mathbf{y})$, because $\mathbf{x}$ and $\mathbf{y}$ only denote the *variable* placeholders). Similarly, instead of differentiating between the random variable $\mathbf{X}$ and the variable argument $\mathbf{x}$ I will only use $\mathbf{x}$ and the context determines the interpretation.

Additionally a probability density function (pdf) $f(\cdot)$ will usually be also denoted by the symbol $p(\cdot)$. Sometimes, only parts of a pdf/pmf are important for a specific derivation. This will be expressed by the symbol $\propto$ (which will also be used in a non-probabilistic content to denote that unimportant terms are left out).

Important distributions

In this subsection I briefly introduce the probability distributions that are used throughout this text. The distributions are given through their probability mass functions or probability density functions, depending on their type. Distributions usually depend on parameters $\theta_1, \theta_2, \cdots, \theta_n$ denoted by $p(\cdot|\theta)$. This is consistent with the previously introduced notation of a conditional probability distribution, as these parameters can also be regarded as random variables. If a random variable x is distributed according to a probability distribution defined by $p(\cdot)$ then I often write $x \sim p(\cdot)$. And finally, if in a set of random variables $\{x_1, x_2, \ldots, x_n$ all random variables are independent and all follow the same distribution this set is denoted as an *i.i.d* (independent and identically distributed) set of random variables.

DISCRETE UNIFORM DISTRIBUTION. If a random variable x only takes on n different values and all values are equally probable then we denote this a (discrete) uniform distribution:

$$p(x = x_i) = \frac{1}{n}, \qquad \forall i \tag{2.108}$$

BERNOULLI DISTRIBUTION. The Bernoulli distribution describes a binary random variable, i.e. the variable can have one of two values. A typical usage example is the description of the outcome of coin flips. The distribution is defined by

$$\text{Bern}(x \mid \theta) = \theta^x (1-\theta)^{1-x} \tag{2.109}$$

Setting $\eta = \log \frac{\theta}{1-\theta}$ one can write

$$\text{Bern}(x \mid \theta) = \frac{\exp(\eta x)}{1 + \exp(\eta)} \tag{2.110}$$

Furthermore

$$\mathbb{E}[x] = \theta \tag{2.111}$$

$$\mathbb{V}[x] = \theta(1-\theta) \tag{2.112}$$

MULTINOMIAL DISTRIBUTION. The Bernoulli distribution is generalized by the Multinomial distribution to describe a discrete random variable that can take on one of k mutually exclusive values. The random variable is represented by a k-dimensional vector x, using a

1-of-k scheme: One element of the vector equals 1 and all remaining elements are 0. The distribution is defined by

$$\text{Multi}(\mathbf{x} \mid n, \theta) = \binom{n}{x_1\ x_2 \ldots x_k} \theta_1^{x_1} \theta_2^{x_2} \ldots \theta_k^{x_k} \tag{2.113}$$

with

$$\sum_k x_k = n \tag{2.114}$$

and

$$\sum_k \theta_k = 1 \tag{2.115}$$

and where

$$\theta_k \geqslant 0, \forall\ k \tag{2.116}$$

and

$$\binom{n}{x_1\ x_2 \ldots x_k} = \frac{n!}{x_1!\, x_2!\, \ldots x_k!} \tag{2.117}$$

Furthermore,

$$\mathbb{E}[\mathbf{x}] = (n\theta_1, n\theta_2, \ldots, n\theta_k) \tag{2.118}$$

and

$$\mathbb{V}[\mathbf{x}] = n \begin{pmatrix} \theta_1(1-\theta_1) & \theta_1\theta_2 & \cdots & \theta_1\theta_k \\ \theta_2\theta_1 & \theta_2(1-\theta_2) & \cdots & \theta_2\theta_k \\ \vdots & & \ddots & \vdots \\ \theta_k\theta_1 & \theta_k\theta_2 & \cdots & \theta_k(1-\theta_k) \end{pmatrix} \tag{2.119}$$

CONTINUOUS UNIFORM DISTRIBUTION. A one-dimensional continuous uniform distribution $U(a, b)$ has a density function that is constant over a finite interval $[a, b]$. If $x \sim U(a, b)$ then its probability density function is given by

$$p(x) = \begin{cases} \frac{1}{b-a}, & x \in [a, b] \\ 0, & \text{otherwise.} \end{cases} \tag{2.120}$$

with

$$\begin{aligned} \mathbb{E}[x] &= \frac{a+b}{2} \\ \mathbb{V}[x] &= \frac{(b-a)^2}{12} \end{aligned} \tag{2.121}$$

So for example, a uniform distribution $U(-a, a)$ which is symmetric around 0 has a variance of $a^2/3$. The cdf (cumulative distribution function) $F_U(x)$ of the continuous uniform distribution has the nice property that it acts as the identity function for the argument, i.e.

$$F_U(x \leqslant c) = c, \quad c \in [0, 1] \tag{2.122}$$

MULTIVARIATE GAUSSIAN DISTRIBUTION. One of the most important probability distributions for continuous variables is the multivariate *normal* or *Gaussian* distribution. The density of a multivariate Gaussian variable $\mathbf{x} \sim \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is given by

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = |2\pi\boldsymbol{\Sigma}|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \tag{2.123}$$

with $\mathbb{E}[\mathbf{X}] = \boldsymbol{\mu}$ and $\mathbb{V}[\mathbf{X}] = \Sigma$. A Gaussian is completely defined by its mean and covariance, i.e. all its higher moments are 0.

The *natural parameterization* of a Gaussian is

$$\mathcal{N}(\mathbf{x} \mid \mathbf{r}, \boldsymbol{\Lambda}) = |2\pi\boldsymbol{\Lambda}|^{1/2} \exp\left(-\frac{1}{2}(\mathbf{x}^{\mathsf{T}}\boldsymbol{\Lambda}\mathbf{x} + 2\mathbf{r}^{\mathsf{T}}\mathbf{x})\right) \tag{2.124}$$

with $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$ and $\mathbf{r} = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}$ (simply multiplying out the square formula from Eq. (2.123) and arranging terms). Using this parameterization, it is easy to show that the product of two Gaussian *densities* is again a Gaussian density (in the functional form). More specifically, if two Gaussian density functions $\mathcal{N}_1(\cdot)$ and $\mathcal{N}_2(\cdot)$ are given, then

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \cdot \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) \propto e^{-\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{A}_1\mathbf{x} + \mathbf{r}_1^{\mathsf{T}}\mathbf{x}} \cdot e^{-\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{A}_2\mathbf{x} + \mathbf{r}_2^{\mathsf{T}}\mathbf{x}} \tag{2.125}$$

The result is *again* a Gaussian density, because we can write it in information form:

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \cdot \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) \propto e^{-\frac{1}{2}\mathbf{x}^{\mathsf{T}}(\mathbf{A}_1+\mathbf{A}_2)\mathbf{x} + (\mathbf{r}_1+\mathbf{r}_2)^{\mathsf{T}}\mathbf{x}} \tag{2.126}$$

Converting it back into moment parameterization gives a new $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$:

$$\boldsymbol{\Sigma} = (\boldsymbol{\Sigma}_1^{-1} + \boldsymbol{\Sigma}_2^{-1})^{-1}, \qquad \boldsymbol{\mu} = \boldsymbol{\Sigma}(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_2^{-1}\boldsymbol{\mu}_2) \tag{2.127}$$

A family of distributions is *closed* under a set of operations if if the outcome of these operations lies in the family as well. The Gaussian family is closed under linear (affine) transformations, marginalization and under conditioning which is one the reasons why it has such a central role in Probability Theory.

The invariance of the type under a linear transformation is a result of change of variable theorem (Eq. (2.106)). If $\mathbf{x} \sim \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ and $\mathbf{y} = \mathbf{D}\mathbf{x}$ for some matrix $\mathbf{D} \in \mathbf{R}^{n \times n}$ with rank n then (see Eq. (2.107))

$$\begin{aligned}
p(\mathbf{y}) &= \mathcal{N}(\mathbf{D}^{-1}\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})/|\mathbf{D}| \\
&= |\mathbf{D}^{(-2/2)}||2\pi\boldsymbol{\Sigma}|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{D}^{-1}\mathbf{x}-\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{D}^{-1}\mathbf{x}-\boldsymbol{\mu})\right) \\
&= |2\pi\mathbf{D}\boldsymbol{\Sigma}\mathbf{D}^{\mathbf{T}}|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{D}\boldsymbol{\mu})^{\mathsf{T}}\mathbf{D}^{-\mathsf{T}}\boldsymbol{\Sigma}^{-1}\mathbf{D}^{-1}(\mathbf{x}-\mathbf{D}\boldsymbol{\mu})\right) \\
&= |2\pi\boldsymbol{W}|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{D}\boldsymbol{\mu})^{\mathsf{T}}\boldsymbol{W}^{-1}(\mathbf{x}-\mathbf{D}\boldsymbol{\mu})\right)
\end{aligned}$$

(2.128)

with $\mathbf{W} = \mathbf{D}\boldsymbol{\Sigma}\mathbf{D}^{\mathsf{T}}$. So $\mathbf{y}$ is also a Gaussian with

$$\mathbb{E}[\mathbf{y}] = \mathbf{D}\boldsymbol{\mu} \tag{2.129}$$

$$\mathbb{V}[\mathbf{y}] = \mathbf{D}\boldsymbol{\Sigma}\mathbf{D}^{\mathsf{T}} \tag{2.130}$$

This result also holds for arbitrary $\mathbf{D} \in \mathbf{R}^{m \times n}$, without having full rank which results in a so called *degenerate Gaussian*. However, the proof is much more difficult and therefore omitted here [202]. A correlation of the above result is that for $\mathbf{x} \sim \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ the *z-transformation* $\mathbf{x}^z$,

$$\mathbf{x}^z = \boldsymbol{\Sigma}^{-1/2}(\mathbf{x} - \boldsymbol{\mu}), \tag{2.131}$$

is distributed according to $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

The results from Eq. (2.129) and Eq. (2.130) automatically follow from the linear nature of the expectation operator and are true for any type of random variable $\mathbf{x}$. Using this rule it is easy to show that the sum of two independent Gaussian random variables is Gaussian. Let $\mathbf{x}_1 \sim \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$, $\mathbf{x}_2 \sim \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, two n dimensional independent Gaussian random variables. *Stack* both variables to form a random variable $\mathbf{y}$. $\mathbf{y}$ is a Gaussian random variable because

$$\mathbf{y} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} \sim \mathcal{N}\left(\mathbf{x} \middle| \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma}_1 & 0 \\ 0 & \boldsymbol{\Sigma}_2 \end{pmatrix}\right) \tag{2.132}$$

Consider $z = \mathbf{x}_1 + \mathbf{x}_2$. z can be written as $z = \mathbf{A}\mathbf{y}$, with $\mathbf{A} = [\mathbf{I}_n \ \mathbf{I}_n]$, so z is Gaussian. Applying Eq. (2.129) and Eq. (2.130) leads to a mean and a covariance matrix that are the sum of the respective means and covariance matrices:

$$\mathbf{x}_1 + \mathbf{x}_2 \equiv z \sim \mathcal{N}(z \mid \boldsymbol{\mu}_1 + \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2) \tag{2.133}$$

Next, let's consider the marginalization operation. For a given n dimensional Gaussian random variable $\mathbf{x}$, we are interested in the marginal distribution of $\mathbf{x}_I$, $I \in \{1, 2, \ldots, n\}$. Usually, one needs to employ explicitly the sum rule for marginalization, however in the Gaussian case marginalization can be expressed as a linear operation, because $\mathbf{x}_I = \mathbf{I}_I \mathbf{x}$, where $\mathbf{I}_I$ is a *selector* matrix (for every index $i \in I$ the matrix $\mathbf{I}_I$ has a row with a single 1 at column i). Thus

$$\mathbf{x}_I \sim \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_I, \boldsymbol{\Sigma}_I) \tag{2.134}$$

according to Eq. (2.128), Eq. (2.129) and Eq. (2.130).

The conditioning operation is straightforward in the natural parameterization. Assume that $\mathbf{x} \sim \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$, $I \in \{1, 2, \ldots, n\}$ and $R = \{1, 2, \ldots, n\} \setminus I$. In order to find the conditional distribution for

$x_I \mid x_R$, we write the joint density $p(x_I, x_R)$ (a Gaussian) but only consider those parts containing x_I:

$$\begin{aligned}\ln p(x_I, x_R) &\propto -\frac{1}{2}\left((x_I - \mu_I)^T \Lambda_I (x_I - \mu_I)\right. \\ &\quad \left. + 2(x_R - \mu_R)^T \Lambda_{R,I}(x_I - \mu_I)\right) \\ &\propto -\frac{1}{2} x_I^T \Lambda_I x_I \\ &\quad + (x_R - \mu_R)^T \Lambda_{R,I} \underbrace{\Lambda_I^{-1} \Lambda_I}_{I} x_I - \mu_I \Lambda_I x_I\end{aligned} \tag{2.135}$$

Note that $\Lambda = \Sigma^{-1}$. The product rule (Eq. (2.91)) gives $p(x_I, x_R) = p(x_I \mid x_R)p(x_R)$, and thus $p(x_I \mid x_R)$ is a Gaussian. One can read of the mean and covariance by matching against its natural parameterization:

$$\begin{aligned}\mathbb{V}[x_I \mid x_R] &= \Lambda_{II}^{-1} \\ \mathbb{E}[x_I \mid x_R] &= \mu_I + \Lambda_I^{-1} \Lambda_{I,R}(x_R - \mu_R)\end{aligned} \tag{2.136}$$

Finally, let's consider *linear Gaussian* systems. Let y be an m dimensional random variable which is defined as follows:

$$y = Ax + b + \varepsilon \tag{2.137}$$

$A \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$ and x, ε are two *independent* random Gaussians (n/m dimensional, respectively):

$$\begin{aligned}x &\sim \mathcal{N}(x \mid \mu, \Lambda^{-1}) \\ \varepsilon &\sim \mathcal{N}(\varepsilon \mid 0, L^{-1})\end{aligned} \tag{2.138}$$

Conditioned on x, $y \mid x$ is a Gaussian because it is the sum of a constant ($Ax + b$) and a Gaussian (ε):

$$p(y \mid x) = \mathcal{N}(y \mid Ax + b, L^{-1}) \tag{2.139}$$

y is a Gaussian, too, because it is the sum of a linearly transformed Gaussian (Ax), a constant (b) and again a Gaussian (ε), so following Eq. (2.133) gives:

$$y \sim \mathcal{N}(y \mid A\mu + b, A\Lambda^{-1}A^T + L^{-1}) \tag{2.140}$$

The covariance matrix between x and y is $\Lambda^{-1}A^T$ because

$$\begin{aligned}\mathbb{E}\left[(x - \mathbb{E}[x])(y - \mathbb{E}[y])^T\right] &= \mathbb{E}\left[(x - \mathbb{E}[x])(Ax + b - A\mathbb{E}[x] - b)^T\right] \\ &= \mathbb{E}\left[(x - \mu)(A(x - \mu))^T\right] \\ &= \mathbb{E}\left[(x - \mu)(x - \mu)^T\right] A^T \\ &= \Lambda^{-1}A^T\end{aligned} \tag{2.141}$$

So stacking x and y is a Gaussian:

$$\begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}\left(x \,\middle|\, \begin{bmatrix} I \\ A \end{bmatrix} x + \begin{bmatrix} 0 \\ b \end{bmatrix}, \begin{bmatrix} \Lambda^{-1} & \Lambda^{-1}A^T \\ A\Lambda^{-1} & A\Lambda^{-1}A^T + L^{-1} \end{bmatrix}\right) \tag{2.142}$$

Then it is simple to *invert* the linear system (as is necessary, e.g., for the Kalman Filter [196, 370, 183]), by applying the conditioning results from Eq. (2.136):

$$p(\mathbf{x} \mid \mathbf{y}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\Sigma}(\mathbf{A}^T \mathbf{L}(\mathbf{y} - \mathbf{b}) + \boldsymbol{\Lambda}\boldsymbol{\mu}), \boldsymbol{\Sigma}) \tag{2.143}$$

with $\boldsymbol{\Sigma} = (\boldsymbol{\Lambda} + \mathbf{A}^T \mathbf{L} \mathbf{A})^{-1}$

2.4 MACHINE LEARNING CONCEPTS

Learning happens on the basis of collected data. The set of n observed data items is denoted by $\mathcal{D}$. These items are either vectors $\mathbf{x}_i \in \mathbf{R}^d$ (with d depending on the task to be solved) or pairs $(\mathbf{x}_i, \mathbf{y}_i)$ forming an input ($\mathbf{x} \in \mathbf{R}^d$) – output ($\mathbf{y} \in \mathbf{R}^o$) relationship. In the first case, *unsupervised learning*, we are interested in modeling the distribution of the observed data, $p(\mathbf{x} \mid \boldsymbol{\theta})$. In the second case, *supervised learning*, we are interested in the *functional relationship* between inputs and corresponding outputs and therefore want to model the conditional distribution $p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})$. A popular choice for modelings distributions in both cases are parametric families of models. The goal is then to find the best parameter vector $\boldsymbol{\theta}$ using the available information in the collected data $\mathcal{D}$[2].

However, what does *best* mean in this case? Given that we only have the observed data $\mathcal{D}$ and the class of distributions $p(\mathbf{x} \mid \boldsymbol{\theta})$ (or, equivalently, $p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})$), a reasonable objective is to find parameters $\boldsymbol{\theta}$ that minimize the Kullback-Leibler divergence (Eq. (2.100)) between the empirical data distribution $\hat{p}(\mathbf{x})$ (Eq. (2.98)), $\hat{p}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \delta(\mathbf{x} = \mathbf{x}_i)$, and $p(\mathbf{x} \mid \boldsymbol{\theta})$:

$$\mathcal{KL}[\hat{p} \| p_\theta] = \int \hat{p}(\mathbf{x}) \log \frac{\hat{p}(\mathbf{x})}{p(\mathbf{x} \mid \boldsymbol{\theta})} d\mathbf{t} \tag{2.144}$$

This can be reformulated as follows, using Eq. (2.99)

$$\begin{aligned}
\mathcal{KL}[\hat{p} \| p_\theta] &= \int \hat{p}(\mathbf{x}) \log \frac{\hat{p}(\mathbf{x})}{p(\mathbf{x} \mid \boldsymbol{\theta})} d\mathbf{t} \\
&= -\mathcal{H}[\hat{p}] - \int \hat{p}(\mathbf{x}) \log p(\mathbf{x} \mid \boldsymbol{\theta}) d\mathbf{t} \\
&= -\mathcal{H}[\hat{p}] - \int \sum_i \delta(\mathbf{x} = \mathbf{x}_i) \log p(\mathbf{x} \mid \boldsymbol{\theta}) d\mathbf{t} \\
&= -\mathcal{H}[\hat{p}] - \sum_i \int \delta(\mathbf{x} = \mathbf{x}_i) \log p(\mathbf{x} \mid \boldsymbol{\theta}) d\mathbf{t} \\
&= -\mathcal{H}[\hat{p}] - \sum_i \log p(\mathbf{x}_i \mid \boldsymbol{\theta})
\end{aligned} \tag{2.145}$$

2 Reinforcement Learning [376] which is not discussed in this text can be modeled in a similar parametric form—the observed data items then comprise state, action and reward information.

So minimizing the KL divergence between the empirical distribution and $p(x \mid \theta)$ is equivalent to maximizing the *log-likelihood* $\ell(\theta) = \sum_i \log p(x_i \mid \theta)$ of the data under the model family $p(x \mid \theta)$, because $\mathcal{H}[\hat{p}]$ is independent of θ. Note that the log-likelihood is a function of θ, as the dataset $\mathcal{D}$ is considered fixed (though unknown apriori). Because the logarithm is a monotone function, the optimum of the log-likelihood function is the same as the one for the *likelihood* function $\prod_i p(x_i \mid \theta) = p(\mathcal{D} \mid \theta)$. Determining parameters θ in this way is called Maximum Likelihood Estimation (MLE).

Though the empirical distribution (the *training set*) is used for learning (i.e. *optimizing* θ in the case of MLE), the overall goal is to find models that *generalize well*, that is, perform well also on data items that are not in the training set. In order to assess the generalization performance of a trained model, an unbiased estimate of it can be obtained by evaluating the trained model on an independent *test set*, a dataset that is generated from the same process that generated the training set, but does not share any specific elements with it. The test set must never be used to adapt parameters θ in any way, it is only utilized for evaluating the log-likelihood of the model on its data samples (for a supervised model this measures its predictive capacity). If the trained model performs well on the training set but not on the test set (that is, it does not generalize well) we say that the model *overfits* the training set.

Sometimes parameters θ themselves are equipped with parameters, so called *hyperparameters*. Determining optimal settings for these hyperparameters often can't be done on the training set. Instead a third independent data set is introduced denoted *validation set* which is used tuning these hyperparameters (and hyperparameters of hyperparameters, ...).

For an item x from the test set what is its likelihood given the items from $\mathcal{D}$? If one considers the parameter θ itself a random variable (the training set $\mathcal{D}$ is a random variable, too, as the collection of the training samples is a random process) then this likelihood can be expressed as (using Bayes' rule):

$$\begin{aligned} p(x \mid \mathcal{D}) &= \frac{p(x, \mathcal{D})}{p(\mathcal{D})} = \frac{\int p(x, \mathcal{D}, \theta) d\theta}{p(\mathcal{D})} \\ &= \frac{\int p(x \mid \mathcal{D}, \theta) p(\mathcal{D} \mid \theta) p(\theta) d\theta}{p(\mathcal{D})} \end{aligned} \tag{2.146}$$

That is the likelihood of a test sample (or, its *posterior distribution*, if one thinks of the test sample as a random variable) is the likelihood of x , $p(x \mid \mathcal{D}, \theta)$ for a given θ weighted by the likelihood $p(\mathcal{D} \mid \theta)$ of θ and the *prior* of θ, $p(\theta)$. The *marginal likelihood* (or *evidence*) $p(\mathcal{D})$ is necessary as a normalization constant. Note that integrating over all possible values of θ and using their likelihood as weights is the best way to avoid overfitting.

The integrals in Eq. (2.146) are often intractable. Maximum Likelihood Estimation circumvents these complicated integrals by setting the likelihood of θ to be a Dirac delta function centered at its maximum and fixes the prior to a constant, i.e.

$$\begin{aligned} p(\mathcal{D} \mid \theta) &= \delta(\theta - \theta_{\text{MLE}}) \\ \theta_{\text{MLE}} &= \arg\max_{\theta} p(\mathcal{D} \mid \theta) \\ p(\theta) &\propto 1 \end{aligned} \tag{2.147}$$

The posterior distribution for $\mathbf{x}$ is then

$$p(\mathbf{x} \mid \mathcal{D}) = \frac{\int p(\mathbf{x} \mid \mathcal{D}, \theta)\delta(\theta - \theta_{\text{MLE}})1\,d\theta}{\int \delta(\theta - \theta_{\text{MLE}})1\,d\theta} = p(\mathbf{x} \mid \mathcal{D}, \theta_{\text{MLE}}) \tag{2.148}$$

Let's consider the application of the MLE principle for both unsupervised and supervised problems. For the unsupervised setting assume the training set consisting of samples from a Bernoulli distribution with unknown θ. Given n i.i.d samples $(b_1, b_2, \ldots, b_n)$ the likelihood function for θ is therefore

$$\prod_i \theta^{b_i}(1-\theta)^{1-b_i} = \theta^{n_0}\theta^{n-n_0} \tag{2.149}$$

where n_0 denotes the number of sampled zeros. Hence the log likelihood $\ell(\theta)$ is

$$\ell(\theta) = n_0 \log\theta + (n - n_0)\log(1-\theta) \tag{2.150}$$

and its first derivative is

$$\frac{n_0}{\theta} - \frac{n-n_0}{1-\theta} \tag{2.151}$$

Therefore, the MLE for θ is

$$\theta_{\text{MLE}} = \frac{n_0}{n} \tag{2.152}$$

For the supervised setting assume that a real valued input $\mathbf{x} \in \mathbf{R}^d$ is mapped to $y \in \mathbf{R}$, a *regression* problem. In particular, assume that the functional relationship between $\mathbf{x}$ and y can be modeled as

$$y = f(\mathbf{x}, \theta) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(\varepsilon \mid 0, \sigma^2) \tag{2.153}$$

That is

$$p(y \mid \mathbf{x}, \theta) = \mathcal{N}(y \mid f(\mathbf{x}, \theta), \sigma^2), \tag{2.154}$$

so the mean of the distribution for y is a deterministic function of $\mathbf{x}$, which is parameterized by θ. An important simplification here is that the variance σ^2 is independent of $\mathbf{x}$. The likelihood function for a dataset $\mathcal{D}$ with n training pairs $(\mathbf{x}_i, y_i)$ is therefore

$$\prod_i \mathcal{N}(y_i \mid f(\mathbf{x}_i, \theta), \sigma^2) \tag{2.155}$$

and hence the loglikelihood function $\ell(\theta)$ is

$$\ell(\theta) = -\frac{n}{2}\ln\sigma^2 - \frac{n}{2}\ln(2\pi) - \frac{1}{2\sigma^2}\sum_i (y_i - f(x_i, \theta))^2 \tag{2.156}$$

So under the assumption of Eq. (2.153) the Maximum Likelihood Estimation of θ for a regression problem corresponds to minimizing the *sum-of-squares error* between the predicted mean $f(x, \theta)$ and the true value y:

$$\begin{aligned} \theta_{\text{MLE}} &= \arg\max_\theta \left(-\frac{1}{2\sigma^2}\sum_i (y_i - f(x_i, \theta))^2 \right) \\ &= \arg\min_\theta \left(\frac{1}{2}\sum_i (y_i - f(x_i, \theta))^2 \right) \end{aligned} \tag{2.157}$$

If, instead to regression, real valued inputs x are mapped to K mutual exclusive natural numbers (*labels*, which are usually abstract representations of properties of the input, e.g. the color of a car) then we denote the underlying task a *classification* problem. y is then a Multinomial random vector (see Eq. (2.113)). One might imagine that this is simply a special case of regression, but this does usually not provide satisfactory results, because there is no ordinal relationship between the labels (see [35, Section 4.1.3] for more discussion on this aspect).

The model $p(y_i = 1 \mid x, \theta)$ denotes the probability that input x is classified with label i. For a given dataset $\mathcal{D} = \{(x_1, l_1), \ldots, (x_n, l_n)\}$ with n pairs of labeled inputs the likelihood is

$$\prod_n \prod_k p(y_{ik} \mid x_i, \theta)^{y_{ik}} = \prod_n p(y_{il_i} \mid x_i, \theta) \tag{2.158}$$

The loglikelihood is then

$$\sum_i \log p(y_{il_i} \mid x_i, \theta) \tag{2.159}$$

which is often denoted the *categorial cross-entropy* function. In order to ensure that $p(y \mid x, \theta)$ is a Multinomial distribution, a common functional form of $p(y \mid x, \theta)$ is the so-called *soft-max* function

$$p(y \mid x, \theta) = \frac{\exp(f(x, \theta))}{\|\exp(f(x, \theta))\|_1} \tag{2.160}$$

where $f : \mathbf{R}^n \to \mathbf{R}^k$ and $\exp(\cdot)$ is the *elementwise* exponential. In this case Eq. (2.159) becomes:

$$\sum_i \log \frac{\exp([f(x, \theta)]_{l_i})}{\|\exp(f(x, \theta))\|_1} = \sum_i [f(x, \theta)]_{l_i} - \log \|\exp(f(x, \theta))\|_1 \tag{2.161}$$

Maximizing the loglikelihood in this case does not only induce maximizing the output for the correct label. The second *contrastive* term

also minimizes the probabilities of all other (wrong) labels for a given input (which is of course a direct consequence of a normalized probability distribution).

In the case of $K = 2$ a scalar random variable y is enough to model the binary outcome with a simple Bernoulli distribution:

$$p(y \mid \mathbf{x}, \theta) = \text{Bern}(y \mid \sigma(f(\mathbf{x}, \theta))) \tag{2.162}$$

with $f : \mathbf{R}^n \to \mathbf{R}$. The loglikelihood function is called *binary cross-entropy*:

$$\sum_i y_i \log \sigma(f(\mathbf{x_i}, \theta)) + (1 - y_i)(1 - \log \sigma(f(\mathbf{x_i}, \theta))) \tag{2.163}$$

with the provided labels $y_i \in \{0, 1\}$.

Certain types of models and tasks can rely on a large body of prior information—this information is then encoded into the *form* of $p(\mathbf{x}|\theta)$ and into the *prior* $p(\theta)$ *over the parameters*. MLE neglects this part of the prior information which especially results in *overfitting* problems when the size of the training set is small. The so called *Maximum-A-Posteriori* (MAP) approach can be a remedy in these situations. First, reformulate Eq. (2.146) by identifying the *posterior distribution of* θ *given the dataset* $\mathcal{D}$:

$$\begin{aligned} p(\mathbf{x} \mid \mathcal{D}) &= \frac{p(\mathbf{x}, \mathcal{D})}{p(\mathcal{D})} = \int p(\mathbf{x} \mid \mathcal{D}, \theta) p(\theta \mid \mathcal{D}) d\theta \\ \text{with } p(\theta \mid \mathcal{D}) &= \frac{p(\mathcal{D} \mid \theta) p(\theta)}{p(\mathcal{D})} \end{aligned} \tag{2.164}$$

That is the posterior distribution of a test sample $\mathbf{x}$ is the average likelihood $p(\mathbf{x} \mid \mathcal{D}, \theta)$, weighted by the posterior of θ. Then approximate the integral in Eq. (2.164), again using a Dirac delta function:

$$\begin{aligned} p(\theta \mid \mathcal{D}) &= \delta(\theta - \theta_{\text{MAP}}) \\ \theta_{\text{MAP}} &= \arg\max_\theta p(\mathcal{D} \mid \theta) p(\theta) \end{aligned} \tag{2.165}$$

The posterior distribution for $\mathbf{x}$ is then

$$p(\mathbf{x} \mid \mathcal{D}) = p(\mathbf{x} \mid \mathcal{D}, \theta_{\text{MAP}}) \tag{2.166}$$

Continuing the regression example from Eq. (2.153), a simple choice for the prior over the parameters θ can be a zero centered Gaussian with independent parameters, i.e.

$$\theta \sim \mathcal{N}(\theta \mid 0, \lambda^2 \mathbf{I}) \tag{2.167}$$

The posterior density $p(\theta|\mathcal{D})$ is

$$p(\theta \mid \mathcal{D}) \propto \prod_i \mathcal{N}(y_i \mid f(\mathbf{x_i}, \theta), \sigma^2) \mathcal{N}(\theta \mid 0, \lambda^2 \mathbf{I}) \tag{2.168}$$

a Gaussian (the product of Gaussian densities, see Eq. (2.126)), which can be written easier in the log domain as

$$\log p(\theta \mid \mathcal{D}) \propto \frac{1}{2} \sum_i (y_i - f(\mathbf{x}_i, \theta))^2 + \frac{1}{2\lambda^2} \theta^T \theta \tag{2.169}$$

MAP with the prior from Eq. (2.167) therefore results in a *regularized* sum-of-squares objective for θ. Regularizing with an L_2 term as in Eq. (2.169) is often called *weight decay* in Machine Learning – the parameters (*weights*) are driven to small absolute values.

Both MLE and MAP have nice statistical properties (e.g. MLE is a *consistent estimator in the limit of an infinite amount of training data*). However, in both cases learning becomes a (difficult) optimization problem. As it turns out, these optimization problems are not invariant to reparameterizations. Therefore it might still be the best approach to tackle the real problem of predictive modeling (Eq. (2.146)) and consider methods that solve high dimensional integrals. This is the topic of the next section.

Quite often, a learning problem can't be formulated in probabilistic terms, in particular in the supervised setting. Instead, a general loss function $L : (\mathbf{R}^n \times \mathbf{R}^n) \to \mathbf{R}$ can be defined for the learning problem which measures the cost of predicting $f(\mathbf{x}, \theta)$ for some input $\mathbf{x}$ when the true observation is $\mathbf{y}$. So $f(\cdot, \cdot)$ is the predictive model, with adaptive parameters θ. In this setting the parameters θ are determined by optimizing the *empirical risk* $R(\theta)$ of the predictive model on the training set $\mathcal{D}$. The empirical risk is usually defined as containing a *data fitting term* induced by $L(\cdot, \cdot)$ on the training set and a *regularization term* $\boldsymbol{\Omega}(\theta)$:

$$R(\theta) = \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x}_i, \mathbf{y}_i} L(f(\mathbf{x}_i, \theta), \mathbf{y}_i) + \lambda \boldsymbol{\Omega}(\theta) \tag{2.170}$$

Learning then means finding parameters θ that minimize $R(\theta)$. Note that this framework of *empirical risk minimization* [403] encompasses MLE and MAP.

2.5 THE MONTE CARLO PRINCIPLE AND SAMPLING

Probability Theory is a powerful language to model uncertainties. Yet, many questions that are related to uncertain modeling are deterministic. Usually these can be written in the form of an expectation: Given a deterministic function $f : \mathbf{R}^m \to \mathbf{R}^n$ and a probability density (or mass) function $p(\mathbf{x})$, one is interested in finding

$$\mathbb{E}[f(\mathbf{x})] = \int f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \tag{2.171}$$

Depending on the form of $f(\mathbf{x})$ and $p(\mathbf{x})$ this integral is often (i.e. for interesting cases) not analytically tractable. Most standard methods

of numerical integration do not work well either, in particular for high-dimensional problems.

A simple yet powerful approach is the *Monte Carlo principle* [242]. The above integral is *approximated* by a weighted sum of $f(\mathbf{x}_i)$ at accordingly chosen points $\mathbf{x}_i$. Specifically

$$\mathbb{E}[f(\mathbf{x})] = \int f(\mathbf{x})p(\mathbf{x})d\mathbf{x} \approx \frac{1}{S}\sum_{i}^{S} f(\mathbf{x}_i), \quad \mathbf{x}_i \sim p(\mathbf{x}) \tag{2.172}$$

So the $\mathbf{x}_i$ are i.i.d samples from $p(\mathbf{x})$. The above estimate has two nice properties:

- It is unbiased (which follows from the linearity of the expectation operator):

$$\begin{aligned}\mathbb{E}\left[\frac{1}{S}\sum_{i=1}^{S} f(\mathbf{x}_i)\right] &= \frac{1}{S}\mathbb{E}\left[\sum_{i=1}^{S} f(\mathbf{x}_i)\right] = \frac{1}{S}\sum_{i=1}^{S}\mathbb{E}[f(\mathbf{x}_i)] \\ &= \mathbb{E}[f(\mathbf{x})]\end{aligned} \tag{2.173}$$

- Its variance shrinks proportionally to S:

$$\mathbb{V}\left[\frac{1}{S}\sum_{i=1}^{S} f(\mathbf{x}_i)\right] = \frac{1}{S^2}\sum_{i=1}^{S}\mathbb{V}[f(\mathbf{x}_i)] = \frac{1}{S}\mathbb{V}[f(\mathbf{x})] \tag{2.174}$$

  Note that this holds *independently* of the dimensionality of $\mathbf{x}$! This is however only true as long as the samples $\mathbf{x}_i$ are independent. Otherwise, Eq. (2.174) is more involved:

$$\begin{aligned}\mathbb{V}\left[\frac{1}{S}\sum_{i=1}^{S} f(\mathbf{x}_i)\right] &= \frac{1}{S}\mathbb{V}[f(\mathbf{x})] \\ &\quad + 2\sum_{1\leqslant i<j\leqslant S}\mathrm{Cov}(f(\mathbf{x}_i), f(\mathbf{x}_j))\end{aligned} \tag{2.175}$$

The Monte Carlo approach can be applied flexibly within a lot of settings. For example, given some dataset $\mathcal{D}$, one goal is usually to find the predictive distribution $p(\mathbf{x} \mid \mathcal{D})$. Assuming the existence of an unknown parameter θ, the Monte Carlo principle can be applied in a straight forward way:

$$p(\mathbf{x} \mid \mathcal{D}) = \int p(\mathbf{x} \mid \theta)p(\theta \mid \mathcal{D})d\theta \approx \frac{1}{S}\sum p(\mathbf{x} \mid \theta_i), \ \theta_i \sim p(\theta \mid \mathcal{D}) \tag{2.176}$$

Overall the problem of solving an integral is replaced by the problem of generating independent (in the best case) samples from some distribution. As it turns out, this is very challenging for high dimensional distributions, but often more manageable than alternative approaches to tackle high dimensional integrals.

SAMPLING. In order to generate representative values from a distribution $p(\mathbf{x})$ a naive idea is to find a simple enough distribution $q(\mathbf{x})$ to sample from instead. Clearly an additional step must be taken, if such a sample is then used in Eq. (2.172) as an *approximation* for a true sample from the difficult $p(\mathbf{x})$. This correction can again be derived applying the Monte Carlo principle and results in *importance sampling* [354]:

$$\begin{aligned}\int p(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} &= \int q(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} f(\mathbf{x}) d\mathbf{x} \\ &\approx \frac{1}{N} \sum_i \frac{p(\mathbf{x}_i)}{q(\mathbf{x}_i)} f(\mathbf{x}), \ \mathbf{x}_i \sim q(\mathbf{x})\end{aligned} \tag{2.177}$$

Usually, only the functional form of $p(\mathbf{x})$ is known, but not its normalization constant, i.e. $p(\mathbf{x}) = \hat{p}(\mathbf{x})/Z$, with $Z = \int p(\mathbf{x}) d\mathbf{x}$. In this case, *weighted importance sampling* can be utilized

$$\begin{aligned}\int p(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} &= \frac{\int \hat{p}(\mathbf{x}) f(\mathbf{x}) d\mathbf{x}}{\int \hat{p}(\mathbf{x}) d\mathbf{x}} = \frac{\int \frac{\hat{p}(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x}) f(\mathbf{x}) d\mathbf{x}}{\int \frac{\hat{p}(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x}) d\mathbf{x}} \\ &\approx \frac{\sum_i \frac{\hat{p}(\mathbf{x}_i)}{q(\mathbf{x}_i)} f(\mathbf{x}_i)}{\sum_i \frac{\hat{p}(\mathbf{x}_i)}{q(\mathbf{x}_i)}} = \sum_i f(\mathbf{x}_i) w_i, \ \mathbf{x}_i \sim q(\mathbf{x})\end{aligned} \tag{2.178}$$

where w_i are *normalized importance weights*:

$$w_i = \frac{\frac{\hat{p}(\mathbf{x}_i)}{q(\mathbf{x}_i)}}{\sum_n \frac{\hat{p}(\mathbf{x}_n)}{q(\mathbf{x}_n)}} \tag{2.179}$$

An obvious problem with this approach is the possibility that very few samples have large weights and therefore dominate the sum in Eq. (2.178) reducing the *effective sample size* drastically. For example, if one considers the unnormalized importance weights $\hat{w}_i = \hat{p}(\mathbf{x}_i)/q(\mathbf{x}_i)$ then the expected variance between two weights $\hat{w}_i$ and $\hat{w}_j$ can be expressed as

$$\mathbb{E}\left[(\hat{w}_i - \hat{w}_j)^2\right] = \mathbb{E}_{p(\mathbf{x})}\left(\frac{\hat{p}(\mathbf{x})}{q(\mathbf{x})}\right)^d - 1 \tag{2.180}$$

which grows exponentially with the dimensionality d of $\mathbf{x}$, because $\mathbb{E}_{p(\mathbf{x})}\left(\frac{\hat{p}(\mathbf{x})}{q(\mathbf{x})}\right) > 1$ for $p(\mathbf{x}) \neq q(\mathbf{x})$[3]. This actually implies that only a single sample will determine the Monte Carlo estimate. A possible remedy for this situation is *Sampling Importance Resampling* [354]: The weights w_i, $i \in S$, induce an S dimensional Multinomial distribution. From this distribution S indices are repeatedly drawn which then form the final Monte Carlo estimate, having each a weight of $1/S$.

If a random variable $\mathbf{y}$ can be represented as some non-linear transformation $f(\mathbf{x})$ of $\mathbf{x}$ and $p(\mathbf{x})$ represents a distribution which it is

3 This derivation assumes that p and q are fully factorized distributions.

easy to sample from, than valid samples of $p(\mathbf{y})$ are easy to generate through sampling from $p(\mathbf{x})$ and applying $f(\cdot)$. A well known application of this *transformation method* is the generation of samples from an arbitrary Gaussian $\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$: The random variable $\mathbf{z} \sim \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I})$ (which is usually generated by a different transformation method, e.g. the Box-Muller method [35]) can be transformed to $\mathbf{x}$ with

$$\mathbf{x} = \mathbf{L}\mathbf{z}, \ \mathbf{z} \sim \mathcal{N}(\mathbf{z} \mid \mathbf{0}, \mathbf{I}) \tag{2.181}$$

where $\mathbf{L}\mathbf{L}^\top = \boldsymbol{\Sigma}$, i.e. $\mathbf{L}$ is the Cholesky factor of $\boldsymbol{\Sigma}$, which always exists because $\boldsymbol{\Sigma}$ is positive semi-definite.

The transformation method can also be applied if the random variable of interest $\mathbf{y}$ is part of a (tractable) joint distribution $p(\mathbf{x}, \mathbf{y})$. Specifically, if $p(\mathbf{y}) = \sum_{\mathbf{x}} p(\mathbf{x}, \mathbf{y})$ then generating a full sample from the joint distribution and simply leaving away the part representing $\mathbf{x}$ generates valid samples $\mathbf{y} \sim p(\mathbf{y})$[4].

If the distribution $p(x)$ is simple, enough an easy way to sample from $p(x)$ is by the *inversion method*. Let X have cdf $F(x)$, which is monotone and therefore invertible. Then Y, defined as

$$Y = F^{-1}(u), \quad u \sim U(0, 1) \tag{2.182}$$

will be distributed according to $F(\cdot)$. The reason is due to the definition of the cumulative distribution function (cdf, Eq. (2.122)) of a continuous uniform random variable:

$$p(Y \leqslant y) = p(F^{-1}(u) \leqslant y) = p(u \leqslant F(x)) = F(x) \tag{2.183}$$

If a distribution $p(\mathbf{x}) = \frac{\hat{p}(\mathbf{x})}{Z}$ has an intractable normalization constant Z, but is otherwise analytically manageable, then *rejection sampling* generates valid samples. Similar to importance sampling a *proposal distribution* $q(\mathbf{x})$ is utilized to generate samples. For the proposal distribution it is important that its scaled density function $q(\mathbf{x})$ is above $\hat{p}(\mathbf{x})$:

$$cq(\mathbf{x}) \geqslant \hat{p}(\mathbf{x}), \quad c > 0, \ \forall \mathbf{x} \tag{2.184}$$

$q(\mathbf{x})$ is used to generate a *candidate* sample $\mathbf{x}_{\text{cand}}$ which is then compared to a sample $u \sim U(0, 1)$ and accepted if $u \leqslant \frac{\hat{p}(\mathbf{x}_{\text{cand}})}{c\, q(\mathbf{x}_{\text{cand}})}$. The conditional probability for acceptance, i.e. $p(\text{accept} \mid \mathbf{x})$, is therefore given by

$$p(\text{accept} \mid \mathbf{x}) = \frac{\hat{p}(\mathbf{x})}{cq(\mathbf{x})} \tag{2.185}$$

The overall acceptance probability represents the statistical efficiency of a sampling method:

$$\begin{aligned} p(\text{accept}) &= \int p(\text{accept}, \mathbf{x}) d\mathbf{x} = \int p(\text{accept} \mid \mathbf{x}) q(\mathbf{x}) d\mathbf{x} \\ &= \int \frac{\hat{p}(\mathbf{x})}{cq(\mathbf{x})} q(\mathbf{x}) d\mathbf{x} = \int \frac{\hat{p}(\mathbf{x})}{c} d\mathbf{x} = \frac{Z}{c} \end{aligned} \tag{2.186}$$

4 More formally, this is equivalent to applying a selector matrix, as demonstrated with marginalizing a Gaussian.

This expression can now be used to show that the sampled distribution (i.e. $p(\mathbf{x} \mid \text{accept})$) is actually the desired distribution $p(\mathbf{x})$:

$$p(\mathbf{x} \mid \text{accept}) = \frac{p(\mathbf{x}, \text{accept})}{p(\text{accept})} = \frac{\hat{p}(\mathbf{x})}{c}\frac{c}{Z} = \frac{\hat{p}(\mathbf{x})}{Z} = p(\mathbf{x}) \tag{2.187}$$

The efficiency of rejection sampling is acceptable for a one dimensional distribution but results in a very impractical sampling method for high dimensional densities: For example, in the simple case that $p(\mathbf{x})$ and $q(\mathbf{x})$ are both d-dimensional factorized distributions the probability of accepting some candidate sample decreases exponentially with the number of dimensions because (using Eq. (2.186) d times for every pair of one dimensional distributions):

$$p(\text{accept}) = \prod_i^d \frac{Z_i}{c_i} \tag{2.188}$$

The previous methods showed that high dimensional distributions are difficult to sample from. Apart from special cases where the transformation method is applicable (like the Gaussian), neither the inverse method (solving integrals which is necessary for the cdf F are generally difficult in high dimensions) nor the rejection method (volumes, in this case the rejection volume, grow exponentially fast with dimensions) are practical methods. Usually even *modes* of a distribution (which could serve as *anchor points* for a sampling procedure) are not known at all.

What is necessary to generate samples in high dimensional spaces is a method that explores efficiently the practically unknown distribution and generates valid (i.e. *representative*) samples from it at the same time.

To model random exploration a popular choice are *first order Markov Chains*, a sequence of random variables whose distribution evolves over time as a function of past realizations [271]. A first order Markov Chain is defined to be a series of random variables $\mathbf{x}_1, \ldots, \mathbf{x}_N$ such that

$$p(\mathbf{x}_k \mid \mathbf{x}_1, \ldots, \mathbf{x}_{k-1}) = p(\mathbf{x}_k \mid \mathbf{x}_{k-1}), \ \forall k \tag{2.189}$$

Mathematically, a Markov Chain is specified by an initial probability distribution $p(\mathbf{x}_0)$ and a transition probability $p(\mathbf{x}_t \mid \mathbf{x}_{t-1})$. In order for the chain to produce actual values from a desired distribution $p(\mathbf{x})$, it must be ensured that the *stationary distribution* $\pi(\mathbf{x})$ of the chain is $p(\mathbf{x})$. That is one can design a desired *global* behavior of a markov chain by shaping the *local* transition operator $T(\mathbf{x}_t, \mathbf{x}_{t-1}) \equiv p(\mathbf{x}_t \mid \mathbf{x}_{t-1})$. For a given desired stationary distribution $\pi(\mathbf{x})$ many possible transition operators exist.

It turns out that only two properties are necessary to construct a Markov Chain with a desired stationary distribution $\pi(\mathbf{x})$:

- the chain must be *ergodic*:

$$p(x_t) \to \pi(x), \text{ for } t \to \infty \text{ and arbitrary } p(x_o) \tag{2.190}$$

  A markov chain is ergodic if it is aperiodic (no state, i.e. the same random variable, is revisited periodically) and irreducible (every state is reachable by every other state). That means it has at most one stationary distribution.

- $p(x)$, the desired distribution, must be an *invariant distribution* with respect to the transition operator $T(x_t, x_{t-1})$.

$$p(x') = \int p(x)T(x', x)dx \tag{2.191}$$

A useful property of a transition operator $T(x', x)$ is *detailed balance* with respect to some distribution $q(x)$:

$$q(x)T(x, x') = q(x')T(x', x), \forall x, x' \tag{2.192}$$

A Markov Chain that satisfies detailed balance is also called a *reversible* markov chain: it is just as likely that we arbitrarily pick some state x and move to x' than the other way around. Detailed balance of a transition operator with respect to some distribution $p(x)$ is a useful property of a chain because it implies that $p(x)$ is the stationary distribution of the induced markov chain [5]:

$$\int p(x)T(x', x)dx = \int p(x')T(x', x)dx = p(x) \cdot 1 \tag{2.193}$$

MCMC methods produce estimates with the Monte Carlo method using samples generated via Markov Chains. The following three methods are currently the most widely used approaches. They are all based on a transition operator that fulfills detailed balance.

GIBBS SAMPLING. *Gibbs sampling* [107] simply picks single dimensions i of the random vector x in turn and resamples $p(x_i \mid x_{-i})$. So the underlying transition operator is

$$T(x', x) = p(x'_i \mid x_{-i})\delta(x'_{-i} = x_{-i}). \tag{2.194}$$

It can be shown that Gibbs Sampling is a special case of Metropolis-Hastings Sampling (Eq. (2.197)), and therefore satisfies automatically detailed balance (because Metropolis-Hastings Sampling satisfies it, Eq. (2.198)).

Standard Gibbs sampling updates only one variable (single-site updates) per Markov Chain step, so successive samples will be highly correlated, which motivates subsampling. It is best used when distributions factorize, i.e. groups of variables are *conditionally* independent. In this case, these variables can be updated in parallel (*blocked Gibbs sampling*). Gibbs sampling is not effective when variables are strongly correlated.

5 Note that detailed balance is a sufficient but not a necessary condition for a stationary distribution.

METROPOLIS-HASTINGS SAMPLING. In the Metropolis-Hastings sampling scheme [257, 143] a proposal distribution $q(\mathbf{x}' \mid \mathbf{x})$ is utilized to generate a step in the Markov Chain. Importantly (and different to the proposal distribution used by e.g. the rejection method), this proposal distribution depends on the current state of the Markov Chain. A candidate $\mathbf{x}'$ for the new state of the Markov Chain is sampled from $q(\mathbf{x}' \mid \mathbf{x})$ and accepted with probability a:

$$a = \min\left(\frac{q(\mathbf{x} \mid \mathbf{x}')p(\mathbf{x}')}{q(\mathbf{x}' \mid \mathbf{x})p(\mathbf{x})}\right) \tag{2.195}$$

So the transition operator for Metropolis-Hastings can be written as

$$T(\mathbf{x}', \mathbf{x}) = q(\mathbf{x}' \mid \mathbf{x})a \tag{2.196}$$

Note that a can even be computed when $p(\mathbf{x}) = \hat{p}(\mathbf{x})/Z$ is only known up to the normalization constant Z! If $\mathbf{x}'$ is rejected, the new state of the Markov Chain is set to $\mathbf{x}$ again.

Given the definition of a, it is easy to see that Gibbs sampling is indeed a special kind of Metropolis-Hastings sampling with an acceptance rate of 1:

$$\begin{aligned} a &= \frac{q(\mathbf{x} \mid \mathbf{x}')p(\mathbf{x}')}{q(\mathbf{x}' \mid \mathbf{x})p(\mathbf{x})} = \frac{p(\mathbf{x}_i \mid \mathbf{x}'_{-i})p(\mathbf{x}')}{p(\mathbf{x}'_i \mid \mathbf{x}_i)p(\mathbf{x})} \\ &= \frac{p(\mathbf{x}_i \mid \mathbf{x}'_{-i})p(\mathbf{x}'_i \mid \mathbf{x}'_{-i})p(\mathbf{x}'_{-i})}{p(\mathbf{x}'_i \mid \mathbf{x}_i)p(\mathbf{x}_i \mid \mathbf{x}_{-i})p(\mathbf{x}_{-i})} = 1, \quad \text{because } \mathbf{x}_{-i} \equiv \mathbf{x}'_{-i} \end{aligned} \tag{2.197}$$

With respect to the desired stationary distribution $p(\mathbf{x})$ the transition operator Eq. (2.196) fulfills detailed balance:

$$\begin{aligned} q(\mathbf{x}' \mid \mathbf{x})ap(\mathbf{x}) &= \min(q(\mathbf{x}' \mid \mathbf{x})p(\mathbf{x}), q(\mathbf{x} \mid \mathbf{x}')p(\mathbf{x}')) \\ &= q(\mathbf{x} \mid \mathbf{x}')p(\mathbf{x}') \min\left(\frac{q(\mathbf{x} \mid \mathbf{x}')p(\mathbf{x}')}{q(\mathbf{x}' \mid \mathbf{x})p(\mathbf{x})}, 1\right) \\ &= q(\mathbf{x} \mid \mathbf{x}')ap(\mathbf{x}') \end{aligned} \tag{2.198}$$

The proposal distribution is chosen such that it is sufficiently simple (computationally efficient) to draw samples from it. For continuous state spaces a common choice is a Gaussian centered at the current state, a reasonable proposal distribution also for correlated high-dimensional random variables. Using a Gaussian as an exemplary proposal distribution, the main problem with Metropolis-Hastings sampling becomes apparent: If the space of the goal distribution $p(\mathbf{x})$ should be adequately explored (because it is highly multimodal), the variance of the proposal Gaussian must be set high enough. However, if the variance is too large then most candidate states will be rejected. So the sampler either degenerates into local exploration or shows poor efficiency. Hence the distance explored by a typical Metropolis-Hastings sampling approach only grows with the square root of the number of computation steps [242], resembling a typical random walk behavior. The more general problem is the fact that while the proposal distribution is state dependent, it does not take the specific functional form of the stationary distribution into account.

HAMILTONIAN MONTE CARLO. Hamiltonian Monte Carlo [35, 242, 358, 32] breaks out of this poor local exploration by utilizing the behavior of physical systems under Hamiltonian dynamics – the state of such systems evolves over multiple time steps, and, if formulated adequately, can then be interpreted as a valid sample of a Markov Chain.

If the stationary distribution $p(\mathbf{x})$ is written as $p(\mathbf{x}) = \frac{\exp(-E(\mathbf{x}))}{Z}$ then the *energy function* $E(\mathbf{x})$ can be interpreted as the *potential energy* of a physical system. The potential energy $E(\mathbf{x})$ is then used to construct a *Hamiltonian* $H(\mathbf{x}, \mathbf{p})$, which additionally needs a kinetic energy term $K(\mathbf{p})$ defined over a *momentum* $\mathbf{p}$:

$$H(\mathbf{x}, \mathbf{p}) = E(\mathbf{x}) + K(\mathbf{p}) \tag{2.199}$$

The Hamiltonian itself can be used to induce a joint probability distribution over $(\mathbf{x}, \mathbf{p})$:

$$p_H(\mathbf{x}, \mathbf{p}) = \frac{\exp(-H(\mathbf{x}, \mathbf{p}))}{Z_H} = \frac{\exp(-E(\mathbf{x}))\exp(-K(\mathbf{p}))}{Z_H} \tag{2.200}$$

Because this distribution is separable its marginal distribution with respect to $\mathbf{x}$ is the stationary distribution $p(\mathbf{x})$. So a sample $(\mathbf{x}, \mathbf{p})$ from $p_H(\cdot)$ becomes a valid sample from the stationary distribution by simply discarding $\mathbf{p}$.

But how is such a joint sample generated to begin with? Given values for $\mathbf{x}$ and $\mathbf{p}$, a Hamiltonian system evolves according to the following dynamic:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{p} \\ \dot{\mathbf{p}} &= -\frac{\partial E(\mathbf{x})}{\partial \mathbf{x}} \end{aligned} \tag{2.201}$$

One can show [35, chapter 11.5] that these Hamiltonian dynamics leave $p_H(\mathbf{x}, \mathbf{p})$ invariant and hence also $p(\mathbf{x})$. However, Eq. (2.201) alone does not ensure that sampling also happens ergodically (it can't, because the value of the Hamiltonian remains constant under the dynamics, ensuring invariance). Hence, after sampling $\mathbf{x}$ from a dynamic evolution of the system, $\mathbf{p}$ must be resampled, according to $p(\mathbf{p}) = \frac{\exp(-K(\mathbf{p}))}{Z}$. A standard distribution for $\mathbf{p}$ is a Gaussian, e.g.

$$K(\mathbf{p}) = \frac{1}{2}\mathbf{p}^T\mathbf{p} \tag{2.202}$$

However, $K(\cdot)$ may also depend on the actual value of $\mathbf{x}$. Note that the approach described up to now does not need a Metropolis correction step—every state $(\mathbf{x}, \mathbf{p})$ evolved via the dynamics in Eq. (2.201) and with regularly resampled $\mathbf{p}$ is accepted.

In practice, Eq. (2.201) must be simulated with finite numerical precision. The leapfrog, or Störmer–Verlet, integration provides such a discrete time approximation for Hamiltonian dynamics [272, 139]. It

depends on one parameter ε, the stepsize of the simulation. Given $\mathbf{x_t}$ and $\mathbf{p_t}$ a step is computed as follows:

$$\begin{aligned} \mathbf{p_{t+1}}' &= \mathbf{p_t} - \frac{\varepsilon}{2}\frac{\partial E(\mathbf{x_t})}{\partial \mathbf{x}} \\ \mathbf{x_{t+1}} &= \mathbf{x_t} + \varepsilon \mathbf{p_{t+1}}' \\ \mathbf{p_{t+1}} &= \mathbf{p_{t+1}}' - \frac{\varepsilon}{2}\frac{\partial E(\mathbf{x_{t+1}})}{\partial \mathbf{x}} \end{aligned} \tag{2.203}$$

The leapfrog integration ensures that the volume in the phase space is conserved, however the value of the Hamiltonian does in general not stay constant. Hence, a correction step is necessary: If $(\mathbf{x}, \mathbf{p})$ represent the initial state of the simulation and $(\mathbf{x}', \mathbf{p}')$ is the state after a series of leapfrog steps, then $\mathbf{x}'$ is accepted with probability

$$\min(1, \exp(H(\mathbf{x}, \mathbf{p}) - H(\mathbf{x}', \mathbf{p}'))) \tag{2.204}$$

With this schema, which is generally referred to as *Hamiltonian Monte Carlo (HMC)*, it is no longer obvious that the transition operator induced by the simulated Hamiltonian dynamics is invariant. However, if at the beginning of a set of simulated leapfrog steps the integration direction (forward or backward in time) is chosen at random (together with the sampled momentum $\mathbf{p}$) then it is possible to show that HMC fulfills detailed balance [35].

Because gradient information of the proposal distribution is used, the system explores the state space linearly with respect to the number of steps of the simulated chain. Note that in general, it is difficult to choose the form of the kinetic energy such that it helps most for $p(\mathbf{x})$. Also, it is not clear for how long the simulation should be run and how big the stepsize should be chosen—both parameters can be itself determined by proposal distributions during running HMC.

In more general terms, HMC is an example of an *auxiliary variable method* which adds additional dimensions to help $\mathbf{x}$ explore more easily and efficiently the state space of the target distribution $p(\cdot)$ [11].

As already mentioned at the beginning of this section many important questions in probabilistic modeling actually involve deterministic quantities, e.g. expectations. In Machine Learning we often look for parameters that optimize these quantities. The following two rules are very helpful if this optimization is done in a gradient–based approach. Both rules (usually) rely on the Monte Carlo principle in order to be implemented efficiently.

If the random variable x is distributed according to some parameterized probability distribution, $x \sim p(x \mid \theta)$, then the *score function (SF)* estimator [101, 339] is given by

$$\frac{\partial \mathbb{E}_{p(x|\theta)}[f(x)]}{\partial \theta} = \mathbb{E}_{p(x|\theta)}\left[f(x)\frac{\partial \log p(x, \theta)}{\partial \theta}\right] \tag{2.205}$$

This equality is derived with the *log derivative identity*:

$$\frac{\partial \log p(x \mid \theta)}{\partial \theta} = \frac{1}{p(x \mid \theta)} \frac{\partial p(x \mid \theta)}{\partial \theta} \tag{2.206}$$

Specifically,

$$\begin{aligned} \frac{\partial \mathbb{E}_{p(x|\theta)}[f(x)]}{\partial \theta} &= \frac{\partial \sum_x p(x \mid \theta) f(x)}{\partial \theta} = \sum_x \frac{\partial p(x \mid \theta)}{\partial \theta} f(x) \\ &= \sum_x p(x \mid \theta) \frac{\partial \log p(x \mid \theta)}{\partial \theta} f(x) \\ &= \mathbb{E}_{p(x|\theta)} \left[f(x) \frac{\partial \log p(x, \theta)}{\partial \theta} \right] \end{aligned} \tag{2.207}$$

Derivative and expectation can be simply swapped if x is a deterministic, differentiable function of θ and another random variable z:

$$\frac{\partial \mathbb{E}_{p(z)}[f(x(\theta, z))]}{\partial \theta} = \mathbb{E}_{p(z)} \left[\frac{\partial f(x(\theta, z))}{\partial \theta} \right] \tag{2.208}$$

This *pathwise derivative (PD)* estimator is valid if and only if $f(x(\theta, z))$ is a continuous function of θ for all z [114, 339], so x can not be a discrete random variable!

Both estimators can be used under different circumstances and have different advantages [339]. The SF estimator is more generally applicable (e.g. $f(\cdot)$ can be discontinuous) and only requires sample values from $f(x)$ instead of the derivative $f'(x)$ (in the case of applying the Monte Carlo principle). However, its gradient estimate tends to have high variance, in particular if $f(x)$ is rather smooth. In this case it is often preferable to transform the SF estimator into an PD estimator by moving the parameter θ from the distribution into the expectation through an appropriate reparameterization [101, 200, 307, 339].

If θ appears in both the probability distribution over the expectation and in $f(\cdot)$ then both Eq. (2.205) and Eq. (2.208) are applied, using the product rule and the derivation from Eq. (2.207) [339]. A detailed technical discussion of the above gradient estimators is given in [114].

2.6 GRAPHICAL MODELS

Probability Theory is the basic language to describe uncertainty, but how can one describe a probabilistic system with a large number of interacting random variables in such a way that all complexities remain manageable? E.g. how does one encode structural information—the covariance matrix expresses such structural information, but only in a limited way.

Relationships between random variables are determined through the concept of statistical independence. A compact but also flexible

way to encode these relationships is the popular framework of *Graphical Models* [204, 410]: Graphical Models are graphs whose nodes correspond to random variables and edges encode statistical dependencies (that is, an absent edge between two nodes *may* indicate statistical independence). This section gives a concise introduction into this framework. Importantly (and distinct from most books on Graphical Models that spend most of the time on tabular Graphical Models) I will only consider *parameterized* Graphical Models, i.e. the edges of the graph are equipped with weights and the goal of learning is to determine these weights. Also differently to most texts I do not cover Factor Graphs and also skip the so called Plate notation [35]. Graphical Models are depicted with directed or undirected edges (representing the learnable parameters of the models) and nodes. Nodes either represent random variables, depicted as circles ◯ or deterministic computations, depicted as diamonds ◇.

2.6.1 *Undirected Graphical Models*

The simplest approach to encode *conditional independence* is a direct representation of this fact: Two nodes are connected by an *undirected* edge if they are conditionally dependent given all others. Or, to put it the other way around, $\mathbf{x}$ is conditionally independent of $\mathbf{y}$ given the set of random variables $\mathcal{Z}$ ($\mathbf{x} \perp\!\!\!\perp \mathbf{y} \mid \mathcal{Z}$), when every path between $\mathbf{x}$ and $\mathbf{y}$ contains some node $\mathbf{z} \in \mathcal{Z}$. Clearly, $\mathbf{x}$ is independent of all other random variables in the graph if one conditions on its direct neighbors. This set is called the *Markov blanket* of $\mathbf{x}$.

An undirected graphical model induces *factorized* probability distributions over its nodes when *non-negative potential functions* $\psi(\cdot)$ are assigned to the set $\mathcal{C}$ of its cliques[6]:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(\mathbf{x}_C, \theta_C) \tag{2.209}$$

The normalizing constant (also *partition function*) $Z(\theta)$ sums (or integrates[7]) over all *configurations* of the random variable $\mathbf{x}$, i.e.

$$Z(\theta) = \sum_{\mathbf{x}} \prod_{C \in \mathcal{C}} \psi_C(\mathbf{x}_C, \theta_C) \tag{2.210}$$

To simplify notation I will write Z instead and leave out the dependence on θ.

As it turns out, if a density (or, in the discrete case, a mass function) $p(\mathbf{x})$ factors according to a density that is induced by an undirected

6 Usually the potential functions are defined over (non-linear) functions $\phi(\cdot)$ of the random variables $\mathbf{x}$, i.e. *features* from the input data. However, to maintain readability, everything is defined on the untransformed random variables.

7 Without loss of generality and no other assumptions given, random variables are considered to be discrete, so marginalization is done with sums.

Graphical Model, then it satisfies all the conditional independence constraints that are defined by the undirected graph. On the other hand, if a probability distribution has all the conditional independence constraints defined by some undirected graph then its associated probability density factors according to the induced probability function from Eq. (2.209) (*Hammersley-Clifford Theorem*, [31]). Actually, the theorem is only true for strictly positive distributions, which I will focus from now on. So we can write Eq. (2.209) in the *Gibbs representation*:

$$p(\mathbf{x} \mid \theta) = \frac{1}{Z} \exp\left(\sum_{C \in \mathcal{C}} \log \psi_C(\mathbf{x}_C, \theta_C)\right) \tag{2.211}$$

with

$$Z = \sum_{\mathbf{x}} \sum_{C \in \mathcal{C}} \log \psi_C(\mathbf{x}_C, \theta_C) \tag{2.212}$$

The argument of the $\exp(\cdot)$ function is often denoted the negative *energy function* $-E(\mathbf{x}, \theta)$.

Given observed data $\mathcal{D}$ the goal is as usual to determine the parameters θ (the weights on the edges of the undirected graph). Note that without any specific partitioning of the random variable $\mathbf{x}$ into input and output elements, I generally consider the case of unsupervised learning (though everything is applicable to the supervised case, too). Following the Maximum Likelihood Estimation principle, the first derivative of the loglikelihood function is necessary: either the optimum can be found analytically in closed form or, in the standard case, the gradient can be used by an iterative optimization procedure. As the elements in $\mathcal{D}$ are assumed to be i.i.d it is enough to compute the first derivative with respect to the parameters θ for one element $\mathbf{x} \in \mathcal{D}$:

$$\begin{aligned}
\frac{\partial \log p(\mathbf{x} \mid \theta)}{\partial \theta} &= \sum_{C \in \mathcal{C}} \frac{\partial \log \psi_C(\mathbf{x}_C, \theta)}{\partial \theta} - \frac{\partial \log Z}{\partial \theta} \\
&= \sum_{C \in \mathcal{C}} \frac{\partial \log \psi_C(\mathbf{x}_C, \theta_C)}{\partial \theta} \\
&\quad - \frac{1}{Z} \sum_{\mathbf{x}'} \frac{\partial \exp\left(\sum_{C \in \mathcal{C}} \log \psi_C(\mathbf{x}'_C, \theta_C)\right)}{\partial \theta} \\
&= \sum_{C \in \mathcal{C}} \left(\frac{\partial \log \psi_C(\mathbf{x}_C, \theta_C)}{\partial \theta} \right. \\
&\quad \left. - \sum_{\mathbf{x}'} p(\mathbf{x}' \mid \theta) \frac{\partial \log \psi_C(\mathbf{x}'_C, \theta_C)}{\partial \theta} \right)
\end{aligned} \tag{2.213}$$

This is a little bit unwieldy and reads simpler in terms of the energy function $E(\mathbf{x}, \theta)$:

$$\frac{\partial \log p(\mathbf{x} \mid \theta)}{\partial \theta} = \frac{\partial E(\mathbf{x}, \theta)}{\partial \theta} - \sum_{\mathbf{x}'} p(\mathbf{x}' \mid \theta) \frac{\partial E(\mathbf{x}', \theta)}{\partial \theta} \tag{2.214}$$

The second term which is due to the normalization constant Z computes an expectation of the energy gradient with respect to the *current* probability model (i.e. the current setting of θ). This expectation may be difficult to evaluate exactly, so the Monte Carlo Principle and efficient sampling strategies become necessary.

The simplest possible model for Eq. (2.211) consists of n binary random variables where the energy is defined over pairs of nodes (if there were no edges at all then learning would result in estimating n independent Bernoulli distributions, resulting in Eq. (2.152)). The negative energy function of this *fully visible Boltzmann Machine vBM* is

$$-E(x,\theta) = \sum_i x_i \hat{\theta}_i + \frac{1}{2}\sum_i \sum_j x_i \tilde{\theta}_{ij} x_j = x^T \hat{\theta} + \frac{1}{2} x^T \tilde{\theta} x \quad (2.215)$$

with $\theta = (\hat{\theta} \in \mathbf{R}^n, \tilde{\theta} \in \mathbf{R}^{n\times n})$, $\tilde{\theta}$ symmetric and $\tilde{\theta}_{ii} \equiv 0$. The first term of Eq. (2.214) is easy to compute, using Eq. (2.70) and Eq. (2.78) for the derivatives of $\hat{\theta}$ and $\tilde{\theta}$ respectively:

$$\begin{aligned} \frac{\partial E(x,\theta)}{\partial \hat{\theta}} &= -x^T \\ \frac{\partial E(x,\theta)}{\partial \tilde{\theta}} &= -(\mathrm{vec}(xx^T))^T \end{aligned} \quad (2.216)$$

But the second term requires a sum over all possible binary configurations and is therefore in general not tractable. However, Gibbs sampling is particularly simple in this case:

$$\begin{aligned} p(x_i \mid x_{-i}) &= \frac{p(x_i \mid x_{-i})}{p(x_{-i})} = \frac{p(x_i \mid x_{-i})}{\sum_{x_i} p(x)} \\ &= \frac{\exp\left(-E(x,\theta)\right)/Z}{\sum_{x_i\in\{0,1\}} \exp\left(-E(x,\theta)\right)/Z} \\ &= \frac{\exp\left((\hat{\theta}_i + \sum_k \tilde{\theta}_{ik} x_k) x_i\right)}{1 + \exp\left(\hat{\theta}_i + \sum_k \tilde{\theta}_{ik} x_k\right)} \end{aligned} \quad (2.217)$$

For $x_i \equiv 1$ this simplifies to

$$\begin{aligned} p(x_i = 1 \mid x_{-i}) &= \frac{\exp\left(\hat{\theta}_i + \sum_k \tilde{\theta}_{ik} x_k\right)}{1 + \exp\left(\hat{\theta}_i + \sum_k \tilde{\theta}_{ik} x_k\right)} \\ &= \frac{1}{1 + \exp\left(-(\hat{\theta}_i + \sum_k \tilde{\theta}_{ik} x_k)\right)} \\ &= \sigma\left(\hat{\theta}_i + \sum_k \tilde{\theta}_{ik} x_k\right) \end{aligned} \quad (2.218)$$

where $\sigma(\cdot)$ is the *logistic sigmoid non-linearity* defined as

$$\sigma(u) = \frac{1}{1+\exp(-u)} = \frac{\exp(u)}{1+\exp(u)} \quad (2.219)$$

So while a simple sampling strategy exists, learning is still very slow due to the inherent inefficiency of Gibbs Sampling and the general usage of iterative gradient ascent (or, in a more abstract way, the learning loop (gradient ascent) relies on an inefficient inference loop (sampling)).

What happens if the same graphical structure is used for real valued random variables $\mathbf{x}$?

$$p(\mathbf{x}) \propto \exp\left(-\frac{1}{2}\mathbf{x}^T\tilde{\theta}\mathbf{x} + \hat{\theta}^T\mathbf{x}\right) \tag{2.220}$$

which can be identified as a Gaussian in natural parameterization (Eq. (2.124)). That is

$$\begin{aligned}\boldsymbol{\Sigma} &= \tilde{\theta}^{-1} \\ \boldsymbol{\mu} &= \boldsymbol{\Sigma}^{-1}\hat{\theta}\end{aligned} \tag{2.221}$$

Differently to the general case, such a *Gaussian Markov Random Field* which is simply a Gaussian can be solved in closed form with respect to its parameters. Given n i.i.d. observations in the training set $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, the likelihood with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ is

$$\prod_{i=1}^{n} \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{2.222}$$

and hence the negative loglikelihood $\ell(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is

$$\ell(\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \underbrace{\frac{nd}{2}\log 2\pi}_{\text{const.}} + \underbrace{\frac{n}{2}\log|\boldsymbol{\Sigma}|}_{\text{depends on } \boldsymbol{\Sigma}} + \underbrace{\frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}_i - \boldsymbol{\mu})}_{\text{depends on } \boldsymbol{\mu},\boldsymbol{\Sigma}} \tag{2.223}$$

Using Eq. (2.70) and Eq. (2.72) the first derivative of the loglikelihood with respect to $\boldsymbol{\mu}$ is

$$\frac{\partial \ell(\boldsymbol{\mu}, \boldsymbol{\Sigma})}{\partial \boldsymbol{\mu}} = \sum_{i=1}^{n} \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu} - \mathbf{x}_i) \tag{2.224}$$

and hence the MLE estimate for $\boldsymbol{\mu}$ is

$$\boldsymbol{\mu}_{\text{MLE}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i \tag{2.225}$$

In order to compute the first derivative of $\ell(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with respect to $\boldsymbol{\Sigma}^{-1}$ Eq. (2.78) and Eq. (2.86) are necessary:

$$\frac{\partial \ell(\boldsymbol{\mu}, \boldsymbol{\Sigma})}{\partial \boldsymbol{\Sigma}^{-1}} = -\frac{n}{2}\boldsymbol{\Sigma} + \frac{1}{2}\sum_{i}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T \tag{2.226}$$

and therefore

$$\boldsymbol{\Sigma}_{\text{MLE}} = \frac{1}{n}\sum_{i}(\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^T \tag{2.227}$$

Usually $\boldsymbol{\mu}$ is not known so $\boldsymbol{\Sigma}_{\text{MLE}}$ must rely on $\boldsymbol{\mu}_{\text{MLE}}$ instead. In order to ensure an unbiased estimate, a tiny correction must be made:

$$\boldsymbol{\Sigma}'_{\text{MLE}} = \frac{1}{n-1}\sum_i (\mathbf{x}_i - \boldsymbol{\mu}_{\text{MLE}})(\mathbf{x}_i - \boldsymbol{\mu}_{\text{MLE}})^\top \tag{2.228}$$

So basically the MLE for the parameters of the Gaussian are simply the *empirical mean* and the *empirical covariance matrix*.

Both the vBM and the Gaussian utilize at most second order information (as evident from Eq. (2.216), Eq. (2.225) and Eq. (2.227)). Already quite simple problems can't be successfully tackled with these types of models [242, p.524]. In general, Graphical Models can be made more powerful by the introduction of *latent variables* [152], usually denoted by $\mathbf{h}$. That is

$$p(\mathbf{x} \mid \boldsymbol{\theta}) = \sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta}) = \frac{1}{Z}\left(\sum_{\mathbf{h}} \exp\left(-E(\mathbf{x}, \mathbf{h}, \boldsymbol{\theta})\right)\right) \tag{2.229}$$

with

$$Z = \sum_{\mathbf{x}}\sum_{\mathbf{h}} \exp\left(-E(\mathbf{x}, \mathbf{h}, \boldsymbol{\theta})\right) \tag{2.230}$$

The general idea behind latent variables is that these either represent theoretical concepts that determine the system in an unobservable manner or actually represent unobservable physical states.

It is straightforward to compute conditionals, e.g. the posterior over latent variables given visible variables $p(\mathbf{h} \mid \mathbf{x})$ is

$$p(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta}) = \frac{p(\mathbf{x}, \mathbf{h} \mid \boldsymbol{\theta})}{p(\mathbf{x} \mid \boldsymbol{\theta})} = \frac{\exp\left(-E(\mathbf{x}, \mathbf{h}, \boldsymbol{\theta})\right)}{\sum_{\mathbf{h}} \exp\left(-E(\mathbf{x}, \mathbf{h}, \boldsymbol{\theta})\right)} \tag{2.231}$$

However, in the special case of a Gaussian this is not true: Adding additional latent variables that are also marginally Gaussian does not enhance the capacity of such a model: it stays a Gaussian (see Eq. (2.134)), the latent variables are unidentifiable.

In order to use the formulation from Eq. (2.211), the free energy $F(\mathbf{x}, \boldsymbol{\theta})$ is introduced:

$$\begin{aligned} p(\mathbf{x} \mid \boldsymbol{\theta}) &= \frac{1}{Z}\exp\left(-F(\mathbf{x}, \boldsymbol{\theta})\right) \\ &= \frac{1}{Z}\exp\left(-\log\left(\sum_{\mathbf{h}} \exp\left(-E(\mathbf{x}, \mathbf{h}, \boldsymbol{\theta})\right)\right)\right) \end{aligned} \tag{2.232}$$

that is

$$F(\mathbf{x}, \boldsymbol{\theta}) = \log\left(\sum_{\mathbf{h}} \exp\left(-E(\mathbf{x}, \mathbf{h}, \boldsymbol{\theta})\right)\right) \tag{2.233}$$

The log-likelihood gradient is then given by:

$$\begin{aligned}
\frac{\partial \log p(\mathbf{x} \mid \theta)}{\partial \theta} &= \frac{\partial F(\mathbf{x}, \theta)}{\partial \theta} - \frac{\partial \log Z}{\partial \theta} \\
&= \frac{\partial \log \left(\sum_{\mathbf{h}} \exp \left(-E(\mathbf{x}, \mathbf{h}, \theta) \right) \right)}{\partial \theta} - \frac{\partial \log Z}{\partial \theta} \\
&= \sum_{\mathbf{h}} \frac{\exp \left(-E(\mathbf{x}, \mathbf{h}, \theta) \right)}{\sum_{\mathbf{h}'} \exp \left(-E(\mathbf{x}, \mathbf{h}', \theta) \right)} \frac{\partial - E(\mathbf{x}, \mathbf{h}, \theta)}{\partial \theta} \\
&\quad - \frac{\partial \log Z}{\partial \theta} \\
&= \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{x}, \theta) \frac{\partial - E(\mathbf{x}, \mathbf{h}, \theta)}{\partial \theta} - \frac{\partial \log Z}{\partial \theta} \\
&= \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{x}, \theta) \frac{\partial - E(\mathbf{x}, \mathbf{h}, \theta)}{\partial \theta} \\
&\quad - \frac{\partial \log \left(\sum_{\mathbf{x}'} \sum_{\mathbf{h}} \exp \left(-E(\mathbf{x}', \mathbf{h}, \theta) \right) \right)}{\partial \theta} \\
&= \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{x}, \theta) \frac{\partial - E(\mathbf{x}, \mathbf{h}, \theta)}{\partial \theta} \\
&\quad - \frac{\partial \log \left(\sum_{\mathbf{x}'} \sum_{\mathbf{h}} \exp \left(-E(\mathbf{x}', \mathbf{h}, \theta) \right) \right)}{\partial \theta} \\
&= \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{x}, \theta) \frac{\partial - E(\mathbf{x}, \mathbf{h}, \theta)}{\partial \theta} \\
&\quad - \sum_{\mathbf{x}'} \sum_{\mathbf{h}} \frac{\exp \left(-E(\mathbf{x}', \mathbf{h}, \theta) \right)}{\sum_{\mathbf{x}''} \sum_{\mathbf{h}} \exp \left(-E(\mathbf{x}'', \mathbf{h}, \theta) \right)} \frac{\partial E(\mathbf{x}', \mathbf{h}, \theta)}{\partial \theta} \\
&= \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{x}, \theta) \frac{\partial - E(\mathbf{x}, \mathbf{h}, \theta)}{\partial \theta} \\
&\quad - \sum_{\mathbf{x}'} \sum_{\mathbf{h}} p(\mathbf{x}', \mathbf{h} \mid \theta) \frac{\partial - E(\mathbf{x}', \mathbf{h}, \theta)}{\partial \theta}
\end{aligned} \tag{2.234}$$

The similarity to Eq. (2.214) is not surprising. Note that now the first term is also an expectation of the energy gradient, this time with respect to the posterior distribution of the latent variables. In general this means that learning now involves two intractable terms. The first term is sometimes called the *positive phase*. It lowers the energy (i.e. increases probability mass) at training samples $\mathbf{x}$ and the *best* (as interpreted by the posterior) latent configurations. The second phase (*the negative phase*) universally increases the energy, in particular at these configurations it considers most probable. The negative phase therefore acts as a *contrastive term* and ensures that learning can end.

A *Boltzmann Machine (BM)* [2] has the same structural terms as the vBM but with (binary) latent variables:

$$-E(\mathbf{x}, \mathbf{h}, \theta) = \sum_i \mathbf{y}_i \hat{\theta}_i + \frac{1}{2} \sum_i \sum_j \mathbf{y}_i \tilde{\theta}_{ij} \mathbf{y}_j = \mathbf{y}^\top \hat{\theta} + \frac{1}{2} \mathbf{y}^\top \tilde{\theta} \mathbf{y} \tag{2.235}$$

where the visible units $\mathbf{x} \in \mathbb{R}^n$ and the latent units $\mathbf{h} \in \mathbf{R}^m$ are stacked into $\mathbf{y} = (\mathbf{x}, \mathbf{h})^\top$ and $\theta = (\hat{\theta} \in \mathbf{R}^{n+m}, \tilde{\theta} \in \mathbf{R}^{(n+m)\times(n+m)})$, $\tilde{\theta}$ again symmetric and $\tilde{\theta}_{ii} \equiv 0$. The gradient of the energy with respect to θ

is already given in Eq. (2.216), substituting $\mathbf{x}$ with $\mathbf{y}$. The overall gradient is a sum of two expectations (with respect to the latent posterior and the joint distribution respectively) over this energy gradient—these expectations are approximated with the Monte Carlo Method, using Gibbs sampling according to Eq. (2.217).

A Boltzmann Machine can be trained *rather* efficiently if its connectivity structure is sparse. In these cases, sampling can become efficient. One widely known instance of such a model is the Restricted Boltzmann Machine (RBM) [355]: Edges exist only between latent and visible units. The associated energy function can then be written as:

$$E(\mathbf{x}, \mathbf{h}, \theta) = \mathbf{x}^\intercal \hat{\theta}_\mathbf{x} + \mathbf{h}^\intercal \hat{\theta}_\mathbf{h} + \mathbf{x}^\intercal \tilde{\theta} \mathbf{h} \tag{2.236}$$

and the free energy $F(\mathbf{x}, \theta)$ is

$$\begin{aligned}
F(\mathbf{x}, \theta) &= \log \left(\sum_\mathbf{h} \exp\left(-E(\mathbf{x}, \mathbf{h}, \theta)\right) \right) \\
&= \mathbf{x}^\intercal \hat{\theta}_\mathbf{x} + \log \left(\sum_\mathbf{h} \exp\left(\mathbf{x}^\intercal \tilde{\theta} \mathbf{h} + \mathbf{h}^\intercal \hat{\theta}_\mathbf{h} \right) \right) \\
&= \mathbf{x}^\intercal \hat{\theta}_\mathbf{x} + \log \left(\sum_\mathbf{h} \prod_{j=1}^{m} \exp\left(\left(\sum_i \mathbf{x}_i \tilde{\theta}_{ij} + \hat{\theta}_j \right) \mathbf{h}_j \right) \right) \\
&= \mathbf{x}^\intercal \hat{\theta}_\mathbf{x} + \log \left(\left(1 + \exp\left(\sum_i \mathbf{x}_i \tilde{\theta}_{i1} + \hat{\theta}_1 \right) \right) \right. \\
&\quad \left. \times \left(\sum_{\mathbf{h}_{\neq 1}} \prod_{j=2}^{m} \exp\left(\left(\sum_i \mathbf{x}_i \tilde{\theta}_{ij} + \hat{\theta}_j \right) \mathbf{h}_j \right) \right) \right) \\
&= \mathbf{x}^\intercal \hat{\theta}_\mathbf{x} + \log \left(\prod_{j=1}^{m} \left(1 + \exp\left(\sum_i \mathbf{x}_i \hat{\theta}_{ij} + \hat{\theta}_{\mathbf{h}j} \right) \right) \right) \\
&= \mathbf{x}^\intercal \hat{\theta}_\mathbf{x} + \sum_{j=1}^{m} \log \left(1 + \exp\left(\sum_i \mathbf{x}_i \hat{\theta}_{ij} + \hat{\theta}_{\mathbf{h}j} \right) \right)
\end{aligned} \tag{2.237}$$

The latent variables are conditionally independent given the visible variables (and, due to symmetric formulation of the energy function, vice versa):

$$\begin{aligned}
p(\mathbf{h} \mid \mathbf{x}, \theta) = \frac{p(\mathbf{x}, \mathbf{h} \mid \theta)}{p(\mathbf{x} \mid \theta)} &= \frac{\exp\left(-E(\mathbf{x}, \mathbf{h}, \theta)\right)}{\sum_\mathbf{h} \exp\left(-E(\mathbf{x}, \mathbf{h}, \theta)\right)} \\
&= \frac{\exp\left(\mathbf{x}^\intercal \hat{\theta}_\mathbf{x} + \mathbf{h}^\intercal \hat{\theta}_\mathbf{h} + \mathbf{x}^\intercal \tilde{\theta} \mathbf{h} \right)}{\sum_\mathbf{h} \exp\left(\mathbf{x}^\intercal \hat{\theta}_\mathbf{x} + \mathbf{h}^\intercal \hat{\theta}_\mathbf{h} + \mathbf{x}^\intercal \tilde{\theta} \mathbf{h} \right)} \\
&= \frac{\exp\left(\mathbf{h}^\intercal \hat{\theta}_\mathbf{h} + \mathbf{x}^\intercal \tilde{\theta} \mathbf{h} \right)}{\sum_\mathbf{h} \exp\left(\mathbf{h}^\intercal \hat{\theta}_\mathbf{h} + \mathbf{x}^\intercal \tilde{\theta} \mathbf{h} \right)} \\
&= \prod_{j=1}^{m} \frac{\exp\left(\left(\sum_i \mathbf{x}_i \hat{\theta}_{ij} + \hat{\theta}_{\mathbf{h}j} \right) \mathbf{h}_j \right)}{\left(1 + \exp\left(\sum_i \mathbf{x}_i \hat{\theta}_{ij} + \hat{\theta}_{\mathbf{h}j} \right) \right)}
\end{aligned} \tag{2.238}$$

where I have used intermediate results from Eq. (2.237). So

$$p(\mathbf{h}_j = 1|\mathbf{x}) = \sigma\left(\sum_i \mathbf{x}_i \hat{\theta}_{ij} + \hat{\theta}_{\mathbf{h}j}\right) \tag{2.239}$$

and

$$p(\mathbf{x}_j = 1|\mathbf{h}) = \sigma\left(\sum_i \mathbf{h}_i \hat{\theta}_{ij} + \hat{\theta}_{\mathbf{x}j}\right) \tag{2.240}$$

relying on the alternative expression for the logistic sigmoid function (Eq. (2.219)). Note that in this case only two (blocked) Gibbs steps are necessary in order to get a new complete sample of the configuration. In particular, given a configuration of the visible variables, a valid sample under the current model from the posterior distribution of the latent variables can be generated in one step, so the first part of the loglikelihood gradient (see Eq. (2.234)) can be evaluated efficiently for the RBM.

However, the negative phase needs a full sample $(\mathbf{x}, \mathbf{h})$ from the current model and therefore one must resort to a Markov Chain that iteratively samples $p(\mathbf{x} \mid \mathbf{h})$ and $p(\mathbf{h} \mid \mathbf{x})$ in a blockwise manner *until equilibrium*.

Contrastive Divergence (CD) [149] circumvents this problem by truncating the Markov Chain after several steps. More specifically, with CD_k the blockwise Gibbs sampling is *started at a sample from the training set* and then *stopped after* k *complete updates* of the latent and visible variables. The basic idea behind CD is that with the correct parameters θ the Markov Chain will leave the data distribution unaltered (because the data distribution is the desired invariant distribution of the chain), and therefore the chain should be started at a data sample (and not at some randomly chosen configuration, as is the standard procedure with Markov Chains). If the data distribution is not (yet) the invariant distribution for the current model than this *divergence* can already be detected after a small number of steps in the chain, without the need to wait until the chain reaches its equilibrium. So instead of computing the log-likelihood gradient $\nabla_\theta \ell(\theta)$ according to the derivation,

$$\begin{aligned}\nabla_\theta \ell(\theta) = \sum_n \Big(&- \mathbb{E}_{p(\mathbf{h}|\mathbf{x},\theta)} \left[\nabla_\theta \left(E(\mathbf{x}, \mathbf{h}, \theta)\right)\right] \\ &+ \mathbb{E}_{p(\mathbf{h},\mathbf{x}|\theta)} \left[\nabla_\theta \left(E(\mathbf{x}, \mathbf{h}, \theta)\right)\right] \Big),\end{aligned} \tag{2.241}$$

Contrastive Divergence approximates it as follows:

$$\begin{aligned}\nabla_\theta \ell(\theta) \approx \sum_n \Big(&- \mathbb{E}_{p(\mathbf{h}|\mathbf{x},\theta)} \left[\nabla_\theta \left(E(\mathbf{x}, \mathbf{h}, \theta)\right)\right] \\ &+ \mathbb{E}_{\hat{p}_k(\mathbf{h},\mathbf{x}|\theta)} \left[\nabla_\theta \left(E(\mathbf{x}, \mathbf{h}, \theta)\right)\right] \Big)\end{aligned} \tag{2.242}$$

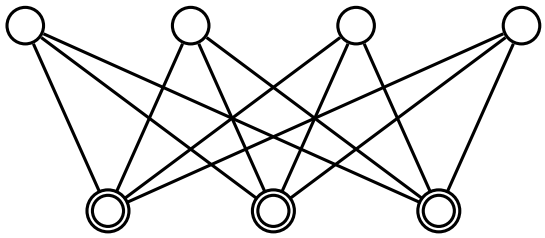


Figure 2.1: A visual representation of an exemplary RBM with 4 latent units and three visible units (identified by a double circle). Edges only exist between latent and visible units.

where $\hat{p}_k(\mathbf{h}, \mathbf{x} \mid \theta)$ is the model distribution over visible and latent units obtained when running blocked Gibbs-sampling for k iterations, starting the chain from samples from the training set.

Starting the Markov Chain at training samples and running it only for a fixed number of steps however leads to suboptimal generative models: Areas in the probability landscape that are far from any training samples have a very low chance of getting reached by a particle through the chain and therefore will never open up the probability mass they occupy. What happens overall is that the model overfits the probability landscape to the samples from the dataset. This effect can be alleviated by setting k high enough (e.g. $k = 10$ is often reported to be good enough), but in theory the problem persists.

A better idea to model true samples from the probability distribution induced by the *current* setting of θ is to have a certain number of particles (i.e. configurations of the variables) representing this distribution that are updated through several steps of iterated Gibbs sampling without resetting them to samples from the training set after every gradient update. So these particles *persist* over the complete learning phase which gives this approach the name *persistent Contrastive Divergence (PCD)* [387, 425, 424]. Because the model parameters θ only change a little bit between every gradient-induced update, it is valid to start the Markov Chain at the persisted particles. Overall, these particles then can roam freely in the complete configuration space and are able to correct the probability density far away from data samples. Combining RMBs and PCD it is possible to train Deep Boltzmann Machines (DBM) in an efficient way [323]. DBM are Boltzmann Machines with many layers where units within a layer are not connected.

2.6.2 *Directed Graphical Models*

Considering the underlying graph of an RBM (Fig. 2.1) efficient conditional sampling is possible because every path between two hidden (visible) units is interleaved by at most one observed visible (hidden) unit. How could we model the reverse semantic, i.e. two units *be-*

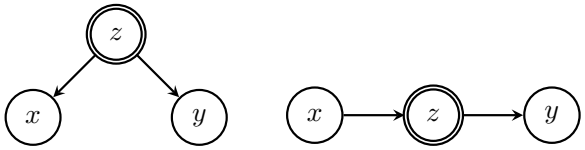


Figure 2.2: The two possible conditional independence relationships in a directed acyclic graph. In both cases $p(x, y \mid z) = p(x \mid z)p(y \mid z)$. The double circle around a unit indicates that it is observed. If z is marginalized out then x and y become dependent.

come dependent conditioned on an observed unit both are connected to? This semantic is very important when modeling cause and effect structures: The observed unit represents the observed effect and the two (or more) unobserved units are possible causes.

As a very simple example, imagine three binary random variables. One variable models whether the lawn in front of our house is wet in the morning. The other two variables represent the two possible causes for a wet lawn, either a nightly rain shower or the sprinkler running *accidentally* very early. Both have a very small probability of being true, and, without any given observation, are independent of each other. The binary variable for the wet lawn is of course a function of the true (unobserved) state of the two causes. Both become dependent on each other as soon as the lawn variable is observed. In particular, in the case of a wet lawn, both would *compete* to explain this observation—if one of the two causes is true the other becomes very improbable (because the unconditional priors are already very low).

This behavior is called *explaining away* and can not be represented by an undirected graphical model. However, a *directed acyclic graph (DAG)* can realize this type of conditional dependence structure. More specifically, a DAG can represent two different *conditional independence* relations (Fig. 2.2) and the above described *conditional dependence* semantic (Fig. 2.3).

Interestingly, there are also sets of conditional independence constraints that can only be expressed by an undirected graph but not with a directed acyclic graph. The simplest example for such a case is given in Figure 2.4.

The set of independence relations that both undirected and directed graphical models can describe are represented by *chordal graphs* [204]. Yet, a Markov Network is better suited to describe soft constraints between the variables of a probabilistic system and on the other hand DAGs are better suited to express causal generative models. Similar to the undirected graphical models, a DAG also induces factorized probability distributions. In this case these are simply prod-

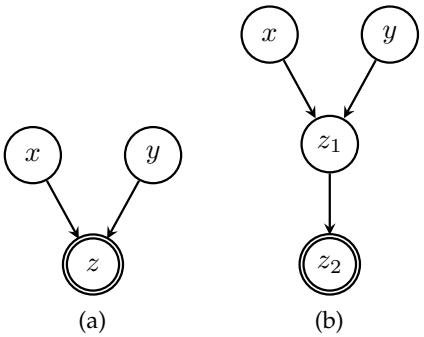


Figure 2.3: x and y are marginally independent. As soon as a *direct descendant* (a) or an *indirect descendant* (b) of x and y is observed, both become statistically dependent. Due to these possible long range interactions along directed paths the *Markov blanket* of a unit in a directed graph comprises the set of parents, children and co-parents of the unit.

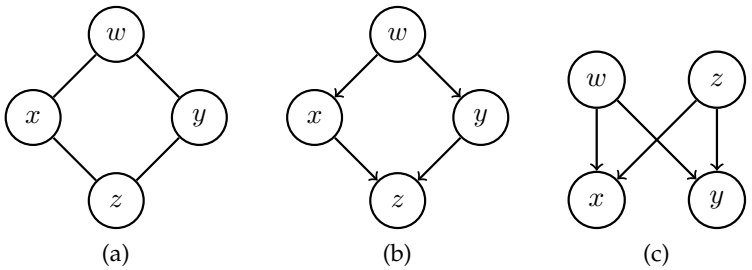


Figure 2.4: (a) Opposite units are conditional independent, given their respective neighbors are observed. No directed *acyclic* graph with the same graph structure can encode these conditional independence relationships over the same units, as can be seen in (b) and (c). Both can only fulfill one of the two conditional independence constraints induced by (a).

ucts of conditional probability distributions derived from the local graph structure

$$p(\mathbf{x}, \mathbf{h} \mid \theta) = \prod_i p(\mathbf{x}_i \mid \pi_i(\mathbf{x}), \theta) \tag{2.243}$$

$\pi_i(\mathbf{x})$ denotes the direct predecessors (parents) of $\mathbf{x}_i$ in the underlying directed graph. Obviously, Eq. (2.243) is a special case of Eq. (2.209) with already normalized potential functions $\psi(\cdot)$, that is $Z = 1$. Re-

membering Eq. (2.234), maybe this way of writing distributions simplifies the loglikelihood gradient?

$$\begin{aligned}\frac{\partial \log p(\mathbf{x} \mid \theta)}{\partial \theta} &= \frac{1}{p(\mathbf{x} \mid \theta)} \frac{\partial p(\mathbf{x} \mid \theta)}{\partial \theta} \\ &= \frac{1}{p(\mathbf{x} \mid \theta)} \frac{\partial \sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{h} \mid \theta)}{\partial \theta} \\ &= \sum_{\mathbf{h}} \frac{p(\mathbf{h} \mid \mathbf{x}, \theta)}{p(\mathbf{x}, \mathbf{h} \mid \theta)} \frac{\partial p(\mathbf{x}, \mathbf{h} \mid \theta)}{\partial \theta} \\ &= \sum_{\mathbf{h}} p(\mathbf{h} \mid \mathbf{x}, \theta) \frac{\partial \log p(\mathbf{x}, \mathbf{h} \mid \theta)}{\partial \theta}\end{aligned} \tag{2.244}$$

So this is very similar to the undirected case, gradients of the complete loglikelihood are weighted by the posterior distribution. If $p(\mathbf{x}, \mathbf{h} \mid \theta)$ is written in the Gibbs representation (Eq. (2.211)), then Eq. (2.244) resembles Eq. (2.234) without the annoying second part that is expensive to compute in the undirected case. While this is a nice effect of the likelihood being the product of normalized conditional probability distributions, $p(\mathbf{h} \mid \mathbf{x}, \theta)$ is still a complex (usually intractable) distribution.

This was also the case for undirected graphical models and there the solution was to resort to the Monte Carlo principle, but now sampling is also difficult due to the explaining away effect, the reason I originally introduced direct models. The complex conditional dependency semantics results in a very large Markov blanket for any unit (see Figure 2.3) so efficient blocked Gibbs sampling (as previously done with RBMs) is in general not possible (unless the DAG has very sparse connectivity, such as trees). At the same time, Metropolis-Hastings based approaches are also inefficient as explaining away leads to highly coupled variables.

If all units are observed then Eq. (2.244) is considerably simplified. In this case, without loss of generality, two groups of units can be identified: Units with only outgoing edges and units with only incoming edges—inputs $\mathbf{x}$ and outputs $\mathbf{y}$. Thus the training set actually comprises n input/output tuples, forming a supervised learning task. Again, without loss of generality, the most basic model in the fully observed instance is then a DAG with a layer of input units $\mathbf{x}$ and a layer of output units $\mathbf{y}$. These two layers are connected by a weight matrix $\boldsymbol{W}$ and the DAG models the conditional distribution $\mathbf{y} \mid \mathbf{x}$. The more complex fully observed models are always several combined instances of this basic model.

If $\mathbf{y} \mid \mathbf{x}$ is distributed according to a multivariate Gaussian then this supervised problem is called *linear regression* (also see Eq. (2.153))[8]:

$$p(\mathbf{y} \mid \mathbf{x}) = \mathcal{N}(\mathbf{y} \mid \boldsymbol{W}\mathbf{x} + \boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{2.245}$$

8 As already pointed out earlier, instead of $\mathbf{x}$ usually non-linear transformations $\phi(\cdot)$ of $\mathbf{x}$ are used in practice.

with $\mathbf{x} \in \mathbf{R}^d$, $\mathbf{y} \in \mathbf{R}^o$, $\theta = (\boldsymbol{\mu} \in \mathbf{R}^o, \boldsymbol{W} \in \mathbf{R}^{o \times d})$. More specifically, if $\mathbf{y}$ is only a scalar then the negative loglikelihood function over the whole training set $\mathcal{D}$ can be written as (see Eq. (2.156)):

$$\ell(\theta) \propto -\frac{n}{2} \ln \sigma^2 - \frac{1}{2\sigma^2} \sum_i \left(y_i - \boldsymbol{w}^T \mathbf{x}_i\right)^2 \tag{2.246}$$

Stacking all inputs $\mathbf{x}_i$ into a matrix $\mathbf{X}$ in a rowwise fashion (and equivalently all scalar targets y_i into a column vector $\mathbf{y}$) this can be expressed more compactly as

$$\ell(\theta) \propto -\frac{n}{2} \ln \sigma^2 - \frac{1}{2\sigma^2} (\mathbf{y} - \mathbf{X}\hat{\theta})^T (\mathbf{y} - \mathbf{X}\hat{\theta}) \tag{2.247}$$

Hereby, $\hat{\theta}^T \equiv (\mu, \boldsymbol{w}^T)$ and every input sample is extended by a 1 (i.e. the first column of $\mathbf{X}$ is $\mathbf{1}_n$). Using Eq. (2.70) and Eq. (2.73) the gradient of Eq. (2.247) with respect to $\hat{\theta}$ is

$$\nabla_{\hat{\theta}} \ell(\theta) \propto \mathbf{X}^T \mathbf{X} \hat{\theta} - \mathbf{X}^T \mathbf{y} \tag{2.248}$$

and therefore the maximum likelihood estimate for $\hat{\theta}$ is

$$\hat{\theta}_{\text{MLE}} = \underbrace{(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T}_{= \mathbf{X}^\dagger} \mathbf{y} \tag{2.249}$$

The expression $\mathbf{X}^\dagger$ is called the Moore-Penrose pseudoinverse [266]. Instead of actually computing the inverse of a possibly numerical problematic matrix, $\hat{\theta}_{\text{MLE}}$ is usually computed using a Singular Valued Decomposition (Eq. (2.27)) of $\mathbf{X}$ [266].

With $\hat{\theta}_{\text{MLE}}$ used in Eq. (2.247), it is straightforward to determine σ^2_{MLE}:

$$\sigma^2_{\text{MLE}} = \frac{1}{n} (\mathbf{y} - \mathbf{X}\hat{\theta}_{\text{MLE}})^T (\mathbf{y} - \mathbf{X}\hat{\theta}_{\text{MLE}}) \tag{2.250}$$

If $\mathbf{y}$ is instead vector valued (e.g. o dimensional), than the above derivation can be done independently o times, as long as $\mathbf{y} \mid \mathbf{x}$ has a diagonal covariance structure (i.e. the regressed outputs are independent given the input). If this is not the case then no closed form solution exists and gradient-based approaches must be employed.

If, as a further example, $y \mid \mathbf{x}$ is distributed according to a Bernoulli distribution, i.e.

$$p(y \mid \mathbf{x}, \theta) = \text{Bern}(y \mid \sigma(\boldsymbol{w}^T \mathbf{x} + \mu)) \tag{2.251}$$

then no closed from solution for $\hat{\theta}^T = (\mu, \boldsymbol{w}^T)$ exists. Instead, the MLE solution for this *logistic regression* model is attained by optimizing the binary cross-entropy function (Eq. (2.163)):

$$\ell(\hat{\theta}) = \sum_i y_i \log \sigma(\hat{\theta}^T \mathbf{x}) + (1 - y_i)(1 - \log \sigma(\hat{\theta}^T \mathbf{x})) \tag{2.252}$$

using gradient information:

$$\nabla_{\hat{\theta}} \ell(\hat{\theta}) = \sum_i \left(\sigma(\hat{\theta}^T x_i) - y_i \right) x_i \tag{2.253}$$

Logistic regression (and its multivariate extension *multinomial regression*) is a convex optimization problem and therefore gradient descent (Eq. (2.42)) will not get stuck in local minima. However, the unique optimum may be at infinity (see [35, p. 222] and therefore a MAP (Eq. (2.164)) approach appears to be reasonable.

A large class of different conditional output distributions can be modeled using the basic model of an affine transformation applied to the inputs x, depending on the employed *link function* after the transformation. See [266] for more details on *Generalized Linear Models (GLM)*.

If latent variables are introduced, the models become more powerful in general, but also more difficult to tackle. Unsurprisingly, if both latent and visible units are Gaussian everything remains simple. Let

$$\begin{aligned} p(h) &= \mathcal{N}(h \mid 0, I) \\ p(x \mid h) &= \mathcal{N}(x \mid Wh + \mu, \sigma^2 I) \end{aligned} \tag{2.254}$$

with $h \in \mathbf{R}^l$, $x \in \mathbf{R}^d$ and $\theta = (\mu, W, \sigma^2)$. This *latent linear model* is called *probabilistic Principal Component Analysis (pPCA)* [389] and is, viewed as a generative model, marginally representing a Gaussian with a low rank structure (see Eq. (2.140)):

$$p(x \mid \theta) = \mathcal{N}(x \mid \mu, WW^T + \sigma^2 I) \tag{2.255}$$

This model is useful if a high-dimensional Gaussian has a low-rank structure. In this case, pPCA allows to estimate the Gaussian with a much lower number of necessary samples compared to the standard Gaussian (this becomes most evident in estimating the covariance matrix, see Eq. (2.227)).

The maximum-likelihood estimates for $\theta = (W, \mu, \sigma^2)$ can be given in closed form [389]:

$$\begin{aligned} \mu_{\text{MLE}} &= \frac{1}{n} \sum_i x_i \\ W_{\text{MLE}} &= L_l (\Lambda_l - \sigma^2 I)^{\frac{1}{2}} R \\ \sigma^2_{\text{MLE}} &= \frac{1}{d-l} \sum_{j=l+1}^{d} \lambda_j \end{aligned} \tag{2.256}$$

with L_l being the $d \times l$ matrix whose columns are the first l eigenvectors of the empirical covariance matrix S (Eq. (2.227)) and Λ_l the diagonal matrix of corresponding eigenvalues (R is an arbitrary orthogonal matrix). The estimated variance σ^2 is the average of the discarded eigenvalues. The best setting for l can't be resolved with MLE.

Instead, one has to rely on a validation set and use the l with highest loglikelihood score on that set.

Inferring the latent representation for an observed $\mathbf{x}$ and given parameters θ can be done using Eq. (2.143):

$$p(\mathbf{h} \mid \mathbf{x}, \theta) = \mathcal{N}(\mathbf{h} \mid \hat{\mu}, \hat{\Sigma}) \tag{2.257}$$

with the posterior mean $\hat{\mu}$ given by

$$\hat{\mu} = \left(\sigma^2 \mathbf{I} + \mathbf{W}^T \mathbf{W}\right)^{-1} \left(\mathbf{W}^T (\mathbf{x}_i - \mu_{\text{MLE}})\right) \tag{2.258}$$

and the *input independent* (i.e. fixed) posterior covariance $\hat{\Sigma}$ given by

$$\hat{\Sigma} = \sigma^2 \left(\sigma^2 \mathbf{I} + \mathbf{W}^T \mathbf{W}\right)^{-1} \tag{2.259}$$

Probabilistic PCA becomes the famous *PCA* [189, 168, 289] for $\sigma \to 0$. Using $\mathbf{W}_{\text{MLE}}$, the posterior distribution is a collapsed Gaussian with posterior covariance $\hat{\Sigma}_{\text{PCA}} = 0$ (because $\mathbf{W}_{\text{MLE}} \mathbf{W}_{\text{MLE}}^T = \mathbf{I}$ in the limit case). The posterior mean is simply the scaled projection of $\mathbf{x}$ on the first l eigenvectors:

$$\hat{\mu}_{\text{PCA}} = \Lambda_l^{-1/2} \mathbf{L}_l^T (\mathbf{x} - \mu_{\text{MLE}}) \tag{2.260}$$

The transformation in Eq. (2.260) is usually referred to as *whitening* [102]. Applied with *all* principle components retained, it results in an empirical covariance matrix $\mathbf{S}$ that is the identity. Without the diagonal scaling Eq. (2.260) represents the standard form of PCA which is usually derived through a maximum variance formulation on the principle components or a minimum error formulation on the reconstructed backprojections [35].

The necessary eigendecomposition of the empirical covariance matrix is often computed using an SVD (Eq. (2.27)) on the data matrix $\mathbf{X}$ (the training samples $\mathbf{x} \in \mathcal{D}$ are stacked in a rowwise fashion to from $\mathbf{X}$).

The latent representation of pPCA is denoted a *distributed representations* [152]. What is happening with the model if it is changed to a *localist representation*? Probabilistically this means that the latent variable is modeled with a (k-dimensional) Multinomial distribution (Eq. (2.113)), resulting in a *Mixture of Gaussians (MoG)* or *Gaussian Mixture Model (GMM)* [35]:

$$\begin{aligned} p(\mathbf{h} \mid \theta) &= \text{Multi}(\mathbf{x} \mid n, \pi_1, \pi_2, \ldots, \pi_k) \\ p(\mathbf{x} \mid \mathbf{h} = i, \theta) &= \mathcal{N}(\mathbf{x} \mid \mu_i, \Sigma_i) \end{aligned} \tag{2.261}$$

So $\theta = (\pi, \mu_1, \Sigma_1, \ldots, \mu_k, \Sigma_k)$. The marginal distribution of $\mathbf{x}$ is a *multimodal* distribution formed by a weighted *sum* of Gaussians:

$$p(\mathbf{x} \mid \theta) = \sum_{m=1}^{k} p(h = m \mid \pi) \mathcal{N}(\mathbf{x} \mid \mu_m, \Sigma_m) \tag{2.262}$$

Maximum Likelihood Estimation can't be done in closed form because the parameter sets for the k different Gaussians interact through the posterior distribution (i.e. explaining away). The gradients of the loglikelihood function have the general form of Eq. (2.244):

$$\begin{aligned}
\frac{\partial \log p(\mathbf{x} \mid \theta)}{\partial \boldsymbol{\mu}_i} &= r_i \boldsymbol{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) \\
\frac{\partial \log p(\mathbf{x} \mid \theta)}{\partial \boldsymbol{\Sigma}_i} &= r_i \left(\boldsymbol{\Sigma}_i - (\mathbf{x} - \boldsymbol{\mu}_i)(\mathbf{x} - \boldsymbol{\mu}_i)^\top \right) \\
\frac{\partial \log p(\mathbf{x} \mid \theta)}{\partial \pi_i} &= \frac{r_i}{\pi_i}
\end{aligned} \tag{2.263}$$

with the *responsibilities* r_i denoting the posterior over the latent variable $\mathbf{h}$:

$$r_i \equiv p(z = i \mid \mathbf{x}) = \frac{\pi_i \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_k \pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)} \tag{2.264}$$

Note that for the covariance matrices $\boldsymbol{\Sigma}_i$ care must be taken that these remain positive definite. This is usually done through some form of projected gradient methods [305].

When the posterior has a simple (i.e. tractable) form as in Eq. (2.264), then a special kind of gradient ascent method is usually much easier to apply, *Expectation Maximization (EM)* [254]. EM is a general algorithm for latent variable problems and corresponds to a coordinate wise gradient ascent method. Instead of optimizing the intractable loglikelihood function (and/or relying on its intractable gradient with respect to θ) EM optimizes a lower bound of the loglikelihood which can be derived from Jensens inequality (Eq. (2.94)):

$$\begin{aligned}
\log p(\mathbf{x} \mid \theta) &= \log \sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{h} \mid \theta) = \log \sum_{\mathbf{h}} \frac{q(\mathbf{h})}{q(\mathbf{h})} p(\mathbf{x}, \mathbf{h} \mid \theta) \\
&\geqslant \sum_{\mathbf{h}} q(\mathbf{h}) \log \frac{p(\mathbf{x}, \mathbf{h} \mid \theta)}{q(\mathbf{h})} := \mathcal{F}(q, \theta)
\end{aligned} \tag{2.265}$$

Hereby $q(\mathbf{h})$ is an arbitrary distribution over the latent variables with $q(\mathbf{h}) > 0$ for all $\mathbf{h}$ where $p(\mathbf{x}, \mathbf{h} \mid \theta) > 0$. The free energy $\mathcal{F}(q, \theta)$ can be expressed differently:

$$\begin{aligned}
\mathcal{F}(q, \theta) &= \sum_{\mathbf{h}} q(\mathbf{h}) \log \frac{p(\mathbf{x}, \mathbf{h} \mid \theta)}{q(\mathbf{h})} \\
&= \sum_{\mathbf{h}} q(\mathbf{h}) \log \frac{p(\mathbf{h} \mid \mathbf{x}, \theta) p(\mathbf{x} \mid \theta)}{q(z)} \\
&= \sum_{\mathbf{h}} q(\mathbf{h}) \log p(\mathbf{x} \mid \theta) + \sum_{\mathbf{h}} q(\mathbf{h}) \log \frac{p(\mathbf{h} \mid \mathbf{x}, \theta)}{q(\mathbf{h})} \\
&= \log p(\mathbf{x} \mid \theta) - \mathcal{KL}[q(\mathbf{h}) \| p(\mathbf{h} \mid \mathbf{x}, \theta)]
\end{aligned} \tag{2.266}$$

The gap between the loglikelihood and the lower bound $\mathcal{F}(q, \theta)$ at some θ corresponds to the Kullback-Leibler divergence (which is always non-negative) between the arbitrarily chosen distribution $q(\mathbf{h})$

and the true posterior $p(\mathbf{h} \mid \mathbf{x}, \theta)$. That is if $q(\mathbf{h})$ is set to the posterior $p(\mathbf{h} \mid \mathbf{x}, \theta)$ for some fixed θ then the lower bound is tight at this θ.

This insight forms the basis for one derivation of the EM algorithm [273]: In the E step, $q(\mathbf{h})$ is set to the posterior of $\mathbf{h}$ at the current parameter θ. In the M step the loglikelihood is (locally) maximized with respect to θ through maximizing the lower bound, relying on the following decomposition of $\mathcal{F}(q, \theta)$:

$$\begin{aligned}\mathcal{F}(q, \theta) &= \sum_{\mathbf{h}} q(\mathbf{h}) \log \frac{p(\mathbf{x}, \mathbf{h} \mid \theta)}{q(\mathbf{h})} \\ &= \sum_{\mathbf{h}} q(\mathbf{h}) \log p(\mathbf{x}, \mathbf{h} \mid \theta) - \sum_{\mathbf{h}} q(\mathbf{h}) \log q(\mathbf{h}) \\ &= \mathbb{E}_{q(\mathbf{h})}\left[p(\mathbf{x}, \mathbf{h} \mid \theta)\right] + \mathcal{H}[q]\end{aligned} \tag{2.267}$$

where the entropy of $q(\cdot)$ is independent of θ. The complete loglikelihood $p(\mathbf{x}, \mathbf{h} \mid \theta)$ is often much easier to optimize with respect to θ which is also true for $\mathbb{E}_{q(\mathbf{h})}\left[p(\mathbf{x}, \mathbf{h} \mid \theta)\right]$. E steps and M steps applied iteratively then ensure that overall the loglikelihood gets maximized locally with respect to θ.

$q(\mathbf{h})$ is usually depending on the input $\mathbf{x}$ so $q(\mathbf{h}) \equiv q(\mathbf{h} \mid \mathbf{x})$. Furthermore, $q(\cdot)$ may also depend on some additional parameters ϕ, either local ones (i.e. the parameters are specific to $\mathbf{x}$) or global ones. The second variant will be discussed in later paragraphs in more detail.

The application of EM is favorable if the posterior $p(\mathbf{h} \mid \mathbf{x}, \theta)$ is somewhat manageable. For example this is the case with Mixtures of Gaussians (Eq. (2.264)). The M step then resembles estimating k many independent Gaussians (see Eq. (2.225) and Eq. (2.227)) where the training samples are *weighted by their respective responsibilities*.

The EM algorithm also provides a simple approach to learning a generalized form of pPCA called Factor Analysis (FA) [359, 35]:

$$\begin{aligned}p(\mathbf{h}) &= \mathcal{N}(\mathbf{h} \mid \mathbf{0}, \mathbf{I}) \\ p(\mathbf{x} \mid \mathbf{h}, \theta) &= \mathcal{N}(\mathbf{x} \mid \boldsymbol{W}\mathbf{h} + \boldsymbol{\mu}, \boldsymbol{\Psi})\end{aligned} \tag{2.268}$$

$\boldsymbol{\Psi}$ is hereby a diagonal matrix with arbitrary positive entries. The marginal distribution $p(\mathbf{x} \mid \theta)$ is again a low-rank Gaussian:

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{W}\boldsymbol{W}^{\mathsf{T}} + \boldsymbol{\Psi}) \tag{2.269}$$

so

$$\boldsymbol{\mu}_{\text{MLE}} = \frac{1}{n} \sum_{i} \mathbf{x}_i \tag{2.270}$$

Note that if $\boldsymbol{\Psi}$ is a *full* covariance matrix then $p(\mathbf{x} \mid \theta)$ represents a *full-rank* Gaussian and the linear latent model would not be useful.

Alternating the following two steps allows learning $\boldsymbol{W}$ and $\boldsymbol{\Psi}$:

- In the E-step the posterior Gaussian $p(\mathbf{h} \mid \mathbf{x}, \theta)$ (see Eq. (2.143)) is determined for the current θ.

$$p(\mathbf{h} \mid \mathbf{x}, \theta) = \mathcal{N}(\mathbf{h} \mid \hat{\mu}, \hat{\Sigma}) \tag{2.271}$$

  with

$$\begin{aligned} \hat{\Sigma} &= (\mathbf{I} + \mathbf{W}^\top \Psi^{-1} \mathbf{W})^{-1} \\ \hat{\mu} &= \hat{\Sigma} \left(\mathbf{W}^\top \Psi^{-1} (\mathbf{x} - \mu_{\text{MLE}}) \right) \end{aligned} \tag{2.272}$$

- In the M step the latent variables are assumed to be known so estimating $\mathbf{W}$ and Ψ results in a standard linear regression problem, where the *posterior means* are the inputs and the observed training samples are the targets. Different to pPCA the learned weights $\mathbf{W}$ are usually not orthogonal.

In this form EM can also be applied to PCA [317]—this is particularly interesting when the covariance matrix does not fit into memory to compute the SVD or if the data becomes available in an online fashion.

The EM algorithm allows to tackle very complicated models in a structured manner. For example, in order to model mixtures of Gaussians that have a low-rank structure (for every cluster the low-rank structure is different) one can simply combine the EM equations for Mixture of Gaussians and Factor Analysis resulting in Mixture of Factor Analyzers (MFA) [111]. However, different from these two simple models (and combinations thereof), $p(\mathbf{h} \mid \mathbf{x}, \theta)$ is in most cases not tractable, so it is not possible to set $q(\mathbf{h})$ to $p(\mathbf{h} \mid \mathbf{x}, \theta)$.

The EM algorithm also works if $q(\mathbf{h})$ only resembles an approximation of $p(\mathbf{h} \mid \mathbf{x}, \theta)$, e.g. through a *variational* distribution. Similarly it is not necessary to *actually maximize* the expected complete loglikelihood with respect to θ, it is enough to *only improve it a little* (e.g. again by an (the hopefully tractable) gradient step) [273]. These last two remarks are often subsumed under *generalized EM* and show that EM can be interpreted as a special case of variational learning [190]: it treats the variational parameters and the model parameters in an alternating fashion instead of optimizing both sets together.

A more probabilistic interpretation of the central decomposition in Eq. (2.267) finally explains the origin of the name *EM*: For the algorithm to work what is necessary is first an expression for the *expected* complete loglikelihood, which then is locally *maximized*. Of course this expectation can also be estimated using the Monte Carlo principle. In this case the samples from the posterior are generated through a MCMC schema, giving rise to MCMC-EM [252, 253]. More details on the derivation of the EM algorithm and its application to Mixture of Gaussians and to Factor Analysis are given in Appendix A.

If the prior distribution in Factor Analysis is changed to a Laplacian distribution the resulting model becomes a representative of *sparse coding* [282]:

$$\begin{aligned} p(\mathbf{h}) &\propto \exp(-\lambda\|\mathbf{h}\|_2) \\ p(\mathbf{x} \mid \mathbf{h}, \theta) &= \mathcal{N}(\mathbf{x} \mid \boldsymbol{W}\mathbf{h}, \mathbf{I}) \end{aligned} \tag{2.273}$$

where the dataset $\mathcal{D}$ has mean $\mathbf{0}$ (so $\boldsymbol{\mu}$ can be omitted) and the conditional covariance matrix is set to $\mathbf{I}$ for simplicity. For some observed $\mathbf{x}$, the associated *negative* complete loglikelihood $-\log(\mathbf{x}, \mathbf{h})$ is

$$-\log(\mathbf{x}, \mathbf{h}) \propto \|\mathbf{x} - \boldsymbol{W}\mathbf{h}\|^2 + \lambda\|\mathbf{h}\|_1 \tag{2.274}$$

If $\boldsymbol{W}$ is known then Eq. (2.274) is sometimes denoted the *LASSO* (Least Absolute Shrinkage and Selection Operator) [386]. In this case it is interpreted as a linear regression problem between *inputs* $\boldsymbol{W}$ (stacked row-wise) and one-dimensional *targets* $\mathbf{x}$ where the *weights* $\mathbf{h}$ are supposed to be sparse, realizing *feature selection*.

Determining the optimal parameters $\theta = \boldsymbol{W}$ can again be done with the EM algorithm. However, the E-step is more difficult than in the case of Factor Analysis due to explaining away effects. It can be formulated as a convex optimization problem (for fixed parameters θ), but there is no closed-form solution:

$$\min_{\mathbf{h}} \|\mathbf{h}\|_1 \quad \text{subject to } \boldsymbol{W}\mathbf{h} = \mathbf{x} \tag{2.275}$$

As it turns out this is also a convex relaxation formulation for the NP-complete problem of *sparse coding* [77, 270, 83, 96]:

$$\min_{\mathbf{h}} \|\mathbf{h}\|_0 \quad \text{subject to } \boldsymbol{W}\mathbf{h} = \mathbf{x} \tag{2.276}$$

The solutions to Eq. (2.275) and Eq. (2.276) are identical if the optimum of Eq. (2.276) is *sparse enough* (see [394] for a much more rigorous mathematical treatment of this statement). For that reason, a lot of effort has been put into finding efficient iterative solutions for Eq. (2.275). A standard approach for this problem is the iterative shrinkage-thresholding algorithm (ISTA) [75, 20]. ISTA belongs to the class of proximal gradient methods [286] for solving non-differentiable convex functions. It finds the minimum of $f + g$ with $f : \mathbf{R}^n \to \mathbf{R}$ a convex and Lipschitz-continuous [147] function and $g : \mathbf{R}^n \to \mathbf{R}$ a convex (but non-differentiable) function using the following iterative algorithm:

$$\mathbf{x}_{t+1} = \arg\min_{\mathbf{x} \in \mathbf{R}^n} \left(g(\mathbf{x}) + \frac{L}{2} \left\| \mathbf{x} - \left(\mathbf{x}_t - \frac{1}{L} \nabla f(\mathbf{x}_t) \right) \right\|_2^2 \right) \tag{2.277}$$

where L is the Lipschitz constant of f. So one iteration is itself an optimization problem. In the case of $g \equiv \|\mathbf{h}\|_1$ and $f \equiv \|\mathbf{x} - \boldsymbol{W}\mathbf{h}\|_2^2$ a closed-form solution for every iteration step exists [130]:

$$\mathbf{h}_{t+1} = \tau_\alpha \left(\mathbf{h} + \frac{1}{L} \boldsymbol{W}^T (\mathbf{x} - \boldsymbol{W}\mathbf{h}) \right). \tag{2.278}$$

$\tau_\alpha(\cdot)$ is a shrinkage operator defined as

$$\tau_\alpha(x) = \operatorname{sgn}(x)(x - \alpha)_+, \quad \alpha = \frac{\lambda}{L}. \tag{2.279}$$

Hereby $\operatorname{sgn}(\cdot)$ is the elementwise sign function and $(\cdot)_+$ sets all negative entries of a vector to 0.

Given a latent representation h for some x the M-step for determining W is, as with Factor Analysis, a linear regression problem over the complete training set $\mathcal{D}$ enhanced with the latent data. It is a good practice to ensure that the columns of W have unit length which is generally realized by re-normalization after the linear regression problem is solved. Otherwise, the latent representation can become very small (i.e. the absolute values of the vector elements) and the entries of W very large.

The marginal distribution of $p(x \mid \theta)$ is complex and in particular no (low-rank) Gaussian. Hence, differently to Factor Analysis, the representation of h can be chosen to be overcomplete, i.e. the dimensionality of h is much larger than the dimensionality of x. This is quite reasonable, considering that h is supposed to be sparse.

If both the visible and latent variables are binary, the underlying model becomes exponentially more difficult to solve. For example, consider a single layer *sigmoid belief network* (SBN) [288, 274], the binary equivalent of Factor Analysis (Eq. (2.268)):

$$\begin{aligned} p(h \mid \theta) &= \operatorname{Bern}(h \mid \mu) = \prod_i \mu_i^{h_i}(1-\mu_i)^{1-h_i} \\ p(x \mid h, \theta) &= \operatorname{Bern}(x \mid \sigma(Wh + b)) \end{aligned} \tag{2.280}$$

with $h \in \{0,1\}^l$, $x \in \{0,1\}^n$, $\theta = (\mu \in \mathbf{R}^l, W \in \mathbf{R}^{l \times n}, b \in \mathbf{R}^n)$ and $\sigma(\cdot)$ being the logistic sigmoid (see Eq. (2.219)) applied elementwise. Using Eq. (2.219) both $p(h \mid \theta)$ and $p(x \mid h, \theta)$ can be written in an alternative way:

$$\begin{aligned} p(h \mid \theta) &= \prod_i \frac{\exp(\eta_i h_i)}{1 + \exp(\eta_i)} = \frac{\exp\left(\eta^T h\right)}{\prod_i (1 + \exp(\eta_i))} \\ p(x \mid h, \theta) &= \prod_i \frac{\exp\left(\left(\sum_j W_{ij} h_j + b_i\right) x_i\right)}{1 + \exp\left(\sum_j W_{ij} h_j + b_i\right)} \\ &= \frac{\exp\left(x^T W h + b^T x\right)}{\prod_i \left(1 + \exp\left(\sum_j W_{ij} h_j + b_i\right)\right)} \end{aligned} \tag{2.281}$$

with $\eta = \frac{\mu}{1-\mu}$. Hence

$$p(x, h \mid \theta) = \frac{\exp\left(x^T W h + b^T x + \eta^T h\right)}{\prod_i \left(1 + \exp\left(\sum_j W_{ij} h_j + b_i\right)\right) \prod_i \left(1 + \exp(\eta_i)\right)} \tag{2.282}$$

The complete loglikelihood is therefore

$$\begin{aligned}\log p(x, h \mid \theta) = x^T W h + b^T x + \eta^T h \\ - \sum_i \log \left(1 + \exp\left(\sum_j W_{ij} h_j + b_i\right)\right) \\ - \sum_i \log\left(1 + \exp\left(\eta_i\right)\right)\end{aligned} \tag{2.283}$$

The energy function of an RBM (Eq. (2.236)) is very similar, but the complete (log)likelihood is easy to calculate here due to the tractable normalization constant. Computing the gradient of the complete log-likelihood with respect to the parameters is better done elementwise in this case, e.g.

$$\begin{aligned}\frac{\partial \log p(x, h \mid \theta)}{\partial W_{ij}} &= x_i h_j - \frac{\exp\left(\sum_j W_{ij} h_j + b_i\right)}{\left(1 + \exp\left(\sum_j W_{ij} h_j + b_i\right)\right)} h_j \\ &= \left(x_i - p(x_i = 1 \mid h, \theta)\right) h_j\end{aligned} \tag{2.284}$$

So learning only happens if the prediction (the conditional distribution of x_i) is significantly different from the observed value x_i. Similarly

$$\frac{\partial \log p(x, h \mid \theta)}{\partial b_i} = x_i - p(x_i = 1 \mid h, \theta) \tag{2.285}$$

and

$$\frac{\partial \log p(x, h \mid \theta)}{\partial \eta_i} = h_i - p(h_i = 1 \mid \theta) \tag{2.286}$$

However, Eq. (2.244) weights these gradient terms for every configuration with the corresponding posterior $p(h \mid x, \theta)$. This posterior, differently from the undirected RBM, is expensive to compute. With

$$p(h \mid x, \theta) = \frac{p(x, h \mid \theta)}{\sum_h p(x, h \mid \theta)} \tag{2.287}$$

an exponential sum needs to be computed. And this sum can't be efficiently simplified, because h appears both in the nominator and the denominator of Eq. (2.282). The only generic approach left is Gibbs sampling. Its mixing time for SBN can be improved when a state is *persisted* for every Markov chain, i.e. every sample has its own persisted chain [274].

An SBN with multiple layers is a simple extension of Eq. (2.280) by introducing intermediate layers h^k (note that layers closer to the

observed *generative output* of the model are numbered higher, i.e. the SBN starts at the layer denoted $\mathbf{h}^0$):

$$\begin{aligned} p(\mathbf{h}^0 \mid \theta) &= \text{Bern}(\mathbf{h}^0 \mid \mu) = \prod_i \mu_i^{\mathbf{h}^0{}_i}(1-\mu_i)^{1-\mathbf{h}^0{}_i} \\ p(\mathbf{h}^k \mid \mathbf{h}^{k-1}, \theta) &= \text{Bern}(\mathbf{h}^k \mid \sigma(\mathbf{W}^k\mathbf{h}^{k-1} + \mathbf{b}^k)), \quad k > 0 \\ p(\mathbf{x} \mid \mathbf{h}^K, \theta) &= \text{Bern}(\mathbf{x} \mid \sigma(\mathbf{W}\mathbf{h}^K + \mathbf{b})) \end{aligned} \tag{2.288}$$

Rewriting Eq. (2.288) according to Eq. (2.281) allows the expression of the joint probability $p(\mathbf{x}, \mathbf{h}^K, \ldots, \mathbf{h}^0)$ according to Eq. (2.282) because

$$p(\mathbf{x}, \mathbf{h}^K, \ldots, \mathbf{h}^0 \mid \theta) = p(\mathbf{x} \mid \mathbf{h}^K, \theta)p(\mathbf{h}^0 \mid \theta) \prod_{i=0}^{K-1} p(\mathbf{h}^{i+1} \mid \mathbf{h}^i, \theta). \tag{2.289}$$

Therefore computing gradients of the complete loglikelihood for arbitrary weight elements remains a purely local operation: For some weight element $\mathbf{W}^k{}_{ij}$ at layer k ($0 < k \leqslant K$) the gradient is

$$\begin{aligned} \frac{\partial \log p(\mathbf{x}, \mathbf{h}^K, \cdots, \mathbf{h}^0 \mid \theta)}{\partial \mathbf{W}^k{}_{ij}} &= \mathbf{h}^k{}_i \hat{\mathbf{h}}_j - \frac{\exp\left(\sum_j \mathbf{W}^k{}_{ij}\hat{\mathbf{h}}_j + \mathbf{b}^k{}_i\right)}{\left(1 + \exp\left(\sum_j \mathbf{W}^k{}_{ij}\hat{\mathbf{h}}_j + \mathbf{b}^k{}_i\right)\right)}\hat{\mathbf{h}}_j \\ &= (\mathbf{h}^k{}_i - p(\mathbf{h}^k{}_i = 1 \mid \hat{\mathbf{h}}, \theta))\hat{\mathbf{h}}_j, \quad \hat{\mathbf{h}} \equiv \mathbf{h}^{k-1} \end{aligned} \tag{2.290}$$

Of course, this gradient has to be evaluated under the expectation of the posterior $p(\mathbf{h}^K, \mathbf{h}^{K-1}, \cdots, \mathbf{h}^0 \mid \mathbf{x}, \theta)$ which is even more complicated than in the single-layer case. Note that the posterior is also not significantly simplified for the case of some hidden units (or complete layers, e.g. $\mathbf{h}^0$ for the case of a *stochastic input-output* mapping) being observed because with multiple layers the Markov blanket of a unit spans three layers (its parents, its children, and the children's parents), so efficient sampling remains very difficult.

So is it possible to train a sigmoid belief network with many layers (sometimes called a *Deep Belief Network (DBN)*) more efficiently? As it turns out the (at first sight) naïve idea of simply stacking greedily trained one-layer models is indeed working. Single layer SBN layers can be stacked on top of each other, if special sparsity constraints are integrated into the basic models, as was recently demonstrated successfully [105].

The idea of stacking directed graphical models was originally considered disadvantageous: A single layer would try to model the latent variables as independent which in general is not possible (because only a single layer of parameters is available). So the resulting *aggregated posterior* $p(\mathbf{h}^0) = \sum_x p(\mathbf{h}^0 \mid \mathbf{x})p(\mathbf{x})$ was considered to be as difficult to model for the next layer as the original data distribution $p(\mathbf{x})$. Instead, stacking *undirected* RBMs on top of each other also leads to a (generative) sigmoid belief network with multiple layers [155]. Using an argument based on the free energy of the model it is possible

to show that with every layer the loglikelihood of the dataset under the deep model can be indeed improved. The top layer of this DBN is still an undirected Restricted Boltzmann Machine, though. The advantage of this approach is that an efficient inference procedure for high level latent variables is provided for free: simply evaluate the stack of RBMs sequentially. Its disadvantage is that, in order to generate new samples, the model first needs to run a Markov Chain of the top-level RBM.

A very different way of learning generative directed graphical models is based on the idea of an inverse recognition model [154, 371]: The complicated inference process for the latent posterior $p(\mathbf{h} \mid \mathbf{x})$ (e.g. represented by an iterative MCMC schema) can be itself described by a machine learning model. Why would that approach be actually reasonable? As shown with the example of a single layer SBN (Eq. (2.283)), the *complete* loglikelihood for directed generative models is easy to evaluate and likewise its associated gradients. So in this case, an efficient method to compute e.g. samples from the posterior solves already the tractability problem (see again Eq. (2.244)). Note that this is basically also what Contrastive Divergence implements, only that CD efficiently produces samples from the *joint distribution* which is necessary in the case of undirected graphical models[9]. A recognition model was also recently used to improve the training of Deep Boltzmann Machines [324].

In order for this idea to work, a very powerful recognition model is necessary and a suitable objective to train this class of models. Neural Networks are considered to be the most flexible Machine Learning methods—in this setting they seem to be the perfect model of choice. A possible consistent training objective (which is missing in the original approach utilizing a recognition mode, the wake-sleep algorithm [154]) is a simple reformulation of the already introduced free energy from the EM algorithm (see Eq. (2.267)) [307, 200]:

$$\begin{aligned}\mathcal{F}(\mathbf{q}, \theta) &= \sum_{\mathbf{h}} q(\mathbf{h}) \log \frac{p(\mathbf{x}, \mathbf{h} \mid \theta)}{q(\mathbf{h})} \\ &= \sum_{\mathbf{h}} q(\mathbf{h}) \log p(\mathbf{x} \mid \mathbf{h}, \theta) + \mathcal{KL}[q(\mathbf{h}) \| p(\mathbf{h} \mid \theta)] \\ &= \mathbb{E}_{q(\mathbf{h})}[\log p(\mathbf{x} \mid \mathbf{h}, \theta)] + \mathcal{KL}[q(\mathbf{h}) \| p(\mathbf{h} \mid \theta)]\end{aligned} \tag{2.291}$$

With this motivation at hand the next section finally introduces Neural Networks. Neural Networks can be considered as special kinds of directed Graphical Models: The probability distributions for the various (hidden) units are delta functions, therefore all computations are entirely deterministic. So it might be not completely surprising that in this case an algorithm exists, that can efficiently train (supervised) models with an arbitrary large number of hidden units. The

9 And Contrastive Divergence works so well because the underlying model is *globally shared* between the samples from the training set.

backpropagation algorithm will be derived in a general way in the next section. Furthermore, the most important developments to successfully train neural networks with multiple layers (*deep* networks) are briefly presented which also encompass approaches that use, among others, Eq. (2.291) to train deep networks in an unsupervised way.

2.7 NEURAL NETWORKS

Looking into physics, biology and engineering, (computational) systems are usually defined by a modular structure. They are built out of smaller systems and often use only a few adjustable basic entities that compose the overall system. So out of the principle of modularity, systems evolve that can solve complex tasks. Algebraically, modularity is represented by *function composition*. In Machine Learning the approach that is most closely connected to function composition are Neural Networks [34, 333].

Functionally, a Neural Network with input $\mathbf{x} \in \mathbf{R}^m$, output $\mathbf{y} \in \mathbf{R}^o$ and parameters (usually called *weights*) $\theta \in \mathbf{R}^n$ is a mapping $N : \mathbf{R}^m \times \mathbf{R}^n \to \mathbf{R}^o$, followed by an *output non-linearity* $M : \mathbf{R}^o \to \mathbf{R}^o$ (which is the identity function in the case of linear outputs) [336]. In order to determine settings for the weights, a loss function $L : \mathbf{R}^o \times \mathbf{R}^o \to \mathbf{R}$ is necessary. In certain cases, M must have a special functional form such that the overall output of the network aligns with the loss L (*matching loss function* [336])[10], e.g. see Eq. (2.160) for the case of multi-valued classification. If the Neural Network is associated with a probabilistic modeling task, the loss function can be the *negative* conditional log-likelihood function, e.g. from regression (Eq. (2.153)) or classification (Eq. (2.159)). Because Neural Networks are quite flexible, additional regularization is usually necessary to avoid overfitting. Hence, the empirical risk minimization framework (Eq. (2.170)) is well suited to describe the *stochastic* optimization problem induced by a training set $\mathcal{D}$ which is equivalent to MAP in the case of a probabilistic form of $L(\cdot,\cdot)$. The expression $\ell(\theta, \mathbf{x}, \mathbf{y})$ then denotes the empirical risk incurred by the Neural Network $M \circ N$ for the parameter setting θ and a sample $(\mathbf{x}, \mathbf{y})$ from the training set, i.e.

$$\ell(\theta, \mathbf{x}, \mathbf{y}) = L(M(N(\theta, \mathbf{x})), \mathbf{y}) + \lambda \boldsymbol{\Omega}(\theta) \tag{2.292}$$

where $\boldsymbol{\Omega}(\theta)$ is a regularization cost for the parameters.

The Neural Network $M \circ N$ itself is considered to be a highly non-linear function constructed out of elementary mathematical expressions. A versatile mathematical building block is the affine transformation

$$\boldsymbol{W}\mathbf{x} + \mathbf{b} \tag{2.293}$$

10 In [34] this is called *a natural pairing of error functions and activation functions*.

with the *weight matrix* $\boldsymbol{W} \in \mathbf{R}^{m \times n}$ and the *bias* $\mathbf{b} \in \mathbf{R}^m$, for some $\boldsymbol{x} \in R^n$.

Because affine transformations are closed under composition, several sequentially combined affine transformations can always be represented by one affine transformation. If however affine transformations are interleaved with non-linearities, a highly non-linear function can be constructed. Combining these two entities results in the most widely used form of a *layer* in a Neural Network:

$$f(\boldsymbol{W}\boldsymbol{x} + \mathbf{b}) \tag{2.294}$$

with the non-linear *activation function* $f : \mathbf{R}^m \to \mathbf{R}^m$ and $\boldsymbol{x}, \boldsymbol{W}, \mathbf{b}$ defined as in Eq. (2.293).

A Neural Network $M \circ N$ is then implemented by the composition of at least two such layers[11] with N having the following general form:

$$\begin{aligned} N(\theta, \boldsymbol{x}) = \mathbf{b}_K + \boldsymbol{W}_K(f_{K-1}(\mathbf{b}_{K-1} + \\ \boldsymbol{W}_{K-1}(\cdots(f_1(\mathbf{b}_1 + \boldsymbol{W}_1\boldsymbol{x}))\cdots))) \end{aligned} \tag{2.295}$$

Hereby, $\theta = (\mathbf{b}_1, \boldsymbol{W}_1, \ldots, \mathbf{b}_{K-1}, \boldsymbol{W}_{K-1}, \mathbf{b}_K, \boldsymbol{W}_K)$. Note that the last non-linearity (at layer K) is the identity (because any non-linearity at this layer is covered by M). All other non-linearities are supposed to be truly non-linear. The *hidden layer* $\mathbf{h}_k$ at depth k is defined recursively as

$$\mathbf{h}_k(\theta, \mathbf{h}_{k-1}) = f_k(\boldsymbol{W}_k\mathbf{h}_{k-1} + \mathbf{b}_k), \qquad k \geqslant 2 \tag{2.296}$$

with

$$\mathbf{h}_1(\theta, \boldsymbol{x}) = f_1(\boldsymbol{W}_1\boldsymbol{x} + \mathbf{b}_1) \tag{2.297}$$

$M \circ N$ is said to have $K - 1$ hidden layers or to have a *depth* of $K - 1$. Neural Networks with $K = 2$ (i.e. one hidden layer) are often denoted *shallow networks* [27]. Conversely, networks with $K > 2$ are denoted *deep networks*. Later, techniques are introduced that allow learning deep Neural Networks with hundreds of hidden layers [367].

An element of a hidden layer is denoted a *hidden unit*. Using hidden layers, Eq. (2.295) can be written as

$$N \equiv \mathbf{h}_K \circ \mathbf{h}_{K-1} \circ \cdots \circ \mathbf{h}_1 \tag{2.298}$$

$M(N(\theta, \boldsymbol{x}))$ then is the prediction (evaluation) of the network for some $\boldsymbol{x}$ at θ. This standard *feed-forward* Neural Network is the most simple *architecture* that can be built out of the two basic entities (affine transformations and non-linearities), where the output of a hidden layer k becomes the input of layer $k + 1$. More complex architectures

11 A Neural Network with one layer is still a linear model with respect to the parameters θ and resembles a Generalized Linear Model (GLM).

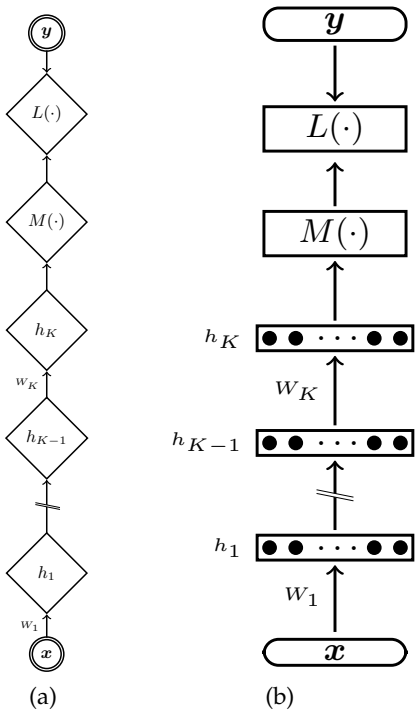


Figure 2.5: Graphical representations of a feed-forward Neural Network. (a) A network with K layers shown in the form of a parameterized Graphical Model. Only input and target nodes are random variables, all other elements represent deterministic computations. (b) The same model shown as a stack of hidden layers represented as vectors. Hereby, rounded rectangles denote random variables and cornered rectangles deterministic variables. Bias units are left out for clarity. Best viewed electronically.

are possible, e.g. some layer k might not only get its input from the directly preceding layer but also from layers $k-2$, $k-3$, ... through so called *skip connections* [34].

If the input to a Neural Network is a sequence of vectors (and the associated target data is also a sequence) then a Recurrent Neural Network (RNN) [313, 415, 419] must be used: every element of the sequence is processed by a network of the above form (and all these networks share their parameters). Additionally these elementwise networks are connected by *recurrent* weights (which are also shared to ensure generalization over sequences of arbitrary length). Mathematically, a Recurrent Neural Network can be described with the following abstract formula:

$$\mathbf{h_t} = F(\mathbf{x_t}, \mathbf{h_{t-1}}, \theta) \tag{2.299}$$

where $\mathbf{x_t}$ is the input at time step t, $\mathbf{h_t}$ is the hidden state of the network at time step t and $F(\cdot)$ is the transfer function for the hidden state, depending on the parameters θ. In fact, the most general Neu-

ral Network architecture is an RNN with $M \circ N$ being the special case of a sequence with length 1. However, Eq. (2.295) is enough to derive the central method that allows efficient learning of the parameters of a Neural Network which then can be applied without modification to the more complex RNNs. Because the models in subsequent chapters are all static feed-forward models, RNNs will be only discussed briefly in this text, see subsection 2.7.1.

Clearly, one can imagine other basic building blocks than the affine transformation from Eq. (2.294) and it is by no means the only possible building block. The important aspect of Eq. (2.295) is the *nesting* of the basic building blocks. However, most alternative architectures can be usually expressed as a stack of matrix-vector operations with interleaved elementwise non-linear activation functions. For example, a *product unit* [87]

$$\mathbf{h}_i = f\left(\prod_j \mathbf{x}_j^{\mathbf{W}_{ij}}\right) \tag{2.300}$$

can be realized as a matrix-vector product in the log-domain,

$$\mathbf{h}_i = f\left(\exp\left(\sum_j \mathbf{W}_{ij} \log \mathbf{x}_j\right)\right), \tag{2.301}$$

with $f(\cdot)$, $\exp(\cdot)$ and $\log(\cdot)$ elementwise and *complex-valued* activation functions [398]. More complex transformations involving *additive and multiplicative* interactions can be realized as affine transformations of *tensor*-vector expression. Hence, I utilize non-linear affine transformations as the standard building block for networks in this section.

The type of activation function hasn't been discussed yet. Traditionally, *sigmoid-shaped* functions have been used, ensuring that computations stay bounded and with the benefit of simple, numerically efficient derivatives. The logistic sigmoid function $\sigma(x)$,

$$\begin{aligned} \sigma(x) &= \frac{1}{1+\exp(-x)} = \frac{\exp(x)}{1+\exp(x)}, \\ \sigma'(x) &= \sigma(x)(1-\sigma(x)), \end{aligned} \tag{2.302}$$

and the Tangens Hyperbolicus function ($\tanh(x)$),

$$\begin{aligned} \tanh(x) &= \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)} = 2\sigma(2x)-1, \\ \tanh'(x) &= 1-\tanh^2(x), \end{aligned} \tag{2.303}$$

are the most prominent members of this class of functions. The logistic sigmoid function is not symmetric around the origin and therefore has some theoretical disadvantage which is empirically measurable (in both training speed and generalization performance). Hence,

the $\tanh(\cdot)$ activation function is preferred, specifically of the form $1.7159 \tanh\left(\frac{2}{3}x\right)$ [224].

Recently, *piecewise linear* functions have been used in many successful applications of deep Neural Networks. These functions are fast to evaluate, have an equally fast (and simple) derivative and have the theoretical benefit that deep networks can be mathematically analyzed to some degree [265, 326]. The *rectifying linear unit (ReLU)* [245, 268, 116, 209],

$$\text{ReLU}(x) = \max(0, x) \equiv [x]_+ \tag{2.304}$$

is the simplest version in this class. More involved variants have some reported benefits, like the leaky ReLU (LReLU) [241],

$$\text{LReLU}(x) = \max(0, x) + c \min(0, x), \; c > 0, \; c \ll 1, \tag{2.305}$$

where c is a very small, positive constant set by hand. The *parametric* PReLU [146] has the same functional form but in this case c is a learnable parameter (either shared within a layer or learned separately for every unit). Finally, this idea can be generalized to a sum of hinge-shaped functions, the adaptive piecewise linear (APL) unit [3]. The APL is defined as

$$\text{APL}(x) = \max(0, x) + \sum_i a_i \min(0, x - b_i) \tag{2.306}$$

Again, the learnable parameters a_i, b_i, $1 \leqslant i \leqslant n$ can be shared between units of a layer or learned separately for every unit.

Based on these approaches, activation functions acting on *disjunctive groups* of units in a hidden layer in a competitive way have been introduced. The *Max-Out* [120] activation function is defined over such a group $H = \{h_1, \ldots, h_m\}$ of hidden units

$$\text{Max-Out}(H) = \max_{h_i \in H}(h_i) \tag{2.307}$$

The hidden units h_i can hereby be the result of a non-linear affine transformation, though the original paper uses linear units. Max-Out also reduces the dimensionality of a layer. And finally, the Local-Winner-Take-All (LWTA) [364] activation function is also defined over disjunctive groups of units. However, it sets the activation of a non-maximum unit in its group to 0 and does not reduce the group to a scalar output, i.e.

$$\text{LWTA}(h_i) = \begin{cases} h_i, & \text{if } h_i \equiv \max_{h_j \in H} \\ 0, & \text{otherwise.} \end{cases} \tag{2.308}$$

It has also been demonstrated to achieve state-of-the-art results on several Computer Vision benchmarks [412].

In fact, a very wide range of reasonable activation functions exist. Interpreting a layer of a Neural Network as a Generalized Linear Model, the link functions utilized for these models can be a valuable source of inspiration. For example, the ReLU activation function is actually the link function used in Tobit regression [391].

An important theoretical question deals with the expressive capabilities of $M \circ N$. What kind of functions can it represent? Does its representational power depend on the number of layers, on the number of hidden units per layer or on the type of non-linear activation functions? Interestingly, a Neural Network of the form in Eq. (2.295) with only one hidden layer and a finite (but apriori indeterminable) number of hidden units can already approximate a large set of continuous functions on compact subsets of $\mathbf{R}^n$, independent of the type of activation function used (as long as some weak assumptions are fulfilled). This is called the *universal approximation theorem* [72, 167] and is of course also true for Neural Networks with more than one layer. However, no constructive proofs exist. An even stronger result can be shown for recurrent Neural Networks which are *Turing complete* [345]. Given such powerful results it comes as no surprise that determining the best parameter setting θ for Neural Networks according to the input/output pairs $(\mathbf{x}, \mathbf{y})$ from a training set $\mathcal{D}$ and some loss function $L(\cdot, \cdot)$ is very difficult, and, in general, NP-complete [192, 36, 347] (this is independent of the depth of a network). So when training a Neural Network, one has to rely on various kinds of heuristics that usually give no guarantees about the quality of the identified solution.

One popular heuristic is gradient descent (Eq. (2.42)). The only guarantee it can give is that some type of local minima in the parameter space can be found [278]. Local minima are usually sub-optimal but in Machine Learning the ultimate goal is generalization and in this case local minima of the training objective need not be harmful. In fact, ending up in a *bad* local minima (i.e. those local minima with inferior generalization capabilities) is highly unlikely for *deep large-scale*[12] Neural Networks. Moreover, for these kinds of networks most *true* local minima seem to behave equally well with respect to the error on a test set [123, 57]. A more important problem with gradient descent heuristics are saddle points with a large extent (*spurious minima*) which actually plague Neural Networks [326, 76, 57].

The naïve way to compute the gradient follows Eq. (2.39) and generally produces (numerically) good enough estimates of the gradient. This finite difference approximation can be improved by the *cen-*

12 These are networks with many hidden layers of non-linearities and many units per hidden layer.

tral difference approximation which produces for an arbitrary function $f : \mathbf{R}^n \to \mathbf{R}$ a gradient estimate with approximation error $O(\varepsilon^2)$:

$$\frac{\partial f}{\partial x_i} \approx \frac{f(\boldsymbol{x} + \varepsilon \boldsymbol{e}_i) - f(\boldsymbol{x} - \varepsilon \boldsymbol{e}_i)}{2\varepsilon} \tag{2.309}$$

However, for every element of the gradient vector, the network must be evaluated twice[13]. So overall $O(n^2)$ operations are necessary to compute the complete gradient in this way which makes this approach unfeasible for Neural Networks. Gradient computation should be at most as expensive as evaluating the network for a given input sample, which is $O(n)$.

The famous *backpropagation* algorithm does exactly this: Computing the gradient of a Neural Network with n parameters in $O(n)$ operations[14]. Technically, backpropagation is a special kind of reverse mode automatic differentiation (AD) [133, 134], however for quite some time AD and Machine Learning remained separate endeavors, unbeknownst to each other [15]. This might explain the many times the algorithm was re-invented but not re-discovered [333].

From a purely mathematically point of view, computing derivatives of some parameter θ_i for a structure like the one presented in Eq. (2.295) is simply done by the chain rule. For efficient gradient calculations it is important to see that a lot of computational effort can be reused which is algorithmically the definition of dynamic programming (or caching computations) [69]. The chain rule and dynamic programming are the central elements of the backpropagation algorithm. Being based on two such simple principles it is not surprising the method itself dates back a very long time. Already with the invention of calculus it was considered that differentiation should be subjected to mechanical automation [15][15]. And around the beginning of the area of electronic computing machines backpropagation was used in the field of stochastic control theory [198, 45, 85, 46] and also to train already Neural network-like architectures with several layers of nonlinearities [173, 172, 233]. For popularizing backpropagation within the field of Machine Learning usually [319] and [218] are credited.

13 In the case of Eq. (2.39), one of these two evaluations computes the loss for the current θ and the given input/output pair(s), so this evaluation only needs to be done once per complete gradient computation.

14 So backpropagation is simply an efficient way to compute gradients. For this reason it is somewhat non-descriptive to say that *a Neural Network is trained by backpropagation*—it only implies that a gradient-based optimization procedure is used for training.

15 Some marginalia from my side: In [15], Leibniz is cited as having done some work on automated differentiation. However, his *Machina arithmetica in qua non additio tantum et subtractio sed et multiplicatio nullo, diviso vero paene nullo animi labore peragantur* does not seem to imply to me that he realized that differentiation of arbitrary expressions can be done by a machine, even though it may be self-evident a posteriori because it is a purely mechanistic task.

A much more detailed historical description of the origins and reinventions of backpropagation and its application for Neural-Network learning is given in [333].

The algorithmic derivation is more important to this work and is explained next. Differently to most texts covering backpropagation I rely solely on the notation and results from Vector and Matrix Calculus (Section 2.2). Hence, the final results are described in a way that can be used without any further transformation for an efficient implementation. Additionally, the application to different architectures is also simplified, because all involved terms are compact matrix expressions and are therefore simpler to map to architectural extensions or changes.

Consider the simple case of N being a stack of non-linear matrix-vector operations (see Eq. (2.295)). $\ell(\theta, \mathbf{x}, \mathbf{y})$ is the risk incurred by the Neural Network $M \circ N$ for the parameter setting θ and a sample $(\mathbf{x}, \mathbf{y})$ from the training set (Eq. (2.292)). In order to compute $\nabla_\theta \ell(\theta, \mathbf{x}, \mathbf{y})$ it turns out that the Jacobians of the risk with respect to hidden layers $\mathbf{h}_k$ are helpful. Following the chain rule for vector-valued functions (Eq. (2.50)) the Jacobian of the risk with respect to some input $\mathbf{x}$ is

$$\mathbf{J}_{L \circ M \circ N} = \mathbf{J}_L \mathbf{J}_M \mathbf{J}_N, \quad \theta \text{ unknown but fixed} \tag{2.310}$$

which is a $1 \times n$ row vector, for $\mathbf{x} \in \mathbf{R}^n$. More specifically, for $\mathbf{h}_k$ one gets

$$\tilde{\mathbf{J}}_{\mathbf{h}_k} \equiv \mathbf{J}_{L \circ M \circ \mathbf{h}_K \circ \mathbf{h}_{K-1} \circ \cdots \circ \mathbf{h}_k} = \mathbf{J}_L \mathbf{J}_M \mathbf{J}_{\mathbf{h}_K} \mathbf{J}_{\mathbf{h}_{K-1}} \cdots \mathbf{J}_{\mathbf{h}_k} \tag{2.311}$$

where $\mathbf{J}_{\mathbf{h}_m}$ is the Jacobian of some (hidden) layer $\mathbf{h}_{m+1}$ with respect to $\mathbf{h}_m$. So the Jacobian with respect to $\mathbf{h}_k$ can be computed recursively given the *local* Jacobian of the next layer $k+1$ with respect to $\mathbf{h}_k$:

$$\tilde{\mathbf{J}}_{\mathbf{h}_k} = \tilde{\mathbf{J}}_{\mathbf{h}_{k+1}} \mathbf{J}_{\mathbf{h}_k} \tag{2.312}$$

Given Eq. (2.294), $\mathbf{J}_{\mathbf{h}_k}$ is simply (see Eq. (2.46) and Eq. (2.69)):

$$\mathbf{J}_{\mathbf{h}_k} = \mathbf{D}_{k+1} \mathbf{W}_{k+1} \tag{2.313}$$

where $\mathbf{D}_{k+1}$ is a square diagonal matrix with $f'_{k+1}(\cdot)$ on its diagonal. So overall

$$\tilde{\mathbf{J}}_{\mathbf{h}_k} = \tilde{\mathbf{J}}_{\mathbf{h}_{k+1}} \mathbf{D}_{k+1} \mathbf{W}_{k+1} \tag{2.314}$$

which is a $1 \times m$ row vector for $\mathbf{h}_k \in \mathbf{R}^m$. Matrix-vector operations are usually read from from right (the vector expression, $\tilde{\mathbf{J}}_{\mathbf{h}_{k+1}}$) to left (the matrix, $\mathbf{D}_{k+1} \mathbf{W}_{k+1}$), so the *gradient* of the loss $\ell(\theta, \mathbf{x}, \mathbf{y})$ with respect to $\mathbf{h}_k$ is

$$\tilde{\mathbf{J}}_{\mathbf{h}_k}^\intercal = \mathbf{W}_{k+1}^\intercal \mathbf{D}_{k+1} \tilde{\mathbf{J}}_{\mathbf{h}_{k+1}}^\intercal \tag{2.315}$$

This formula explains the name *backpropagation*: Compared to the way of evaluating the loss for some input $\mathbf{x}$, information for computing gradients flows backwards (hence *reverse* mode automatic differentiation). Computing these Jacobians is efficient because the recursive definition reuses previous computations. However, it is necessary to save some information from the forward computation, as becomes evident when computing $\mathbf{D}_{k+1}$: The layer-wise *preactivations* $\mathbf{a}_k = \boldsymbol{W}_k \mathbf{h}_k + \mathbf{b}_k$ are necessary for evaluating the first derivative of the respective activation functions. From Computer Science, it is a well known property of algorithms to trade savings in time for costs in space.

Given $\tilde{\mathbf{J}}_{\mathbf{h}_k}$, computing the first derivative of $\ell(\theta, \mathbf{x}, \mathbf{y})$ with respect to $\boldsymbol{W}_k$ becomes straightforward (Eq. (2.62)):

$$\ell'(\theta, \mathbf{x}, \mathbf{y}) = \tilde{\mathbf{J}}_{\mathbf{h}_k} \left[\mathrm{D}\mathbf{f}_k(\boldsymbol{W}_k \mathbf{h}_{k-1} + \mathbf{b}_k) \right], \tag{2.316}$$

And

$$\mathrm{D}\mathbf{f}_k(\boldsymbol{W}_k \mathbf{h}_{k-1} + \mathbf{b}_k) \equiv \mathbf{D}_k \frac{\partial \mathrm{vec}(\boldsymbol{W}_k \mathbf{h}_{k-1} + \mathbf{b}_k)}{\partial \mathrm{vec}(\boldsymbol{W}_k)^\mathsf{T}} \tag{2.317}$$

where $\mathbf{D}_k$ is defined as previously. Finally, from Eq. (2.66),

$$\frac{\partial \mathrm{vec}(\boldsymbol{W}_k \mathbf{h}_{k-1} + \mathbf{b}_k)}{\partial \mathrm{vec}(\boldsymbol{W}_k)^\mathsf{T}} = \mathbf{h}_{k-1}{}^\mathsf{T} \otimes \mathbf{I} \tag{2.318}$$

so in total

$$\ell'(\theta, \mathbf{x}, \mathbf{y}) = \tilde{\mathbf{J}}_{\mathbf{h}_k} \mathbf{D}_k (\mathbf{h}_{k-1}{}^\mathsf{T} \otimes \mathbf{I}_m) \tag{2.319}$$

This $1 \times nm$ vector expression can be simplified (see Eq. (2.67)) to

$$\ell'(\theta, \mathbf{x}, \mathbf{y}) = \mathrm{vec}(\mathbf{D}_k \tilde{\mathbf{J}}_{\mathbf{h}_k}{}^\mathsf{T} \mathbf{h}_{k-1}{}^\mathsf{T})^\mathsf{T} \tag{2.320}$$

And hence the *gradient* of the loss $\ell(\theta, \mathbf{x}, \mathbf{y})$ with respect to $\boldsymbol{W}_k$ is

$$\nabla_{\boldsymbol{W}_k} \ell(\theta, \mathbf{x}, \mathbf{y}) = \mathrm{vec}(\mathbf{D}_k \tilde{\mathbf{J}}_{\mathbf{h}_k}{}^\mathsf{T} \mathbf{h}_{k-1}{}^\mathsf{T}) \tag{2.321}$$

which is (in words) the outer product of the scaled backpropagated Jacobian $\tilde{\mathbf{J}}_{\mathbf{h}_k}$ and the input to layer k. Similarly,

$$\nabla_{\mathbf{b}_k} \ell(\theta, \mathbf{x}, \mathbf{y}) = \mathrm{vec}(\mathbf{D}_k \mathbf{J}_{\mathbf{h}_k}{}^\mathsf{T}) \tag{2.322}$$

It is instructive to compare Eq. (2.321) with Eq. (2.290), the gradient of a weight-matrix in a Sigmoid Belief Network with respect to the complete log-likelihood. It also has the form of an outer product between the input to a layer and an error signal for this layer and involves forward computations and backward computations (in the form of sampling). However, being overall a stochastic computation, the error signal has to be estimated locally and can't be transferred in a direct and deterministic way from higher layers to lower layers

(information between layers is transported through conditional sampling).

Finally, the recursive computation must also be initialized. Because the gradient of the parameters with respect to the regularization term $\boldsymbol{\Omega}(\theta)$ is usually simple to compute, the backpropagation algorithm is initialized with the Jacobian of $N(\theta, x)$ with respect to the loss $L(\cdot,\cdot)$. For matching loss functions this Jacobian has usually a particular simple form, e.g. in the case of non-linear regression (Eq. (2.153)) or classification (Eq. (2.159)) it is simply the distance vector between the last layer h_K and the observed output, i.e.

$$\tilde{J}_{h_K} = N(\theta, x) - y \tag{2.323}$$

So overall

$$\nabla_\theta \ell(\theta, x, y) \equiv (\nabla_{W_K}, \nabla_{b_K}, \nabla_{W_{K-1}}, \ldots, \nabla_{W_1}, \nabla_{b_1}), \tag{2.324}$$

which takes $O(n)$ to compute if the cost to evaluate the non-linearities and their respective derivatives is negligible. Note that from an implementation perspective, the above derivations assume that weight matrices are stored in a column-first order (i.e. the way FORTRAN stores matrices).

How successful is the direct improvement signal J_L, telling the output how it should change locally to improve the assessed risk $\ell(\theta, x, y)$, relayed to the parameters, e.g. some weight matrix at layer k? A reasonable approach to this question is to *compare the norms* $\|J_L\|_2$ and $\|\tilde{J}_{h_k}\|_2$. When the length of a vector is considered to express its information content then a skewed proportion between the two lengths indicates that information gets lost. Reiterating Eq. (2.311), $\tilde{J}_{h_k}$ is

$$\tilde{J}_{h_k} = J_L J_M J_{h_K} J_{h_{K-1}} \cdots J_{h_k} \tag{2.325}$$

The spectral norm Eq. (2.29) is consistent (Eq. (2.31)), hence

$$\|\tilde{J}_{h_k}\|_2 \leqslant \|J_L\|_2 \|J_M\|_2 \prod_{i=k+1}^{K} \|W_i\|_2 \|D_i\|_2 \tag{2.326}$$

Eq. (2.326) shows that the transfer of information can become arbitrarily bad. Gradient information either *vanishes*, e.g. in the case of activation functions with (absolute) derivatives $\|D_i\|$ that are strictly smaller than 1, or *explodes*, e.g. in the case of badly initialized weight matrices. The problem of vanishing/exploding gradients was first identified for Recurrent Neural Networks [159, 26, 163] because it only manifests itself with networks of some depth. Overall it leads to a highly varying distribution of gradient information per layer and therefore destabilizes the optimization process. For a very long time it was considered to be the main obstacle to train Neural Networks with many layers of non-linearities.

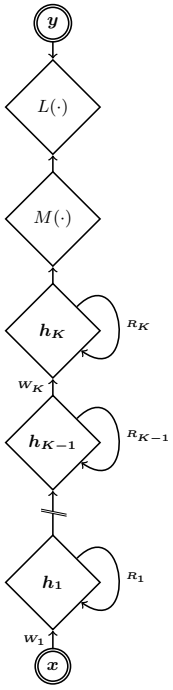


Figure 2.6: A simple recurrent Neural Network, as defined in Eq. (2.327). The recurrent connections are only within the particular layers. Bias units are left out for clarity. Best viewed electronically.

2.7.1 *Recurrent Neural Networks*

Given the derivation of the backpropagation algorithm for a feed-forward network, it is not difficult to extend the algorithm to a general Recurrent Neural Network. The basic idea is to unfold the recursive structure of the RNN for a given input/output pair $(\mathbf{X}, \mathbf{Y})$ into a large (deep) Neural Network with massively shared parameters. Consider the following simple RNN with K layers and an input sequence of length n:

$$\begin{aligned} \mathbf{h}_k^i &= f_k(\mathbf{W}_k \mathbf{h}_{k-1}^i + \mathbf{R}_k \mathbf{h}_k^{i-1} + \mathbf{b}_k)\ ,\ 1 \leqslant i \leqslant n, 2 \leqslant k \leqslant K \\ \mathbf{h}_1^i &= f_k(\mathbf{W}_1 \mathbf{x}^i + \mathbf{R}_1 \mathbf{h}_1^{i-1} + \mathbf{b}_1)\ ,\ 1 \leqslant i \end{aligned} \tag{2.327}$$

Hereby, the input $\mathbf{X}$ is a sequence of vectors $\mathbf{X} = (\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^n)$ and the target $\mathbf{Y} = (\mathbf{y}^1, \mathbf{y}^2, \ldots, \mathbf{y}^n)$ an associated sequence of target vectors. The parameters θ encompass the standard feed-forward parameters $(\mathbf{W}_k, \mathbf{b}_k), 1 \leqslant k \leqslant K$, the recurrent weights $\mathbf{R}_k, 1 \leqslant k \leqslant K$ and the k many initial states $\mathbf{h}_k^0, 1 \leqslant k \leqslant K$. Eq. (2.327) is the simplest version of a recurrent Neural Network, usually the recurrence structure is fully connected between all layers and/or has additional elements beyond standard affine transformations (e.g. Long short-term Memory (LSTM) [161]), but it is simple to depict graphically (see Fig-

ure 2.6). For a training pair $(\mathbf{X}, \mathbf{Y})$ the empirical risk for the network is defined per time step, i.e.

$$\ell(\theta, \mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{n} L(M(\mathbf{h}_k^i), \mathbf{y}^i) + \lambda\Omega(\theta) \tag{2.328}$$

In order to compute the gradient of θ with respect to $\ell(\theta, \mathbf{X}, \mathbf{Y})$ the computational graph induced by Eq. (2.327) for the specific training sample $(\mathbf{X}, \mathbf{Y})$ is *unrolled in time*. This is depicted in Figure 2.7 for a sequence of length n. Most of the parameters are reused over the course of the computation (they are *shared over time*), but in order to facilitate the backpropagation algorithm from Eq. (2.321), the shared parameters are unaliased per time step. That is, for the time being, the parameters are unshared, and then it is straightforward to compute the gradient for every parameter. The overall gradient for a shared parameter is then simply the sum of the gradients of the unaliased parameters (Backpropagation Through Time (BPTT) [416]). For example, consider $\mathbf{R}_1$:

$$\frac{\partial \ell(\theta, \mathbf{X}, \mathbf{Y})}{\partial \mathbf{R}_1} = \sum_{i=1}^{n} \frac{\partial \ell(\theta, \mathbf{X}, \mathbf{Y})}{\partial \mathbf{R}_1^i} \tag{2.329}$$

Hereby, $\frac{\partial \ell(\theta, \mathbf{X}, \mathbf{Y})}{\partial \mathbf{R}_1^i}$ is simply computed with Eq. (2.321), using the underlying feed-forward computation graph shown in Figure 2.7.

2.7.2 *Convolutional Neural Networks*

Eq. (2.327) induces parameter sharing over time. Sharing parameters is also a necessity with respect to generalization—the RNN should be able to handle sequences of arbitrary length. However, also for a standard feed-forward Neural Network parameter sharing is possible and reasonable. In this case, single elements of a weight matrix (or associated groups of elements) are shared, possibly only limited to dedicated layers or over several layers. A very general approach for parameter sharing in this regard is *soft weight sharing* [279, 35] (which can of course also be applied to RNN's), acting as a regularization method. With soft weight sharing the hard constraint of equal weights (as is the case with RNNs with respect to sharing over time steps) is replaced by a set of weights that have *similar* weights. A special case for soft weight sharing is weight decay, where the weights are put into exactly one group and encouraged to have values close to 0. Soft weight sharing generalizes this MAP based approach. The prior $p(\theta)$ with

$$p(\theta) = \prod_i p(\theta_i) \tag{2.330}$$

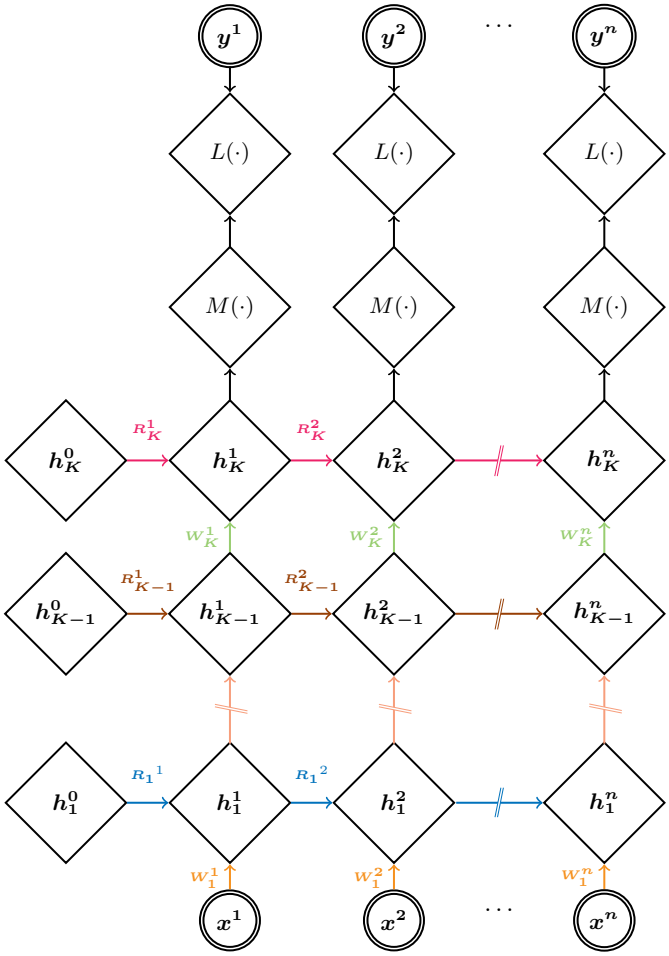


Figure 2.7: Eq. (2.327) unrolled in time. Shared parameters are identified with the same color, but are unaliased in the unrolled computation graph (indicated by the superscript). Best viewed electronically.

uses a flexible mixture model for $p(\theta_i)$:

$$p(\theta_i) = \sum_{j=1}^{M} \pi_j \mathcal{N}(\theta_i \mid \mu_j, \sigma^2{}_j) \tag{2.331}$$

Identical to the MAP approach, the negative log-likelihood of the prior is then used as a regularization term for the empirical risk term:

$$\Omega(\theta) = -\sum_i \log \left(\sum_{j=1}^{M} \pi_j \mathcal{N}(\theta_i \mid \mu_j, \sigma^2{}_j) \right) \tag{2.332}$$

The soft weight parameters π, μ and σ^2 are then also adaptively determined on the training set together with the parameters θ. One possibility would be to utilize the EM algorithm (Eq. (2.265)), alternating between the soft weight parameters and θ. However, both sets

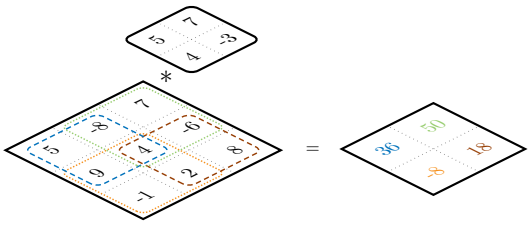


Figure 2.8: Convolving a 3×3 image (angular corners) with a 2×2 filter (rounded corners). The convolution is only computed at those positions where both image and filter overlap (indicated by the colored rectangles). Best viewed electronically.

of parameters heavily depend on each other and therefore a joint optimization using gradient-based approaches is reasonable, relying on the gradients for the mixture model as given in Eq. (2.263).

Weight sharing is also reasonable if the input data has a certain *topological* structure that implies that weights should be shared. The premier example for this case are (natural) images. Images considered as two-dimensional signals are *position invariant* and *local*. In mathematical terms this is a *linear shift invariant (LSI)* system. The standard method to analyze LSI systems is *convolution* [405]. Convolution is an operator defined on two functions f (the signal to be analyzed) and g (the *filter*), resulting in the new function $(f * g)$. If $f : \mathbf{R} \to \mathbf{R}$ and $g : \mathbf{R} \to \mathbf{R}$ are continuous functions then $(f * g) : \mathbf{R} \to \mathbf{R}$ with

$$(f * g)(x) = \int f(y)g(x-y)dy = \int f(x-y)g(y)dy \tag{2.333}$$

which can be readily extended to the n-dimensional case.

For discrete functions $f : \mathbf{Z} \to \mathbf{R}$ and $g : \mathbf{Z} \to \mathbf{R}$, the *discrete convolution* $(f * g)$ is defined by

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m] = \sum_{m=-\infty}^{\infty} f[n-m]g[m] \tag{2.334}$$

For the specific case of a two-dimensional signal $f : \mathbf{Z} \times \mathbf{Z} \to \mathbf{R}$ and a two-dimensional filter $g : \mathbf{Z} \times \mathbf{Z} \to \mathbf{R}$, Eq. (2.334) is defined as

$$\begin{aligned}(f * g)[n, m] &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l]g[n-k, m-l] \\ &= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[n-k, m-l]g[k, l]\end{aligned} \tag{2.335}$$

Note that $(f * g)$ is a valid function on the domain of f and thus can serve as the signal for a subsequent convolution. So while convolution is an inherently local operation, by repeated applications to a given signal global properties of the signal can be determined.

In both the continuous and discrete case the convolution *exists* if at least one function has a compact support[16]. In the case of discrete functions, if the filter g has finite support on the set $\mathcal{M} \equiv \{-M, -M+1, \ldots, M-1, M\}$ (that is, $g[n] = 0$, for $n \notin \mathcal{M}$), then

$$(f * g)[n] = \sum_{m=-M}^{M} f[n-m]g[m] \tag{2.336}$$

The discrete convolution can be transformed into a matrix-vector multiplication where one of the two functions is converted into a Toeplitz matrix (Eq. (2.9)). Without loss of generalization assume that both $\mathbf{f}$ and $\mathbf{g}$ have compact support and let $\mathbf{f} = (f_1, f_2, \ldots, f_n)^T$ and the filter $\mathbf{g} = (g_1, g_2, \ldots, g_m)^T$ with $m \ll n$. Then $\mathbf{h} = \mathbf{f} * \mathbf{g}$ can be written as

$$\mathbf{h} = \begin{pmatrix} g_1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ g_2 & g_1 & 0 & \cdots & \cdots & \cdots & 0 \\ g_3 & g_2 & g_1 & 0 & \cdots & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ g_m & g_{m-1} & \ddots & g_1 & 0 & \cdots & 0 \\ 0 & g_m & g_{m-1} & \ddots & g_1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & g_m & g_{m-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & g_m \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix} \tag{2.337}$$

So $\mathbf{h}$ is a discrete signal with $n + m - 1$ elements. In order to avoid *border effects* within the convolved signal, the sum of the convolution operation (Eq. (2.336)) is sometimes restricted to the domain where both signal and filter have support[17]. In this case the matrix-vector expression results in an $n - m + 1$ dimensional signal (with $\mathbf{f}$ and $\mathbf{g}$ defined as previously):

$$\mathbf{h} = \begin{pmatrix} g_m & g_{m-1} & \cdots & g_1 & 0 & 0 & \cdots & 0 \\ 0 & g_m & g_{m-1} & \cdots & g_1 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & g_m & g_{m-1} & \cdots & g_1 & 0 \\ 0 & 0 & 0 & 0 & g_m & g_{m-1} & \cdots & g_1 \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{n-1} \\ f_n \end{pmatrix} \tag{2.338}$$

Eq. (2.337) and Eq. (2.338) are also possible for n-dimensional convolutions, the filter elements g_i in the rows of the Toeplitz matrix are

16 The convolution between two functions exists under much more permissible conditions, but these are mathematically more involved and not relevant for this text.

17 This *mode* of convolution is usually denoted *valid*, while the computation according to the definition in Eq. (2.336) is called *full*.

then interleaved with zeros, while the input signal f is vectorized according to some regular schema, e.g. Eq. (2.53). As a two-dimensional example for Eq. (2.338) consider Figure 2.8: A 3×3 image gets convolved with a 2×2 filters, ignoring border patterns, i.e. the convolution is only computed at these positions of the resulting functions where the support of the image and the filter overlap completely. The result (also shown in Figure 2.8 to the right) is a 2×2 image. If images are represented as vectors according to the $\text{vec}(\cdot)$ operator (Eq. (2.53)), then Eq. (2.338) can be written in the following way in this specific case:

$$\begin{pmatrix} 36 \\ -8 \\ 50 \\ 18 \end{pmatrix} = \begin{pmatrix} -3 & 7 & 0 & 4 & 5 & 0 & 0 & 0 & 0 \\ 0 & -3 & 7 & 0 & 4 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & -3 & 7 & 0 & 4 & 5 & 0 \\ 0 & 0 & 0 & 0 & -3 & 7 & 0 & 4 & 5 \end{pmatrix} \begin{pmatrix} 5 \\ 9 \\ -1 \\ -8 \\ 4 \\ 2 \\ 7 \\ -6 \\ 8 \end{pmatrix} \tag{2.339}$$

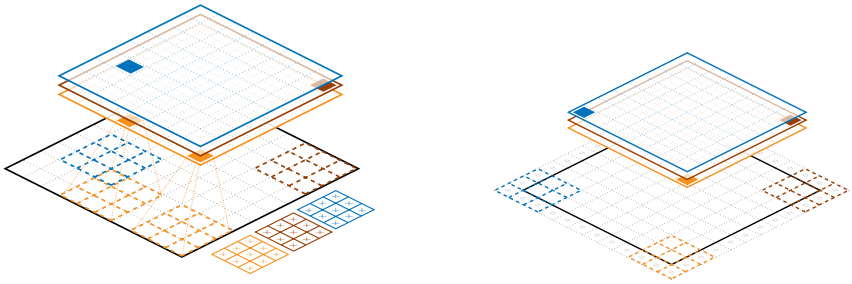
In practical applications, it is sometimes necessary to have a certain degree of border information. This can be realized through *zero padding* the signal on its borders and then applying a *valid* convolution. Furthermore, the signal to be analyzed may be very high-frequent but also rather redundant. In this case, convolution can be applied only every s-th input sample, increasing the *effective stride* of Eq. (2.336) to s. So if a one-dimensional discrete signal with n elements is zero-padded with z zeros on every end and then convolved with a filter of length m using a stride of s the resulting signal has length $(n - m + 2z + 1)//s$ where $//$ denotes integer division.

A related operation to convolution is the *cross-correlation* [405] operator $\star$ (here only defined for the discrete case):

$$(f \star g)[n] = \sum_{m=-\infty}^{\infty} f^*[m]\, g[n+m] \tag{2.340}$$

with f^* the complex conjugate of f. From a Machine Learning point of view, convolution and cross-correlation are identical operations in the case of real-valued signals, because differently to the usual application of convolution in either mathematics or signal processing, for Machine Learning the goal is to learn the filter operator g.

Considering Eq. (2.337) it is easy to see how this can be integrated into a Neural Network: a convolution can be expressed as an affine transformation where the rows of the associated weight matrix are shared parameters. Also differently from standard applications, a so



(a) (b)

Figure 2.9: A convolutional layer with three different filters resulting in an output with three channels. (a) The three different filters are visualized in a stylized way to the right of the input image. Every filter computes the respectively colored feature map. For the orange filter, the convolution operation for two different positions in the feature map are sketched in more detail. (b) The three filters applied to the same image with a zero padded border of width 1. In this case, resulting feature maps have the same dimensionality as the input. In both (a) and (b) the filters are applied with a stride of 1. Best viewed electronically.

called *convolutional layer* consists not only of one filter g but usually comprises a *set of learnable filters*. This set of filters then computes a set of *feature maps* where every feature map retains the basic structure of the input signal. That is, if the input is for example an image, a feature map also represents an image, see Figure 2.9 for the case of a two-dimensional input signal. These sets of feature maps are organized into *channels*.

If the input to a convolutional layer already spans a number of channels $|c|$ (for example, in the case of images these might be the red, green and blue color channels, or several consecutive (gray-valued) frames from a video) than a learnable filter g for this kind of input consists of the same channel-like structure:

$$g \equiv (g_1, g_2, \dots, g_{|c|}) \tag{2.341}$$

A feature map h associated with such a filter g is then computed by summing up the individual convolutions per channel, i.e.

$$h = \sum_{i=1}^{|c|} f_i * g_i \tag{2.342}$$

Again, the underlying structure of the input (as given by the structure of the individual channels) is retained by this definition. Figure 2.10 shows a simple example where several channels of two-dimensional images are convolved with three different filters.

A Neural Network that has layers made up of convolutions of the form Eq. (2.342), equipped with an additional bias and followed by

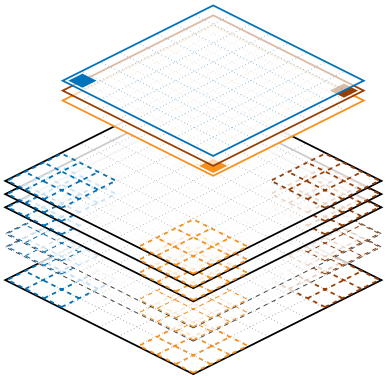


Figure 2.10: A convolutional layer with a stack of channels as input. The feature maps are computed by summing up the individual convolutions of the respective filters. Again, the three different sets of filters produce an output containing three channels. Best viewed electronically.

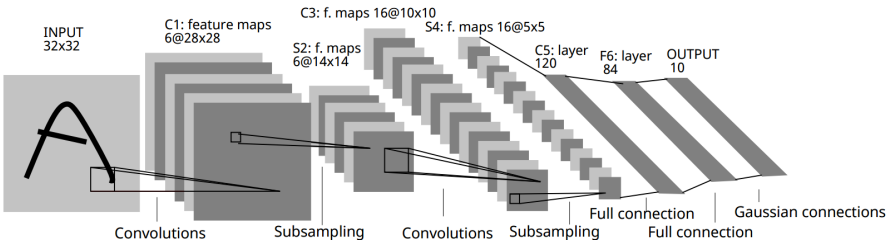
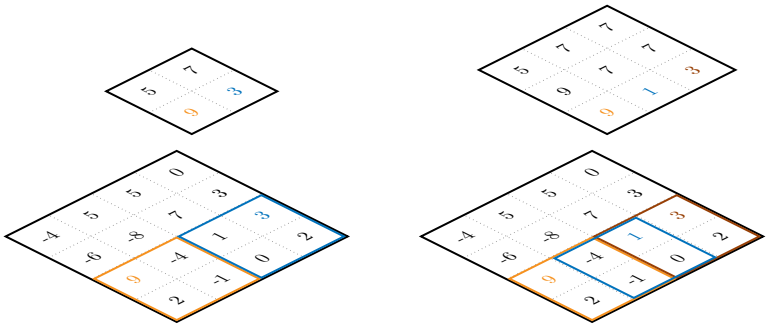


Figure 2.11: The architecture of LeNet-5, a convolutional Neural Network used to recognize human handwriting. The figure is taken from [221].

non-linearities is called a *Convolutional Neural Network (CNN)* [103, 219, 221]. Figure 2.11 shows a well-known instance of a CNN, LeNet-5 [221]. The figure depicts the typical form of CNNs, layers of convolutions followed by standard *fully-connected* layers. In a CNN, the convolutional layers are typically interleaved with reduction operations. These usually *pool* over a small area of the individual feature maps (but never between channels). Pooling operations are typical aggregation functions like averaging [221] or maximum [414, 299] operations. Recently, more involved pooling functions have been proposed, e.g. L_p pooling [341] or distance transform pooling [112]. Pooling implements not only a form of dimensionality reduction but also encodes invariances of the network to geometric transformations of the input, e.g. the repeated application of max-pooling over several layers leads to a certain degree of *translation invariance* of the network. Figure 2.12 shows variants of the maximum operation over a 2×2 pooling region.

These pooling operations can also be expressed as matrix-vector operations, the elements of the respective matrices are no learnable



(a) (b)

Figure 2.12: *Max-Pooling* over 2×2 regions with (a) non-overlapping or (b) overlapping [208] pooling areas. In both cases some pooling regions with the respective dominant element are colored. Best viewed electronically.

parameters, though. Instead, these matrices are usually determined adaptively, depending on the current parameters θ and the input x to the network. However, it is also possible to use a standard learnable convolutional layer instead of a pooling layer [363]. In order to simulate the dimensionality reduction aspect, the stride of the convolution is increased and the convolution keeps the number of feature maps identical. In its basic form, convolution can only simulate simple aggregation functions. For more complex pooling operations, specialized activation functions are also necessary, e.g. the learned-norm pooling function [137]. Compared to the state-of-the-art performance of max-pooling based CNNs for object recognition, this type of CNN with only convolutions (and alternating subsampling) performs competitively [363].

Because a CNN can be formulated as a stack of non-linear affine transformations (Eq. (2.295)), computing the gradients of the filters g of some convolutional layer with respect to a loss function $\ell(\theta, x, y)$ can be done by the backpropagation algorithm (Eq. (2.321)). Consider only one filter g, with its Toeplitz matrix from Eq. (2.337) denoted as $G \in \mathbf{R}^{n \times m}$ (the exact values for n and m depend not only on the size of the filter and the (channeled) input but also on the chosen stride(s) and zero padding(s)). $\nabla_G \ell(\theta, x, y)$ is easy to derive, but it does not yet take into account that the rows of G are actually shared. This is however straightforward, if the Toeplitz structure of G is exploited: First, reshape $\nabla_G \ell(\theta, x, y)$ (an nm-dimensional vector) into an $n \times m$ matrix using the $\operatorname{cev}(\cdot, n, m)$ operator (Eq. (2.55)). Then build the sum of the elements along all possible $n + m - 1$ diagonals, forming an $n + m - 1$ vector. This vector can then be reshaped into the same form

as the original filter g. This reshape however depends again on the size of the filter, the chosen strides and the chosen zero padding(s).

$$\nabla_g \ell(\theta, x, y) = \sum_{\backslash} \nabla_G \operatorname{cev}(\ell(\theta, x, y), n, m) \tag{2.343}$$

where $\sum_{\backslash}$ denotes forming a vector by summing up elements along all possible diagonals of a matrix.

As it turns out $\nabla_g \ell(\theta, x, y)$ can also be computed through convolving a mirrored Jacobian and the input to the respective convolutional layer. Again, let g be a filter that is applied convolutionally to some arbitrary input x (a tensor), though with one channel only (without loss of generality, as will be seen shortly). Furthermore, let h denote the result of the convolution, i.e.

$$h = x * g \tag{2.344}$$

The Jacobian $\tilde{J}_h$ of h (Eq. (2.311)) with respect to $\ell(\theta, x, y)$ has the same structure as x. So in general $\tilde{J}_h$ is representable as a tensor, e.g. in the case of x being a 2d image, $\tilde{J}_h$ also represents a 2d image. Given the elementwise definition of the convolution operator $*$ (Eq. (2.336)), $\nabla_g \ell(\theta, x, y)$ can also be derived elementwise which results in

$$\nabla_g \ell(\theta, x, y) = \operatorname{vec}(([\tilde{J}_h]^\circ * x)^\circ) \tag{2.345}$$

where $[\tilde{J}_h]$ indicates that the Jacobian is represented as a tensor and $\circ$ denotes the point reflection operator for tensors, see Eq. (2.38). If the input x has several channels then $[\tilde{J}_h]$ is simply convolved with every channel separately to get the respective gradients, because Eq. (2.341) only involves a sum over the filtered channels.

For example in the case of x being an image, this means that in order to compute $\nabla_g \ell(\theta, x, y)$ the 2d interpretation of $\tilde{J}_h$ is rotated by 180 degrees, convolved with x and then rotated again by 180 degrees.

$\tilde{J}_h$ can also be computed using convolution explicitly. However it depends heavily on the actually used stride s and zero-padding z. Assume $s = 1$ and $z = 0$. Furthermore, assume that $\tilde{J}_k$ denotes the Jacobian of the layer *after* h and that both layers only have one feature map (again, without loss of generalization). If f is a dimensionally consistent filter such that

$$k = h * f \tag{2.346}$$

then $\tilde{J}_h$ is computed from a zero-padded Jacobian $\tilde{J}_k$. More specifically, denote $\tilde{J}_k^m$ the Jacobian $\tilde{J}_k$ that is zero-padded by $m - 1$ zeros on both ends per every dimension of the Jacobian. Then

$$\tilde{J}_h = \tilde{J}_k^m * f^\circ \tag{2.347}$$

If h has n elements per dimension then k has $n - m + 1$ elements, if f has m elements per dimension. The zero-padded Jacobian $\tilde{J}_k^m$ has

$n - m + 1 + 2(m-1) = n + m - 1$ elements per dimension. Hence, $\tilde{J}_h$ has $n + m - 1 - m + 1 = n$ elements.

The backward path from Eq. (2.347) for one convolutional layer can also be used to realize *the forward path of an upsampling convolutional operation*. More specifically, an upsampling convolutional layer with an upsampling factor of k can be implemented by the backward pass of a convolutional layer with stride k. This operation is sometimes denoted *fractional strided convolution* [236]. A much simpler realization of fractional strided convolution is using *perforated upsampling followed by standard convolution* [285][18]. In perforated upsampling every element of a feature map is replaced by a cube (same dimensions as the feature map) having k zeros per dimension. The elements of the original feature map are placed at the first element of their respective cubes. This upsampling operator is then followed by a standard convolution operator of the desired size.

An important practical aspect is to consider which of the two different technical approaches to compute the gradients of the filters is more efficient. The widely used Caffe framework [188] implements both forward and backpropagation paths in CNNs as matrix operations, relying on the highly optimized matrix-matrix operations of the underlying hardware. This approach requires more memory though, because the Toeplitz matrix requires more space than its associated filter. On the other hand, it might be reasonable to use the Fast Fourier Transformation (FFT) to realize efficient forward and backward convolutions [405]. However, for CNNs the filters used in the forward computations are usually small, so an FFT-based approach only is promising for the forward path if high-dimensional inputs (i.e tensors) are used. However, in the backward path for the backpropagation algorithm Jacobians and feature maps are convolved with each other and then an FFT-based approach should be beneficial [251].

From an optimization point of view, minimizing some loss function induced by a Neural Network is considered as one the most challenging real-world optimization problems. Already the invariance of the evaluation of a network to symmetric re-orderings of weights leads to the existence of an exponential amount of equivalent local minima [34]. However, one has to realize that the overall goal of optimizing a Neural Network is to find settings with high generalization capabilities: The network has to perform well on unseen future data, and hence the training data must be considered as a *stochastic proxy* for the truly unknown objective. Therefore, finding the absolute minimum on an apriori stochastic cost function is not worthwhile. Indeed, recent research hints at this being be counterproductive to the goal of generalization [57]. This makes *stochastic optimization* techniques a necessity. Interestingly, simple *stochastic gradient descent* [312] itera-

18 Hubert Soyer came up with the idea and the name of perforated upsampling. Back then we didn't know that it implements fractional strided convolution.

tively evaluated on a small set of randomly sampled training data (a so called *minibatch*) and equipped with a *momentum term* [319] often leads to very good parameter settings.

More specifically, for minibatch stochastic gradient descent the update direction $\delta\theta_t$ at the t-th optimization iteration ($t \geqslant 1$) is evaluated as:

$$\delta\theta_t = \frac{1}{|\mathcal{D}'|} \sum_{x_i, y_i \in \mathcal{D}'} \nabla_\theta \ell(\theta_{t-1}, x_i, y_i), \quad \mathcal{D}' \subset \mathcal{D},\ |\mathcal{D}'| \ll |\mathcal{D}| \tag{2.348}$$

The minibatch $\mathcal{D}'$ is hereby sampled randomly from $\mathcal{D}$ without replacement. With a fixed learning rate η_t the new parameter value θ_t at iteration t is

$$\theta_t = \theta_{t-1} + \eta_t \delta\theta_{t-1} \tag{2.349}$$

η_t is often independent of iteration t and set to a small value (between 0.1 and 0.0001), ensuring the validity of the underlying Taylor approximation. Yet, in order to exact convergence, η_t can also be annealed to zero with a simple schedule [312, 327], e.g.

$$\eta_t = \eta_0 (1 + \gamma t)^{-1} \tag{2.350}$$

with η_0 and γ being two hyperparameters.

An important question is how θ_0 is chosen. A widely used *initialization schema* is to sample the elements of the weight matrices from a Gaussian with mean 0 and standard deviation 0.1 and set the biases to 0. However, as will be pointed out later in the text, choosing the initial parameter values for a Neural Network can have a heavy influence on its successful application to the task at hand, in particularly for deep networks[19]. A simple guiding principle for choosing an initialization is to ensure that at the beginning of the training the *linear regime* of the non-linearities is utilized, which allows gradient information to be propagated to all layers of the network (cf Eq. (2.326)). For the $\tanh(\cdot)$ activation function a good rule is to draw the elements of a weight-matrix $W_k \in \mathbf{R}^{m \times n}$ from a distribution centered at 0 with variance $1/n$[20], i.e. $W_k \sim \mathcal{N}(W_k \mid 0, 1/n\mathbf{I})$ or $W_k \sim \mathcal{U}(-\sqrt{3}/n, \sqrt{3}/n)$ [224].

How is $\mathcal{D}'$ at some iteration t chosen? A typical approach in Machine Learning is to split the training set $\mathcal{D}$, $|\mathcal{D}| = m$, into $\lceil \frac{m}{k} \rceil$ many minibatches $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_{\lceil \frac{m}{k} \rceil}$ with

$$\begin{aligned} |\mathcal{D}_i| &= k, \quad \forall\, 1 \leqslant i < \left\lceil \frac{m}{k} \right\rceil, \\ \mathcal{D} &= \mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_{\lceil \frac{m}{k} \rceil} \text{ and} \\ \mathcal{D}_i \cap \mathcal{D}_j &= \varnothing, \quad \forall\, 1 \leqslant i < j \leqslant \left\lceil \frac{m}{k} \right\rceil \end{aligned} \tag{2.351}$$

19 The initial value of the bias can have a large influence on the overall optimization process, depending on the utilized non-linearities [109, 146, 191].

20 n is sometimes denoted the *fan-in* of a layer.

The next $\lceil \frac{m}{k} \rceil$ many update directions are then evaluated on the respective minibatches $\mathcal{D}_i$. After one such *epoch* (a complete learning pass through the training set where every sample in $\mathcal{D}$ is used exactly once to calculate an update direction) the minibatch splits are resampled anew. k is considered a hyperparameter, for $k = 1$ the standard stochastic gradient descent approach is recovered [38, 37].

Finally, from a more practical point of view, the sum in Eq. (2.348) may seem cumbersome at first, after having streamlined the derivation of the gradient for one input/output pair (i.e. Eq. (2.321) and Eq. (2.322)). However, reformulating the involved expressions in terms of matrices, the above sum can be represented as a simple matrix-matrix product. To see this, first consider the specific update direction for some weight matrix $\boldsymbol{W}_k \in \mathbf{R}^{m \times n}$ at iteration t in the minibatch case:

$$\begin{aligned} \delta \boldsymbol{W}_{k,t} &= \sum_{\boldsymbol{x}_i, \boldsymbol{y}_i \in \mathcal{D}'} \operatorname{vec}(\boldsymbol{D}_{k,i} \tilde{\boldsymbol{J}}_{\boldsymbol{h}_{k,i}}{}^{\mathsf{T}} \boldsymbol{h}_{(k-1),i}{}^{\mathsf{T}}) \\ &= \operatorname{vec}\left(\sum_{\boldsymbol{x}_i, \boldsymbol{y}_i \in \mathcal{D}'} \boldsymbol{D}_{k,i} \tilde{\boldsymbol{J}}_{\boldsymbol{h}_{k,i}}{}^{\mathsf{T}} \boldsymbol{h}_{(k-1),i}{}^{\mathsf{T}} \right) \end{aligned} \tag{2.352}$$

where the index i at $\boldsymbol{D}_{k,i}$, $\tilde{\boldsymbol{J}}_{\boldsymbol{h}_{k,i}}$ and $\boldsymbol{h}_{(k-1),i}$ indicates the association of these three terms with the training pair $(\boldsymbol{x}_i, \boldsymbol{y}_i)$. This is a sum over the outer products of the respectively scaled Jacobians and the associated input to the k-th hidden layer. Using Eq. (2.22) this can be written much more elegantly (and, important for practical purposes, computationally much more efficient) as a matrix-matrix product. Let's define a matrix $\boldsymbol{H}_{k-1} \in \mathbf{R}^{k \times n}$ that collects the k many hidden activations $\boldsymbol{h}_{k-1} \in \mathbf{R}^n$ in a row-wise manner. Additionally, let the matrix $\hat{\boldsymbol{J}}_k \in \mathbf{R}^{m \times k}$ collect the k Jacobians $\tilde{\boldsymbol{J}}_{\boldsymbol{h}_{k,i}}$ in a *column*-wise manner. Finally, $\boldsymbol{D}_{k,i}$ being k many diagonal matrices, a matrix $\hat{\boldsymbol{D}}_k \in \mathbf{R}^{m \times k}$ collects the diagonals as vectors also in column-wise manner. Given these three matrices, Eq. (2.352) can be rewritten as

$$\begin{aligned} \delta \boldsymbol{W}_{k,t} &= \operatorname{vec}\left(\sum_{\boldsymbol{x}_i, \boldsymbol{y}_i \in \mathcal{D}'} \boldsymbol{D}_{k,i} \tilde{\boldsymbol{J}}_{\boldsymbol{h}_{k,i}}{}^{\mathsf{T}} \boldsymbol{h}_{(k-1),i}{}^{\mathsf{T}} \right) \\ &= \operatorname{vec}\left((\hat{\boldsymbol{D}}_k \odot \hat{\boldsymbol{J}}_k)\, \boldsymbol{H}_{k-1} \right), \end{aligned} \tag{2.353}$$

where $\odot$ is the elementwise multiplication of two matrices (see the Linear Algebra section, e.g. Eq. (2.8) and Eq. (2.22) for a detailed review of the utilized identities). $\delta \boldsymbol{b}_{k,t}$ needs a vector $\mathbf{1}$ of ones:

$$\delta \boldsymbol{b}_{k,t} = (\hat{\boldsymbol{D}}_k \odot \hat{\boldsymbol{J}}_k)\, \mathbf{1} \tag{2.354}$$

If a momentum term $\boldsymbol{\nu}_t$ is utilized, the updates at iteration $t \geqslant 1$ are as follows:

$$\begin{aligned} \boldsymbol{\nu}_t &= \mu \boldsymbol{\nu}_{t-1} + \eta_t \delta \boldsymbol{\theta}_{t-1} \\ \boldsymbol{\theta}_t &= \boldsymbol{\theta}_{t-1} - \boldsymbol{\nu}_t \end{aligned} \tag{2.355}$$

with $\mu \in (0;1)$ being a decay (hyper-) parameter and $\boldsymbol{v}_0 = \mathbf{0}$. The basic motivation for a momentum term $\boldsymbol{v}_t$ is to accelerate the optimization process along those dimensions in which the gradient behaves consistently and, on the other hand, soften the negative impact of dimensions which have inconsistent gradient information and thus mitigates oscillations. For that it builds an exponential moving average over update directions which results in the cancellation of gradient information for those inconsistent parameter dimensions. Yet, being a purely additive method its usefulness can be quite limited for challenging cases.

One hypothesis why stochastic gradient descent works so well is that stochastic minibatches ensure the optimization process never getting stuck in bad local minima, in particular at the beginning of the process [375]. As pointed out already earlier, large Neural Networks should not suffer from bad local minima, as the probability of finding one is exponentially small [326, 123, 57]. Instead, it is important that the optimization does not get stuck at points that look locally like minima but are actually saddle points (*spurious minima*). Results from Random Matrix Theory [8, 7] indicate that saddle points are prevalent in Neural Networks (for both small- as well as large-scale networks). Large numbers of saddle points can explain why classical second-order Newton methods never gained traction for Neural Networks, as such methods have difficulties dealing with them [76].

A theoretically motivated approach resolving this problem is to utilize additional second order information *in a stochastic manner*. While several promising stochastic second order optimization algorithms where proposed ([336, 246, 76]), stochastic gradient descent methods with simple *preconditioning* [108, 327] ensuring different multiplicative scaling factors for every parameter perform surprisingly robust and good in general [86, 388, 427] and only rely on first order information. One such approach is AdaDELTA [427]. It is based on the following two premises:

- The optimization should progress in a similar fashion along all dimensions. This is particularly a good idea with deep networks because the scale of the gradients in each layer differs often in the order of several magnitudes. So a direction that *tends* to have *predominantly* large (absolute) gradient values should have *predominantly* small learning rates. And vice versa. A good local estimate for the behavior of a direction's absolute gradient values is the average of the square gradient values over a fixed window [86]. An efficient approximation (efficient with respect to memory requirements) of such a window is a simple exponential moving average of the elementwise squared gradient:

$$\mathrm{E}(\mathbf{g}^2)_t = \rho \mathrm{E}(\mathbf{g}^2)_{t-1} + (1-\rho)\mathbf{g}_t{}^2 \tag{2.356}$$

with $g_t \equiv \delta\theta_t$, the update direction at iteration t. ρ is a *decay* constant, a hyperparameter. Learning rate adaptation per dimension then happens with the squared root of $E(g^2)_t$ resulting in an approximation of the root mean squared length of all update directions up to iteration t. That is, the improved parameter update rule at iteration t is then

$$\theta_t = \theta_{t-1} + \frac{\eta_t}{\text{RMS}(g)_t}\delta\theta_{t-1} \tag{2.357}$$

with

$$\text{RMS}(g)_t = \sqrt{E(g^2)_t + \varepsilon} \tag{2.358}$$

where a small constant $\varepsilon \approx 1e-6$ is added for improving the condition of the denominator [22]. Eq. (2.357) is also known as the *RMSprop* update rule [388].

- The problem with the update rule in Eq. (2.357) (and also for Eq. (2.349) and Eq. (2.355)) is that it is dimensionally inconsistent (*not covariant* [242, 203]): The left hand side has a vector of units suitable for the parameter vector θ, the right hand side is dimensionless, though. Inspired by second order methods only using a diagonal Hessian [220], AdaDELTA multiplies the update direction from Eq. (2.357) with an exponentially moving average *over the parameter updates* $\Delta\theta$ to get a covariant descent direction:

$$\theta_t = \theta_{t-1} + \frac{\text{RMS}(\Delta\theta)_{t-1}}{\text{RMS}(g)_t}\delta\theta_t \tag{2.359}$$

Identically to Eq. (2.358), $\text{RMS}(\Delta\theta_t)$ is defined as

$$\text{RMS}(\Delta\theta_t) = \sqrt{E((\Delta\theta)^2)_t + \varepsilon} \tag{2.360}$$

with the same ε as used in Eq. (2.358). $E((\Delta\theta)^2)_t$ is also an exponential moving average with the previous decay constant ρ:

$$E((\Delta\theta)^2)_t = \rho E((\Delta\theta)^2)_{t-1} + (1-\rho)(\Delta\theta_t)^2 \tag{2.361}$$

Hereby $\Delta\theta_t$ is the actual update at iteration t, i.e.

$$\Delta\theta_t = \theta_t - \theta_{t-1} \tag{2.362}$$

Note that in Eq. (2.359) $\text{RMS}(\Delta\theta_{t-1})$ must be used. This may lead to a more robust behavior of the updates in case of large (sudden) gradients, as these are directly damped by the denominator [427]. Furthermore, with ε also in the nominator, progress is ensured even in the case of previously very small updates.

2.7.3 *Deep supervised Neural Networks*

> Since we know that, with a single hidden layer, we can approximate any mapping to arbitrary accuracy we might wonder if there is anything to be gained by using any other network topology, for instance one having several hidden layers. One possibility is that by using extra layers we might find more efficient approximations in the sense of achieving the same level of accuracy with fewer weights and biases in total. Very little is currently known about this issue [34, p.132].

As some theoretical evidence has been accumulated that deep networks may actually be advantageous with respect to their shallow counterparts for statistical efficiency reasons [142, 27, 25], interest in training discriminative deep networks always remained high.

However it *was* generally considered difficult to successfully train such networks. Empirically, they often performed worse than networks with one layer [383]. For some time only two instances of deep networks existed that were *generally* applicable: recurrent LSTM networks and Convolutional Neural Networks. Both cope exceptionally well with the vanishing/exploding gradient (Eq. (2.326)) effect.

An LSTM network [161] is specifically designed to resolve this problem by using an architecture that ensures unhampered flow of gradient information through multiplicative gating mechanisms. It replaces simple hidden units with *LSTM-cells* (Figure 2.13), storing each an analog value in a *memory cell*. An LSTM-cell has input and output *gates* controlling when network input can affect its stored value and when it can influence the network's output. At time t the recurrent computation for an LSTM network with one hidden layer is as follows (using the extensions of *forget* gates [109] and *peephole* connections [110]):

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\boldsymbol{W}_i \mathbf{x} + \mathbf{R}_i \mathbf{h}_{t-1} + \mathbf{p}_i \odot \mathbf{c}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\boldsymbol{W}_f \mathbf{x} + \mathbf{R}_f \mathbf{h}_{t-1} + \mathbf{p}_f \odot \mathbf{c}_{t-1} + \mathbf{b}_f) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot g(\boldsymbol{W}_c \mathbf{x} + \mathbf{R}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{o}_t &= \sigma(\boldsymbol{W}_o \mathbf{x} + \mathbf{R}_o \mathbf{h}_{t-1} + \mathbf{p}_o \odot \mathbf{c}_{t-1} + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot h(\mathbf{c}_t) \\
\mathbf{y}_t &= M(\boldsymbol{W}_M \mathbf{h}_t + \mathbf{b}_M)
\end{aligned} \tag{2.363}$$

Hereby $\mathbf{i}_t$, $\mathbf{f}_t$ and $\mathbf{o}_t$ denote the respective *vector-valued* input, *forget* and output gates, $g(\cdot)$ and $h(\cdot)$ are the input and output activation functions, $\mathbf{p}$ denotes the respective peephole connections (written as vectors, which means that there is no direct leakage of information *between* memory cells) and $\mathbf{c}_t$ denotes the values of the memory cells at time t. Note that I again chose $\mathbf{h}_t$ to denote the hidden state of the network which is distinct from the value of the memory cells.

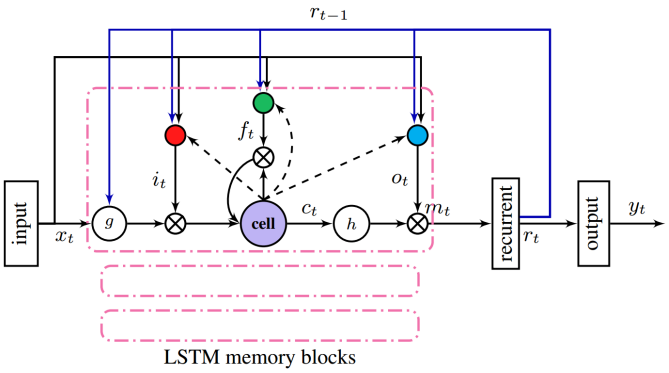


Figure 2.13: An LSTM cell (or an LSTM memory block). This Figure visualizes one element (i.e. a row in each) of the matrix expressions in Eq. (2.363). Instead of $\mathbf{h_k}$ the Figure uses $\mathbf{m_k}$. Additionally, it shows a linear projection step for the hidden state $\mathbf{h_t}$, depicted as *recurrent* and resulting in $\mathbf{r_k}$ which is not given in Eq. (2.363). The peephole connections are depicted by the dashed lines. The Figure is taken from [320].

Convolutional Neural Networks seem not to suffer from the vanishing/exploding gradient effect because sharing parameters happens *within* a layer and leads to summing up of many noisy gradient contributions which mitigates effects of Eq. (2.326).

If the unstable gradient information, caused by the vanishing/exploding gradient problem, is a reason for the difficulties of training deep networks than one approach is to employ optimization methods that are robust in these scenarios, like the previously introduced AdaDELTA approach (Eq. (2.359)). A very different approach is to ensure that vanishing or exploding gradients can't start to happen at all because the weights are *initialized* such that the parameters are overall well-behaving. This general approach became popular over the last years because it happens *before* learning starts and hence is simpler to analyze and understand.

One possible initialization scheme is driven by the objective to maintain approximately the variances of the forward activations and also the variances of the backpropagated gradients at some fixed level for all layers [115]. Obviously this approach depends on the precisely used activation function. For the $\tanh(\cdot)$ activation this *normalized initialization* draws the parameters of the weight-matrices according to the following uniform distribution:

$$\boldsymbol{W}_k \sim \mathcal{U}\left(-\frac{\sqrt{6}}{m+n}, \frac{\sqrt{6}}{m+n}\right), \; k \geqslant 1 \tag{2.364}$$

where $\boldsymbol{W}_k \in \mathbf{R}^{m \times n}$. For a rectifying activation function the optimal initialization is [146]:

$$\boldsymbol{W}_k \sim \mathcal{N}\left(\boldsymbol{W}_k \mid 0, \frac{2}{n}\mathbf{I}\right), \; k \geqslant 1 \tag{2.365}$$

The *orthogonal initialization* method is based on an analysis of the learning dynamics in deep linear networks. It suggests to initialize weight-matrices as the product of two orthonormal weight-matrices that are chosen in such a way that as much information as possible can be propagated from the input layer to the loss function [326]. Assume that $\boldsymbol{W}_k \in \mathbf{R}^{m \times n}$ with $m > n$ without loss of generality. $\boldsymbol{W}_k$ is then initialized as the product of two orthonormal matrices $\mathbf{R}_{k+1} \in \mathbb{R}^{m \times m}$ and $\mathbf{R}_k \in \mathbb{R}^{n \times n}$ as follows:

$$\boldsymbol{W}_k = \mathbf{R}_{k+1}{}^{(n)} \mathbf{R}_k{}^{\mathsf{T}} \tag{2.366}$$

where $\mathbf{R}_{k+1}{}^{(n)}$ is the matrix $\mathbf{R}_{k+1}$ with the last $(m-n)$ columns removed. $\boldsymbol{W}_{k+1}$ is then also initialized according to Eq. (2.366) with $\mathbf{R}_{k+1}{}^{\mathsf{T}}$ chosen as the right orthonormal matrix. The orthonormal matrices can be constructed through an SVD of Gaussian noise matrices of the respective dimension.

A much simpler initialization method ensures that for every hidden unit most incoming parameters are set to 0 (e.g. only 15% of all incoming weights have some random value other than 0). This *sparse initialization scheme* works well for both feed-forward as well as for recurrent Neural Networks [375, 276], independently of the activation function (so, in particular, it seems to work well for the venerable logistic sigmoid function, too). Specifically for recurrent Neural Networks it is necessary to limit the spectral radius of the recurrent weight matrices—this idea originates from work with *echo-state networks* [178, 179, 240, 375]. In the case of a ReLU activation function learning seems to be successful when the recurrent connections are initialized to the identity matrix, or a scaled variant thereof [217, 357].

The most involved initialization schema is based on the idea of constructing the network consecutively in a layer-wise fashion. This can be done in a purely supervised manner [93, 331, 226, 28, 350, 315], in a semi-superwised fashion [297, 417] or in a completely unsupervised way [10, 335, 155, 25]. After the network with the required depth is constructed, the complete stack is *fine-tuned* in a supervised way.

The most widely recognized variant of the latter approach was originally developed to facilitate learning of Deep Belief Networks [155]. However, it was shown empirically that this approach also improves learning of discriminative deep networks [25]. This idea received (and still receives) a lot of attention but it is currently not used very often as other methods mentioned in this section are faster, simpler and more flexible and overall lead to better results with deep networks (both feed-forward and recurrent ones) on a wide range of tasks. Yet,

this approach is attractive because the pretraining phase works completely in an unsupervised way and hence could enable to learn deep networks even when very little supervised data is available.

Pretraining a layer can be done with different kinds of (shallow) unsupervised (or semi-supervised [297, 417]) models that must have a simple *inference* procedure. The RBM (Eq. (2.236)) is one such shallow model. Because the latent units are conditionally independent given the visible units, inference is fast, though training a layer is generally slow. After training one RBM on the training set, the data is passed through the model and the latent representations are now used as the new training set for the next RBM.

A conceptually simpler approach is to use the first layer of a *shallow autoencoder* [318, 10, 342, 141, 89, 39] for stacking. A shallow autoencoder is a Neural Network with one hidden layer that maps the input onto itself, so the loss is defined only in terms of the input $\mathbf{x}$:

$$\ell(\theta, \mathbf{x}) = L(M(N(\theta, \mathbf{x})), \mathbf{x}) + \lambda \Omega(\theta) \tag{2.367}$$

with

$$N(\theta, \mathbf{x})) = \mathbf{b_d} + \mathbf{W_d} \underbrace{f(\mathbf{b_e} + \mathbf{W_e x})}_{\textit{encoder}} \tag{2.368}$$

The *encoder* part of this autoencoder is then used to form a layer in the deep network. In [326] it is suggested to use standard shallow Autoencoders to initialize the layers of deep networks with true nonlinearities, based on their analysis of the dynamics of linear deep networks. In order to ensure that the single layer encoder learns an interesting representation of the input the shallow model is usually somehow constrained. Several possible variations of this approach are presented in the following. A simple constraint is that both encoder and decoder are transposes of each other [28], i.e. they share the weight matrix $\mathbf{W}$:

$$N(\theta, \mathbf{x})) = \mathbf{b_d} + \mathbf{W^T} f(\mathbf{b_e} + \mathbf{W x}) \tag{2.369}$$

Another widely used approach is to inject distortions to the network sampled from some limited random process e.g. through additive or multiplicative Gaussian or Bernoulli noise [346, 5, 267, 292]. The Denoising Autoencoder [407] combines a shared weight-matrix with distortions applied to the input. In the case of additive Gaussian noise (with variance σ^2) this approach can also be interpreted as minimizing not only the reconstruction cost but also the following additional penalty $\Omega(\theta, \mathbf{x})$ [292]:

$$\Omega(\theta, \mathbf{x}) = \sigma^4 \sum_{i,j} \left(\|\boldsymbol{w_j}^T \boldsymbol{w_i}\| f'(\boldsymbol{w_i}^T \mathbf{x}) \right)^2 \tag{2.370}$$

where $\boldsymbol{w_i}^T$ are the rows of the weight matrix $\mathbf{W}$. Note that differently to the usual structural risk minimization framework (Eq. (2.170)) the

penalty now also depends on the input $\mathbf{x}$. This penalty encourages the hidden units to learn orthogonal representations of the input, similar to [215] that modeled overcomplete Independent Component Analysis [193, 68, 23, 162, 170] with a shallow autoencoder.

A different constraint on the shallow Autoencoder may be that the model should learn representations that are insensitive to small changes in the input space [311, 310] . This can be achieved by penalizing the Frobenius norm (Eq. (2.30)) of the Jacobian of the encoder function with respect to the input which is equivalent to using the standard shallow autoencoder (Eq. (2.368)) and inject white Gaussian noise at the hidden representation [292]:

$$\Omega(\theta, \mathbf{x}) = \|\mathbf{J}_{\mathbf{x}}(\mathbf{h})\|_F = \sum_{i=1}^{h} f'(\boldsymbol{w}_i{}^T \mathbf{x}) \|\boldsymbol{w}_i\|^2 \tag{2.371}$$

Finally, a very different constraint on the hidden units enforces *sparsity*. While various forms of *sparse Autoencoders* have been suggested, the k-sparse autoencoder [244] seems to be theoretically the best motivated approach. For the hidden layer $f(\boldsymbol{W}_e \mathbf{x} + \mathbf{b}_e)$ only the k largest values are retained, all other units are set to 0. If f is chosen to be linear then it can be shown that this approach approximates a sparse coding algorithm (Eq. (2.275)) [244].

The best initialization method is worthless in practice if the inputs to the respective layers are in a bad range requiring input normalization, in particular if the input data is heterogeneous or uses different kinds of units. The input data is usually normalized such that the empirical distribution for every input dimension approximates a simple Gaussian centered at 0 and with standard deviation of 1 [224]. This can be done through a z-transformation (Eq. (2.131)) using the empirical mean $\boldsymbol{\mu}_{\text{MLE}}$ and the empirical diagonal variances $\boldsymbol{\sigma}^2_{\text{MLE}}$ over the training set. During evaluation the data from the test set must also be normalized with these parameters. An even better but computationally more demanding approach is to *whiten* (Eq. (2.260)) the input data [418].

Not only the input should be normalized but also all hidden layers With a point-symmetric activation function (and normalized inputs) this is ensured in the case of well-initialized weight-matrices for the beginning of the training process. However, learning is a complicated dynamical process so for hidden units this means that their (empirical) distributions permanently change because the weights leading to the respective layers permanently change (*covariate shift*, [343]). However, in the case of iterative training methods it is prohibitive to keep track of the *exact hidden non-stationary distributions*. A simple and effective method is to constantly normalize the hidden representations based on minibatches (*batch normalization* [171]). In this approach a hidden layer k is normalized through a z-transformation (Eq. (2.131))

using the empirical mean μ_{MLE} and the empirical standard deviation σ_{MLE} of the preactivation a_k[21], computed over a minibatch:

$$\hat{a}_k = \frac{a_k - \mu_{\text{MLE}}}{\sqrt{(\sigma_{\text{MLE}})^2 + \varepsilon}} \tag{2.372}$$

where ε is chosen as a small constant for numerical stability. In order to ensure that the non-linearity $f_k(\cdot)$ can use its complete non-linear range, $\hat{a}_k$ is additionally scaled and shifted before $f_k(\cdot)$ is applied:

$$\tilde{a}_k = \gamma \odot \hat{a}_k + \beta \tag{2.373}$$

γ and β are parameters that are learned, too[22]. It is important that Eq. (2.372) is included in the backpropagation procedure. In particular, the Jacobian $\tilde{J}_{k-1}$ also gets contributions from μ_{MLE} and σ_{MLE}. For convolutional layers, μ_{MLE} and σ_{MLE} are computed per feature map, γ and β are shared *within* a feature map. During evaluation on the test-set, μ^k and σ^k are substituted by the respective population statistics over the training set, computed e.g. by moving averages already during training. Batch normalization performs exceptionally well in experiments [171]. It not only improves the training speed but also the generalization capabilities of trained networks.

Architecturally the flow of information can be improved through so called *skip connections* [34, 401, 294]. Skip connections connect a hidden layer h_k with some later layers h_l, $l > k+1$ or directly with the network output. Clearly, this ensures that the learning signal can be more directly disseminated over all layers. However, it is not clear where and how the skip connections should be placed. The best option would be to learn this, too.

Very recently an adaptive gating mechanism allowed the training of very deep feed-forward networks. It builds paths along which information can flow across many layers without attenuation in an adaptive manner [367, 195]. Differently to fixed skip connections this architecture can build skip connections in a dynamic way depending on the respective input. The idea is inspired by the LSTM cell (Eq. (2.363)) and utilizes multiplicative gating elements. More specifically, for the k-th layer y_k a *transform gate* $T_{k-1}(y_{k-1}, \theta)$ and a *carry gate* $C_{k-1}(y_{k-1}, \theta)$ are introduced and combined with $\hat{y}_k = f_k(W_k y_{k-1} + b_k)$ such that y_k is computed as [367]

$$y_k = \hat{y}_k \odot T_{k-1}(y_{k-1}, \theta) + y_{k-1} \odot C_{k-1}(y_{k-1}, \theta) \tag{2.374}$$

The functional form of both gates is arbitrary but one is inclined to choose again a non-linear affine transformation. The parameters

21 μ and σ^2 are supposed to have a superscript k to indicate that they are related to layer k, but it is left out for readability.

22 Eq. (2.373) can be considered as an additional layer with a diagonal weight matrix $\text{diag}(\gamma)$.

of the respective gates might be shared (i.e. $C_{k-1}(\mathbf{y}_{k-1}, \theta) = 1 - T_{k-1}(\mathbf{y}_{k-1}, \theta)$. The multiplications in Eq. (2.374) are defined in an elementwise way, so some dimensions might be carried over unaltered and others not. This can induce quite complex transformation idioms, in particular if redundant information is contained in $\mathbf{y}_{k-1}$. In the case of shared parameters a good choice for the transform gate is the logistic sigmoid, i.e.

$$T_{k-1}(\mathbf{y}_{k-1}, \theta) = \sigma(\mathbf{W}_{T,k-1}\mathbf{y}_{k-1} + \mathbf{b}_{T,k-1}) \tag{2.375}$$

$\mathbf{b}_{T,k-1}$ is then initialized to a larger negative value [367, 109], ensuring unattenuated gradient flow at the beginning of training.

Given such a large pool of methods that allow the training of networks with many layers, underfitting during training is often no longer a problem. Yet, even with large-scale datasets, very big (and hence rather deep) networks still (may) face the problem of overfitting. The standard approach to tackle this problem is *early stopping* on a validation set [35, 293]: During training, the risk $\ell(\theta, \mathbf{x}, \mathbf{y})$ is constantly monitored on an additional validation set and as soon as its empirical risk clearly starts to deteriorate, training is stopped, even if the empirical training risk would still improve. Furthermore, *weight decay* (Eq. (2.169)) is usually applied, too.

A simple but quite effective method to avoid specifically the *coadaptation* of hidden units in the same layer is *dropout* [365]. During training units in hidden layers are randomly set to 0 (*dropped out*), which happens according to some fixed probability, depending on layer k. A hidden layer $\mathbf{y}_k$ is then computed according to

$$\mathbf{y}_k = \mathbf{B} f_k(\mathbf{W}_k \mathbf{y}_{k-1} + \mathbf{b}_k) \tag{2.376}$$

where $\mathbf{B}$ is a binary diagonal matrix, with $\mathbf{B}_i \sim \text{Bern}(\mathbf{B}_i \mid p_k)$. That is, at layer k a hidden unit gets dropped out with probability $1 - p_k$ and hence during evaluation the weight-matrix $\mathbf{W}_k$ gets scaled by p_k. Dropout works specifically well with ReLU and Max-Out activation functions[23]. Formulated as a chained matrix multiplication it is seamlessly integrated into the backpropagation framework, though for the optimization procedures more extreme settings seem to be preferable as well as additional norm constraints on the weight matrices [364]. Interestingly, with batch normalization (Eq. (2.372)) dropout seems to be less beneficial or necessary [171].

The dropout idea can also be applied directly to the weight matrices (*dropconnect*, [411]) and in theory it should be more flexible than dropout, as the overall noise pattern acting on the network has

23 Dropout is inspired by some insights from genetics [234]. From this point of view it is obvious that it should work very well with Max-Out, as genes usually co-act in groups. However, recent research shows that Dropout is better interpreted as a variational inference method[104]. I do not know how this aspect can explain the improvement seen with Max-Out

much less structure. Yet, its efficient implementation is much more challenging (several noisy versions of the parameter vector θ must be kept in memory for every minibatch) and the reported results [411] do not seem to justify this additional overhead from a practical point of view.

Finally, for standard recurrent Neural Networks it was recently shown that training instabilities due to vanishing or exploding gradients can be handled very successfully by simply clipping gradients [260, 287].

The previous approaches are very appealing from a theoretical point of view. However, the following two aspects had a much more resounding effect for the successful application of deep Neural Networks to tasks that were traditionally dominated by alternative methods:

- Faster computers: In order to avoid getting stuck in spurious minima it is often enough to train longer (while avoiding overfitting). This simple insight however only became economically viable (with respect to real world computation time) over the last 10 years through the advent of powerful graphics processing units (GPGPU—General Purpose Graphical Processing Units). Their optimized atomic operations are perfectly aligned for realizing efficient matrix-matrix operations found in the elementary affine transformations (Eq. (2.294)) and the backpropagated Jacobians (Eq. (2.314)) [281]. GPUs not only allow tackling large-scale models but prove specifically helpful with *abundant labeled training data*.

- Large datasets (e.g. ImageNet [80]): Ubiquitous powerful and cheap computational units not only lead to a large amount of available data, the dimensions of the gathered data samples also increase (which means, in the case of classification, the number of classes). This increase of information density helps avoiding the problematic overfitting trend that Neural Networks sometimes show due to their high flexibility. Not only do more classes mean more bits per training sample (the number of bits a training samples conveys to a learning algorithm is about $O(\log|c|)$, where $|c|$ is the number of classes). Large datasets simple reduce the risk of stopping the training in spurious minima by mistake which manifests itself in overfitting on the test set. Large datasets can be made even bigger by augmenting them with transformed input samples. These are changed in such a way that the associated target information stays invariant [63, 209, 84]. Datasets that are augmented in this way allow the model to detect the important invariances it otherwise would have a very hard time to identify.

> Large, augmented datasets are only helpful for training Neural Networks if they can be tackled computationally. That is, the wall-clock time for training must remain within reasonable limits. GPUs are the right tool to solve this kind of problem.

The idea of artificially augmenting the dataset with *valid* is extremely powerful and at the same time so simple—but it is not possible for those cases where humans themselves don't know the underlying invariances. But these instances are be the premier cases to apply Machine Learning in the first place!

One possible approach to tackle this problem is to generate so-called *adversarial* inputs that are close to a given input but have a lower cost compared to this reference sample [123]. Such adversarial inputs can be generated by relying on an adversarial epsilon neighborhood around an input sample, defined through the first derivative of the cost function with respect to the input. *Increasing* the cost for samples in the (dynamically determined) adversarial neighborhood embeds the labelled training samples in a loss surface that is mostly flat around the embedded training cases, ensuring a much more stable generation of the resulting model (also compare to *Flat Minima* [160]).

On a more abstract level this method (decrease a loss for some input space, increase the loss for some (other) input space) resembles some well-known forms of unsupervised learning, as detailed for directed and undirected graphical models (see Section 2.6). An under-appreciated form of unsupervised learning for Neural Networks (*contrastive backpropagation* [156]) produces adversarial inputs using Hybrid Monte Carlo sampling.

Therefore, the final part of this section covers therefore ways to train deep Neural Networks in an unsupervised fashion. If done right, only very little labeled training data can already produce powerful discriminative models, as recently shown for the simple MNIST dataset [303].

This compact review covered those aspects only that *subjectively* form the basics of successful approaches to training deep supervised Neural Networks. It can't delve into the ever increasing set of problems successfully tackled[24] by either recurrent (LSTM-based) Neural Networks or Convolutional Networks[25]. A small excerpt of these encompasses major successes in unconstrained handwriting recognition [128], speech recognition [129, 320], machine translation [373, 9], image captioning [409, 197], object recognition [59, 60, 209, 304, 350, 380], medical image processing [62], image segmentation [94,

24 *Successfully* hereby means that deep Neural Networks are performing better than or comparable to the state-of-the-art.

25 Many of the successful applications of CNNs use the ReLU non-linearity. It would be interesting to see whether this activation function has a positive effect on the gradient scalings compared between different layers.

369] and pose estimation [393, 55, 421, 280]. It also can't present the overwhelming number of new algorithmic approaches that seemingly appear on a weekly basis at the moment. Exciting developments hereby include fast approximations of Dropout [413, 19, 18], Bayesian Neural Networks [206], parameter compression [81, 413, 176], attention-driven models [9, 368] and completely new approaches to train deeply nested systems [50].

2.7.4 *Unsupervised Deep Neural Networks*

If one considers a Neural Network as a computational architecture with directed edges the connection to directed graphical models is obvious and directly leads to one possibility for using Neural Networks for unsupervised learning: the training data is considered the *deterministic output* of the Neural Network, generated by a set of *unknown* hidden units. Learning the parameters can be done with gradient descent (relying for example on the backpropagation algorithm). This approach is only reasonable if inference of the hidden activations is efficient.

On the other hand if the data is considered the input to a Neural Network, a score function must be defined that represents the unsupervised modelling task for the data at hand. In this case learning also happens by gradient descent using the gradient of the score function with respect to the input and Hybrid Monte Carlo (HMC) to produce adversarial input data ([156], also see the discussion in the previous paragraph). However, identifying a valid score function is difficult (overall it represents a large part of the unsupervised learning challenge) and HMC sampling is slow in general.

A simplification for this type of unsupervised Neural Network learning is to define the score for some given sample as its *reconstruction cost*. That is, the Neural Network is tasked to reproduce its input. This *Autoencoder* then resembles a self-supervised problem and training its parameters works as described in Section 2.7.3.

An Autoencoder with one hidden layer (and linear output units and a sum-of-squares objective) is actually no more powerful than PCA (irrespective of the non-linearity of the hidden layer) [39]. So for more expressive models *deep* Autoencoders are necessary. Training such models seems to resemble the standard supervised problem detailed in the previous subsection, but in fact it is not. With a proper supervised problem the targets already contain the important information, having abstracted away distracting aspects of the data. However, *identifying the defining content of the data is the task of an autoencoder*.

So while for unsupervised learning unlimited training data is available, it is still a huge challenge to obtain good models with deep Autoencoders—this is clearly different to supervised deep models.

Therefore *algorithmic* techniques like those mentioned in Section 2.7.3 are much more important for training deep Autoencoders.

Similar to the supervised setup one possible way to construct deep Autoencoders is to stack single layer models on top of each other. The decoder of the deep Autoencoder is often just the encoder transposed (i.e. the weight matrices are transposed and stacked in the inverse order). Single layer models like RBMs [321] or Denoising Autoencoder [408] are widely used for initializing deep Autoencoders which are then fine-tuned on the unlabelled dataset using the reconstruction cost. The output of the encoder (which can be defined using any of the hidden layers in the deep model) is then used as the representation for the respective input data. However, from a more theoretical point of view these approaches lack a proper probabilistic formulation (and are also difficult to generalize to recurrent models). For example while it is possible to associate the cost function of the Denoising Autoencoder with a probabilistic criterion by a variational lower bound, this lower bound only is related to the log-probability of the corrupted data [30].

It is therefore instructive to consider other methods that *induce* deep Autoencoders. Generative directed graphical models can be used to define principled methods that result in deep Autoencoder models. In the following paragraphs, two such approaches are presented and extended in novel ways. Both rely on the fact that a complicated inference procedure associated with the generative model is either directly or indirectly approximated with deep Neural Networks.

Mapping inference algorithms into deep Autoencoders

In Section 2.6 the probabilistic formulation for sparse coding is introduced:

$$\begin{aligned} p(\mathbf{h}) &\propto \exp(-\lambda \|\mathbf{h}\|_2) \\ p(\mathbf{x} \mid \mathbf{h}, \theta) &= \mathcal{N}(\mathbf{x} \mid \boldsymbol{W}\mathbf{h}, \mathbf{I}) \end{aligned} \tag{2.377}$$

A standard approach to fit the model to a dataset is the EM algorithm. While the M-step is straightforward (estimate $\boldsymbol{W}$ given $\mathbf{x}$ and the respective $\mathbf{h}$, using standard linear regression) the E-step for inferring $\mathbf{h}$ for a given $\mathbf{x}$ and $\boldsymbol{W}$ is complicated due to the explaining-away phenomenon in directed graphical models.

A popular iterative algorithm for the E-step is ISTA [75] which iterates the following recursion until convergence in order to determine the sparse approximation for some $\mathbf{x}$:

$$\mathbf{h}_{t+1} = \tau_\alpha \left(\mathbf{h}_t + \frac{1}{L} \boldsymbol{W}^\top (\mathbf{x} - \boldsymbol{W}\mathbf{h}_t) \right). \tag{2.378}$$

Hereby, $\tau(\cdot)$ is a shrinkage operator defined as

$$\tau_\alpha(\mathbf{x}) = \text{sgn}(\mathbf{x})(\mathbf{x} - \alpha)_+, \quad \alpha = \frac{\lambda}{L} \tag{2.379}$$

with $\text{sgn}(\cdot)$ the elementwise sign function, $(x)_+ \equiv \max(0, x)$ and L is lower bounded by the largest eigenvalue of $\boldsymbol{W}^T\boldsymbol{W}$. Eq. (2.378) can be slightly rewritten [130] as

$$\mathbf{h}_{t+1} = \tau_\alpha\left(\frac{1}{L}\boldsymbol{W}^T\mathbf{x} + \boldsymbol{S}\mathbf{h}_t\right). \tag{2.380}$$

$\boldsymbol{W}^T$ is also denoted the *filter* matrix and $\boldsymbol{S} \equiv (\mathbf{I} - \frac{1}{L}\boldsymbol{W}^T\boldsymbol{W})$ is denoted the *inhibition* matrix. Eq. (2.380) is a recursive relation of an elementwise non-linearity applied to a set of matrix-vector operations and therefore resembles a recurrent Neural Network with the input *shared over time*. Conversely, it can also be interpreted as a static deep network where the layers share the same weights, the input is routed to every layer and the network has a dynamic depth.

With a *fixed* number of layers, Eq. (2.380) can be used to *induce* a deep Neural Network. The results of the ISTA algorithm for a converged sparse coding model act as target data for the network which therefore resembles a fast approximation of ISTA (learned ISTA, *LISTA* [130]).

At first this might simply seem to be some variant of an explicitly *truncated* form of ISTA. However, the involved parameters ($\boldsymbol{W}$, $\boldsymbol{S}$) are learned and no longer tied to each other as in the case of ISTA, see Eq. (2.380). Additionally, after some initial training period the parameters shared by all layers can be *untied* and optimized individually on a per-layer basis. This allows LISTA to outperform ISTA and its variants by a large margin with respect to computational aspects [130] and still perform competitively with respect to the resulting latent representations.

If the last layer of this model is itself projected back into the input space, an *Autoencoder model is induced* [360, 362]. Differently to standard deep Autoencoders, the last hidden layer of such a model forms the latent (sparse) representation. Also, differently from the standard deep Autoencoder model the network has the same number of units per layer. This might seem to be restrictive at first but the connection to sparse coding allows a straightforward training of an overcomplete latent representation.

In this case a *process* (the iterative algorithm from Eq. (2.380)) induces a deep (autoencoding) network structure and provides guidance (the (initially) shared weight structure) for the difficult optimization procedure. This kind of support to learn a deep network is very different from simply enhancing the dataset through viable input samples.

Sparse coding can be interpreted as a model that combines the advantages of a purely local coding approach with a distributed rep-

resentation [98]. A different approach to sparse coding is to aim explicitly for a set of features that are statistically uncorrelated.

One realization for this idea is Predictability Minimization (PM) [330]: The model for learning the latent representation (e.g. a shallow Autoencoder) is paired with a predictor network that tries to infer one element of a latent representation given all the other elements. Training then happens in an alternating way: In one phase the unsupervised learner finds representations that minimize the reconstruction error and at the same time maximize the prediction error of the predictor network. In the other phase the predictor network minimizes its prediction error.

Building unsupervised models with multiple layers of nonlinearities in an iterative way [337] with this approach is difficult (see Appendix B): The greedily trained single layers usually get stuck at bad locally minima, or, more likely given the alternating optimization approach, at saddle points [76].

An alternative approach to PM is to use explicit inhibitory weights instead of a predictor network [98]. Because of the inhibitory weights and the involved non-linearities (in the form of activation functions for the hidden units), such a model must also be simulated by an iterative method. More specifically, a single-layer forward model with inhibitory weights between the latent units can be simulated by the following differential equation:

$$\frac{d\mathbf{h}}{dt} = f(\mathbf{W}\mathbf{x} + \mathbf{Q}\mathbf{h} + \mathbf{b}) - \mathbf{h} \tag{2.381}$$

f is some arbitrary non-linearity, $\mathbf{W}$ is the filter matrix and $\mathbf{Q}$ is the inhibitory matrix that has negative off-diagonal elements and a diagonal that is fixed to 0. For stability reasons $\mathbf{Q}$ is also normalized to have rows of length 1. For a symmetric $\mathbf{Q}$ it is guaranteed that the above equation settles in a stable state [165]. The underlying Neural Network is shown in Figure 2.14.

Training the parameters ($\mathbf{W}$, $\mathbf{Q}$, $\mathbf{b}$) is done through hebbian learning for $\mathbf{W}$ and $\mathbf{b}$ and anti-hebbian learning for $\mathbf{Q}$, using inputs $\mathbf{x}$ and stable outputs $\mathbf{h}$ from the differential equation.

To reach a stable output pattern $\mathbf{y}$, Eq. (2.381) must be simulated numerically. The Euler method [147] is the easiest numerical simulation approach to differential equations. For Eq. (2.381) one iteration is as follows:

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \alpha\frac{d\mathbf{h}_t}{dt} = (1-\alpha)\mathbf{h}_t + \alpha f(\mathbf{W}\mathbf{x} + \mathbf{Q}\mathbf{h}_t + \mathbf{b}), \tag{2.382}$$

where α is a fixed stepsize. For a given $\mathbf{x}$ a stable output $\mathbf{y}$ is computed by iterating Euler steps until convergence. Similar to the idea of unrolling the inference algorithm for sparse coding as described

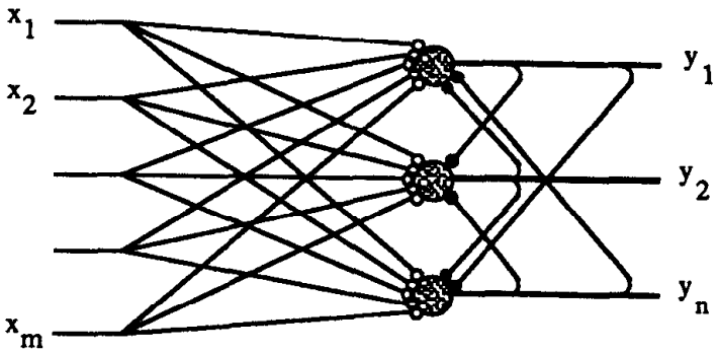


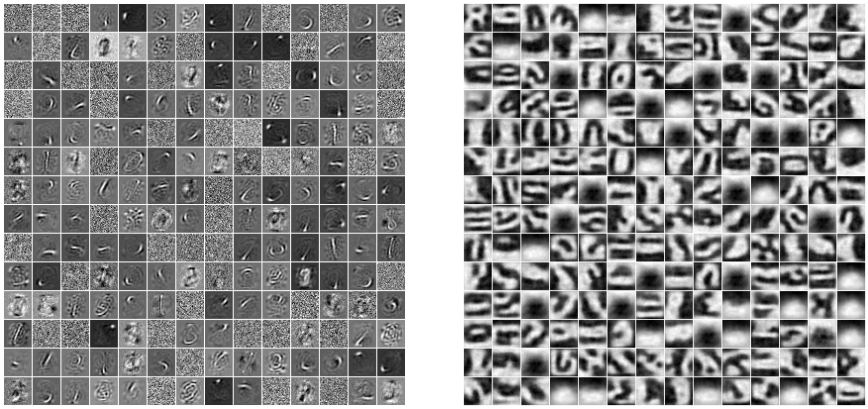
Figure 2.14: The architecture of the model that gives rise to the differential equation in Eq. (2.381). Inhibitory connections are depicted by filled circles. Figure is taken from [98]. Best viewed electronically.

previously for LISTA, the iterative Euler steps can also be used to construct a deep network with multiple layers of nonlinearities. The depth will be fixed in order to achieve a computationally efficient approximation of the underlying mathematical model.

The parameters ($\boldsymbol{W}$, $\mathbf{Q}$ and $\mathbf{b}$) and the stepsize α are shared by all layers, and can gradually be unshared as training progresses. The necessary training signals can be either obtained by the correctly simulated differential equation or again by a reconstruction criterion. The latter is chosen in the subsequent discussion. Note that in the initial model, no aspects of autoencoding where present! Additional objectives like minimizing the correlation between latent units, minimizing some sparsity penalty on the latent units or minimizing a supervised criterion [360] are also possible. The resulting autoencoder can be further regularized if the reconstruction matrix is simply the transpose of $\boldsymbol{W}$.

It has not been possible to evaluate the proposed model (a *Hobbesian network* [158]) in a detailed manner yet, due to time constraints. However a short empirical analysis of the learned filters was conducted. Figure 2.15(a) shows filters (that is, rows of $\boldsymbol{W}$) when trained on images of the MNIST dataset. The figure depicts three different types of filters: (i) the typical local strokes, (ii) several global patterns and (iii) quite a high number of random filters. The local filters seem to be unique: None of the filters of this type appear several times—this is due to the fact that the model tries to decorrelate the features. The latter also explains the existence of the high number of random filters, which is the easiest way to decorrelate features. Without the reconstruction criterion, setting all filters to white noise would be a viable solution. That is, the reconstruction criterion can be interpreted as a regularizing factor for the original unsupervised model.

An interesting observation can be made when the model is applied to natural gray-scale images from the CIFAR10 dataset [207].



(a) (b)

Figure 2.15: Filters and reconstruction filters learned by Hobbesian Networks. (a) Filters ($\boldsymbol{W}$) of a fully connected Hobbesian Network with 5 layers, 196 hidden units and tanh non-linearity, trained on MNIST [222]. The decorrelation matrix $\mathbf{Q}$ leads to many unused filters and thus sparse codes. (b) *Reconstruction* filters from a Hobbesian Network (same configuration as in (a)) trained on gray-scale images from CIFAR10. $\boldsymbol{W}$ shows the same type of filters but with a very high degree of white noise added, so it is less useful for visualization purposes. Figure is best viewed electronically.

Figure 2.15(b) shows the reconstruction filters which are obtained on this dataset. Compared to other unsupervised models, the observed structures are very different[26] and also rather difficult to explain. One might think some of these are decomposing the image into different parts (e.g. foreground and background), but there is no consistent way to achieve a valid interpretation. It is apparent that more analysis is necessary, for example even without evaluating the learned features on some additional tasks (e.g. supervised classification problems) it is interesting to investigate how the filters change when the input set is changed, e.g. in the case of CIFAR10 through augmenting the dataset with typical geometric transformations.

VARIATIONS. As a very general mathematical model, Eq. (2.381) can be implemented in different ways[27]. In the following several variations or extensions that Eq. (2.381) allows are presented.

26 The images were *not* whitened. Clearly, this is an important preprocessing step if one wants to achieve competitive performance to other models. Without whitening the model will get stuck with resolving only second-order correlations.

27 Considering the fact that many important models are described by differential equations, it could be a worthwhile investigation to see how existing models from different domains can be used to induce deep Neural Networks as fast approximators. This is especially useful for the case of time-consuming simulations.

- For a d-dimensional latent variable $\mathbf{h}$, d different learning rates can be used, i.e.

$$\mathbf{h}_{t+1} = (1 - \boldsymbol{\alpha})\mathbf{h}_t + \boldsymbol{\alpha} f(\mathbf{W}\mathbf{x} + \mathbf{Q}\mathbf{h}_t + \mathbf{b}), \quad \boldsymbol{\alpha} \in \mathbf{R}^d \tag{2.383}$$

  More generally the stepsize $\boldsymbol{\alpha}$ can be determined adaptively at every layer t:

$$\boldsymbol{\alpha}_t = g(\mathbf{U}\mathbf{x} + \mathbf{V}\mathbf{h}_t) \tag{2.384}$$

  The parameters ($\mathbf{U}$ and $\mathbf{V}$) might be shared over all layers to avoid overfitting. With an adaptive learning rate per layer the resulting deep architecture becomes similar to the previously introduced gating mechanism for deep supervised networks (see Eq. (2.374)) [367, 195].

- A very different type of variation can be achieved by using a more powerful numerical method to simulate the differential equation Eq. (2.381). The basic form of a *Runge-Kutta* method is the so called *midpoint* method [147]:

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \alpha \frac{d\tilde{\mathbf{h}}_t}{dt} \tag{2.385}$$

  with

$$\tilde{\mathbf{h}}_t = \mathbf{h}_t + \frac{\alpha}{2}\frac{d\mathbf{h}_t}{dt} \tag{2.386}$$

  Applied to Eq. (2.381) this results in the following update rule:

$$\begin{aligned} \tilde{\mathbf{h}}_t &= (1 - \frac{\alpha}{2})\mathbf{h}_t + \frac{\alpha}{2} f(\mathbf{W}\mathbf{x} + \mathbf{Q}\mathbf{h}_t + \mathbf{b}), \\ \mathbf{h}_{t+1} &= \mathbf{h}_t - \alpha\tilde{\mathbf{h}}_t + \alpha f(\mathbf{W}\mathbf{x} + \mathbf{Q}\tilde{\mathbf{h}}_t + \mathbf{b}) \end{aligned} \tag{2.387}$$

  The network architecture is thus extended by one additional layer $\tilde{\mathbf{h}}_t$ per hidden layer $\mathbf{h}_t$. The resulting structure can be made more interesting if the idea from Eq. (2.384) is also integrated. This means that different network topologies are implied depending on the numerical method that is used to simulate Eq. (2.381).

- The described approach can also be used to induce a deep *convolutional* autoencoder. The matrix $\mathbf{Q}$ then resembles a $1 \times 1 \times f$ convolution and implements competition between the f feature maps [231].

- $\mathbf{Q}$ can have several levels of structure, identifying groups of units that should not compete with each other [131].

RELATED WORK. The proposed Hobbesian Network model first needs to be validated in more detail on several typical benchmarks in order to assess its feasibility as an unsupervised learning algorithm. Overall one might argue that it is only a small variation to LISTA [130], because for LISTA the matrix $\mathbf{S}$ (see Eq. (2.380)) is itself setup as an inhibitory matrix. However the fact that different deep architectures can be induced (see Eq. (2.387)) clearly shows that the presented idea is more general. In fact, one might argue that LISTA can be interpreted as a special case of a Hobbesian network.

Given only the algorithmic idea, there are some connections to other published approaches:

- Setting the stepsize α to 1, allowing an arbitrary matrix $\mathbf{Q}$ and selecting the ReLU activation function as the non-linearity f recovers the autoencoding part of the Discriminative Recurrent Sparse Auto-Encoding model [314].

- There is a surprising connection to a recently proposed recurrent network architecture, the Gated Recurrent Unit (GRU) [56]. If in Eq. (2.383) the input $\mathbf{x}$ is different at every time-step t and the learning rate $\boldsymbol{\alpha}_t$ is computed adaptively as proposed above, i.e.

$$\begin{aligned} \mathbf{h}_{t+1} &= (1 - \boldsymbol{\alpha}_t)\mathbf{h}_t + \boldsymbol{\alpha} f(\mathbf{W}\mathbf{x}_t + \mathbf{Q}\mathbf{h}_t + \mathbf{b}), \\ \boldsymbol{\alpha}_t &= g(\mathbf{U}\mathbf{x}_t + \mathbf{V}\mathbf{h}_t) \end{aligned} \tag{2.388}$$

  then the *functional* form of the GRU is recovered (the *reset* gate of a GRU unit is not explicitly modeled, but it can be subsumed into the $\mathbf{Q}$ operator that can be defined to be time-dependent). That is, GRU resembles some differential equation (with additional aspects mixed in) simulated by the Euler method. Conversely various instantiations of the GRU exist, depending on the concrete numerical method used to simulate the dynamics (e.g. see Eq. (2.387)).

- Inhibitory weights were used with an ISTA-like algorithm before [131], combining a generative model with an additional penalty cost derived from the correlation between the latent units. Several new algorithmic developments would also allow to unroll this model into a feed-forward model [361] which could inspire additional variations to the basic Hobbesian network.

Future work will not only investigate the feasibility of the basic Hobbesian Network model in more detail but also its possible variations and connections to related work. The overall goal is to define a rich class of models that allow efficient approximation of models with explicit lateral inhibition, for both static as well as dynamic data (i.e. recurrent networks).

Learning stochastic inverses

With sparse coding the inference algorithm (the inverse of the model) is readily available and can be used to induce a much more efficient approximation using a deep (autoencoding) network. However, for arbitrary directed graphical models such a tractable inference algorithm does not exist. Instead a very successful approach is to find a tractable lower bound on the *marginal likelihood* of the data and use this bound to identify an efficient (approximate) inference method. Lower bounding the marginal likelihood was already introduced with the EM algorithm (see 2.265, the Evidence Lower BOund (ELBO)).

Consider a generative model $p(\mathbf{x} \mid \theta)$ with parameters θ and latent variables $\mathbf{h}$. Using an approximate posterior distribution $q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi})$ for the latent variables $\mathbf{h}$, the ELBO can be written as (discrete latent variables are assumed without loss of generality, mostly because sums can be written more compactly than integrals):

$$\log p(\mathbf{x} \mid \theta) \geqslant \sum_{\mathbf{h}} q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi}) \log \frac{p(\mathbf{x}, \mathbf{h} \mid \theta)}{q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi})} := \mathcal{F}(\mathbf{x}, \theta, \boldsymbol{\phi}) \tag{2.389}$$

The lower bound $\mathcal{F}(\mathbf{x}, \theta, \boldsymbol{\phi})$ can be expressed in alternative, but equivalent ways, each giving rise to a different approach to tackle the inference problem. For example, if the approximate posterior distribution of the latent variables equals the true posterior then the ELBO is tight (see Eq. (2.390)):

$$\mathcal{F}(\mathbf{x}, \theta, \boldsymbol{\phi}) = \log p(\mathbf{x} \mid \theta) - \mathcal{KL}[q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi}) \| p(\mathbf{h} \mid \mathbf{x}, \theta)], \tag{2.390}$$

where the second term is the Kullback-Leibler divergence between the approximate and true posterior distribution. This way of formulating the bound leads to the E-step in the EM algorithm. The E-step is tractable in the case of a simple posterior $p(\mathbf{h} \mid \mathbf{x}, \theta)$. However, this formulation is not helpful for intractable posteriors. The M-step can be derived with a different formulation:

$$\mathcal{F}(\mathbf{x}, \theta, \boldsymbol{\phi}) = \mathbb{E}_{q(\mathbf{h}|\mathbf{x}, \boldsymbol{\phi})}[p(\mathbf{x}, \mathbf{h} \mid \theta)] + \mathcal{H}[q], \tag{2.391}$$

where the second term denotes the entropy of $q(\cdot)$. The central problem here is to compute the expectation with respect to the (approximate) posterior. If this is possible, then optimizing Eq. (2.391) with respect to θ is straightforward. However, even in the case of a non-convex problem for θ gradient based approaches can be applied (*generalized EM*)—the gradient of the above expression with respect to θ can be computed using the rule for *pathwise derivatives* (see Eq. (2.208), assuming a reasonable well-behaving complete log-likelihood):

$$\nabla_\theta \mathbb{E}_{q(\mathbf{h}|\mathbf{x}, \boldsymbol{\phi})}[p(\mathbf{x}, \mathbf{h} \mid \theta)] = \mathbb{E}_{q(\mathbf{h}|\mathbf{x}, \boldsymbol{\phi})}[\nabla_\theta p(\mathbf{x}, \mathbf{h} \mid \theta)] \tag{2.392}$$

Computing the expectation with respect to the posterior distribution can be done with the Monte Carlo principle (see Section 2.5) which

needs (unbiased) samples. One possible way to generate valid samples is to actively forego the approximate posterior distribution $q(\cdot)$ and produce samples from the true posterior $p(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\theta})$. Without any additional knowledge of the true posterior distribution this is only possible with MCMC methods (e.g. using the gradient of the posterior with respect to $\mathbf{h}$ and employing Hybrid Monte Carlo). While mathematically the best approach (because the produced samples are unbiased samples from the true posterior distribution), it is computationally unfeasible for large datasets.

An alternative approach is to choose a *local variational* approximation $q(\cdot)$. It usually allows to solve the resulting pathwise derivative integral (Eq. (2.392)) in an analytic form [190, 35]. However local variational approximations imply an optimization process for *every sample in the dataset* and therefore are also not applicable to large amounts of data.

A *global variational* approximation model $q(\cdot)$ circumvents this problem, because only one set of parameters $\boldsymbol{\phi}$ is used for the approximate posterior [153, 79, 262, 200, 307], *amortizing the inference cost over all data samples* [306]. The main obstacle here is estimating the parameter $\boldsymbol{\phi}$. A straightforward solution is to rely on the score function estimator Eq. (2.205) for Eq. (2.391). However this estimator usually is associated with a high degree of variance [164].

Using yet another formulation for $\mathcal{F}(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi})$ eventually allows the application of a pathwise derivative estimator for $\boldsymbol{\phi}$ (at least in the case of continuous latent variables):

$$\begin{aligned}\mathcal{F}(\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi}) = &\mathbb{E}_{q(\mathbf{h}|\mathbf{x},\boldsymbol{\phi})}\left[\log p(\mathbf{x} \mid \mathbf{h}, \boldsymbol{\theta})\right] \\ &- \mathcal{KL}[q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi}) \| p(\mathbf{h} \mid \boldsymbol{\theta})]\end{aligned} \tag{2.393}$$

Similar to Eq. (2.392) $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{q(\mathbf{h}|\mathbf{x},\boldsymbol{\phi})}\left[\log p(\mathbf{x} \mid \mathbf{h}, \boldsymbol{\theta})\right]$ can be readily evaluated with a pathwise derivative. For $\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q(\mathbf{h}|\mathbf{x},\boldsymbol{\phi})}\left[\log p(\mathbf{x} \mid \mathbf{h}, \boldsymbol{\theta})\right]$ the high-variance score estimator is always applicable:

$$\begin{aligned}\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q(\mathbf{h}|\mathbf{x},\boldsymbol{\phi})}\left[\log p(\mathbf{x} \mid \mathbf{h}, \boldsymbol{\theta})\right] = &\mathbb{E}_{q(\mathbf{h}|\mathbf{x},\boldsymbol{\phi})}\left[\log p(\mathbf{x} \mid \mathbf{h}, \boldsymbol{\theta})\right. \\ &\left. \times \left(\nabla_{\boldsymbol{\phi}} \log q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi})\right)\right]\end{aligned} \tag{2.394}$$

If $\mathbf{h}$ is continuous then an estimator with much less variance can be computed using a pathwise derivative. In order to apply Eq. (2.208), a random variable $\mathbf{z}$ (with distribution $r(\mathbf{z})$) and a mapping $f(\mathbf{z}, \boldsymbol{\phi})$ must exist, such that for $\tilde{\mathbf{h}} = f(\mathbf{z}, \boldsymbol{\phi})$ it holds that:

$$\tilde{\mathbf{h}} \sim q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi}) \qquad \text{for } \mathbf{z} \sim r(\mathbf{z}) \tag{2.395}$$

In this case

$$\nabla_{\boldsymbol{\phi}} \mathbb{E}_{q(\mathbf{h}|\mathbf{x},\boldsymbol{\phi})}\left[\log p(\mathbf{x} \mid \mathbf{h}, \boldsymbol{\theta})\right] = \mathbb{E}_{r(\mathbf{z})}\left[\nabla_{\boldsymbol{\phi}} \log p(\mathbf{x} \mid f(\mathbf{z}, \boldsymbol{\phi}), \boldsymbol{\theta})\right] \tag{2.396}$$

For some distributions the Kullback-Leibler divergence between the approximate posterior distribution of $\mathbf{h}$ and its prior (the second part of Eq. (2.393)) can be computed analytically. In this case

$\nabla_\phi \mathcal{KL}[q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi}) \| p(\mathbf{h} \mid \theta)]$ can also be derived analytically, otherwise either score function estimates or pathwise derivative estimates are applicable, too.

If the global variational approximation $q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi})$ is parameterized by a deep Neural Network [79, 307, 200] then Eq. (2.393) describes a deep Autoencoder model[28]: The first term is the reconstruction cost and the second term acts as a regularizer. It is important to point out that this kind of regularized Autoencoder is very different from the standard approach of combining the reconstruction cost haphazardly with some arbitrary regularization method: Eq. (2.393) is a lower bound to the log-likelihood of the data under the model to be trained and is therefore meaningful from a probabilistic perspective. Estimating the true log-likelihood of some sample $\mathbf{x}$ under the trained model can therefore also be accomplished through principled manners, using importance sampling (Eq. (2.177)) [307]:

$$\begin{aligned} p(\mathbf{x} \mid \theta) &= \int \frac{p(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi})}{p(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi})} p(\mathbf{x} \mid \mathbf{h}, \theta) p(\mathbf{h} \mid \theta) d\mathbf{h} \\ &\approx \frac{1}{S} \sum_{i=1}^{S} \frac{p(\mathbf{x} \mid \mathbf{h}_i, \theta) p(\mathbf{h}_i \mid \theta)}{q(\mathbf{h}_i | \mathbf{x}, \boldsymbol{\phi})}, \quad \mathbf{h}_i \sim q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi}) \end{aligned} \tag{2.397}$$

Training this *variational Autoencoder (vAE)* [200] happens in the standard way with (minibatched) stochastic gradient descent methods, relying on the previously described gradient computations through *stochastic backpropagation* [307]. The expectation over these gradients with respect to the posterior distribution relies on the Monte Carlo principle, so the overall approach is sometimes described as a *doubly stochastic estimation* [390].

The most basic example for a vAE assumes Gaussian latent variables, with a diagonal prior $p(\mathbf{h} \mid \theta) = \mathcal{N}(\mathbf{h} \mid \mathbf{0}, \mathbf{I})$, $\mathbf{h} \in \mathbb{R}^d$. The inference model is also considered a diagonal Gaussian. More specifically, $q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi})$ is conditionally Gaussian, that is $q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi}) = \mathcal{N}(\mathbf{h} \mid \boldsymbol{\mu}(\mathbf{x}, \boldsymbol{\phi}), \boldsymbol{\sigma}(\mathbf{x}, \boldsymbol{\phi}))$.

In order to apply the pathwise derivative estimator a (valid!) reparameterization [200] of $\mathbf{h}$ is necessary. In the case of a Gaussian this can be the inverse of the z-transformation:

$$\tilde{\mathbf{h}} = \boldsymbol{\mu}(\mathbf{x}, \boldsymbol{\phi}) + \boldsymbol{\varepsilon} \odot \boldsymbol{\sigma}(\mathbf{x}, \boldsymbol{\phi}), \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{\varepsilon} \mid \mathbf{0}, \mathbf{I}) \tag{2.398}$$

With a diagonal Gaussian for both the prior and the inference model, the Kullback-Leibler divergence from Eq. (2.393) is also analytically tractable:

$$\mathcal{KL}[q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi}) \| p(\mathbf{h} \mid \theta)] = -\frac{1}{2} \sum_{i=1}^{d} (1 + 2 \log \sigma_i - \mu_i^2 - \sigma_i^2) \tag{2.399}$$

28 In the case of a $q(\cdot)$ being a Neural Network the variational approximation is also called a *recognition network*.

Hence, $\nabla_\phi \mathcal{KL}[q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi}) \| p(\mathbf{h} \mid \theta)]$ is easy to compute, relying on standard backpropagation.

The conditional log-likelihood model $p(\mathbf{x} \mid \mathbf{h}, \theta)$ does not need to be specified in a detailed manner—gradients with respect to θ can be computed using standard backpropagation, so any type of (differential) model can be chosen, depending on the task at hand.

A large class of continuous distributions allow the kind of reparameterization demonstrated in Eq. (2.398). Therefore it is straightforward to choose different kinds of prior and (conditional) inference distributions. It is also possible to capture posterior correlations between the latent variables either by explicit modelling [12] or by utilizing volume-preserving probability flows [306].

The vAE framework is very expressive and flexible. It can be easily adapted to supervised [153, 12, 124], semi-supervised [201] and generative recurrent [16, 58] models. In the following, I suggest three small extensions of the basic framework:

- Inspired by previous work on directed and undirected graphical models, I suggest to utilize a factorized latent representation [296, 71, 256].
- This factorized representation gives rise to variational Autoencoders for multi-view data with private and shared latent representations [325, 187].
- Finally, I propose to utilize the variational Autoencoder framework to *regularize an unsupervised clustering approach* in a principled manner.

FACTORIZING LATENT VARIABLES. Instead of allowing the posterior distribution to model arbitrary correlations between latent dimensions [12], an alternative approach to a more expressive model is to change the conditional model $p(\mathbf{x} \mid \mathbf{h}, \theta)$, which is usually assumed to be a standard directed graphical model. An instance of this idea is a bi-linear directed graphical model [71]. In the basic case $p(\mathbf{x} \mid \mathbf{h}, \theta)$ is a directed graphical model with two layers. The layer that generates $\mathbf{x}$ is a linear generative model

$$\mathbf{x} = \mathcal{N}(\mathbf{x} \mid \mathbf{W}\mathbf{a}, \boldsymbol{\sigma}^2), \tag{2.400}$$

a diagonal Gaussian. The second layer is a (deterministic) factorized representation of $\mathbf{a}$, i.e.

$$\mathbf{a} = \mathbf{U}\mathbf{b} \odot \mathbf{V}\mathbf{c} \tag{2.401}$$

In this case, $\mathbf{h} \equiv [\mathbf{b}, \mathbf{c}]$, $\theta \subset \{\mathbf{U}, \mathbf{V}, \mathbf{W}\}$ and the prior distributions $p(\mathbf{b} \mid \theta)$ and $p(\mathbf{c} \mid \theta)$ implicitly define the functional roles of $\mathbf{b}$ and $\mathbf{c}$. Of course, both can be chosen to be diagonal Gaussians. Even in this case the factorized model is more expressive compared to the basic

vAE model. As an example, with $\mathbf{U} = \mathbf{V} \equiv \mathbf{I}$ the model represents a Gaussian Scale Mixture [71].

The factorization is particularly interesting in combination with the recently introduced Normalizing Flows for vAE [306] which allow arbitrary complex posterior approximations: Due to the multiplicative interaction of the latent variables a very compact description of data can be achieved.

As a first empirical test, a vAE model with factorized latent units was compared to a standard vAE model, evaluated on the binarized version of MNIST [211]. The basic vAE has an encoder and a decoder with two hidden layers each. The encoder is defined as follows:

$$\begin{aligned} q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi}) &= \mathcal{N}(\mathbf{h} \mid \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \\ \boldsymbol{\mu} &= \mathbf{V}_3 \tilde{\mathbf{h}} + \mathbf{a}_3 \\ \log \boldsymbol{\sigma} &= \mathbf{V}_4 \tilde{\mathbf{h}} + \mathbf{a}_4 \\ \tilde{\mathbf{h}} &= \tanh\left(\mathbf{V}_2(\tanh\left(\mathbf{V}_1 \mathbf{x} + \mathbf{a}_1\right) + \mathbf{a}_2\right), \end{aligned} \tag{2.402}$$

with $\boldsymbol{\phi} = \{\mathbf{V}_i, \mathbf{a}_i\}$, $i = 1, 2, 3, 4$, $\boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbf{R}^{100}$ and $\mathbf{x} \in \mathbf{R}^{784}$. The conditional log-likelihood model $\log p(\mathbf{x} \mid \mathbf{h}, \boldsymbol{\theta})$ is given by

$$\begin{aligned} \log p(\mathbf{x} \mid \mathbf{h}, \boldsymbol{\theta}) &= \sum_{i=1}^{768} \mathbf{x}_i \log \mathbf{y}_i + (1 - \mathbf{x}_i) \log(1 - \mathbf{y}_i) \\ \mathbf{y} &= \sigma(\mathbf{W}_3\left(\tanh\left(\mathbf{W}_2(\tanh\left(\mathbf{W}_1 \mathbf{h} + \mathbf{b}_1\right) + \mathbf{b}_2\right)\right) + \mathbf{b}_3\right). \end{aligned} \tag{2.403}$$

So $\boldsymbol{\theta} = \{\mathbf{W}_i, \mathbf{b}_i\}$, $i = 1, 2, 3$ and $\mathbf{h} \in \mathbf{R}^{100}$.

The factorized model has the same basic architecture with 2 hidden layers in the encoder and decoder. The encoder is identical to Eq. (2.402), but semantically splits the linear output units into two separated classes:

$$\begin{aligned} q(\mathbf{h} \mid \mathbf{x}, \boldsymbol{\phi}) &= \mathcal{N}(\mathbf{h}_1 \mid \boldsymbol{\mu}_1, {\boldsymbol{\sigma}_1}^2)\mathcal{N}(\mathbf{h}_2 \mid \boldsymbol{\mu}_2, {\boldsymbol{\sigma}_2}^2) \\ \boldsymbol{\mu}_1 &= \mathbf{V}_3 \tilde{\mathbf{h}} + \mathbf{a}_3 \\ \log \boldsymbol{\sigma}_1 &= \mathbf{V}_4 \tilde{\mathbf{h}} + \mathbf{a}_4 \\ \boldsymbol{\mu}_2 &= \mathbf{V}_5 \tilde{\mathbf{h}} + \mathbf{a}_5 \\ \log \boldsymbol{\sigma}_2 &= \mathbf{V}_6 \tilde{\mathbf{h}} + \mathbf{a}_6 \\ \tilde{\mathbf{h}} &= \tanh\left(\mathbf{V}_2(\tanh\left(\mathbf{V}_1 \mathbf{x} + \mathbf{a}_1\right) + \mathbf{a}_2\right), \end{aligned} \tag{2.404}$$

with $\boldsymbol{\phi} = \{\mathbf{V}_i, \mathbf{a}_i\}$, $i = 1, 2, 3, 4, 5, 6$, $\boldsymbol{\mu}_j, \boldsymbol{\sigma}_j, j = 1, 2 \in \mathbf{R}^{50}$ and $\tilde{\mathbf{h}} \in \mathbf{R}^{784}$. The log-likelihood model $p(\mathbf{x} \mid \mathbf{h}, \boldsymbol{\theta})$ is a non-linear extension of the bi-linear model from Eq. (2.401):

$$\begin{aligned} \log p(\mathbf{x} \mid \mathbf{h}, \boldsymbol{\theta}) &= \sum_{i=1}^{768} \mathbf{x}_i \log \mathbf{y}_i + (1 - \mathbf{x}_i) \log(1 - \mathbf{y}_i) \\ \mathbf{y} &= \sigma(\mathbf{W}_4 \tanh\left(\mathbf{W}_3 \tilde{\mathbf{y}} + \mathbf{b}_1\right) + \mathbf{b}_2) \\ \tilde{\mathbf{y}} &= \tanh(\mathbf{W}_1 \mathbf{h}_1) \odot \tanh(\mathbf{W}_2 \mathbf{h}_2). \end{aligned} \tag{2.405}$$

with $\theta = \{\boldsymbol{W}_i, \mathbf{b}_1, \mathbf{b}_2\}$, $i = 1, 2, 3, 4$ and $\mathbf{h}_1, \mathbf{h}_2 \in \mathbf{R}^{50}$.

In a brief experiment the marginal negative log-probability of the standard vAE model (Eq. (2.397)) for the test dataset was evaluated to 120 nats, the factorized model achieved 119 nats. The difference is not significant though the factorized model has less parameters overall. Also, both numbers are far from the best reported negative log-probabilities on the dataset (which are around 85 [399]).

Future work will cover more experiments with larger models and also with different datasets. It is interesting to investigate whether the two latent variables $\mathbf{h}_1$ and $\mathbf{h}_2$ will perform semantically different roles. For this type of experiment, at least for one random variable distributions different from a Gaussian should be employed (e.g. the log-Gaussian distribution). Finally, going beyond the original bilinear model, more than two latent variables should be considered. In this case it is not clear how the latent variables are combined. One possibility is to use a hierarchical process.

PRIVATE AND SHARED LATENT VARIABLES. A factorized latent representation is particularly useful for modeling multi-view data. In this case a good modeling assumption is that on the one hand different modalities share some general latent information (the abstract concepts defining the multi-view data), but on the other hand latent representations specific to every modality are necessary. The shared variables encode the *content* of the multi-modal data while the *private* variables encode *style* [302]. Models with latent spaces factorized into private and shared parts are easily integrated into the general vAE framework and thus can be efficiently applied to large scale data sets. Figure 2.16 depicts an architecture that finds private and shared latent representations for the special case of a two-fold multi-view dataset. Unlike other Neural Network-based multi-view models [256] the presented architecture is a proper generative model and does not need one of the modalities to be available in order to generate the others.

Apart from this architectural specifications, no experiments with this model have been conducted so far. Future work will encompass a larger set of experiments with different kinds of multi-view datasets and investigate the following questions:

- How should encoders of different modalities be combined in order to generate a latent representation (e.g. additively vs. multiplicatively)?
- What types of (continuous) distributions should be chosen for private and latent representations?
- Should private and latent spaces them-self be factorized?
- How are the resulting models evaluated? Standard multi-view datasets are usually labeled, so for comparison with competitive

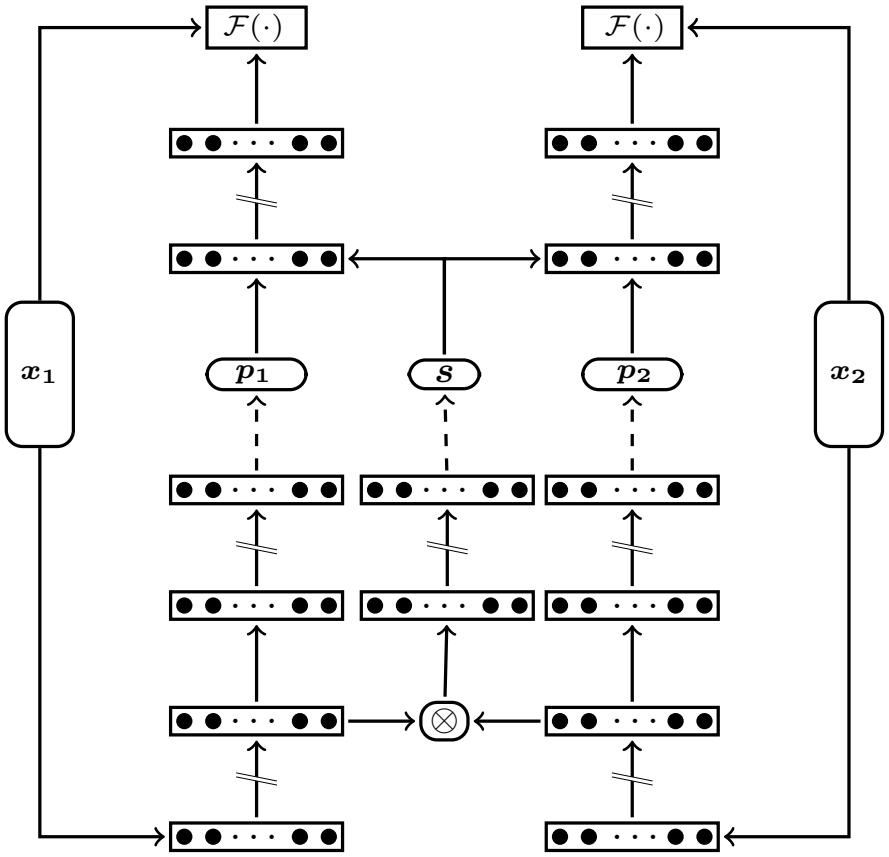


Figure 2.16: Variational Autoencoding a multi-view dataset into shared and private latent variables. The operator $\otimes$ is a placeholder for various possible ways to combine representations from the two views (e.g. addition or multiplication). The model heavily relies on very good single-view generative models. Solid lines in the graph denote deterministic forward propagation of activations, dashed lines denote forward propagation of samples.

models the architecture in Figure 2.16 should be expanded by a semi-supervised mechanism [201].

- How are missing modalities tackled? One idea is to use the neutral element of the combination operator in the shared inference network during training. Then training of the parameters associated with the available modalities can be conducted without problems. Inferring a missing modality happens in a similar way. In this case the missing private latent variable (responsible for the *style* of the data) must be integrated out, e.g. with Monte Carlo.

- The existence of additional modalities should have a profound impact on attentional mechanisms [132]. This aspect is important for the question of sensor fusion.

- Finally, relying on recent developments for generative recurrent models [16, 58], the architecture can be extended to modeling temporal multi-view data.

VARIATIONAL AUTOENCODED REGULARIZED INFORMATION MAXIMIZATION. Regularized Information Maximization (RIM) [119] is a principled probabilistic approach for discriminative clustering. It is based on maximizing the Mutual Information between the empirical distribution on the inputs and the induced cluster distribution, regularized by a complexity penalty. More specifically, let $p(y \mid \mathbf{x}, \boldsymbol{\psi})$ be a conditional model that predicts a distribution over label values (cluster memberships) $y \in \{1, \ldots, K\}$ given some input $\mathbf{x}$. The variable $\boldsymbol{\psi}$ denotes the parameters of this conditional model.

For a *discriminative* clustering task, $p(y \mid \mathbf{x}, \boldsymbol{\psi})$ should fulfill the following competing characteristics:

- *class balance*, i.e. the category labels should be assigned equally across the dataset,
- *class separation*, i.e. data points should be classified with a large margin ($p(y \mid \mathbf{x}, \boldsymbol{\psi})$ should put as much probability mass as possibly to one specific label).

Maximizing the empirical Mutual Information $I(\mathbf{x}, y \mid \boldsymbol{\psi})$ between $\mathbf{x}$ and y under $p(y \mid \mathbf{x}, \boldsymbol{\psi})$ is one possible approach to realize the above two requirements. The Mutual Information $I(\cdot)$ is defined as

$$I(\mathbf{x}, y \mid \boldsymbol{\psi}) = \mathcal{H}[\hat{p}(y \mid \boldsymbol{\psi})] - \frac{1}{N} \sum_i \mathcal{H}[p(y \mid \mathbf{x}, \boldsymbol{\psi})] \tag{2.406}$$

where $\mathcal{H}[p(\mathbf{x})]$ denotes the entropy of $p(\mathbf{x})$ and $\hat{p}(y)$ is the empirical label distribution given as

$$\hat{p}(y \mid \boldsymbol{\psi}) = \int \hat{p}(\mathbf{x}) p(y \mid \mathbf{x}, \boldsymbol{\psi}) d\mathbf{x} \approx \frac{1}{N} \sum_i p(y \mid \mathbf{x}_i, \boldsymbol{\psi}) \,. \tag{2.407}$$

$I(\mathbf{x}, y \mid \boldsymbol{\psi})$ may be trivially maximized by classifying each point $\mathbf{x}_i$ in its own category y_i [40]. RIM therefore extends the objective of maximizing $I(\mathbf{x}, y \mid \boldsymbol{\psi})$ by a regularization term $R(\lambda, \boldsymbol{\psi})$, where the form of $R(\lambda, \boldsymbol{\psi})$ depends on the specific choice of $p(y \mid \mathbf{x}, \boldsymbol{\psi})$:

$$\mathcal{L}_{\text{RIM}}(\mathbf{x}, y, \boldsymbol{\psi}, \lambda) := I(\mathbf{x}, y \mid \boldsymbol{\psi}) - R(\lambda, \boldsymbol{\psi}) \tag{2.408}$$

For example, if $p(y \mid \mathbf{x}, \boldsymbol{\psi})$ is chosen as a multi-class logistic regression model then a suitable regularizer for this linear model is the squared L2 norm of the parameters. However, this simple model is not sufficient to model complex distributions $p(y \mid \mathbf{x}, \boldsymbol{\psi})$. In case of more powerful models, such as deep Neural Networks, appropriate regularization needs to be reconsidered. Clearly, the squared L2 norm

is a viable suggestion also for the case of deep models. However, L2 provides limited expressibility (e.g. when considering that parts of $p(y \mid \mathbf{x}, \boldsymbol{\psi})$ are formed by a Convolutional Network) and, even more importantly, does not align well with the probabilistic principles introduced by RIM[29].

I introduce a novel method vaeRIM that provides probabilistic regularization of complex RIM models based on the vAE framework. While previous Neural Network-based approaches combine unsupervised and supervised learning problems by various forms of nonlinear embedding methods (i.e. Autoencoders) [321, 297, 201] the presented idea combines two unsupervised learning methods in a principled manner.

In the specific case of RIM a straightforward approach is to put a multinomial logistic regression layer on top of some part of the inference network from the vAE framework. This idea is depicted schematically in Figure 2.17. The objective function to be optimized for learning $\boldsymbol{\psi}$, $\boldsymbol{\phi}$ and θ is

$$\mathcal{L}_{\text{vaeRIM}}(\mathbf{x}, \boldsymbol{\psi}, \boldsymbol{\phi}, \theta) = \mathcal{F}(\mathbf{x}, \theta, \boldsymbol{\phi}) + I(\mathbf{x}, y \mid \boldsymbol{\psi}) - R(\lambda, \boldsymbol{\psi} \setminus \boldsymbol{\phi}) \quad (2.409)$$

with $\boldsymbol{\psi} \cap \boldsymbol{\phi} \neq \emptyset$. Hereby $\mathcal{F}(\mathbf{x}, \theta, \boldsymbol{\phi})$ denotes the ELBO (Eq. (2.393)), i.e. $\boldsymbol{\phi}$ and θ denote the parameters of the inference network and the conditional log-likelihood model respectively. Note that parameters from the conditional clustering model $p(y \mid \mathbf{x}, \boldsymbol{\psi})$ that are not shared with the recognition model still need to be regularized in some appropriate way.

As a first benchmark the popular MNIST dataset (with the typical training/validation/test set split) is used to evaluate vaeRIM. The baseline model is a Convolutional Network with two convolutional layers (with 2×2 max-pooling and tanh nonlinearity each) followed by a fully connected layer. The convolutional layers have feature maps of size 32 and 64 respectively and filter sizes of 5×5 each. The parameters (except the biases) of this network are regularized with the squared L2 norm. The baseline network has a 50 dimensional multinomial output, representing 50 possible clusters. A given input $\mathbf{x}$ is associated with the cluster that has the highest activity (i.e. probability under the multinomial distribution). Given cluster memberships for all samples in the training set, it is straightforward to determine the class a cluster represents by looking at the dominating (true) label per cluster. A classifier learned in this way achieves an average classification rate of 73% on the test set, see the first line in Table 2.1.

The recognition network of the variational Autoencoder also consists of the same stack of two convolution layers as the baseline model.

29 In combination with a maximum likelihood based objective, the L2 regularization term lends itself to a MAP interpretation of the resulting cost function. But in the case with RIM the empirical Mutual Information objective does not imply any MAP interpretation.

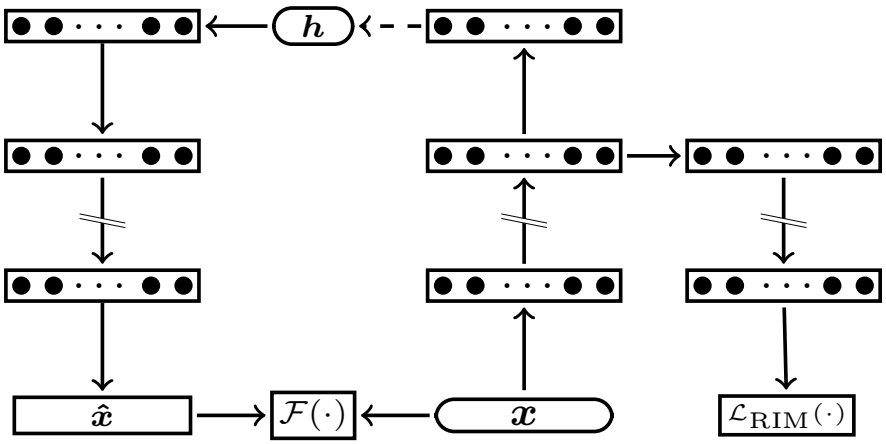


Figure 2.17: The computational graph of vaeRIM. The conditional clustering model $p(y \mid x, \psi)$ used by RIM and the recognition network $q(h \mid x, \phi)$ from the variational Autoencoder framework share a set of parameters. The parameters from $p(y \mid x, \psi)$ that are not shared are still regularized with a squared L2 norm. Solid lines in the graph denote deterministic forward propagation of activations, dashed lines denote forward propagation of samples.

Table 2.1: Average classification performance over 40 runs for the baseline model and vaeRIM on the standard MNIST test set. Out of 40 runs, the best model for vaeRIM achieved 86.9% classification accuracy, the worst model 72.2%.

	MNIST
BASELINE	73.2%± 0.7%
VAERIM	78.9%± 3.2%

On top of this convolutional stack, two fully connected layers are employed *in parallel*: One represents a 50 dimensional multinomial logistic regression layer, feeding into the RIM objective. This layer is regularized with the squared L2 norm. The other layer represents the parameters of the approximated posterior distribution, a 20-dimensional diagonal Gaussian. The generative model $p(x \mid h, \theta)$ is represented by an MLP with one hidden layer of dimensionality 500 and the ReLU nonlinearity. The trained vaeRIM then again induces a classifier on the input data, which achieves a (mean) classification accuracy of 79% on the test set, see the second entry in Table 2.1.

Optimization of both the baseline model as well as vaeRIM was done with Adadelta [427] for 200 epochs—optimal hyperparameters, e.g. the weighting of the L2 regularization terms, were identified on the standard validation set of MNIST.

Figure 2.18 shows the mean input for every cluster identified by an instance of vaeRIM (models based on RIM always only utilize a

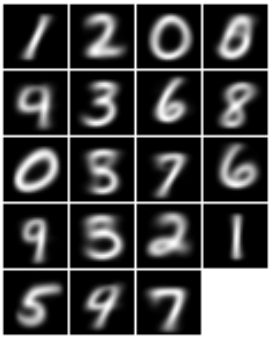


Figure 2.18: Mean images per cluster. In the experiments, vaeRIM can allocate at most 50 different clusters. Out of these possible choices it usually allocates a small number of actually used clusters (i.e. pseudo-classes), in the specific case it settles on 19 different clusters. Every sub-image above is the mean of all MNIST instances from the training set allocated to the respective pseudo-class. Figure best viewed electronically.

subset of the available 50 clusters, i.e. this is also true for the baseline model). The images indicate that inputs from classes 3, 5 and 8 and from classes 4 and 9 are easily confused. This is confirmed by considering the associated confusion matrix, see Figure 2.19. In particular the bad performance of class 5 consistently prohibited average classification rates above 85% for vaeRIM.

The presented vaeRIM model is not the only way to combine RIM and variational Autoencoders. Another obvious approach [201] is to use the inferred latent (variational) representation from a given input and apply a multinomial logistic regression model trained with the RIM objective. In preliminary experiments, this model however produced classification results below 60%.

The most principled approach treats the cluster membership of some input as a latent variable itself. In this case the L2 regularization for the unshared parameters of the discriminative clustering network would become unnecessary. If the two types of latent variables ($\mathbf{h}$ and y) are independent given some data point $\mathbf{x}$, the marginal log-likelihood of $\mathbf{x}$ can be lower bounded as follows [201]:

$$\log p(\mathbf{x} \mid \theta) \geqslant \sum_{y} q(y \mid \mathbf{x}, \boldsymbol{\phi}) \mathcal{L}(\mathbf{x}, y) + \mathcal{H}[q(y \mid \mathbf{x}, \boldsymbol{\phi})] \tag{2.410}$$

where

$$\begin{aligned} \mathcal{L}(\mathbf{x}, y) := \mathbb{E}_{q(\mathbf{h}|\mathbf{x},\boldsymbol{\phi},y)}[\log p(\mathbf{x} \mid \mathbf{h}, y, \theta) + p(y \mid \theta) \\ + p(\mathbf{h} \mid \theta) - \log q(\mathbf{h} \mid \mathbf{x}, y, \boldsymbol{\phi})] \end{aligned} \tag{2.411}$$

and $q(y \mid \mathbf{x}, \boldsymbol{\phi})$ is the recognition model for the cluster membership. Compared to $I(\mathbf{x}, y \mid \boldsymbol{\psi})$ from RIM, an important difference becomes

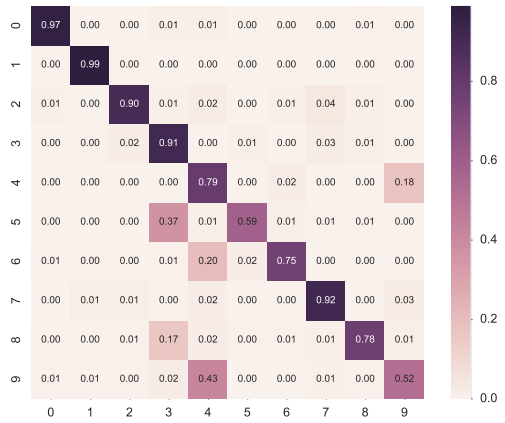


Figure 2.19: The confusion matrix of the ten classes in MNIST for vaeRIM. Rows represent the true class labels, columns the predictions. The classification result for this instance on the test set is 81.3%. Figure best viewed electronically.

apparent: The above model aims at a high entropy for the cluster distribution of a sample $\mathbf{x}$. RIM however aims at an *overall* balanced cluster assignment (first term of $I(\mathbf{x}, y \mid \boldsymbol{\psi})$) but strives for a discriminative clustering behavior for a given sample $\mathbf{x}$ (second term of $I(\mathbf{x}, y \mid \boldsymbol{\psi})$). Future work is concerned with resolving this conflicting goals and combining the RIM cost function with the lower bound objective, essentially realizing a parametric form of an infinitely warped mixture model [174].

3

DATASET

A large body of research for Machine Learning in Computer Vision deals with the problem of object recognition. Supervised deep learning approaches lead to several impressive results in this domain over the last years [60, 61, 209]. In particular the large scale ImageNet [80] dataset became a prominent benchmark to showcase the superiority of deep models and also to investigate new architectural ideas [350, 380] for Neural Networks. This dataset also gained some wider high profile, as popular media outlets started to base absurd claims with respect to artificial intelligence mostly on performance numbers on ImageNet. This high profile in turn lead to an overly focus of deep learning models on high-level object recognition tasks.

In the following I will argue why (high-level) object recognition as represented by ImageNet is not an optimal benchmark for learning systems in the domain of Computer Vision. Instead, I suggest to use datasets that are closer to physically measurable facts and introduce such a dataset which is used in the subsequent sections.

Looking more closely at the ImageNet dataset it becomes clear that this benchmark must be regarded with some care. Its overall construction has a large amount of very similar *micro-classes* of the same abstract classes (e.g. there are hundreds of different *dog* categories). Together with the fact that images mostly show the object in question in a centered (or at least rectified) manner, a learning algorithm is tasked to a large degree to do *elaborate texture analysis*. Of course it was demonstrated that deep models pretrained on ImageNet can also be used for more complicated object recognition datasets [304, 283, 113], but the overall critique can not be refuted: An elaborate texture analysis tries to *emulate a tiny aspect of human intelligence*.

Until very recently (i.e. end of 2014) actually only a small number of other Computer Vision problems have been tackled with deep models: object detection [62, 379], image segmentation [94, 340, 291] and (human) pose estimation [393]. If one defines Computer Vision as the task of inferring (true) physical properties of an environment observed through a camera-like artifice then only the latter task somehow resonates with this basic definition. And even with pose estimation the model is exclusively tasked to emulate human intelligence, as the labels for these datasets are generated completely by humans.

Computer Vision tasks like Tracking, Stereo Vision or Structure-From-Motion can be adequately defined relying on measurable *physical properties* only. Hence these tasks can serve as less biased bench-

marks for deep learning models compared to high-level object recognition.

A common subproblem for these three tasks is identifying similar low-level image patches. Different from high-level object patches these kind of image patches don't show any abstract structure and resemble much more a *seemingly* arbitrary collection of pixels. Nevertheless these patches have a high inherent structure, as they map a very detailed aspect of the underlying three dimensional reality to a two-dimensional representation. Identifying such structures goes beyond simple texture analysis. Note that the notion of similarity on low-level image patches can be extended to similarity on high-level images and hence allows also to tackle object recognition tasks from this point of view (the converse is in general not true). Building an adaptive vision system in a bottom-up way will make the resulting model not only more flexible but also much less prone to artifacts unconsciously introduced into the top-down oriented dataset: High-level concepts will emerge in a bottom-up approach automatically, reducing the necessity of large labelled datasets. The flexibility of a bottom-up approach stems from the fact that adding invariances to the model can be simply achieved by extending the training set with samples that exhibit these invariances—this is not always possible for top-down architectures [290].

Being a common subtask to several Computer Vision problems, low-level image similarity estimation is accompanied by several practical constraints, usually in the form of computational or memory requirements. Of course it is challenging to obtain a dataset that is endowed with the similarity measure induced by the physical reality (basically a weakly labelled dataset with binary targets—patches are similar or not similar). A reliable approach that produces realistic labels and introduces as little algorithmic artifacts as possible is to use additional sensors that allow to build dense surface models which then are used to infer the similarity score.

The dataset [43] used in this work was generated in this way, using multi-view stereo matching to obtain the dense surface models. One of the reasons for generating this dataset is to help finding representations (*descriptors*) for these kind of patches that have a specifically *compact* representation (i.e. a low memory footprint which is essential for real-world applications).

To be closer to the real application of local patch similarity measures, the similarity measure was not defined for arbitrary image patches, but only for these patches centered at so-called *keypoints* [232]. This setting resembles the underlying approach of typical Vision systems for the previously mentioned problems—these systems rely on keypoints in order to stay within computational requirements. Using keypoints to define low-level image patches also allowed to model a main source of uncertainty in such Vision system (i.e. key-

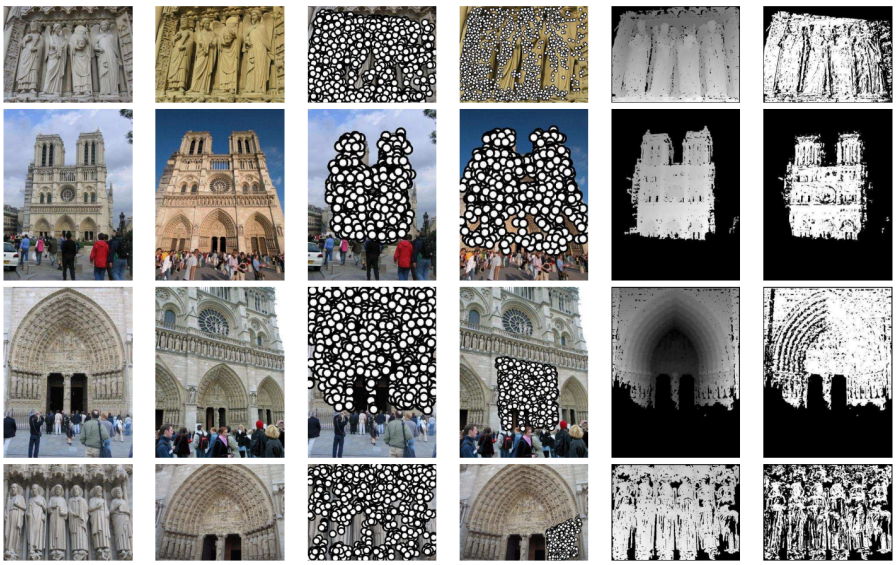


Figure 3.1: A range of images taken from the Notre Dame dataset demonstrate how candidate pairs of low-level image patches are generated. Keypoints from a reference image (first column) are projected into neighboring images (second column) using dense surface models (i.e. depth maps, see second-to-last column). If the projected keypoint is visible in the neighboring image (using *visibility maps*, see last column) any keypoints within 5 pixels in position, 0.25 octaves in scale and $\pi/8$ in orientation of the projected keypoint are considered *matches*. Keypoint pairs that are outside of two times these ranges are considered as *non-matches*. The third and forth column show keypoints that have a matching keypoint in the respective other image. Figure taken from [43]. Best viewed electronically.

points are defined algorithmically and therefore are subject to uncertainty).

The overall approach to determining matching low-level image pairs is outlined in Figure 3.1. After an identified keypoint is projected into neighboring images using the parameters obtained by dense surface models, all keypoints in this image are deemed to be matching ones as long as their location, scale and orientation is within a certain range compared to the projected keypoint. In particular this results in the inclusion of relationships that exist on different *scales*, an important aspect for (natural) images [232]. The journal article [43] Figure 3.1 is taken from contains more details on the dataset generation process.

Because the approach relies on multi-view methods, the available data sources are limited. Hence only three different scenes are available to generate matching and non-matching image patches. These encompasse more than 1.5 million image patches (64×64 pixels) collected from scenes from the Statue of Liberty (450,000 patches), Notre Dame (450000 patches) and Yosemite's Half Dome (650000 patches).

(a) (b)

Figure 3.2: Patches in the same column may depict the same 3D point. (a) Examples for false positive matches (i.e. the two patches represent a different 3D point). (b) Examples for false negatives, i.e. the two patches represent the same 3D point but might be considered as not similar. Best viewed electronically.

While the dataset itself is very challenging (Figure 3.2 shows instances of false positives and false negatives) and the patches from Half Dome resemble a wide selection of typical natural image patches (see Figure 3.3) the overall dataset is limited. For example, no indoor scenes are available which usually have a very different kind of illumination profile, different scales, different level-of-detail and different kinds of viewpoints. In the case of generating a new dataset for this kind of task it is advisable to extend the multi-view approach with a LIDAR-based registration process like it is done for a recently published dataset that covers mostly road-type scenes [106].

Irrespective of the shortcomings of the dataset it is one of the few Computer Vision benchmarks for Machine Learning that represents a task that goes *beyond imitating human intelligence* and allows to realize a low-level type of *understanding*. This fact makes it particularly interesting as a benchmark for unsupervised learning algorithms, an aspect that was not considered so far. Usually such algorithms are evaluated indirectly through supervised object recognition problems or based on probabilistic measurements, e.g. negative log-likelihood scores. Both approaches are however unsatisfactory to some degree.

The first approach becomes obsolete as soon as purely supervised algorithms perform better on the respective object recognition tasks—which will happen with any kind of dataset equipped with human-defined categories because essentially *unlimited labelled data for these tasks exist if enough time for human labelers is allotted*. Also it is highly debatable how an unsupervised algorithm is supposed to perform well on a very narrowly defined supervised task when it does not know of the said task. The latter is theoretically sound but doesn't demonstrate the practical applicability of an unsupervised approach. It is also limited to probabilistic models only and even for this class of algorithms many interesting scores can only be approximately evaluated. Additionally it was recently pointed out that several popular evaluation methods for unsupervised models seem to be inconsistent with each other [385].

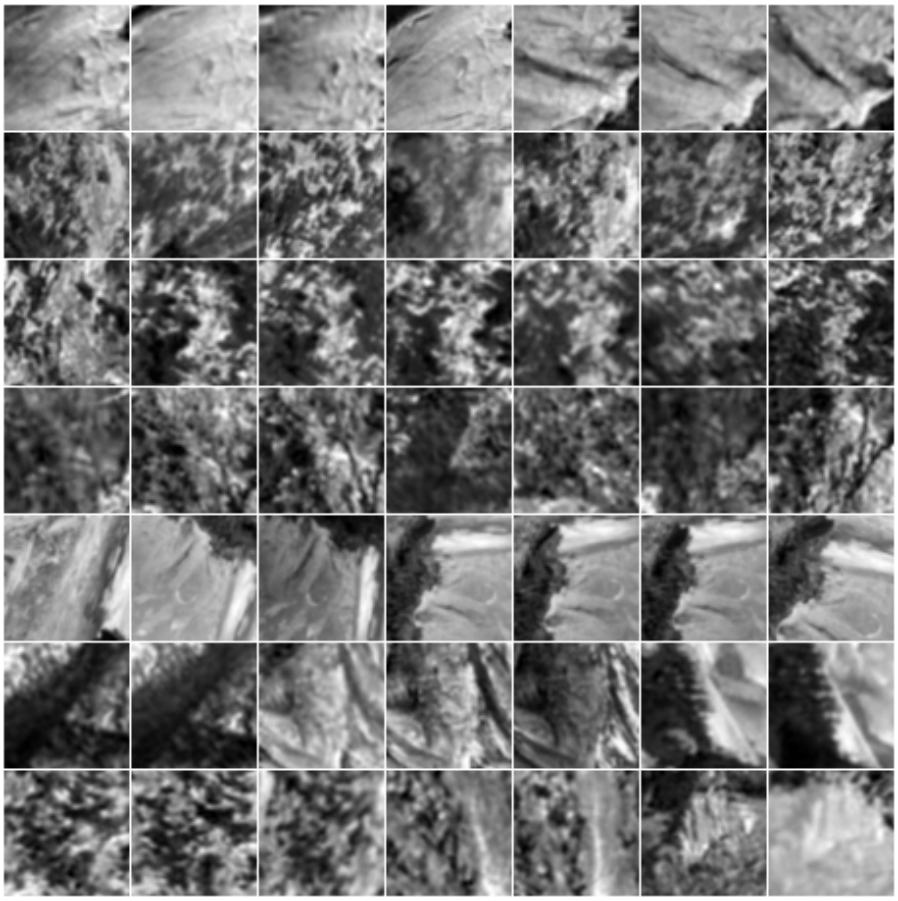


Figure 3.3: Patch correspondences from the Half Dome dataset. Two consecutive images form valid pairs. For this dataset the wide variation of viewpoints and level-of-detail is the most prominent feature. Note that some patches are better identified by only considering the texture present in the patch, while others (e.g. the first row) need some kind of different analysis. All patches are centered on interest points but otherwise can be considered random.

This void can be filled perfectly by the dataset at hand (and more generally by any kind of *unbiased* similarity dataset defined by physical properties). Any unsupervised learning algorithm defines explicitly or implicitly a latent representation for the input space and also induces a distance metric on it. This distance metric then can be used to evaluate the dataset on the similarity scores which can be done without any other supervised component.

As already alluded earlier if unsupervised learning algorithms can be extended to semi-supervised algorithms with a small amount of similarity-based targets (derived from physical properties like proximity in space or time [264]), vision systems for various kinds of problems can be built in a bottom-up manner. The high-level tasks can share most of the processing stack and will overall require considerably less labelled data. This is particularly important for those tasks where it is very difficult (or costly) to get a large amount of labels. So the dataset presented in this chapter can also be interpreted as a first step to build a deep Computer Vision system for a wide range of tasks in a bottom-up way.

BASELINES. The evaluation metric used to quantitatively determine the performance of a (learned) descriptor on the dataset is the *false positive error rate at 95% recall (fp@95)*. In order to determine this error rate any two descriptors must be map-able to a number representing their degree of similarity. Using this number a threshold has to be determined such that 95% of the matching descriptors are considered as actually matching. In turn a certain amount of non-matching descriptor pairs will also be below this threshold (false positives)– this is the considered error metric. The goal is to have as low a false positive error rate as possible.

In order to evaluate the error rates of different models in a principled manner for every scene a preselected dataset comprising 50,000 matching pairs and 50,000 non-matching pairs exists. These test sets are used to evaluated the descriptors which are of course not trained on the respective test scene.

The following baselines on this dataset (see Table 3.1 for the quantitative evaluation) are used to assess the performance of the deep models introduced in the subsequent sections

- RAW: The most basic model is to use the pixels themselves as features. Obviously this is not a very compact representation, and also not a very *invariant* one. Two descriptors are compared by computing the sum-of-square difference on the vectorized images.

- SIFT [238]/SIFTb: The 128 dimensional descriptor from David Lowe. These are actually 128 float numbers, so the memory footprint for one SIFT descriptor is not that small. *SIFTb* therefore

describes the SIFT descriptor where every dimension is represented by one byte.

- DAISY [43]: The best descriptor learned with the pipeline defined in the original paper of the dataset. The DAISY descriptor is either 29 or 36 dimensional.
- L-BGM [396]: A compact (64 dimensional) descriptor that is learned by applying boosting to gradient-based weak learners.
- CVX [351]: This descriptor (in the case of this work we look at the 32 dimensional version) learns both pooling regions and dimensionality reduction using convex optimization applied to a discriminative large margin objective.

Table 3.1: Baseline models used throughout this work. The table shows the false positive error rate at 95% retrieval. Descriptor models are trained on one dataset and evaluated on the remaining two datasets. RAW, SIFT and SIFTb do not need any prior training phase.

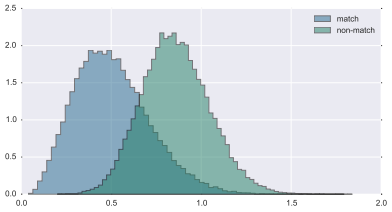
	LY		**ND**		**HD**	
	ND	HD	LY	HD	LY	ND
RAW	52.4	51.1	58.3	51.1	58.3	52.4
SIFT	20.9	24.7	28.1	24.7	28.1	24.7
SIFTb	22.8	25.6	31.7	25.6	31.7	25.6
DAISY	–	–	16.8	13.5	18.3	12.0
L-BGM	14.2	19.6	18.0	15.8	21.0	13.7
CVX	9.1	14.3	14.2	13.4	16.7	10.0

As additional baselines two special cases of deep Convolutional Networks were also considered:

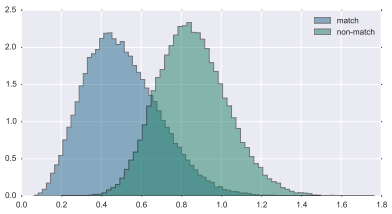
- The deep Convolutional Network referred to as *AlexNet* [209] encoded similarity on an *instance-based* metric surprisingly well by relying on the final hidden layer (a fully connected layer of size 1024) [209, Figure 4 (right)]. Hence, it could be the case that the pretrained network also performs well on the dataset at hand. The original network is configured for images of size 224×224. Being a convolutional architecture, it is straightforward to adapt the network to the new input size of 64×64. It is only necessary to change the stride lengths of the convolutional layers in order to ensure that the fully connected layers have the necessary shape of the original AlexNet. In this way

all pretrained parameters can be utilized. However, the resulting error rates were very poor and never decreased below 70% (this performance was consistent independently of the layer of the network that was chosen to act as a descriptor for a given patch). One possible way to improve the score would have been to fine-tune the network with the labelled similarity data. However, the resulting descriptor would have been at least a 1024-dimensional vector, and thus too large, at least for the supervised setting. Additionally, due to the large number of parameters evaluating the network on a patch took about twice as long as evaluating SIFT. Therefore I did not continue to pursue this direction.

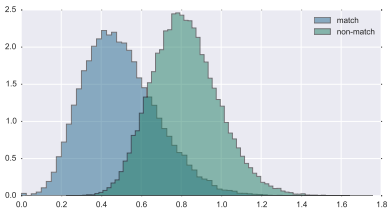
- Mapping high-dimensional samples with random projections [33] to low-dimensional embeddings often results in very good nearest neighbor performance. However, this was not the case here. A randomly initialized Convolutional Network with 4 layers (and tanh activation function) mapping a 64×64 patch to a 32 dimensional vector resulted in an error rate of roughly 50% for all three datasets. Figure 3.4 depicts the distance histograms of this network for all three test sets.



(a) Liberty



(b) Notre Dame



(c) Yosemite

Figure 3.4: Distance histograms for a random Convolutional Network on the respective test set for the three scenes. The blue histogram shows the empirical distribution of distances (L2 norm) of matching patches. It is noticeable that the distribution of the non-matching patches is much more *Gaussian* than the distribution of the matching patches. This can be interpreted as an indication that the matching patches form a kind of lower-dimensional manifold. Best viewed electronically.

4

SUPERVISED MODELING OF LOCAL IMAGE PATCHES

A central goal of Computer Vision is to describe the content of images, possibly at different levels of detail. Given such descriptions tasks like object recognition, scene understanding, 3d reconstruction, structure from motion, tracking and image search can be solved. One possible approach to identify such a description is to rely on geometric properties of the objects that are supposedly depicted in images—model-based object recognition [258]. Clearly, this approach finds its limits when describing non-geometric objects (e.g. trees or clouds) and is also limited computationally with respect to the number of objects it can handle within an image.

Utilizing photometric information on the other hand is often an excellent tool to differentiate between a large number of objects. Simple ideas like color co-occurrence histograms [52] work surprisingly well. These *appearance-based* methods can be extended by computing *local* gray value descriptors, e.g. steerable filters [99] or Gabor filters [181] on a *global* grid. However, being global methods they have difficulties with partially visible objects.

The seminal paper by Schmid and Mohr [329] introduced the idea of describing an image as a *set of salient* image points only. Salient points (or *interest* points) are characteristic locale regions (*features*) in an image with high informational content. The *descriptor* of an interest point is computed in such a way that it shows a certain degree of *invariance* with respect to typical transformations that leave the actual content unchanged, e.g. illumination transformations, displacement transformations or scale transformations.

This idea was subsequently refined and extended by David Lowes SIFT [239]: Inspired by findings from neuroscience [88, 235] SIFT relies (among other heuristic ingredients) on a non-linear, edge-based descriptor transformation and improves on illumination, scale, rotation and translation invariance. It resembles a simplified model of the primary V1 cortex to discriminate between edge orientations and pools into small spatial as well as orientation bins. Based on these concepts a plethora of other descriptors has been hand-crafted during the last 10–15 years, e.g. SURF [14] and HOG [74], see [316] for a summarizing article.

These descriptors are specifically constructed to show a high level of invariance with respect to appearance, viewpoint and lightning variations of a local image patch. Being tolerant to non-rigid changes in object geometry but maintaining high selectivity at the same time

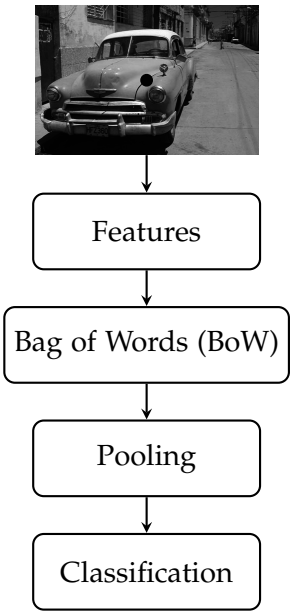


Figure 4.1: The prototypical structure of a processing pipeline for an object recognition system before 2013. A given image (not limited to grayscale) is converted into a large set of *local* descriptors, using a feature extraction module. The feature extraction module resembles typical hand-engineered approaches like SIFT. Dictionaries for Bag-of-Word (BoW) representations are learned with unsupervised methods like k-means or sparse coding. Sometimes, an additional *pooling* module is employed which often encodes prior knowledge about the input data, e.g. spatial pyramid pooling [214]. The final vectorized representation of the original image is used training a *shallow* supervised classifier. This figure is adapted from a set of slides by Yann LeCun.

makes these local descriptors central to a wide range of Computer Vision tasks like tracking [429], stereo vision [392], panoramic stitching [42], structure from motion [144] and specifically object recognition [352, 214]. E.g. for object classification, local features are used through *vector quantization*—at least until fairly recently (before 2013) *visual words* were at the center of many successful classification systems [352, 353, 214] that have the overall compositional structure depicted Figure 4.1. The descriptor plays a central role in this pipeline which is the reason why the task of defining the descriptor was reserved for the human: if the descriptor is chosen right the whole system will perform well.

Interestingly the approach depicted in Figure 4.1 is not limited to object recognition in Computer Vision systems: Typical classification tasks in natural language processing are solved in the same way – in this case engineered *syntactic parse trees* resemble the handcrafted descriptors. Similarly, the area of speech recognition relied heavily on such an architecture for more than 10 years. In this case

the handcrafted features are variants of Mel-frequency cepstral coefficients (MFCC) [78] and a standard Gaussian Mixture Model [35] implements the Bag of Word module and also realizes pooling.

Widespread interest in deep learning methods for solving general perception problems (image, audio, natural language and video) was (re)ignited by several astounding results of Neural Network-based systems for classification tasks in Computer Vision and Speech Recognition [126, 125, 64, 60, 62, 73] where deep networks outperformed their opponent approaches by a large margin. More specifically, deep *Convolutional* Networks have emerged as the dominating model for a successful Computer Vision pipeline[1] (and, interestingly enough, are also competitive for speech recognition [1] and natural language processing [194]). This new pipeline bears a lot of resemblance to the one shown in Figure 4.1, however it expands the feature processing element of the old pipeline into multiple layers, see Figure 4.2. It no longer relies on human engineering but learns everything on its own, given only input/output relations (i.e. such a system is trained in an *end-to-end* manner). The advantage of learning methods is that they provide a way to *learn filters in the subsequent stages* of a processing pipeline: Prior knowledge about image statistics point to the usefulness of oriented edge detectors at the first stage, but there is no similar apriori knowledge that would allow to design reasonable filters for later feature modules. Therefore, basically all human designed feature extractors need to stay *shallow*.

A possible criticism of these successful Computer Vision systems can be that they are nearly all instances of classification problems: Even object detection and image segmentation can be solved by considering it as a (pixelwise) classification problem [340, 94]. Yet, many of the typical vision problem domains mentioned in the opening paragraph are not instances of classification problems. Instead solutions in theses domains are not structured like the pipeline from Figure, but are a mix of algorithmic components and low-level image descriptions—the hand-crafted features from the first stage of the pipeline in Figure 4.1.

It would come as no surprise if this first stage of these algorithms can also be substituted by a deep model. Remarkably, hand-crafted descriptors them-self have a structure that resembles the one shown in Figure 4.1: a layer of filters is followed by a pooling stage and possibly by a normalization layer, see Figure 4.3. Obviously, every stage of the pipeline in Figure 4.3 can benefit from learning with the goal to exploit statistical properties of low-level image patches in order to find *discriminative* representations that are invariant to the many unwanted transformations such a patch is usually exposed to. Recently exactly this has been done with various discriminative learning tech-

1 It might be the case that Convolutional Networks get superseded by the more general RNNs in the Vision domain, too [384, 369].

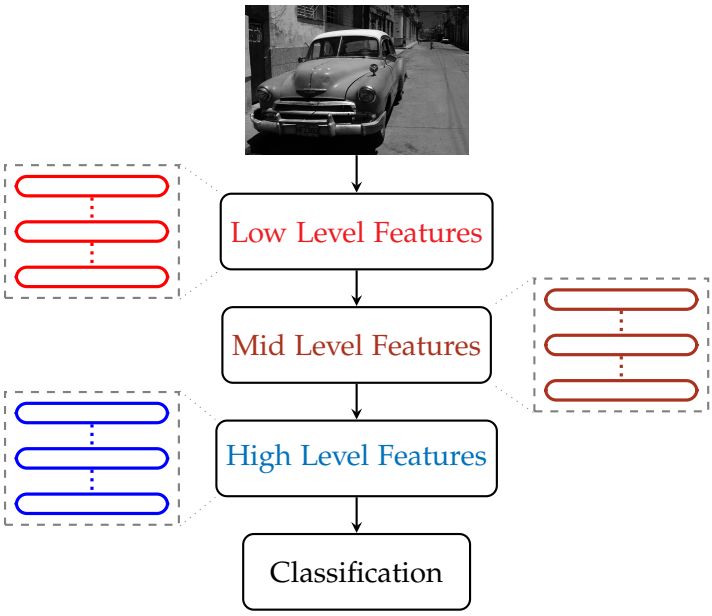


Figure 4.2: A deeply nested feed-forward system that is trained in an end-to-end way. Note that in such systems conceptual entities like *Low Level Features* loose their actual meaning. The system consists of different kinds of *stackable* learning modules, represented by colored rectangles. Different colors indicate that different types of learning modules can be combined. As of Summer 2015, this overall architecture is the dominating paradigm for e.g. object classification systems of static images. It is straightforward to add a time-dependent component to this architecture through so-called *recurrent* connections. The only requirement for the overall system is that information from the classification problem (i.e. a learning signal) somehow reaches all trainable modules. *Back-propagation* is a popular way to provide such a signal, but as it turns out, more general approaches also work very well [50].

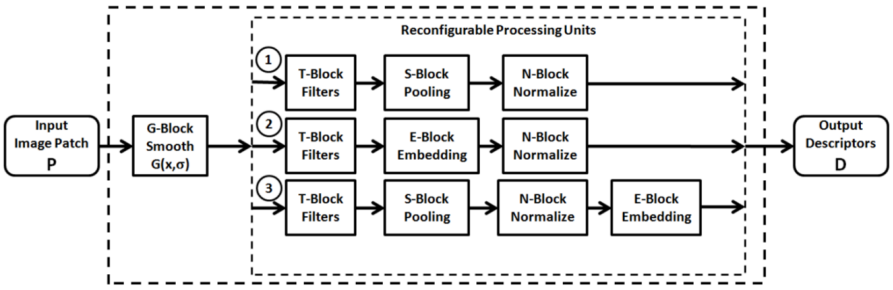


Figure 4.3: A typical descriptor design: After smoothing the pixels around an identified interest point (represented by a rectangular or circular image patch), a bank of filters computes *feature maps*. These feature maps are then non-linearly processed through normalization layers and/or dimensionality reduction layers (pooling and embedding). Figure is adapted from [43].

niques [344, 43, 396, 351]. For example, in [351], a convex optimization formulation has been utilized to learn the pooling configuration and the subsequent dimensionality reduction.

The parallel between Figure 4.1 and Figure 4.3 is evident and therefore it is not only conceptually reasonable but actually very promising to learn a deep version of the original descriptor pipeline in an end-to-end manner.

The resulting learning task is apriori no classification problem, so the learning objective needs to be formulated differently. In this case an interesting question for the successful realization of the deep descriptor learning system is to find out how much of the wisdom identified for training deep networks on object classification tasks is true in this new setting.

While this question is interesting and important for Machine Learning practitioners, Computer Vision experts are probably evaluating the approach according to different criteria[2]: Is the considered dataset representative enough? Do the learned descriptors also perform acceptably on other tasks than the one they were trained on? Are typical constraints imposed on descriptors considered?

The last question is particularly important for the chosen deep architecture: Descriptors are extracted in a dense fashion, so it is important that both the memory footprint for a descriptor is as small as possible as well as the computational demand for computing it.

Therefore, the overall goals of this chapter are as follows:

- Identify possible ways to formulate descriptor learning in a way suitable for deep networks but usable in generic Computer Vision tasks.

2 Deep Convolutional Networks were only embraced by the majority of the Computer Vision community after they beat standard approaches on ImageNet [80], an accepted benchmark in this research community.

- Empirically evaluate the vast range of design decisions that need to be resolved when working with deep networks for the specific problem of descriptor learning.
- Possibly perform better than other state-of-the-art approaches on a meaningful benchmark, while taking relevant side constraints for descriptors into account.
- Evaluate the learned models on datasets that are significantly different from the one used to train the descriptors.

However, learning descriptors with a deep architecture is not only a goal in itself. Instead it can serve as a building block for solving Computer Vision problems in a *bottom-up* manner. This idea is contrary to the currently dominating paradigm of training nested architectures in a *top-down* end-to-end way. It particularly necessitates a very large amount of labeled data which is time consuming and expensive to collect. It is even more so for more detailed and more fine grained labels that are necessary for tasks that go beyond standard object recognition.

A bottom-up approach can solve this problem. It still builds a deeply nested architecture that is trained in an end-to-end way. However, much less (exactly) labeled data for the overall task is necessary because parts of the architecture are trained on auxiliary tasks for which it is either easier to get labeled data or the labels itself need only to be weak (e.g. correspondences between two patches can easily be gathered through simple temporal coherence principles or with systems that have additional sensors, e.g. depth measurements that help to determine the weak correspondence label). Additionally, these parts (usually in the lower layers of deep architectures) can also be shared by different high-level tasks.

The chapter is structured as follows: In Section 4.1 I briefly describe the dataset we are looking at in this chapter—a more in-depth presentation is given in Chapter 3. Convolutional Networks and suitable cost functions for the descriptor learning task are presented in Section 4.2. Section 4.3 is an extensive quantitative evaluation of the wide range of possible design decisions when training Convolutional Networks. This is followed by a qualitative evaluation of some of the trained models, showing visualizations of various properties of these models (Section 4.4). Two transfer learning experiments are covered in Section 4.5. The chapter concludes with a section dedicated to related work (Section 4.6) and a summary (Section 4.7).

4.1 DATASET

In order to convince a Computer Vision practitioner to use deep Convolutional Networks as a building block to solve her *geometric* vision

problems, it is important to demonstrate their capabilities in modeling low-level image patches on a relevant dataset.

The dataset from Chapter 3 encompasses a large and, more importantly, a realistic data set of patch correspondences. Importantly it is built by well respected Computer Vision researchers, working in the field of feature descriptors. The main reason for constructing the dataset was to establish a benchmark to evaluate *compact* local image descriptors [43]. It is based on more than 1.5 million image patches (64×64 pixels) collected from three different scenes (450,000 patches from the Statue of Liberty (this scene is denoted *LY* in the experiments) and Notre Dame (*ND* each, 650,000 patches from Yosemite's Half Dome (*HD*)). The patches are sampled around interest points detected by Difference of Gaussians [239] and are normalized with respect to scale and orientation[3]. Figure 4.4 shows some random patches from the Notre Dame scenery. Clearly, the dataset has a wide variation in lighting conditions, viewpoints, and scales.

The dataset contains also approximately 2.5 million image correspondences. In order to produce ground truth data stereo matching is used to obtain dense surface models. These dense surface models in turn are then used to establish correspondences between image patches. As actual 3D correspondences are used, the identified 2D patch correspondences show substantial perspective distortions resulting in a much more realistic dataset than previous approaches [227, 259]. In order to facilitate comparison of various descriptor algorithms a large set of predetermined match/non-match patch pairs is provided. For every scene, sets comprising between 500 and 500,000 pairs (with 50% matching and 50% non-matching pairs) are available.

The evaluation metric used to quantitatively determine the performance of a descriptor is the *false positive error rate at 95% recall (fp@95)*. This means that a threshold has to be determined such that 95% of the matching descriptors are considered as actually matching. In turn a certain amount of non-matching descriptor pairs will also be below this threshold—this is the considered error metric. The goal is to lower this percentage as much as possible.

At a first glance, the dataset appears very similar to an earlier benchmark of the same authors [420], yet the correspondences in the novel dataset resemble a much harder problem. The error rate at 95% detection of correct matches for the SIFT descriptor [239] raises from 6% to 26%, the error rate for evaluating patch similarity in pixel space (using normalized sum squared differences) raises from 20% to at least 48% (all numbers are take from [420] and [43] respectively).

3 A similar dataset of patches centered on multi-scale Harris corners is also available.

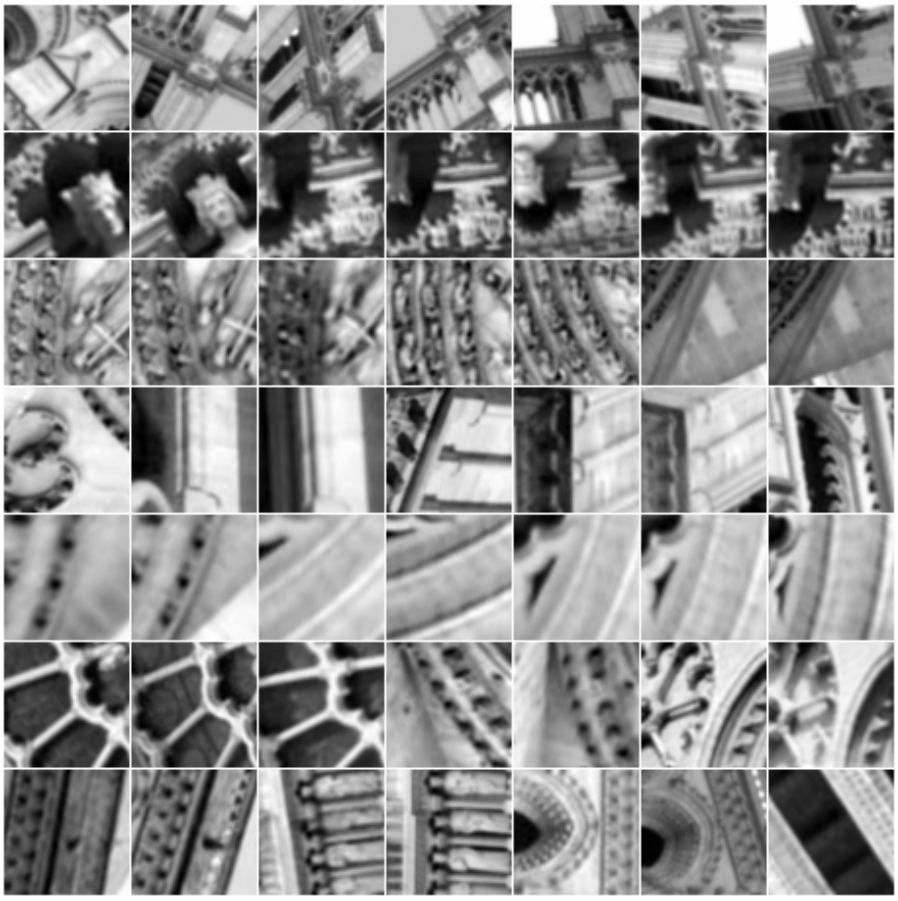


Figure 4.4: Patch correspondences from the Notre Dame dataset. Note the wide variation in lighting, viewpoint and level of detail. The patches are centered on interest points but otherwise can be considered random, e.g. there is no reasonable notion of an object boundary possible.

Table 4.1: The false positive error rate at 95% retrieval (fp@95) for the baseline models. Descriptor models are trained on one dataset and evaluated on the remaining two datasets. RAW, SIFT and SIFTb do not need any prior training phase.

	LY		**ND**		**HD**	
	ND	HD	LY	HD	LY	ND
RAW	52.4	51.1	58.3	51.1	58.3	52.4
SIFT	20.9	24.7	28.1	24.7	28.1	24.7
SIFTb	22.8	25.6	31.7	25.6	31.7	25.6
DAISY	–	–	16.8	13.5	18.3	12.0
L-BGM	14.2	19.6	18.0	15.8	21.0	13.7
CVX	9.1	14.3	14.2	13.4	16.7	10.0

4.1.1 *Baselines*

In order to asses the performance of the many trained models in Section 4.3, I consider the following baselines on this dataset (see Table 4.1 for the quantitative evaluation):

- RAW: The most basic model is to use the pixels themselves as features. Obviously this is not a very compact representation, and also not a very *invariant* one. Two descriptors are compared by computing the sum-of-square difference on the vectorized images.
- SIFT [238]/SIFTb: The 128 dimensional descriptor from David Lowe. These are actually 128 float numbers, so the memory footprint for one SIFT descriptor is not that small. *SIFTb* therefore describes the SIFT descriptor where every dimension is represented by one byte.
- DAISY [43]: The best descriptor learned by optimizing the free parameters of the descriptor pipeline from Figure 4.3. It has either 29 or 36 dimensions.
- L-BGM [396]: A compact (64-dimensional) descriptor that is learned by applying boosting to gradient-based weak learners.
- CVX [351]: This descriptor (in the case of this work I consider the 32-dimensional version) learns both pooling regions and dimensionality reduction using convex optimization applied to a discriminative large margin objective.

4.2 CONVOLUTIONAL NETWORKS

As described in detail in section 2.7.3 a standard Convolutional Network is a feed forward network with an architecture that is specialized for processing datasets where the constituents of a sample show a high degree of local neighborhood relations. Typical examples for such local neighborhood relations are the 1D structure of audio data, the 2D structure of images or the 3D structure of videos. While these local neighborhoods are defined through a regular graph structure (a lattice over multiple dimensions), recent publications [44, 250] applied Convolutional Networks to general graph structures, too.

In order to find patterns in such locally defined structures the main elements of a Convolutional Network are its *filters* that are applied in a *convolutional* way to all elements of a given sample, e.g. to all pixels of a given image. As it turns out this convolutional operation can be simulated in a straightforward manner by a standard feed forward network through an adequately structured Toeplitz matrix [118] (most weights are zero) and weight sharing[4]. Such a convolutional layer is followed by an elementwise applied nonlinearity and then (usually) by a so called *pooling* layer. This special type of feed forward operation aggregates information within a small local region of a filtered sample. The local region is hereby implied by the local geometry of the dataset, e.g. in 2d images, a local region is usually defined by a square patch encompassing 2x2 or 3x3 pixels.

The pooling operation can be considered a special form of a convolution with a fixed set of weights [363]. Differently to a standard convolution layer the pooling regions are usually disjunct. The goal of the pooling operation is to achieve a certain degree of invariance with respect to a basic geometric operation on the underlying structure. This is particularly helpful for classification tasks because a range of geometric operations are invariant with respect to object identity but obfuscate the overall image. For example, in regular lattices, pooling achieves some form of *translation* invariance.

Finally, in recent years additional types of specialized layers were proposed mostly for normalization purposes [182, 209], but again these types are actually special cases of convolution.

A classic Convolutional Network is composed of alternating layers of convolution (with a nonlinearity) and pooling. Notably, a convolution layer not only consists of one filter but a set of filters. A filter f_i applied to a given input produces a *feature map* m_i. A convolutional layer with n filters therefore produces n feature maps (that is in the case of 2d images a *three* dimensional tensor with n being the first dimension). After pooling every *single* feature map, a new layer of convolutional filters is applied to the new *stack* of pooled feature maps. Due to the backpropagation algorithm (see Section 2.7) train-

4 This approach is for example taken in the widely popular *Caffe toolkit* [188].

ing a Convolutional Network can be done efficiently through iterative optimization algorithms, e.g. stochastic gradient ascent on a suitable objective function.

4.2.1 *Loss functions for correspondences*

While the architecture of a deep network defines a simple yet powerful and flexible computational pipeline the accompanying loss function determines what task actually is solved by the network. So here it is necessary to look at the specific dataset at hand, consider all available information and the problem that should be solved.

For the matching task the problem description is easy to formulate: Find a representation for patches such that patches identifying the same 3D point in a scene are close by under some undefined (i.e. to be determined, too) metric.

When *evaluating* learned representations an additional scalar is necessary: all pairs of patches with a distance of their representations below this threshold are deemed similar, i.e. representing the same 3D point. Truly similar patches therefore need representations that are *close* to each other. In this case a suitable cost function seems to be straightforward: *Pull* representations of matching patches together, i.e. minimize the sum over all squared distances of matching pairs:

$$\sum_{x_1, x_2 \in \mathcal{M}} d_\theta(x_1, x_2)^2 \tag{4.1}$$

Here $\mathcal{M}$ is the set of all matching patches and $d_\theta(x_1, x_2)$ is the distance between the representations of the two patches x_1 and x_2. One possible choice is the standard euclidean distance:

$$d_\theta(x_1, x_2) = \|f(x_1, \theta) - f(x_2, \theta)\|_2 \tag{4.2}$$

In turn, $f(x, \theta)$ represents a Convolutional Network that implements a non-linear transformation of some patch $x \in \mathcal{X}$: $f : \mathcal{X} \times \mathbf{R}^n \to \mathbb{R}^D$. $\theta \in \mathbb{R}^n$ denotes all learnable parameters of the network.

However, Eq. (4.1) is a flawed objective function: One possible optimal solution is to map all patches to the same (constant) representation, e.g. the zero vector. This problem occurs because the objective is not formulated in a probabilistic way: a contrastive term (see Section 2.4) is missing.

For datasets where similarities between inputs are defined by object identities, Neighborhood Component Analysis (NCA) [117, 17] is a nice probabilistic formulation. However, for the dataset at hand similarities are defined at a very fine granularity so a better approach is to define a contrastive term in a heuristic way (avoiding at the same time the computational problems of a probabilistic formulation like NCA, because no costly normalization with respect to probabilities is

necessary). An obvious idea is to *push* dissimilar pairs apart, that is (because of the negative sign!) *minimize*

$$-\sum_{x_1,x_2\in\mathcal{N}} d_\theta(x_1,x_2)^2 \tag{4.3}$$

where $\mathcal{N}$ is the set of non-matching pairs. Overall, the objective function is defined as

$$\sum_{x_1,x_2\in\mathcal{M}} d_\theta(x_1,x_2)^2 - \sum_{x_1,x_2\in\mathcal{N}} d_\theta(x_1,x_2)^2 \tag{4.4}$$

If a label y indicates whether the pair $(x_1,x_2)\in\mathcal{M}$ (that is $y\equiv 1$) or $(x_1,x_2)\in\mathcal{N}$ (i.e. $y\equiv 0$) then the loss from Eq. (4.4) can be written as:

$$\sum_i y^i d_\theta(x_1^i,x_2^i) - (1-y^i) d_\theta(x_1^i,x_2^i) \tag{4.5}$$

Here i iterates over all available pairs of matching as well as non-matching patches.

Investigating the objective function from Eq. (4.5) with respect to the evaluation criterion formulated at the beginning of this section, it becomes clear that too much work is done by the contrastive term: In comparison to the distance of matching pairs it is only important that dissimilar patches are pushed apart *far enough*. Pushing them apart indefinitely (as implied by Eq. (4.3)) results in unnecessary work and actually makes the task much more difficult to solve. From a theoretical point of view the problem with Eq. (4.3) is its *unbounded* nature. The training algorithm will overfit to this aspect of the loss function and important information from matching pairs that are *still too far apart* is drowned out by contributions from dissimilar but already far apart pairs.

The solution to this problem is as simple as elegant: If two dissimilar patches are farther apart than some margin m_{push} then these pairs are no longer considered in the objective (i.e. they are no longer *pushed* farther apart, which explains the naming of this margin). That is, *minimize*

$$\sum_{x_1,x_2\in\mathcal{N}} [m_{\text{push}} - d_\theta(x_1,x_2)]_+^2 \tag{4.6}$$

with

$$[x]_+ = \max(0,x) \tag{4.7}$$

Setting $m_{\text{push}}\equiv\infty$ one gets Eq. (4.3) as a special case. In the best case, m_{push} should be somehow related to the threshold that is utilized during the evaluation procedure, i.e. the threshold that determines whether two patches are considered similar or not. The overall loss function

$$\ell_{\text{DrLim}}(\theta) = \sum_i y^i d_\theta(x_1^i,x_2^i)^2 + (1-y^i)[m_{\text{push}} - d_\theta(x_1^i,x_2^i)]_+^2 \tag{4.8}$$

was introduced in 2006 under the name *DrLim* (dimensionality reduction by learning an invariant mapping) [138]. The goals for DrLim obviously align well with the matching task at hand:

- *Simple distance measures in the output space* (such as euclidean distance) should approximate neighborhood relations in the input space.
- The mapping should not be constrained to implementing simple distance measures in the input space and should be able to learn complex transformations.
- It should be faithful even for samples whose neighborhood relationships are unknown.

The last two goals are important because many metric distance learning algorithms [210] need a tractable distance metric in the input space and can not provide out-of sample extensions.

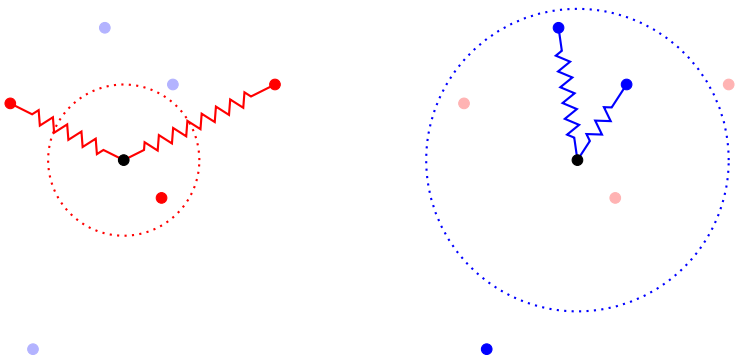
From a conceptual point of view, DrLim can be described with a physical spring analogy: Similar patches are connected by *attract-only* springs, dissimilar patches by *m-repulse-only* springs. Figure 4.5 depicts this conceptual point of view for a stylized setting in 2D (already showing the generalized idea of *m-attract-only* springs introduced in the next paragraphs).

While the DrLim loss seems very plausible, a close inspection of Eq. (4.8) indicates that room for improvement exists: In particular, if two matching inputs x_1 and x_2 are considered, Eq. (4.8) demands that their final representations should be identical. However, this is an unreasonable requirement with respect to the evaluation criterion: The two representations should only be *close enough*. Additionally if one imagines that the Convolutional Network transforms patches to some complicated manifold than it is a bad idea to require that *similar* patches have *identical* representations. Instead one hopes that similar patches actually define some space on this manifold that behaves locally in a nice way, e.g. it is locally linear with respect to the patches associated with the same (or close-by) 3D point(s). In more geometric terms, one wants to achieve *equivariance* instead of *invariance* under different 3D transformations.

Therefore, a simple relaxation similar to the push margin m_{push} can be introduced via a *pull margin* m_{pull}:

$$\sum_{x_1, x_2 \in \mathcal{M}} [d_\theta(x_1, x_2) - m_{\text{pull}})]_+^2 \tag{4.9}$$

That is, all matching pairs with a distance below the pull margin are ignored by the objective function (see Figure 4.5). Not only allows it to learn an overall smoother mapping but it also simplifies the optimization procedure: Pairs that are already close enough can be ignored by the network and it can focus its capacity on solving for



(a) Attracting Forces

(b) Repulsive Forces

Figure 4.5: Physical spring analogy for the two terms involved in the objective. The black point represents a reference object: red points resemble similar objects, blue points dissimilar objects. Clearly, the overall system tries to find a complicated equilibrium, balancing compressive and stretching forces between all involved objects. (a) Red springs act as attracting springs, that is similar points are pushed closer to the black point. However, these springs have a certain rest length (indicated by the red dotted circle with respect to the black point), if this length (or a smaller length) is achieved, such a spring no longer exerts a *compressive* force—so this kind of spring does not show any stretching forces. The similar red object within the dotted red circle is therefore not pulled any closer to the reference object (but also not pushed away!). Note that the rest length of the original DrLim formulation for the attracting springs is 0, so it will always exert a compressive force. (b) Blue springs act as repulsive springs, they push apart (i.e. only exert stretching forces). These springs also have a rest length, if this length is attained (or takes values beyond) then a spring stops pushing apart, too. This rest length is indicated by the blue dotted circle, the dissimilar blue object outside this circle is not pushed any further from the reference object. Figure is adapted from [138].

difficult pairs (i.e. similar to the previous reasoning, one can avoid overfitting of the network). The improved formulation for DrLim is

$$\begin{aligned}\ell_{\text{DrLim}^+}(\theta) = \sum_i \big[& y^i [d_\theta(x_1^i, x_2^i) - m_{\text{pull}}]_+^2 \\ & + (1 - y^i)[m_{\text{push}} - d_\theta(x_1^i, x_2^i]_+^2\big]\end{aligned} \tag{4.10}$$

$\ell_{\text{DrLim}^+}(\theta)$ is simply the sum of two hinge-like losses. The difference to the standard hinge loss is only the square operation at the respective pushing and pulling terms. Is the square actually beneficial in this case? Clearly it smooths the otherwise discontinuous *gradient* of the loss with respect to the embedding. Consider the part of Eq. (4.10) that pulls similar pairs together:

$$\sum_{x_1, x_2 \in \mathcal{M}} [d_\theta(x_1, x_2) - m_{\text{pull}})]_+^2 \tag{4.11}$$

If the distance for a matching pair (x_1, x_2) is between m_{pull} and $m_{\text{pull}} + 1$ then the contribution of this pair to the overall cost is *scaled down quadratically*. Similarly, the contribution to the gradient used for optimizing the parameters of the Convolutional Network are scaled down respectively. This can have a negative effect with respect to the overall difficulty of the resulting optimization problem: if many matching pairs have distances between m_{pull} and $m_{\text{pull}} + 1$ all accompanying gradients get scaled down. It therefore seems to be a good idea to consider a pure hinge loss for the part concerning the matching pairs:

$$\sum_{x_1, x_2 \in \mathcal{M}} [d_\theta(x_1, x_2) - m_{\text{pull}})]_+ \tag{4.12}$$

resulting in the $\ell_{\text{DrLim}^{++}}(\theta)$ loss:

$$\begin{aligned}\ell_{\text{DrLim}^{++}}(\theta) = \sum_i \big[& y^i [d_\theta(x_1^i, x_2^i) - m_{\text{pull}}]_+ \\ & + (1 - y^i)[m_{\text{push}} - d_\theta(x_1^i, x_2^i)]_+^2\big]\end{aligned} \tag{4.13}$$

Due to symmetry the same reasoning can be applied to the part that pushes dissimilar pairs apart yielding $\ell_{\text{DrLim}^{+++}}(\theta)$:

$$\begin{aligned}\ell_{\text{DrLim}^{+++}}(\theta) = \sum_i \big[& y^i [d_\theta(x_1, x_2) - m_{\text{pull}}]_+ \\ & + (1 - y^i)[m_{\text{push}} - d_\theta(x_1, x_2)]_+\big]\end{aligned} \tag{4.14}$$

The usage of the hinge loss exposes the underlying task that is *approximately* solved by DrLim and the proposed variants: If a matching pair is above some threshold m it contributes one unit to a total loss. Symmetrically if a non-matching pair is not far enough apart, this pair also contributes one unit. In classification problems this loss function

is usually called the 0-1 loss. The 0-1 loss is non-convex (and non-continuous) so it is often approximated by a convex surrogate loss where the hinge loss is the most popular candidate.

Boosting algorithms use the *exponential loss* [35] as a surrogate loss for the 0-1 loss. Like the hinge loss it is convex with respect to the decision variable but also everywhere differentiable. Applied to the task at hand the exponential loss is given by

$$\ell_{\text{exp}}(\theta)) = \sum_i \exp(y_i' d_\theta(\mathbf{x}_1, \mathbf{x}_2)) \tag{4.15}$$

where $y' = 2y - 1$ and y indicates whether two samples $\mathbf{x}_1$ and $\mathbf{x}_2$ are a corresponding pair or not, i.e. $y' \in \{-1, 1\}$. This is the loss function used in two recent publications for compact descriptor learning [396].

Finally, another popular surrogate loss function for the 0-1 loss is the *log loss* [35], which is closely connected to the exponential loss:

$$\ell_{\log}(\theta) = \sum_i \log\left(1 + \exp(y_i' d_\theta(\mathbf{x}_1, \mathbf{x}_2))\right) \tag{4.16}$$

4.3 EXPERIMENTS

Using a deep Convolutional Network for Computer Vision tasks is always a valid decision. However, the question of the detailed configuration of such a network remains elusive. Already the choice of the correct cost function is challenging if the task is not the prototypical classification problem with a fixed number of object classes as demonstrated in the previous section. Decisions with respect to depth, filter sizes, number of feature maps, type of activation functions and so on heavily impact the overall performance of the network.

While these hyperparameter settings can be automatically determined with meta learning algorithms [356, 378] a systematic evaluation of such settings should help to improve the understand of Convolutional Network from an empirical point of view. This is even more so for tasks that are not in the already well explored space of object classification.

The first problem of a systematic evaluation for the various types of hyperparameters is the combinatorial explosion of possible settings. There is however a second, much bigger problem with a systematic evaluation: overfitting. At the end of a thorough evaluation it is simply not possible to report an unbiased performance score—the test set is touched too often [5]. The goal of this work is however not to report the best performance on the selected benchmark, it is the empirical evaluation of some basic design decisions. Therefore it is more interesting what effect the various settings have, compared to a basic model.

5 In Statistics this is referred to as *double dipping*.

This basic model is chosen with the help of a small training set and a very small evaluation set, both constructed by hand. More specifically, I selected 15000 patch pairs from every scene as training data (in total 45000 pairs) and 500 pairs from every scene as test data (the patches in these 1500 pairs in the test set are not contained in the large evaluation sets used later). On this newly created training and test set I evaluated a large number of possible basic configurations. The search space encompassed different cost functions ($\ell_{\text{DrLim}}(\theta)$, $\ell_{\text{DrLim}^+}(\theta)$ and $\ell_{\text{DrLim}^{++}}(\theta)$[6].), filter sizes ($3 \times 3$, 5×5 and 7×7), activation functions ($\tanh(\cdot)$ and $\text{ReLU}(\cdot)$), depth (3 and 4 convolutional layers) and number of feature maps (20, 30 and 40 feature maps per convolutional layer). The various models had between 120000 and 150000 parameters, so at least an order of magnitude smaller than standard Convolutional Networks as of 2015. Optimization of the respective architecture was done with stochastic gradient descent with a mini batch size of 100 and a learning rate $\eta = 0.01$. Every candidate architecture was trained for exactly 15 epochs and then evaluated on the test set. The combined fp@95 error rate on all three scenes was then used as the final evaluation criteria. The architecture that performed best had the following configuration:

- A convolutional layer with 40 5×5 filters, followed by a standard 2x2 max-pooling operation. The activation function is tanh.
- A convolutional layer with 40 5×5 filters, followed by a standard 2x2 max-pooling operation. The activation function is tanh.
- A convolutional layer with 40 4×4 filters, followed by a standard 2x2 max-pooling operation. The activation function is tanh.
- A convolutional layer with 64 5×5 filters, which reduces every feature layer to a scalar (Why 64? Because $64 = 2 \times 32$). The activation function is tanh.
- Finally a fully connected linear layer of size 64 x 32.
- The cost function is $\ell_{\text{DrLim}^{++}}(\theta)$, with $m_{\text{pull}} = 1.5$ and $m_{\text{push}} = 5$.

The output of the four layer network is 32 dimensional real valued vector. Assuming 4 bytes of memory requirements for every float this compact patch descriptor takes up as much memory the the 128 *byte* descriptor of SIFTb.

Given this basic architecture the following variations were systematically investigated on the training sets with 250000 matching and non-matching pairs each and the respective test sets from the three scenes:

- Loss functions.

6 Luckily (as it becomes clear later), I had 3 GPUs available for this kind of evaluation and therefore it was possible to include also $\ell_{\text{DrLim}^{++}}(\theta)$

- Activation functions.
- Pooling operations.
- Optimization procedures.
- Depth.
- Data augmentation.

I only varied one of these aspects at any point in time. The other aspects used the settings from the reference model.

TRAINING AND EVALUATION PROTOCOL. For both training and evaluation, every image patch (represented as a real valued vector) is preprocessed by subtracting its mean value and dividing by its standard deviation. The evaluation protocol is the same as used by the baseline models (see section 4.1.1): train on one dataset and evaluate the learned model on the remaining two datasets, reporting the false positive rate at 95% retrieval. Training a deep Convolutional Network usually uses a validation set (in the specific setting of this work for avoiding overfitting on the training set, i.e. *early stopping*). In order to be able to use all available training data from one dataset, I utilize the respective third scene as a validation set when evaluating on the second scene. Note that this is a rather difficult setup, as training set, validation set and test set are all from different distributions. Training happens always until the optimization converges on the training set or shows clear overfitting signs on the validation set.

LOSS FUNCTIONS. Based on the systematic development of suitable loss functions for the correspondence problem in section 4.2.1 the following loss functions where investigated with the base architecture.

- $\ell_{\text{DrLim}}(\theta)$, that is
$$\sum_i y^i d_\theta(x_1, x_2)^2 + (1 - y^i)[m_{\text{push}} - d_\theta(x_1, x_2)]_+^2$$
- $\ell_{\text{DrLim}^+}(\theta)$, that is
$$\sum_i y^i [d_\theta(x_1, x_2) - m_{\text{pull}}]_+^2 + (1 - y^i)[m_{\text{push}} - d_\theta(x_1, x_2)]_+^2$$
- $\ell_{\text{DrLim}^{++}}(\theta)$, that is
$$\sum_i y^i [d_\theta(x_1, x_2) - m_{\text{pull}}]_+ + (1 - y^i)[m_{\text{push}} - d_\theta(x_1, x_2)]_+^2$$
- $\ell_{\text{DrLim}^{+++}}(\theta)$, that is
$$\sum_i y^i [d_\theta(x_1, x_2) - m_{\text{pull}}]_+ + (1 - y^i)[m_{\text{push}} - d_\theta(x_1, x_2)]_+$$

Table 4.2: fp@95 error rates for different kind of loss functions. For comparison, SIFT and CVX from the set of baseline models are also shown.

	LY		**ND**		**HD**	
	ND	HD	LY	HD	LY	ND
$\ell_{\text{DrLim}}(\theta)$	11.4	19.0	18.1	17.1	22.1	13.2
$\ell_{\text{DrLim}^{+}}(\theta)$	10.3	17.6	14.8	13.6	19.4	10.6
$\ell_{\text{DrLim}^{++}}(\theta)$	8.9	16.7	14.1	14.9	16.7	8.9
$\ell_{\text{DrLim}^{+++}}(\theta)$	10.9	18.5	15.8	16.2	20.4	9.8
$\ell_{\text{exp}}(\theta)$	53.9	65.4	60.1	63.6	67.0	56.6
$\ell_{\log}(\theta)$	41.0	53.2	49.1	52.2	55.5	46.0
SIFT	20.9	24.7	28.1	24.7	28.1	24.7
CVX	9.1	14.3	14.2	13.4	16.7	10.0

- $\ell_{\text{exp}}(\theta)$, that is

$$\sum_i \exp(y_i' d_\theta(x_1, x_2)) \tag{4.17}$$

- $\ell_{\log}(\theta)$, that is

$$\sum_i \log\left(1 + \exp(y_i' d_\theta(x_1, x_2))\right) \tag{4.18}$$

Table 4.2 shows the final result for this type of evaluation. The theoretical consideration for relaxing the push loss (manifold assumption) lead to a significant improvement (compare $\ell_{\text{DrLim}}(\theta)$ and $\ell_{\text{DrLim}^{+}}(\theta)$). In particular, $\ell_{\text{DrLim}^{++}}(\theta)$ (which was already identified with the small model selection dataset) performed best, having a standard hinge loss for the positive pairs but a squared hinge loss for the negative pairs. It is difficult to assess why the squared hinge loss for the negative pairs performs better than the standard hinge loss for negative pairs.

The low performance of both the exponential and logistic loss stems from an underfitting problem. Widely used remedies for this problem (that are strictly speaking not allowed in to be used in this evaluation), e.g. more powerful optimizers, different kinds of initializations or various forms of regularizations did not help.

ACTIVATION FUNCTIONS. The standard activation functions for forward as well as recurrent networks have the well-known sigmoid shape: Both the logistic sigmoid and the tangens hyperbolicus not only have mathematically nice properties (the derivatives are bounded

and exist everywhere) but are also computationally efficient to evaluate (also with respect to their derivatives). However, recent deep learning systems [207] for object recognition identified the rectifying linear unit (ReLU) [245, 268, 116] and its *multivariate* generalization, Max-Out [120], as crucial for achieving state-of-the-art results. While some theoretical results [265] try to argue why ReLU is a good activation function (apart from the fact that it does not suffer from the vanishing gradient problem), on a practical side it speeds up the training time due to its simple form and hence allows the training of bigger models with the same computational budget. This advantage is however not relevant here as the architecture is deliberately kept small.

A systematic investigation of different activation functions for a given problem is usually not done. Using the basic architecture I do so and look at the following candidates (see Section 2.7 for their mathematical definitions):

- the logistic sigmoid function.
- the Tangens Hyperbolicus (the activation function of the reference model).
- the rectifying linear unit (ReLU).
- the Max-Out activation function, applied to *disjunctive groups* of *feature maps* of one convolutional layer.
- the Local-Winner-Take-All (LWTA) [366] function, also applied to disjunctive groups of feature maps of one convolutional layer.

Table 4.3 shows that tanh performs much better than the other possible functions, in particular better than ReLU. One possible explanation might be that the small model size is overall problematic for ReLU activations: the ReLU activation usually leads to a larger number of inactive units (i.e. they produce a 0 for a given input), which effectively limits the capacity of the model. From a statistical point of view already at the beginning of training half of all units will be inactive[7]: the preactivation can be modeled as a Gaussian with mean zero. So only about $\frac{1}{4}$th of all units can be utilized for training (in order to get gradient information for a given weight, both units connected by this weight must be non-zero). In large models this implicit capacity reduction is not a problem (and actually might be helpful to avoid overfitting), but with small models as in this work the performance may be limited by this fact. First experiments with models 4 to 6 times larger as the reference model show improvements in the range of 1% to 1.5% for ReLU based models (but do not show any further improvements for tanh).

7 For every sample, a different set of units will be inactive, but the basic argument holds.

Table 4.3: fp@95 error rates for different kinds of activation functions. Both Max-Out and LWTA activation functions are applied to groups of two feature maps. σ denotes the logistic sigmoid function.

	LY		**ND**		**HD**	
	ND	HD	LY	HD	LY	ND
σ	16.5	24.3	25.6	24.4	27.2	17.5
tanh	8.9	16.7	14.1	14.9	16.7	8.9
ReLU	10.6	18.6	17.0	17.5	18.0	10.2
Max-Out	9.3	17.5	16.6	17.3	18.1	9.5
LWTA	12.8	20.7	20.4	19.9	21.5	13.3
SIFT	20.9	24.7	28.1	24.7	28.1	24.7
CVX	9.1	14.3	14.2	13.4	16.7	10.0

POOLING OPERATIONS. Pooling is an important operation for the matching task, as the various patches show a high range of different scales—so pooling is not only responsible for translation invariance but also for scale invariance in this case. In principle pooling implements a spatial dimensionality reduction restricted to the dominant feature space of the Convolutional Network. More formally, if we consider a layer $\mathbf{m}$ of some arbitrary Convolutional Network with dimensions $f \times w \times h$ (f is the number of feature maps in a layer, w is the width and h is the height of one feature map) then a pooling operation with pooling size p and stride s can be defined in the following elementwise way:

$$\pi(\mathbf{m})_{k,i,j} = \left(\sum_{x=0}^{p} \sum_{y=0}^{p} \mathbf{u}_{x,y}^{k} |\mathbf{m}_{k,si+x,sj+y}|^{l} \right)^{\frac{1}{l}} \tag{4.19}$$

$\mathbf{u}$ is hereby a three dimensional non-negative weigh tensor and $\mathbf{u}_{x,y}^{k}$ may depend on the value of $\mathbf{m}_{k,si+x,sj+y}$. $\pi(\mathbf{m})_{k,i,j}$ then denotes the value of the pooled feature layer m in feature map k at position (i, j). Notably if p is uneven the above formula can be rewritten:

$$\pi(\mathbf{m})_{k,i,j} = \left(\sum_{x=-p/2}^{p/2} \sum_{y=-p/2}^{p/2} \mathbf{u}_{x,y}^{k} |\mathbf{m}_{k,ri+x,rj+y}|^{l} \right)^{\frac{1}{l}} \tag{4.20}$$

As pointed out recently [363] this can be written even more generally like a standard convolution operation in a Convolutional Network:

$$\pi(\mathbf{m})_{k,i,j} = \left(\sum_{x=-p/2}^{p/2} \sum_{y=0}^{p} \sum_{k'} \hat{\mathbf{u}}_{x,y}^{k,k'} |\mathbf{m}_{k',ri+x,rj+y}|^{l} \right)^{\frac{1}{l}} \tag{4.21}$$

$\hat{\mathbf{u}}$ is now a 4-dimensional weight tensor. Eq. (4.21) still implements a spatially restricted pooling operation (i.e. the number of feature maps in layer $\mathbf{m}$ is not reduced). In particular, the pooling operation from Eq. (4.20) can be recovered by setting most entries of $\hat{\mathbf{u}}$ to 0:

$$\hat{\mathbf{u}}_{x,y}^{k,k'} = \begin{cases} 1, & \text{if } k = k' \\ 0, & \text{otherwise.} \end{cases}$$

In the following we consider 5 different types of specific instantiations of Eq. (4.20) or Eq. (4.21).

- Max-Pooling (mp): Currently the most widely used pooling operation is max-pooling [414, 309, 328, 64]. Considering Eq. (4.20), max-pooling can be realized by letting $l \to \infty$ (i.e. the maximums norm) and setting all entries of $\mathbf{u}$ to 1.

- Average-pooling (ap): With $l = 1$ and $\mathbf{u}_{x,y}^{k} \equiv \frac{1}{p^2}$ a simple averaging operation over a pooling region [221] is implemented. The absolute value avoids that canceling of positive and negative values happens [182].

- Overlapping max-pooling (ov-mp) realizes the standard max-pooling with a stride size s that is *smaller* than the pooling size p. Similar to recently successful Convolutional Network-architectures [207], max-pooling with a pooling size of 3 and a stride size of 2 is investigated.

- Stochastic max-pooling (sto-mp) [428]: Standard max-pooling only considers the maximal element in a pooling region—this hard thresholding operation neglects many values that should be considered during computation, e.g. because they are very close to the maximum value in their pooling region. Stochastic max-pooling ensures that such values are also utilized and implements a *multimodal* pooling operation. During the training phase, $\mathbf{u}_{x,y}^{k}$ is *sampled probabilistically* for every input from a multinomial distribution. This multinomial is defined by the value of $m_{k,ri+x,rj+y}$ for $x, y \in [0, p] \times [0, p]$. More precisely, the probability mass function of the random variable $\mathbf{U}^k$ is given by

$$p(\mathbf{u}_{x,y}^{k} = 1) = \frac{e^{m_{k,ri+x,rj+y}}}{\sum_{x=0}^{p} \sum_{y=0}^{p} e^{m_{k,ri+x,rj+y}}} \tag{4.22}$$

  At test time the elements of a given pooling region are summed with weights $\mathbf{u}_{x,y}^{k}$ defined by the probabilities from Eq. (4.22).

- All convolutional (all-conv) pooling [363]. Pooling can be considered as a special case of convolution (see Eq. (4.21)). The *all-conv* model substitutes the specialized pooling layer by a

Table 4.4: fp@95 error rates for different kinds of pooling operations. Max-Pooling is used in the reference architecture.

	LY		**ND**		**HD**	
	ND	HD	LY	HD	LY	ND
mp	8.9	16.7	14.1	14.9	16.7	8.9
ap	12.5	20.4	16.6	16.8	19.2	11.7
ov-mp	17.4	24.5	22.0	22.1	26.0	17.1
st-mp	16.9	24.3	36.1	28.1	38.3	22.8
all-conv	9.2	18.8	18.7	18.4	19.7	9.4
SIFT	20.9	24.7	28.1	24.7	28.1	24.7
CVX	9.1	14.3	14.2	13.4	16.7	10.0

standard convolutional layer, with stride 2 in the case at hand. This convolutional layer is also followed by an elementwise non-linearity, which is tanh in this set of experiments[8].

Table 4.4 shows the results for the various types of pooling operations. Standard max-pooling (mp) outperforms the other pooling operations (similar result is obtained when the task is object classification [328]). This might come as no surprise because max-pooling is apriori best suited for scale invariance. The results for stochastic max-pooling must be considered with a word of warning: Training one model with this pooling operation takes a very long time (about 4 days on the available hardware setup), therefore only one experiments for every training/test set is realized—it may well be the case that a large amount of underfitting is present in this specific results. Furthermore it might be that the inference procedure for Eq. (4.22) is leading to suboptimal results and that instead a proper Monte Carlo approximation should be used. This in turn would require a distance measure that can handle probability densities. See Section 6.2 for some preliminary work in this direction.

OPTIMIZATION. Learning the parameters of a Convolutional Network is a difficult non-convex optimization problem. Interestingly, first-order methods are widely used optimization methods, though powerful second order methods have been specifically designed for deep learning problems [247, 336].

8 The recently published paper that suggested the all convolutional architecture [363] used rectifying linear units throughout the network. Experiments with this activation function performed worse, however.

From a theoretical point of view, first-order methods are suitable enough as long as one ensures that learning gets not stuck in regions that look like local minima but actually are not (i.e. saddle points [76]) [9]. However, this requirement may be simply fulfilled by training long enough on extremely large datasets. From a practical point of view, simple first-order methods allow scalable training of large models and large datasets and are simple to implement. Therefore, I only consider first-order optimization procedures:

- sgd: The stochasticity in *stochastic gradient descent* (sgd) [312, 37] comes from selecting a small set (i.e. a *minibatch*) of random samples from the training set when computing one gradient step. This stochasticity moves the network quickly through bad regions in weight space [375] and is therefore overall very popular. The update rule for one iteration of sgd is (assuming a fixed learning rate η):

$$\theta_t = \theta_{t-1} + \eta_t \delta\theta_{t-1} \tag{4.23}$$

  θ_t denotes the parameter values at iteration t and $\delta\theta_{t-1}$ denotes the gradient of the objective function with respect to the parameter values at iteration $t-1$.

- Momentum based sgd (m-sgd): In order to avoid spurious movements in certain directions in weight space a momentum term is built by collecting gradient directions from the optimization iterations. The weighted accumulation of the instantaneous directions leads to filtering out spurious directions and therefore smooths the overall optimization procedure. The momentum ν_t at iteration t is an exponential moving average with some decay factor $\mu \in [0, 1]$:

$$\begin{aligned} \nu_t &= \mu\nu_{t-1} + \eta_t \delta\theta_{t-1} \\ \theta_t &= \theta_{t-1} - \nu_t \end{aligned} \tag{4.24}$$

- Nesterov accelerated gradient (nag) was invented as a descent method that converges quadratically for *non-stochastic, convex* problem domains [275]. For stochastic optimization problems it is instructive to express the update rules as follows [375]:

$$\nu_t = \mu\nu_{t-1} + \eta\nabla_\theta \ell(\theta_{t-1} + \mu_t \nu_{t-1}) \tag{4.25}$$
$$\theta_t = \theta_{t-1} - \nu_t \tag{4.26}$$

  At a coarse glance this approach looks like standard momentum based gradient descent. However, nag *looks ahead* when computing the derivative of the loss at time t and thus avoids

9 Note that it is not considered a disadvantage that one can only identify *local* minima when training neural networks, as was pointed out several times in Section 2.7.

high entries for the momentum vector which easily lead to overshooting during difficult optimization phases—instead it allows a much more timely correction of accumulated momentum directions. As it turns out this equation can be rewritten such that *looking ahead* is no longer specifically necessary [29].

- RMSprop. It would be beneficial to have an adaptable learning rate *per parameter*. A straightforward and theoretically correct approach would be to take the diagonal hessian for determining individual learning rates [21]. A different, more heuristically driven approach is RProp [308], but it fails with stochastic mini-batches. A conceptually simple heuristic for a stochastic variant of RProp was recently introduced under the name RMSProp [388]:

$$\theta_t = \theta_{t-1} + \frac{\eta_t}{\text{RMS}(g)_t}\delta\theta_{t-1} \tag{4.27}$$

with

$$\text{RMS}(g)_t = \sqrt{E(g^2)_t + \varepsilon} \tag{4.28}$$

where a small constant $\varepsilon \approx 1e-6$ is added for improving the condition of the denominator [22]. $E(g^2)_t$ is an efficient approximation of the diagonal Hessian, implemented as a simple exponential moving average of the elementwise squared gradient:

$$E(g^2)_t = \rho E(g^2)_{t-1} + (1-\rho){g_t}^2 \tag{4.29}$$

with $g_t \equiv \delta\theta_t$, the update direction at iteration t. ρ is a *decay* constant.

- AdaDELTA. From a theoretical point of view, RMSProp computes a gradient step that is inconsistent with respect to units: parameters and gradients of these parameters do not fit together [427]. The updates introduced by AdaDELTA ensure that the quantities involved in computing the gradient step fit together with respect to units:

$$\theta_t = \theta_{t-1} + \frac{\text{RMS}(\Delta\theta)_{t-1}}{\text{RMS}(g)_t}\delta\theta_t \tag{4.30}$$

Similar to RMSprop, $\text{RMS}(\Delta\theta_t)$ is defined as

$$\text{RMS}(\Delta\theta_t) = \sqrt{E((\Delta\theta)^2)_t + \varepsilon} \tag{4.31}$$

with the same ε as used previously. $E((\Delta\theta)^2)_t$ is also an exponential moving average with the previous decay constant ρ:

$$E((\Delta\theta)^2)_t = \rho E((\Delta\theta)^2)_{t-1} + (1-\rho)(\Delta\theta_t)^2 \tag{4.32}$$

Table 4.5: fp@95 error rates for different kinds of optimization procedures. sgd is the method used in the reference model, with learning rate $\eta = 0.1$. A momentum term of 0.9 is used where applicable. ρ is set to 0.9. The learning rate for RMSprop is $1e-5$, for AdaDELTA it is 0.1. AdaDELTA denotes the variant described in the text.

	LY		**ND**		**HD**	
	ND	HD	LY	HD	LY	ND
sgd	8.9	16.7	14.1	14.9	16.7	8.9
m-sgd	10.1	17.6	14.1	14.7	17.3	9.6
nag	10.8	18.5	18.0	18.2	18.7	10.9
RMSprop	8.9	15.6	15.3	13.8	16.4	8.6
AdaDELTA	8.6	15.9	13.6	13.9	16.6	8.6
SIFT	20.9	24.7	28.1	24.7	28.1	24.7
CVX	9.1	14.3	14.2	13.4	16.7	10.0

Note that $\text{RMS}(\Delta\theta_{t-1})$ must be used. This may lead to a more robust behavior of the updates in case of large (sudden) gradients, as these are directly damped by the denominator [427]. However, in my experiments it turned out that this dampening effect had a negative impact on the overall performance: Models trained with AdaDELTA showed very good performance development on the validation set in the first few epochs but suddenly stalled. I was able to circumvent this problem by using $\text{RMS}(g)_{t-1}$ in the denominator instead.

Table 4.5 summarizes the results for the set of optimizers. AdaDELTA performs best, improving significantly in particular on the difficult training/test set combinations. It also outperforms the strongest baseline model, CVX by a small overall margin! RMSProp performs extremely well during the first 10 epochs but starts to heavily fluctuate. I briefly investigated different schemes of annealing the learning rate in this case, without any success, though. RMSprop is the optimization procedure where early-stopping needs to be employed, for all other methods no overfitting tendencies with respect to training error were observed.

DEPTH. The depth (that is, the number of layers) of a model can be varied in two principled ways: Changing the number of convolutional layers or adding fully connected layers to the stack of convolutional layers. When only considering different numbers of convolutional layers I ensure that the overall number of parameters stays roughly the

same compared to the base model. However, I don't try to balance capacity when adding fully connected layers. In this section I consider the following configurations with respect to depth:

- L4, the reference model with 4 convolutional layers and one final linear layer.
- L3, 3 convolutional layers, with (roughly) the same number of parameters as the reference model. Specifically, the first layer has 64 5×5 filters, followed by a max-pooling operator with stride 3. The second has the same filter configuration but is followed by a max-pooling operator with stride 2. The third layer has 32 5×5 filters followed by the final linear layer.
- L5, 5 convolutional layers, again adjusted for the number of parameters with respect to L4. Overall the filter sizes had to become smaller. The first layer has 64 3×3 filters, followed by three convolutional layers with 64 2×2 filters each. The fifth convolutional layer has 128 3×3 filters. All max-pooling operations have stride 2.
- L7, 7 convolutional layers. The first two convolutional layers have 32 3×3 filters. The next convolutional layer has 64 2×2 filters, followed by 2 layers with 64 2×2 filters, a layer with 64 3×3 filters and a final convolutional layer with 64 2×2 filters. The max-pooling operations after the second and third layer are missing.
- FC1 enhances L4 with one fully connected layer with 256 units. The last convolutional layer is adapted to this setting, it no longer contains 5×5 filters, but instead 4×4 filters. With 64 feature maps (and no max-pooling operation) this results in 256 convolutional output units.
- FC2 enhances the reference model with two fully connected layers. In the spirit of FC1, another hidden layer with 256 units is added.

It is clearly visible that a certain amount of depth helps, both in the convolutional stack as well as with fully-connected layers. Both FC1 and FC2 clearly outperform CVX, but both models do have substantially more parameters than the reference model.

DATA AUGMENTATION For training a deep Neural Network labeled data can never be enough. In the specific setting of this chapter a simple (and highly effective) way to extend the training data is to fuse two of the three scenes into one training set. In the training protocol of the base model the validation scene is missing, however. Using

Table 4.6: fp@95 error rates for models with different overall depth. L4 is the reference model. For FC1 and FC2 the learning rate η is set to 0.001.

	LY		**ND**		**HD**	
	ND	HD	LY	HD	LY	ND
L3	14.3	23.3	27.4	26.3	27.5	16.5
L4	8.9	16.7	14.1	14.9	16.7	8.9
L5	8.9	17.2	14.8	15.3	16.9	9.2
L7	9.3	18.2	21.0	20.3	19.6	11.4
FC1	7.7	14.7	13.1	13.1	15.2	7.9
FC2	8.1	14.6	13.5	13.7	16.0	8.0
SIFT	20.9	24.7	28.1	24.7	28.1	24.7
CVX	9.1	14.3	14.2	13.4	16.7	10.0

the standard settings of the base model I forgo the usage of a validation set and train on the fused training set until the optimization procedure converges. The reference model can be substantially improved using this large dataset: On the Liberty evaluation set, 13.6% are achieved, on the Notre Dame dataset 7.4% and on the Half Dome evaluation set 12.5%.

Another way to handle models with high capacity is to utilize prior knowledge and encode this knowledge into the overall learning procedure. For example a convolutional Network encodes the prior knowledge that the input is spatially structured according to a 2d lattice. A different (and easy to realize in practice) approach to encode prior information is to augment the training data in a manner consistent with this prior knowledge: given a labeled training sample more labeled samples can be generated by consistent functional transformations of this sample. Of course this not always possible, usually one resorts to Machine Learning because these valid transformations of the input space are not known.

For images a wide set of invariant transformations are known, that, at least for object classification, encode prior knowledge. Generating additional valid training data through translating, rotating and changing color information is currently considered another key enabler for the recent success of deep learning approaches for Computer Vision approaches [63, 64, 60, 209].

However, with the dataset at hand translations and rotations can not be freely chosen, though: Matching pairs were chosen according to strict rules that are defined with respect to translational, rotational

and scale characteristics of the keypoint patches. I therefore limit possible translations to at most 5 pixels and rotations are only possible in full amounts of 90 degrees (and both patches in a pair are rotated in an identical way). Nonetheless, even with this specific schedule for generating additional data, I was not able to improve on the best performance. Instead, as it turns out, performance degraded significantly on the reference model. Overall, the absolute performance number decrease by at least 10%. The reason for this unexpected drop in performance might be the highly specific way the dataset was constructed—introducing additional synthetic data may introduce further statistical aspects that can not be handled by the compact base model.

COMPRESSION THROUGH BINARIZATION. While the learned descriptor is already quite compact, an improvement with respect to both memory requirements as well as matching speed can be achieved if the descriptor is inherently binary. In this case, for example, matching two descriptors is realized through a simple binary XOR operation, which is usually available as an extremely efficient low-level operation on current computer architectures[10].

Learning such a binary representation can be achieved with the same cost function and the same architecture used previously, except for one tiny change: instead of a linear activation function at the end of the descriptor pipeline a sigmoid function is chosen. At evaluation time, every dimension then is binarized through a threshold function. The threshold is determined as the *median* activation vector over the training set. Note that this resembles the prior assumption that the marginal probability of a dimension having a binary one should be 0.5 (i.e. highest marginal entropy).

While straightforward (and very successful in other cases with large networks [230]), this approach is not competitive with respect to the state-of-the-art results, as Table 4.7 shows. Adding additional regularization terms that encourage for example saturating outputs or utilizing more powerful optimizers did not improve the evaluation performance. Considering the training performance of this approach it seems that the problem is that the network underfits. Solving this problem is left as future work.

A very different approach for finding binary representations is to use random projections of learned real valued descriptors [184, 351]. The idea is based on increasing the dimensionality of a given descriptor through multiplying with a random matrix and then simply thresholding the resulting higher dimensional representation (i.e. only taking the sign).

10 This instruction is called POPCOUNT: After an XOR operation between two bit strings, POPCOUNT counts the number of bits set to 1 in the result (i.e. the Hamming distance) very efficiently.

Table 4.7: Error rates for binary descriptors. The table shows the percent of incorrect matches when 95% of the true matches are found. CVX-64b denotes the 46-dimensional CVX floating point descriptor projected to 64bits. L4-σ denotes the binarized descriptor (32 dimensions) learned from scratch. L4-32b and L4-64b denote binarized descriptors obtained by random projections of the 32-dimensional floating point descriptor of the reference model. BinBoost [397] is a 64-dimensional binary descriptor learned with boosting.

	LY		**ND**		**HD**	
	ND	HD	LY	HD	LY	ND
SIFT	22.8	25.6	31.7	25.6	31.7	25.6
L4-σ	32.5	39.7	37.9	36.1	36.3	30.4
L4-32b	23.9	33.1	33.1	33.6	33.0	23.5
L4-64b	15.3	23.6	23.7	19.3	23.7	14.9
SIFTb	22.8	25.6	31.7	25.6	31.7	25.6
BinBoost	16.9	22.9	20.5	19.0	21.7	14.6
CXV-64b	15.2	24	20.4	18.5	23.5	14.4

More specifically, a *Parseval tight frame* is some matrix $F \in \mathbf{R}^{q \times d}$ with $q \geqslant d$ and $F^T F = \mathbf{I}_d$. The new representation $Ff_\theta(\mathbf{x})$ (i.e. the matrix F gets left multiplied to the d-dimensional representation of $f(\mathbf{x})$, leads to an *overcomplete* representation that *preserves Euclidean distance*, though:

$$\begin{aligned} \|Ff_\theta(\mathbf{x}_1) - Ff_\theta(\mathbf{x}_2)\|^2 &= (f_\theta(\mathbf{x}_1) - f_\theta(\mathbf{x}_2))F^T F(f_\theta(\mathbf{x}_1) - f_\theta(\mathbf{x}_2)) \\ &= \|f_\theta(\mathbf{x}_1) - f_\theta(\mathbf{x}_2)\| \end{aligned} \tag{4.33}$$

The binarized version of $Ff_\theta(\mathbf{x}_1)$ then is simply

$$f^b_\theta(\mathbf{x}) = \operatorname{sgn}(Ff_\theta(\mathbf{x})) \tag{4.34}$$

where $\operatorname{sgn}(\cdot)$ is the sign function, i.e.

$$\operatorname{sgn}(x) \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{4.35}$$

Note that the descriptors $\{f_\theta(\mathbf{x})\}_\mathcal{X}$ must be zero centered in order to get a meaningful distribution of binary vectors. This is achieved through computing the mean descriptor on the respective training set.

Clearly this binarization procedure can only approximate the true distance $\|f_\theta(\mathbf{x}_1) - f_\theta(\mathbf{x}_2)\|$ for two patches $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$. The more overcomplete the representation (i.e. the larger q is) the better this approximation will get. For a given binary descriptor $f^b(\mathbf{x})$ its memory

footprint will be smaller then the one of the original 32 dimensional floating point descriptor as long as $q < 1024$ (32 bits for every floating point dimension). Table 4.7 shows performance results for various values of q. The underlying floating point descriptor is computed using the models trained with the AdaDELTA optimization procedure (see Table 4.5). It shows that descriptors binarized in this way perform comparable to other state-of-the-art approaches.

FULLY CONNECTED ARCHITECTURE. I end this long list of computational experiments with a brief review of a different kind of network architecture, a simple fully connected deep forward network.

The networks are trained with the $\ell_{\text{DrLim}^{++}}(\theta)$ cost function, again with the goal of finding a 32 dimensional real valued descriptor for the image patches. In order to avoid overfitting (the 4096 dimensional input induces a large number of parameters apriori) I choose the following two approaches to reduce the input dimensions.

- Resizing patches to 32×32 pixels. This reduction of the input size to 1024 dimensions allows a deep network with 5 layers that has about 750000 parameters: 1024-512-256-256-128-32. The nonlinear activation function is again tanh, training is done with AdaDELTA, with a learning rate $\eta = 1$ and a decay factor of $\rho = 0.9$. Figure 4.6 shows some random receptive fields from the first layer for the network trained on the Half Dome training set. Interestingly these are focused around the keypoint (i.e. the center) of a given patch.

- Reducing the dimensionality with Principal Component Analysis (PCA). As it turns out only between 350 and 500 principal components are necessary to cover 95% of the variance within a training set. After transforming the input data to its lower dimensional representation a deep network with 3 layers computes the 32 dimensional descriptor. Training is again done with AdaDELTA ($\eta = 1$ and $\rho = 0.9$).

The results for both models are shown in Table 4.8. Compared to the results achieved by a Convolutional Network and compared to the standard SIFT descriptor, both models do not perform very well. However, I could not spent a lot of time to run a large number of different experiments with this architecture, so it is quite likely that these results can be improved substantially, in particular if the best architecture is selected following the same approach as with the reference model for Convolutional Networks. It is interesting that filters from the first layer focus solely on a very small area around the patch center (Figure 4.6). In order to foster a more diverse set of filters a L1 prior on the first layer-weights might be a good idea, possibly combined with a penalty term for the first hidden layer derived from sparse coding.

Table 4.8: Error rates for fully connected networks. FC is the fully connected network with 5 hidden layers, FC-PCA denotes the architecture that has PCA-transformed inputs.

	LY		**ND**		**HD**	
	ND	HD	LY	HD	LY	ND
FC	23.8	29.5	28.3	28.9	31.2	25.1
FC-PCA	37.2	46.5	43.9	42.0	46.3	36.9
SIFT	20.9	24.7	28.1	24.7	28.1	24.7
CVX	9.1	14.3	14.2	13.4	16.7	10.0

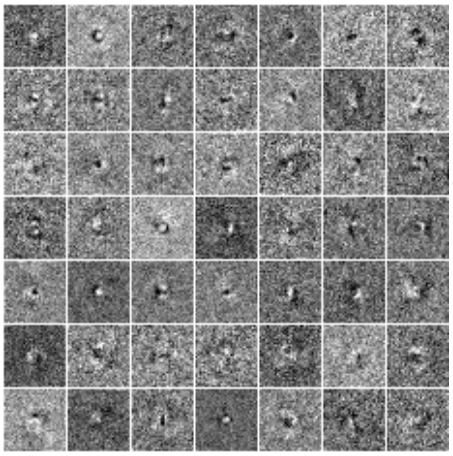


Figure 4.6: Some filters from the first layer of the fully connected network with 5 layers. The network focuses on the area around the center where the reference-keypoint of every patch is defined.

4.4 QUALITATIVE ASSESSMENT

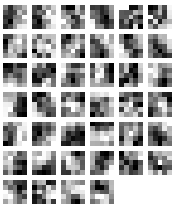
In this section, I briefly give some qualitative characterizations of the learned convolutional architecture.

- Filters: It is often instructive to visualize the learned filters, as these can indicate what is actually going on in the model (see also the receptive fields in the fully connected model, Figure 4.6). For a Convolutional Network, all filters can be visualized, not only these from the first layer.
- Propagation path: What is happening to an image when it is propagated through the network? Again, for Convolutional Networks this path can be visualized in a simple way: Plotting the various feature maps after the max-pooling operation as two dimensional images.

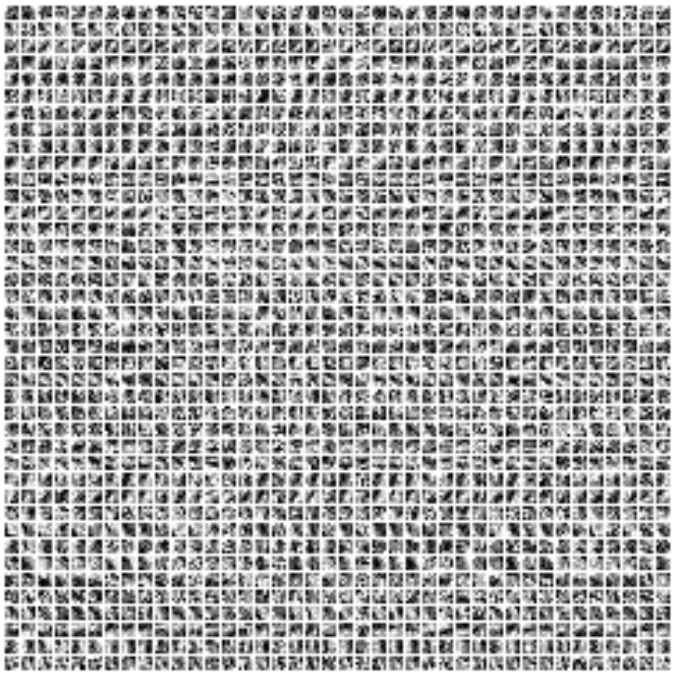
- Most offending image pairs: What are those image pairs that the network gets wrong the most? That is which true pairs are furthest apart (and therefore probably considered as non matching, i.e. *false negatives*) and, conversely, which false pairs are closest (i.e. considered matching, i.e. *false positives*). These visualizations are accompanied by the respective overall distance histograms.

In more detail the model used for these qualitative evaluations is the reference model trained on the Half Dome training set, using the variant of AdaDELTA described previously as an optimizer (i.e the model from Table 4.5, last row). With respect to the filters of the different layers all investigated models give similar visual impressions. Figure 4.7 depicts the filters from the first and second convolutional layer. Overall the filters do no show any visually discernable patterns. Maybe some of the filters in the first layer might be interpreted as edge detectors but only with some amount of goodwill.

Figure 4.8 and Figure 4.9 show the transformation path of images through the network. Two arbitrary images from the Notre Dame training set are fed through the same model. Finally, the resulting distance histograms for the three training sets (reference model optimized with AdaDELTA) are shown in Figure 4.10 and derived from these histograms the corresponding most offending image pairs (both false negatives and false positives) are shown in Figure 4.11. The four image pairs with the smallest (in the case of false positive pairs) and the largest (in the case of false negative pairs) distances respectively are shown for the six combinations of training sets and evaluation sets. Even for humans some of the presented pairs would be difficult to get correctly classified. For example, for images from the Notre Dame evaluation set (Figure 4.11(f),(h)), repeated structural elements from *different* 3D points are easily falsely to form a matching pair.



(a)



(b)

Figure 4.7: Visualizations of the first two filter layers. (a) The 40 filters of size 5×5 forming the first convolution layer. Most of the filters do not look completely random, but it is challenging to detect specific patterns.(b) The second layer has 40 feature maps. Every feature map (one row in the figure) has 40 filters of size 5×5. Again, filters seem not to be random, but also do not show easily interpretable patterns. Best viewed electronically.

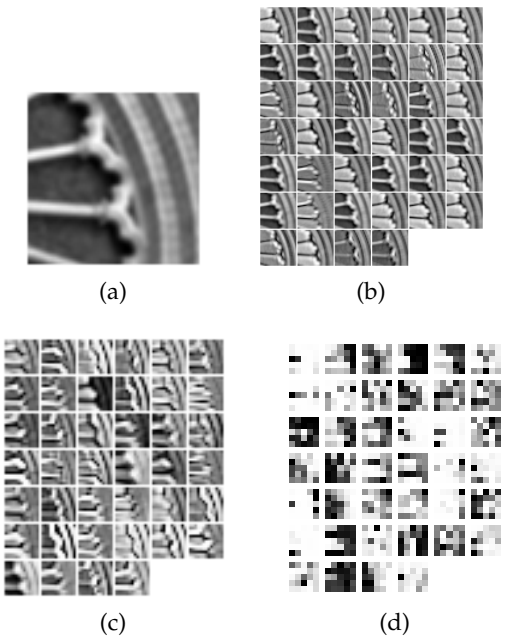


Figure 4.8: Visualizing the activations at the various layers produced by an input image. (a) The 64×64 inputs patch shows a detailed structural element of a window from Notre Dame. (b) After the first max-pooling layer a feature map has 30×30 pixels. (c) 40 feature maps of size 13×13 pixels after the second max-pooling layer. (d) The final max-pooling layer with 40 filters of size 5×5 pixels. Best viewed electronically.

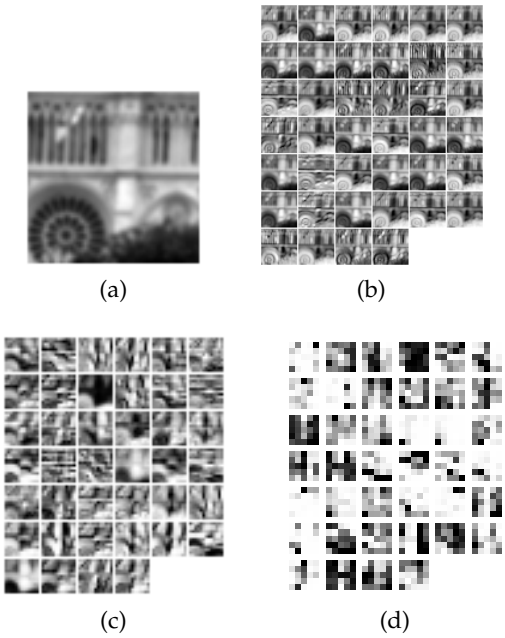
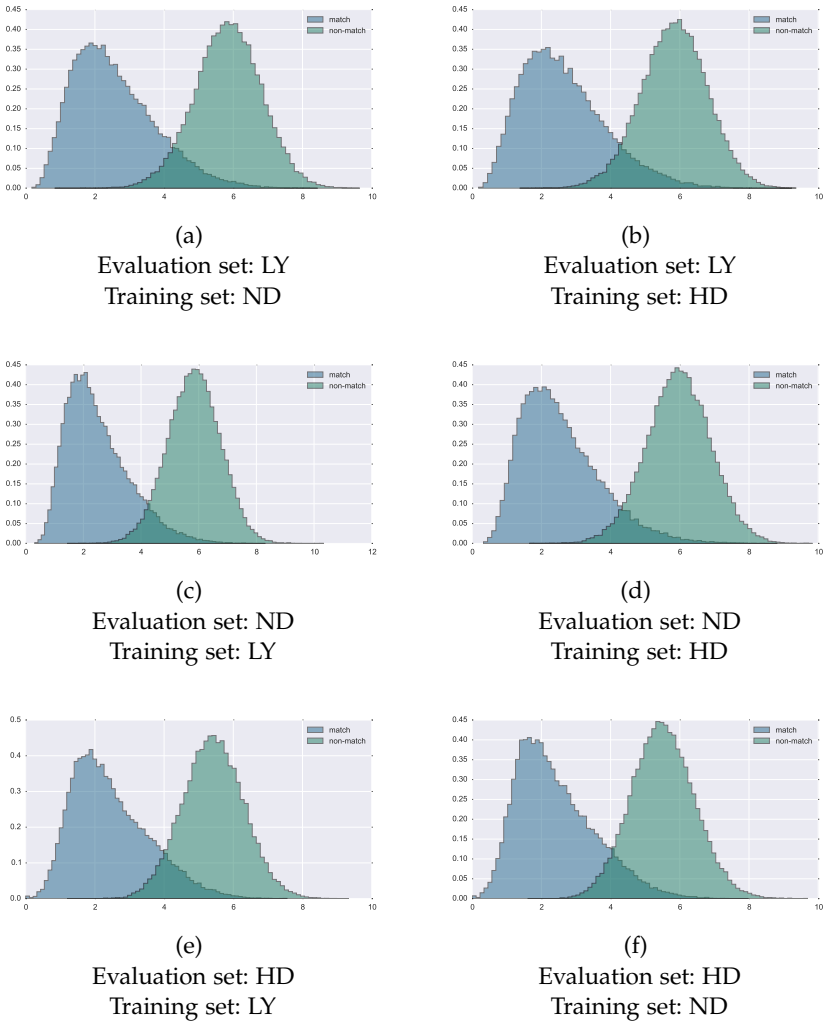


Figure 4.9: Visualizing the activations at the various layers produced by an input image. (a) The 64×64 input depicts content at a large scale. (b) After the first max-pooling layer a feature map has 30×30 pixels. (c) 40 feature maps of size 13×13 pixels after the second max-pooling layer. (d) The final max-pooling layer with 40 filters of size 5×5 pixels. Compared to Figure 4.8, in particular the second layer behaves qualitatively very differently on this type of input. Best viewed electronically.



(a)
Evaluation set: LY
Training set: ND

(b)
Evaluation set: LY
Training set: HD

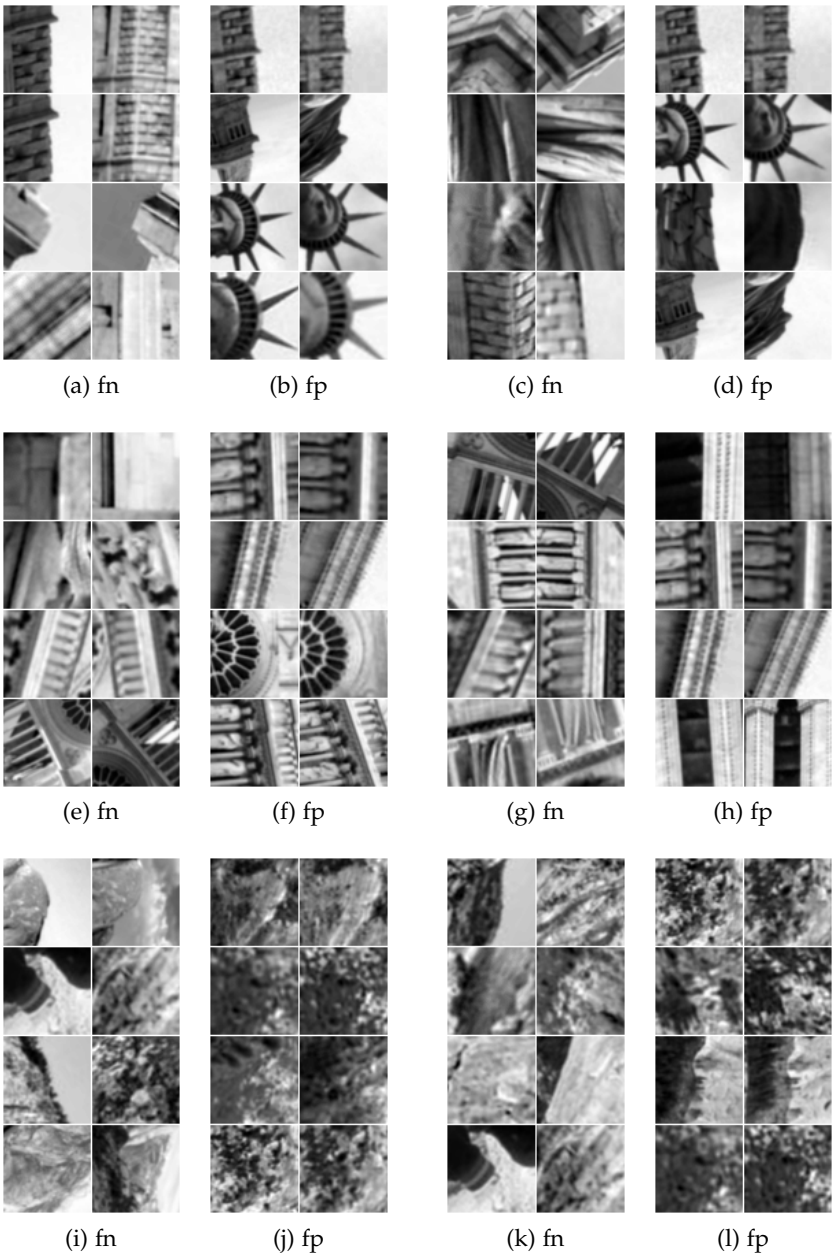
(c)
Evaluation set: ND
Training set: LY

(d)
Evaluation set: ND
Training set: HD

(e)
Evaluation set: HD
Training set: LY

(f)
Evaluation set: HD
Training set: ND

Figure 4.10: Distance histograms for the three evaluation sets (Liberty (first row), Notre Dame (second row) and Half Dome (third row)), using the reference model optimized with the variant of AdaDELTA from Table 4.5. The x-axis is the euclidean (L2) distance, the y-axis are normalized counts. Best viewed electronically.



(a) fn (b) fp (c) fn (d) fp

(e) fn (f) fp (g) fn (h) fp

(i) fn (j) fp (k) fn (l) fp

Figure 4.11: False negatives (fn) and false positives (fp), the four most *wrong* image pairs for every evaluation set/training set combination. Image pairs are shown next to each other. The first row shows image pairs from the Liberty evaluation set when the model is trained on the Notre Dame ((a), (b)) or the Half Dome ((c), (d)) training set. Similarly, the second row shows wrongly classified pairs from the Notre Dame scene with a model trained on the Liberty ((e), (f)) or the Half Dome ((g), (h)) training set. Finally, the third row shows false negatives and false positives from the Half Dome scene when the descriptor is trained on the Liberty ((i), (j)) or the Notre Dame ((k), (l)) training set. Best viewed electronically.

4.5 TRANSFER LEARNING

In this section I evaluate the learned descriptors on tasks it was not specifically trained for (*transfer learning*)[11]. The underlying goal of learning a low-level descriptor is to use it for different kinds of (high-level) tasks. In the best case the descriptor is integrated in a larger learning framework that itself has a deep structure and can be learned through backpropagation (or a more flexible framework [50]). Here, I consider two simpler tasks from the standard domain of object recognition. In the first task the new descriptor substitutes SIFT in the old classification pipeline from Figure 4.1. The dataset chosen here is the PASCAL Visual Object Classes [91, 92]: Its images have a large spatial extend, so a Bag-of-Word approach seems a good idea[12].

Indeed a much more interesting experiment would be to apply the learned descriptor *convolutionally* to these images and form a representation that is a three dimensional tensor with 32 channels. These representations would then be fed into a second Convolutional Network which is trained in a supervised manner on the associated classification task and *backpropagates information also into the low-level descriptor network*. However, due to limited computational resources, this approach must be postponed to future work. So the experiments performed in this case merely investigate how well the learned descriptor can replace SIFT in a typical use case for hand-engineered local image descriptors.

In the second task the new descriptor is directly used for object recognition on the STL-10 [66] dataset: Given (downscaled) images of objects, convert every image to a 32 dimensional descriptor and classify the image according to a simple nearest neighbor scheme. Conversely, given matching and non-matching pairs of images from STL (based on object identity), learn a compact representation with the base model and apply it to the low-level correspondence task.

4.5.1 *PASCAL VOC 2007*

The Pascal Visual Object Classes (VOC) [92] challenge is a benchmark in visual object category recognition and detection. The dataset from the year 2007[13] consists of annotated consumer photographs collected from FLICKR[14], a public photo storage website. There are two princi-

11 As of 2015 this has become a standard approach for many Computer Vision settings: Deep Convolutional Networks are trained on ImageNet and then adapted to the actual task.

12 This was true for 2013 when these experiments have been conducted. However, as of 2015, Deep Convolutional Networks pretrained (and even without that pretraining!) on ImageNet are the state-of-the-art approaches for the VOC challenge, too [145, 54].

13 Only the data for the 2007 competition has a test set with available ground truth annotations.

14 www.flickr.com

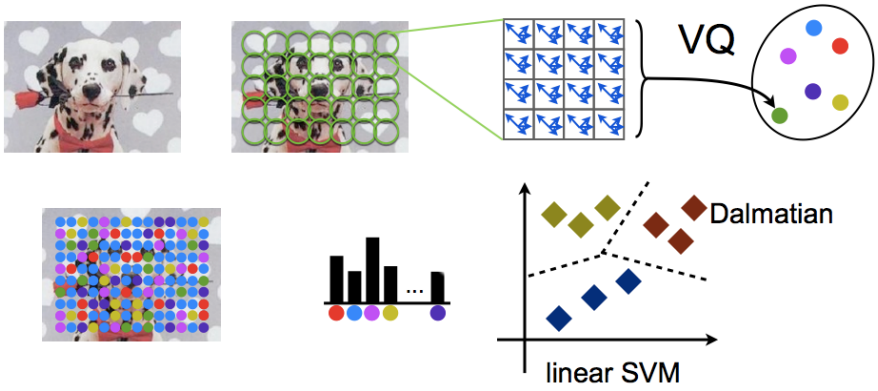


Figure 4.12: The abstract pipeline from Figure 4.1 visualized on a specific example. Descriptor representations are computed with the reference model trained with AdaDELTA on the Half Dome training set. The figure is adapted from a presentation by Andrea Vedaldi.

ple challenges for this benchmark: classification and detection. In the classification challenge, which I am considering in this section, the task is to predict the presence or absence of at least one object from 20 classes, so multiple objects can occur per image. The 20 classes are *aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa, train, tvmonitor*. Figure 4.13 shows two example images from this dataset, already converted to gray scale (the descriptor model from this chapter needs gray valued input patches).

In order to build a representation for a given image, as a first step descriptors are densely extracted using the reference model trained with AdaDELTA on the Half Dome training set. Figure 4.14 and Figure 4.15 show several *feature layers* of the new representation from the two images in Figure 4.13. These images are generated by interpreting the extracted tensor of size $h \times w \times 32$ as a stack of 32 images.

The *set of descriptors* per image are then vector-quantized via a visual dictionary and converted into a Bag-of-visual-Words representation. In order to avoid any effects of biased learning, the visual dictionary was trained via a simple clustering algorithm (K-Means, i.e. the dictionary consists of cluster centers) on a *different* dataset, the MIRFLICKR-1M [169], which consists of one million random images from FLICKR.

The BoW representation is used to train a standard multi-class support vector machine on the images from the training set, about 10000 samples. The baseline model on this dataset utilizes densely extracted SIFT descriptors instead and also employs a visual dictionary trained via K-Means on the MIRFLICKR-1M [169]. Table 4.9 shows the results of this evaluation. Compared to the best results on this dataset [53, 54] the performance is obviously very bad, for both Convolutional Net-

(a)

(b)

Figure 4.13: Two randomly chosen pictures from the VOC 2007 dataset in their original colored version and the employed gray scale variant. (a) The label associated with this image is *car*. (b) The labels associated with this image are *diningtable, person, chair*.

works as well as SIFT based systems [54]. However, no other tricks to improve the performance have been tried (for example spatial pyramid pooling [214] or more sophisticated encoding methods [53]). The important result is that the descriptors learned with a Convolutional Network perform as well as (or even slightly better) than SIFT while being much more compact.

Particularly the recent results (as of 2015) achieved with deep Convolutional Networks pretrained on ImageNet [54] make a comparison with a *bottom-up constructed* and end-to-end trained architecture an interesting future work. Additionally, it will also be interesting to see qualitatively how the feature maps (see Figures 4.14 and 4.15) change if a high-level object recognition task is used to fine-tune the low-level descriptors.

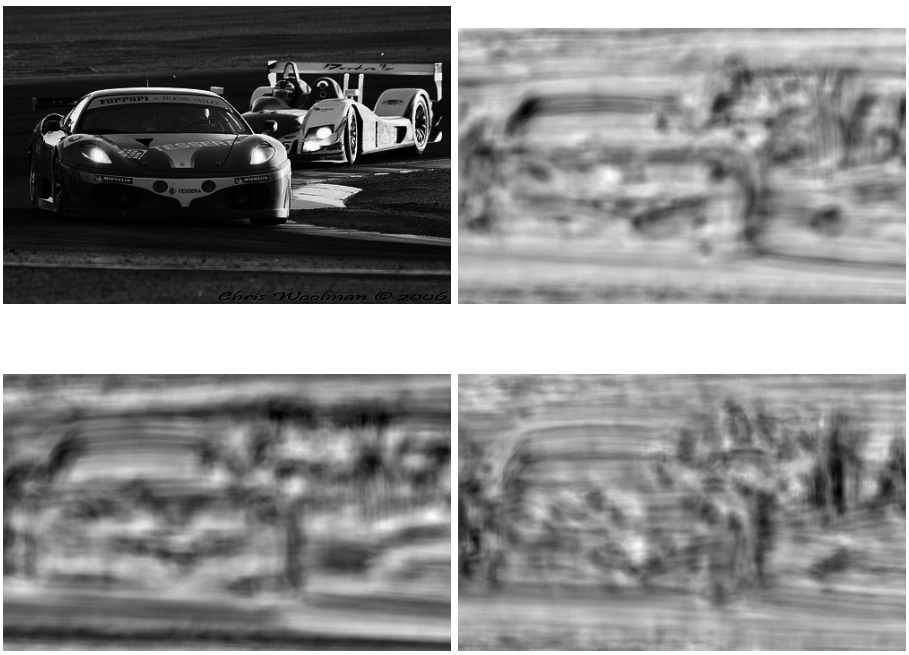


Figure 4.14: Randomly extracted feature maps from the car image. At least the car in the front is reflected in the feature maps.

Figure 4.15: The second exemplary image filtered with the CNN descriptor. The original scene is very cluttered, the descriptor cannot retain a lot of information.

Table 4.9: Image classification results on VOC 2007. The column denoted SIFT_{256} represents the results from the classification pipeline (Figure 4.12) using SIFT as the descriptor method and a dictionary size of 256 atoms. Equivalently, columns denoted with CNN_{xxx} denote pipelines based on the compact descriptor learned with the convolutional architecture. Descriptors are densely extracted with a stride of 4 pixels. The Bag-of-Word representation is formed through standard term-frequency inverse-document-frequency (tf-idf) coding. For the four different dictionary sizes, the convolutional descriptors perform marginally better than the SIFT descriptor.

	CNN_{256}	CNN_{512}	CNN_{1024}	CNN_{2048}	SIFT_{256}	SIFT_{512}	SIFT_{1024}	SIFT_{2048}
aeroplane	45.1	47.9	51.7	54.0	49.0	51.2	55.2	55.8
bicycle	26.1	32.9	35.2	38.4	25.8	30.5	30.7	30.4
bird	22.5	21.7	25.2	28.9	23.7	17.0	24.8	25.2
boat	29.9	32.6	34.1	44.4	35.1	35.1	45.6	46.6
bottle	19.0	18.8	19.5	19.7	18.2	16.7	13.8	17.7
bus	24.6	22.8	25.2	26.5	26.0	27.9	29.2	26.1
car	50.2	53.6	53.0	53.4	50.2	51.2	51.2	52.1
cat	23.9	30.3	29.7	31.8	25.4	26.8	34.1	30.8
chair	31.2	34.0	34.5	35.2	32.5	33.2	32.7	33.6
cow	15.2	21.7	20.3	23.2	15.4	17.7	24.4	19.3
diningtable	18.2	17.6	22.0	27.1	21.4	23.5	23.4	22.8
dog	24.2	30.3	29.9	29.0	25.4	28.5	30.4	30.8
horse	32.6	33.2	34.4	43.5	31.9	31.3	36.8	38.8
motorbike	34.3	34.7	36.6	37.1	18.2	22.5	23.4	26.7
person	66.6	68.7	70.1	69.4	64.8	65.7	68.8	68.3
pottedplant	17.7	16.9	17.4	12.1	18.9	18.5	18.4	13.9
sheep	21.5	26.0	31.0	22.5	19.2	21.4	22.1	24.1
sofa	21.0	16.0	19.3	18.7	14.9	16.2	16.2	19.9
train	33.3	36.6	42.8	45.4	37.8	40.6	39.8	42.6
tvmonitor	25.8	22.7	23.9	28.5	27.9	29.1	31.7	28.7
mAP	28.0	30.2	31.5	32.9	27.8	28.8	30.7	31.0

Figure 4.16: Randomly chosen images from STL, already resized to 64×64 pixels and converted to gray scale.

4.5.2 *STL-10*

The STL-10 dataset [66] is used to investigate how well knowledge can be transferred between high-level and low-level learning tasks. The STL-10 dataset is quite different from the VOC2007 dataset. It represents a line of datasets [207, 95, 135] that might be criticized by their limited *real-worldness*: images show only one object and are largely without any clutter, the single object is well aligned with the center, spatial extend of an image itself is limited (in the case of STL-10 the image size is 96×96 pixels) and the number of objects is small. However, these characteristics allow two nicely defined learning experiments: Given the learned low-level image descriptor from the correspondence task, how well does the embedding vector represent high-level object categories. And, vice versa, how well does a descriptor perform on the low-level matching task when it is trained with high-level inputs and similarities are formed according to category information.

STL-10 is built from labeled examples from ImageNet [80], cropped and rescaled to 96x96 pixels. The 10 classes have each 500 training images and 800 test images. Additionally, the dataset contains 100000 unlabeled images from a much broader set of classes. These images are supposed to be used by unsupervised algorithms and thus remain unused with the current experiments. Figure 4.16 shows some example images from the dataset, already converted to grayscale.

Table 4.10: Classification accuracy for STL using 32 dimensional descriptors learned with a Convolutional Network on a low-level correspondence task. Classification is done using a k nearest neighbour approach. The baseline approach (RAW) uses the pixel values as descriptors (i.e. the descriptor is 4096 dimensional). Higher numbers are better.

	$k = 1$	$k = 3$	$k = 5$
CNN	29.8	33.4	38.9
RAW	27.6	31.8	35.3

FROM LOW-LEVEL TO HIGH-LEVEL REPRESENTATIONS. For the object classification task the images from both the training and test set are converted to gray scale, rescaled to 64×64 pixels and contrast normalized. Every preprocessed image is mapped to a 32 dimensional vector by the descriptor model (again, I choose the reference architecture trained with AdaDELTA on the Half Dome training set). Classification is performed with a simple non-parametric nearest neighbour classifier, directly evaluating the image representation. Table 4.10 shows the results when evaluating the descriptor model with various numbers of neighbors (k) on the test set. The baseline is a simple Sum-of-Squares Difference metric applied to the raw pixel values.

FROM HIGH-LEVEL TO LOW-LEVEL REPRESENTATIONS. Given the object categories from the STL-10 dataset, pairs of matching (same object class) and non-matching (different object class) image pairs are formed. The overall architecture (a convolutional Network with $\ell_{\text{DrLim}^{++}}(\theta)$ cost function) is the same as the reference architecture from Section 4.3. Training happens with stochastic gradient descent ($\eta = 0.01$) until convergence.

The learned 32 dimensional descriptor is evaluated on the three evaluation sets from that correspondence task. Overall the result is as worse as with a random initialization—the performance on the three evaluation sets is around 50%. So if one takes into account that the AlexNet stack (see Chapter 3) did also perform not very well (without any additional adjustments) on the correspondence task it seems that for different levels of abstractions in the image content different convolutional models can not be simply interchanged. More work is necessary here in the future to determine how such networks can be adapted in an efficient way to different settings.

4.6 RELATED WORK

Local, fine-grained correspondence is such an elementary task in Computer Vision, it should be a typical problem domain for Convolutional Networks. However, only in 2008 did first work appear in this direction [180]. Its evaluation data is limited though, only covering in-plane affine transformations. Without a properly constructed training set no further development in this area was observed for some time. The large dataset from Chapter 3 made training deep Convolutional Networks an obvious choice, prone to succeed in this domain. The preliminary work for this chapter showed the feasibility of size-constraint Convolutional Networks for compact descriptor learning [284].

Very recently, several papers also investigated local descriptor learning with deep Convolutional Networks [97, 237, 426, 140, 423, 348][15]. Two of these papers have much better results on the matching task than reported in this work. However, these results are achieved with networks and representations that are several times larger than the models used in this work. For both papers some educated guesswork can be done which indicates that their performance would be at most as good as the one reported here if their models map to 32-dimensional descriptors (but still keep their larger number of parameters). Additionally both papers do not investigate the generalization capabilities of the learned descriptor using transfer learning tasks (see Section 4.5). Transfer learning is considered in a another recent paper [348] but its results are not comparable to my work as the paper utilizes an evaluation methodology different from fp@95. Of course other methods than deep networks have been applied to the matching task, too [43, 396, 395, 351]. Table 4.1 already showed the best performing approaches with a compact memory footprint.

Learning a similarity measure between entities belongs to the field of metric learning [210]. A very comprehensive literature review for this research field is given in the Ph.D. thesis from Karen Simonyan [349]. For Neural Network-based models, similarity measures can be learned with DrLim [138, 382] or ranking-based objectives [338]. The *siamese network* concept [334, 41] can also be applied to Recurrent Neural Networks [17] and to learn similarity preserving binary representations for multimodal data [249].

4.7 SUMMARY

Given the recent overwhelming success of deep Convolutional Networks for Computer Vision problems (mostly object classification, though) the good results reported in this chapter might appear a

15 Apart from [348] these papers seem to miss the previously mentioned earlier references.

Table 4.11: False positive error rates at 95% retrieval (fp@95) for some of the baseline models. L4 is the reference model trained with the adapted AdaDELTA optimization algorithm, FC1 is the reference model followed by one fully-connected hidden layer before mapping to a 32-dimensional descriptors. It is trained with stochastic gradient descent. In particular FC1 clearly outperforms all other models.

	LY		**ND**		**HD**	
	ND	HD	LY	HD	LY	ND
SIFT	20.9	24.7	28.1	24.7	28.1	24.7
DAISY	–	–	16.8	13.5	18.3	12.0
CVX	9.1	14.3	14.2	13.4	16.7	10.0
L4	8.6	15.9	13.6	13.9	16.6	8.6
FC1	7.7	14.7	13.1	13.1	15.2	7.9

bit uninspiring. However, considering the constraints with respect to model size and representations dimensionality the performance may be surprising, nevertheless. Theoretical considerations [57] and empirical investigations [123] so far only indicate that very high capacity networks will generally be able to show good performance as long as enough training data is available. Good results with capacity-restriced models are therefore so much more interesting.

It might be surprising that tanh outperformed the ReLU activation function, but this is probably due to the limited number of parameters. This result should make one cautious against black-box recipes when using deep networks. Not only would these clearly advocate for the ReLU (or similar half-plane activation functions) but also for Dropout, which neither did help with respect to performance (see a longer comment with respect to Dropout in Section 6.2). Similarly, a certain level of depth appears beneficial in particular in combination with a fully connected layer. But again, with restricted model sizes deeper does not mean better.

Properly modeling the embedding task turned out to be crucial for achieving good results with a small representation. Integrating the manifold hypothesis into the DrLim objective to form a new cost function improved the performance more than any other measurement (like fully-connected layers or optimizers). In absolute terms up to 5% improvement could be gained, which is roughly 25% relative improvement.

Overall the reference model identified by a small model selection stage at the very beginning turned out to perform very well when

combined with a powerful optimizer that is adopted to the task. It therefore is valid to compare the best performing models in this chapter to other state-of-the-art models without risking (too much) of double dipping. So far, as Table 4.11 shows, the Convolutional Network developed in this Chapter seems to be the best performing model when the number of parameters and the size of the representation is severely limited.

Could these results be still obtained if the descriptor representation is reduced even more? A singular value decomposition of the last linear layer consistently identified only between 24 and 26 non-negligible singular values. However, first experiments were not successful to use this decomposition in order to achieve a more compact descriptor having comparable performance.

The most compact descriptor would be binary. However, as reported earlier, finding binary descriptor from scratch turned out to be difficult. Nevertheless, good binary descriptors can be obtained by random projections and sign-hashing of real-valued descriptors. Hence, a possible approach to find good binary descriptors would use a model pretrained with a real-valued embedding criterion which is then changed to a hashing-based objective, following the cost functions outlined in several recent papers [230, 249].

5

UNSUPERVISED MODELING OF LOCAL IMAGE PATCHES

In the following I tackle the dataset from Chapter 3 from the viewpoint of pure unsupervised feature learning. Why is this actually an interesting problem? It seems that there are already enough datasets for evaluating unsupervised feature learning algorithms. In particular for feature learning from image data, several well-established benchmarks exist like Caltech-101 [95], CIFAR-10 [207] and NORB [223], to name only a few. Notably, these benchmarks are all object classification tasks. Unsupervised learning algorithms are evaluated by considering how well a *subsequent supervised classification* algorithm performs on high-level features that are found by aggregating the learned low-level representations.

However, I think that mingling these steps makes it difficult to assess the quality of the unsupervised algorithms. A more direct way is needed to evaluate these methods, preferably where a subsequent supervised learning step is completely optional.

I am not only at odds with the *methodology* of evaluating unsupervised learning algorithms. General object classification tasks are always based on orientation- and scale-rectified pictures with objects or themes firmly centered in the middle. One might argue that object classification acts as a good proxy for a wide range of Computer Vision tasks beyond object classification, like tracking, stereo vision, panoramic stitching or structure from motion. But this hypothesis was not yet shown to be correct either theoretically or through empirical evidence. Instead, the dataset from Chapter 3 provides the most basic task that is elementary to all typical problems in Computer Vision: matching low-level representations, i.e. determining if two data samples are similar given their learned representation.

Matching image descriptors is a central problem in Computer Vision, so hand-crafted descriptors are always evaluated with respect to this task [259]. Given a dataset of labeled correspondences, *supervised* learning approaches will find representations *and* the accompanying distance metric that are optimized with respect to the induced similarity measure (as it was empirically demonstrated for Convolutional Networks in Chapter 4). It is remarkable that hand-engineered descriptors perform well under this task *without the need to learn such a measure* for their representations in a supervised manner.

To the best of my knowledge it has never been investigated whether any of the many unsupervised learning algorithms developed over the last couple of years can match this performance. While I do not

introduce a new learning algorithm, in essence I propose to make the dataset from Chapter 3 a standard benchmark when evaluating new unsupervised feature learning algorithms.

Specifically, I investigate the performance of the Gaussian RBM [381], its sparse variant and the mean covariance RBM [296] without any supervised fine tuning with respect to the matching task. As it turns out, under very favourable conditions performs the mcRBM comparably to hand-engineered feature descriptors. In fact using a simple heuristic, the mcRBM produces a *compact binary* descriptor that performs better than several state-of-the-art hand-crafted descriptors.

For the sake of seclusiveness, the main characteristics of the matching dataset are briefly repeated (section 5.1). Section 5.2 introduces the investigated models in more detail while Section 5.3 describes the evaluation protocol. In section 5.4 I present the empirical results, both quantitatively and qualitatively. Section 5.5 concludes with a brief summary.

RELATED WORK Only a few papers from the last years are similar in spirit to the work in this chapter: The popular press discussed two papers [67, 216] that are actually interested in the behaviour of unsupervised learning approaches without any supervised steps afterwards. However, again only high-level representations (i.e. class instances) are considered. Similarly, very deep Autoencoders [208] learn a compact, binary representation in order to do fast content-based image search (*semantic hashing*, [322]). Again, these representations are only studied with respect to their capabilities to model high-level object concepts.

Finding (compact) low-level image descriptors is an excellent supervised learning task: Even hand-designed descriptors have many free parameters that cannot (or should not) be optimized manually. Given ground truth data for correspondences, the performance of supervised learning algorithms is impressive, as e.g. demonstrated in Chapter 4 of this work. See Section 4.6 for more related work in the space of supervised metric learning.

5.1 DATASET

The matching dataset was originally introduced for studying discriminative (i.e. supervised) learning algorithms for local image descriptors [43]. It fosters supervised learning of optimal low-level image representations using a large and realistic training set of patch correspondences.

The dataset is based on more than 1.5 million image patches (64×64 pixels) of three different scenes: the Statue of Liberty (about 450,000 patches), Notre Dame (about 450,000 patches) and Yosemite's Half

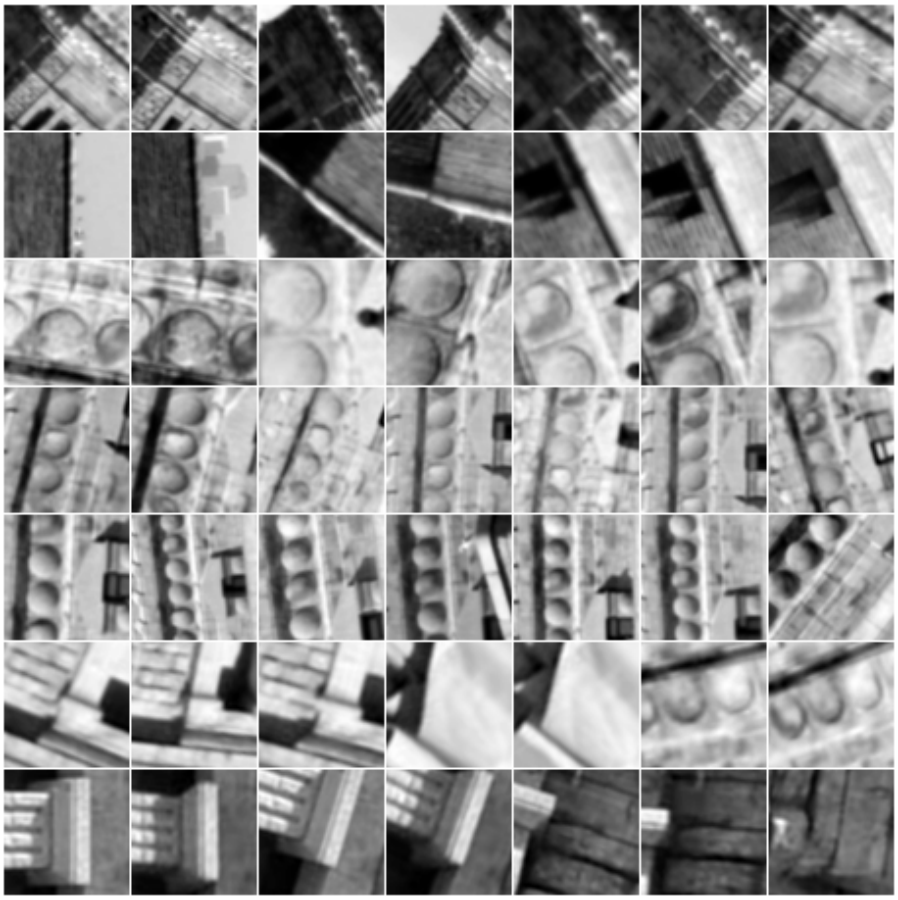


Figure 5.1: Patch correspondences from the Liberty dataset. Note the wide variation in lighting, viewpoint and level of detail. The patches are centered on interest points but otherwise can be considered random, e.g. there is no reasonable notion of an object boundary possible. Figure taken from [43].

Dome (about 650,000 patches). The patches are sampled around interest points detected by Difference of Gaussians [239] and are normalized with respect to scale and orientation[1]. As shown in Figure 5.1, the dataset has a wide variation in lighting conditions, viewpoints, and scales.

The dataset contains also approximately 2.5 million image correspondences. Correspondences between image patches are established via dense surface models obtained from stereo matching (stereo matching, with its epipolar and multi-view constraints, is a much easier problem than unconstrained 2D feature matching). As actual 3D correspondences are used, the identified 2D patch correspondences show substantial perspective distortions resulting in a much more realistic dataset than previous approaches [227, 259]. The dataset appears very similar to to an earlier benchmark of the same authors [420], yet the

1 A similar dataset of patches centered on multi-scale Harris corners is also available.

correspondences in the novel dataset resemble a much harder problem. The error rate at 95% detection of correct matches for the SIFT descriptor [239] raises from 6% to 26%, the error rate for evaluating patch similarity in pixel space (using normalized sum squared differences) raises from 20% to at least 48% (all numbers are take from [420] and [43] respectively), for example. In order to facilitate comparison of various descriptor algorithms a large set of predetermined match/non-match patch pairs is provided. For every scene, sets comprising between 500 and 500,000 pairs (with 50% matching and 50% non-matching pairs) are available.

The dataset constitutes an excellent test-bed for unsupervised learning algorithms. A wide range of experiments can be conducted in a controlled manner:

- Self-taught learning [295],
- Semi-supervised learning,
- Supervised transfer learning over input distributions with a varying degree of similarity (the scenes of Statue of Liberty and Notre Dame show architectural structures, while Half Dome resembles a typical natural scenery),
- Enhancing the available dataset with arbitrary image patches at keypoints and
- Evaluating systems trained on different tasks (e.g. large scale classification problems [209]).

5.2 METHODS

The basics of undirected graphical models are extensively covered in Section 2.6. Therefore this section only describes briefly the extensions of these basic models.

5.2.1 *Gaussian-Binary Restricted Boltzmann Machine*

The Gaussian-Binary Restricted Boltzmann Machine (GRBM) is an extension of the Binary-Binary RBM [100] that can handle continuous data [151, 381]. It is a bipartite Markov Random Field over a set of visible units, $\mathbf{v} \in \mathbb{R}^n$, and a set of hidden units, $\mathbf{h} \in \{0,1\}^m$. Every configuration of units $\mathbf{v}$ and units $\mathbf{h}$ is associated with an energy $E(\mathbf{v}, \mathbf{h}, \theta)$, defined as [381]

$$E(\mathbf{v}, \mathbf{h}, \theta) = \frac{1}{2}\mathbf{v}^T\boldsymbol{\Lambda}\mathbf{v} - \mathbf{v}^T\boldsymbol{\Lambda}\mathbf{a} - \mathbf{h}^T\mathbf{b} - \mathbf{v}^T\boldsymbol{\Lambda}^{1/2}\mathbf{W}\mathbf{h} \tag{5.1}$$

with $\theta = (\mathbf{W} \in \mathbb{R}^{n\times m}, \mathbf{a} \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^m, \boldsymbol{\Lambda} \in \mathbb{R}^{m\times m})$, the model parameters. $\mathbf{W}$ represents the visible-to-hidden symmetric interaction

terms, $\mathbf{a}$ and $\mathbf{b}$ represent the visible and hidden biases respectively and $\boldsymbol{\Lambda}$ is the precision matrix of $\boldsymbol{v}$, taken to be diagonal. $E(\boldsymbol{v}, \mathbf{h}, \theta)$ induces a probability density function over $\boldsymbol{v}$ and $\mathbf{h}$:

$$\begin{aligned} p(\boldsymbol{v}, \mathbf{h}, \theta) &= \frac{\exp\big(-E(\boldsymbol{v}, \mathbf{h}; \theta)\big)}{Z(\theta)} \\ Z(\theta) &= \int \sum_{\mathbf{h}} \exp\big(-E(\boldsymbol{v}, \mathbf{h}, \theta)\big)\, d\boldsymbol{v} \end{aligned} \tag{5.2}$$

where $Z(\theta)$ is the normalization partition function.

Learning the parameter set θ is accomplished by gradient ascent in the log-likelihood of θ given N i.i.d. training samples. The log-probability of one training sample is

$$\begin{aligned} \log p(\boldsymbol{v}, \theta) = &-\frac{1}{2}\boldsymbol{v}^T \boldsymbol{\Lambda} \boldsymbol{v} + \boldsymbol{v}^T \boldsymbol{\Lambda} \mathbf{a} \\ &+ \sum_{j}^{m} \log\left(1 + \exp\left(\sum_{i}^{N_v} \boldsymbol{v}_i^T (\boldsymbol{\Lambda}^{\frac{1}{2}} \boldsymbol{W})_{ij} + \mathbf{b}_j\right)\right) \\ &- Z(\theta) \end{aligned} \tag{5.3}$$

Evaluating $Z(\theta)$ is intractable, therefore algorithms like Contrastive Divergence (CD) [149] or persistent CD (PCD) [387] are used to compute an approximation of the log-likelihood gradient. The bipartite nature of an (G)RBM is an important aspect when using these algorithms: The visible units are conditionally independent given the hidden units. They are distributed according to a diagonal Gaussian:

$$p(\boldsymbol{v} \mid \mathbf{h}, \theta) \sim \mathcal{N}(\boldsymbol{\Lambda}^{-\frac{1}{2}} \boldsymbol{W} \mathbf{h} + \mathbf{a}, \boldsymbol{\Lambda}^{-1}) \tag{5.4}$$

Similarly, the hidden units are conditionally independent given the visible units. The conditional distribution can be written compactly as

$$p(\mathbf{h} \mid \boldsymbol{v}, \theta) = \sigma(\boldsymbol{v}^T \boldsymbol{\Lambda}^{\frac{1}{2}} \boldsymbol{W} + \mathbf{b}) \tag{5.5}$$

where σ denotes the element-wise logistic sigmoid function, $\sigma(z) = 1/(1 + e^{-z})$.

5.2.2 *Sparse GRBM*

In many tasks it is beneficial to have features that are only rarely active [269, 65]. Sparse activation of a binary hidden unit can be achieved by specifying a sparsity target ρ and adding an additional penalty term to the log-likelihood objective that encourages the actual probability of unit j of being active, q_j, to be close to ρ [269, 150]. This penalty is proportional to the negative Kullback-Leibler divergence

between the hidden unit marginal $q_j = \frac{1}{N}\sum_n p(\mathbf{h}_j = 1 \mid \boldsymbol{v}_n, \theta)$ and the target sparsity:

$$\lambda_{sp}\big(\rho \log q_j + (1-\rho)\log(1-q_j)\big), \tag{5.6}$$

where λ_{sp} represents the strength of the penalty. This term enforces sparsity of feature j *over* the training set, also referred to as *lifetime sparsity*. The hope is that the features for one training sample are then encoded by a sparse vector, corresponding to *population sparsity*. I denote a GRBM with a sparsity penalty $\lambda_{sp} > 0$ as *spGRBM*.

5.2.3 *Mean-Covariance Restricted Boltzmann Machine*

In order to model pairwise dependencies of visible units gated by hidden units, a third-order RBM can be defined with a weight w_{ijk} for every combination of $\boldsymbol{v}_i$, $\boldsymbol{v}_j$ and $\mathbf{h}_k$. The cubic scaling of the weight parameter $\boldsymbol{W}$ (a three-dimensional tensor) of such a *third-order* RBM makes learning models of even moderate size prohibitive. A widely used approach in such cases is to factorize the weight matrix and possibly share weights. The parameters are therefore reduced to a filter matrix $\mathbf{C} \in \mathbb{R}^{n\times F}$ connecting the input to a set of *factors* and a *pooling* matrix $\mathbf{P} \in \mathbb{R}^{F\times m}$ mapping factors to hidden variables. The energy function for this *cRBM* [300] is

$$E_c(\boldsymbol{v}, \mathbf{h}_c, \theta') = -(\boldsymbol{v}^\mathsf{T}\mathbf{C}^\mathsf{T})^2\mathbf{P}\mathbf{h}_c - \mathbf{c}^\mathsf{T}\mathbf{h}_c \tag{5.7}$$

with $\theta' = \{\mathbf{C}, \mathbf{P}, \mathbf{c}\}$ and where $(\cdot)^2$ denotes the element-wise square operation. Note that $\mathbf{P}$ has to be non-positive [300, Section 5]. The hidden units of the cRBM are still conditionally independent given the visible units, so inference remains simple. Their conditional distribution (given visible state $\boldsymbol{v}$) is

$$p(\mathbf{h}_c \mid \boldsymbol{v}, \theta') = \sigma(\mathbf{P}^\mathsf{T}(\mathbf{C}^\mathsf{T}\boldsymbol{v})^2 + \mathbf{c}) \tag{5.8}$$

On the other hand the visible units are conditionally Gaussian:

$$p(\boldsymbol{v} \mid \mathbf{h}_c, \theta') \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}) \tag{5.9}$$

with

$$\boldsymbol{\Sigma}^{-1} = \mathbf{C}\mathrm{diag}(-\mathbf{P}\mathbf{h}_c)\mathbf{C}^\mathsf{T} \tag{5.10}$$

Note that this *inverse* covariance matrix depends on the hidden state. Repeated conditionally sampling $\mathbf{x}$ which is necessary for example in MCMC is therefore computationally very costly as the inference can not be amortized over samples: for every sample a new matrix inversion must be computed.

As Eq. (5.9) shows, the cRBM can only model Gaussian inputs with zero mean. For general Gaussian-distributed inputs the cRBM and the

GRBM can be combined into the *mean-covariance RBM* (mcRBM) by simply adding their respective energy functions:

$$E_{mc}(\mathbf{v}, \mathbf{h}_m, \mathbf{h}_c, \theta, \theta') = E_m(\mathbf{v}, \mathbf{h}_m, \theta) + E_c(\mathbf{v}, \mathbf{h}_c, \theta') \tag{5.11}$$

$E_m(\mathbf{v}, \mathbf{h}_m, \theta)$ denotes the energy function of the GRBM (see Eq. (5.1)) with $\mathbf{\Lambda}$ fixed to the identity matrix. The resulting conditional distribution over the visible units, given the two sets of hidden units $\mathbf{h}_m$ (*mean* units) and $\mathbf{h}_c$ (*covariance* units) is

$$p(\mathbf{v} \mid \mathbf{h}_m, \mathbf{h}_c, \theta, \theta') \sim \mathcal{N}(\mathbf{\Sigma W h}_m, \mathbf{\Sigma}) \tag{5.12}$$

with $\mathbf{\Sigma}$ defined as in Eq. (5.10). The conditional distributions for $\mathbf{h}_m$ and $\mathbf{h}_c$ are still as in Eq. (5.5) and Eq. (5.7) respectively. The parameters θ, θ' can again be learned using approximate Maximum Likelihood Estimation, e.g. via CD or PCD. These methods require to sample from $p(\mathbf{v}|\mathbf{h}_m, \mathbf{h}_c, \theta, \theta')$, which involves an expensive matrix inversion (see Eq. (5.10)). Instead, samples are obtained by using Hybrid Monte Carlo (HMC) [271] on the mcRBM free energy [296].

5.3 EVALUATION PROTOCOL

For the results presented in the following section I loosely follow the standard evaluation procedure [43]: For every scene (Liberty (denoted by LY), Notre Dame (ND) and Half Dome (HD)), I use the labeled dataset with 100,000 image pairs to assess the quality of a trained model on this scene. In order to save space I do not present ROC curves and only show the results in terms of the 95% error rate which is the percent of incorrect matches when 95% of the true matches are found.

The presented models are trained in an unsupervised fashion on the available patches while the original work was purely supervised [43]. Therefore I investigate two scenarios. In the first scenario, I train on one scene (400,000 randomly selected patches from this scene) and evaluate the performance on the test set of every scene. This allows to investigate the self-taught learning paradigm [295]. In the second scenario I train on all three scenes jointly (represented by 1.2 million image patches) and then evaluate again every scene individually.

5.3.1 *Distance metrics*

As I explicitly refrain from learning a suitable (with respect to the correspondence task) distance metric with a supervised approach, standard distance measures must be used. The Euclidean distance is widely used when comparing image descriptors. Yet, considering the generative nature of the models a rather general argumentation [186] indicates that the Manhattan distance, denoted L1, should be chosen

for matching tasks. I also consider two normalization schemes for patch representations, $\ell 1$ and $\ell 2$ (i.e. after a feature vector $\mathbf{x}$ is computed, its length is normalized such that $\|\mathbf{x}\|_1 = 1$ or $\|\mathbf{x}\|_2 = 1$).

Given a visible input both (sp)GRBM and mcRBM compute features that resemble parameters of (conditionally) independent binary random variables (i.e. Bernoulli variables). The Kullback-Leibler divergence [70] $\mathcal{KL}[\cdot\|\cdot]$ between two probability mass functions p and q over a set $\mathcal{X}$ is defined as

$$\mathcal{KL}[p\|q] = \sum_{x\in\mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \tag{5.13}$$

For example, the Kullback-Leibler divergence of a Bernoulli variable with bias p to a Bernoulli variable with bias q is given by

$$p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q} \tag{5.14}$$

A symmetric version of the Kullback Leibler divergence is the Jensen-Shannon divergence JSD [229]:

$$\mathrm{JSD}(p \parallel q) = \frac{1}{2}\mathcal{KL}[p\|m] + \frac{1}{2}\mathcal{KL}[q\|m] \tag{5.15}$$

with $m(x) = \frac{1}{2}(p(x) + q(x))$ for all $x \in \mathcal{X}$. Finding $m(x)$ for discrete probability mass functions $p(x)$ and $q(x)$ is straightforward—however it is very difficult for continuous probability density functions!

5.4 RESULTS

This section presents the results for GRBM, its sparse variant and mcRBM. As the three main baselines serve SIFT [239], Principle Components Analysis (PCA, see Section 2.6.2) and raw pixel values (i.e. the image patch is the descriptor itself), see Table 5.1.

5.4.1 *Baselines*

SIFT. SIFT [239] (both as interest point detector and descriptor) is a landmark for image feature matching. Because of its good performance it is one the most important basic ingredients for many different kinds of Computer Vision algorithms. I use it as the main baseline for evaluating the three models.

Table 5.1 shows two entries for SIFT: The first line shows the result for a slightly post-processed SIFT descriptor: it is ℓ_1 normalized. ℓ_1 normalization provides better results than ℓ_2 normalization or no normalization at all. SIFT performs descriptor sampling at a certain scale relative to the Difference of Gaussians peak. In order to achieve good results, it is essential to optimize this scale parameter [43, Figure 6] on

Table 5.1: Error rates of various baseline models. The table shows the percent of incorrect matches when 95% of the true matches are found.

		Test set		
Method	**Training set**	LY	ND	HD
SIFT	–	28.1	20.9	24.7
RAW	–	58.3	52.4	51.1
PCA (128d)	LY	51.3	50.9	53.2
	ND	51.9	46.4	49.5
	HD	53.8	47.4	47.3
SIFTb	–	31.7	22.8	25.6
BRIEF	–	59.1	54.5	54.9
BRISK	–	79.3	74.8	73.2
SURF	–	54.0	45.5	43.5

every dataset. As it turns out, with the SIFT implementation used for this work [404], the scale parameter is identical for all three datasets. Therefore, one line in Table 5.1 suffices for presenting the results for a normalized SIFT vector.

A second aspect that is also investigated here is the performance of compact descriptors (compact here means that the memory footprint is small). SIFT in its unnormalized form can be represented with 128 bytes only. The performance of this 128-byte descriptor is shown in the line labeled *SIFTb* of Table 5.1. The last lines in this table also show other widely used descriptors that where specifically designed to improve over SIFT with respect to compactness: BRIEF (32 bytes) [49] and BRISK (64 bytes) [228] are binary descriptors, SURF [14] is a real valued descriptor with 64 dimensions. Interestingly, the performance of these descriptors is far from the performance of SIFT and yet these descriptors are often used in matching tasks.

PCA. The simplest descriptor for any entity is always the entity itself. In the case of the image patches of the dataset at hand this results in 4096 dimensional descriptors. Obviously, the resulting matching performance is not expected to be very good. Therefore this performance serves well as a general lower bound. The line denoted RAW in Table 5.1 depicts the error rates for this case. A definite weakness of this descriptor is its high memory footprint (4096 bytes, because no normalization is applied to the byte values of the pixels). PCA is a widely used technique to achieve simple dimensionality reduction.

Table 5.2: Error rates, i.e. the percent of incorrect matches when 95% of the true matches are found. All numbers for GRBM and spGRBM are given within $\pm 0.5\%$. $(L_1 \ell_1)$ indicates that descriptors are ℓ_1 normalized and compared under the L1 distance metric.

		Test set		
Method	**Training set**	LY	ND	HD
SIFT	–	28.1	20.9	24.7
GRBM $(L_1 \ell_1)$	LY	47.6	33.5	41.4
	ND	50.0	33.4	42.5
	HD	49.0	34.0	41.5
	LY/ND/HD	48.7	33.5	42.1
spGRBM $(L_1 \ell_1)$	LY	37.9	26.9	34.3
	ND	40.0	28.0	35.4
	HD	39.1	27.9	34.9
	LY/ND/HD	37.5	26.6	33.6

The entry in Table 5.1 shows the performance of 128-dimensional descriptors (real valued). PCA needs a training set, therefore the entry consists of three sub-entries.

5.4.2 *GRBM/spGRBM*

The GRBM and spGRBM only differ in the setting of the sparsity penalty λ_{sp}, therefore the results for both models are presented together. I use CD_1 [149] to compute the approximate gradient of the log-likelihood and the recently proposed RMSprop [388] (also see Eq. (2.357)) method as gradient ascent method. Table 5.2 shows the overall error rates for both models (together with the SIFT baseline).

Before learning the parameters all image patches are rescaled to 16×16 pixels. Then I preprocess all training samples by subtracting from every patch its mean and dividing by the standard deviation of its elements. This is a common practice for visual data and corresponds to local brightness and contrast normalization. There is also a theoretical justification for why this preprocessing step is necessary to learn a reasonable precision matrix $\boldsymbol{\Lambda}$ [381, Section 2.2]. It is the only preprocessing scheme that allows GRBM and spGRBM to achieve good results. In addition, it is important to learn $\boldsymbol{\Lambda}$—setting it to the identity matrix, a common practice [150], also produces dissat-

isfying error rates. Note that originally it was considered that learning $\boldsymbol{\Lambda}$ is mostly important when one wants to find a good density (i.e. generative) model of the data.

Both GRBM and spGRBM have 512 hidden units. The elements of $\boldsymbol{W}$ are initialized according to $\mathcal{N}(0, 0.1)$, the biases are initialized to 0. RMSprop uses a learning rate of 0.001, the decay factor is 0.9, the minibatch size is 128. Both models train for 10 epochs only. The spGRBM uses a sparsity target of $\rho = 0.05$ and a sparsity penalty of $\lambda_{sp} = 0.2$. spGRBM is very sensitive to settings of λ_{sp} [377]—setting it too high results in dead representations (samples that have no active hidden units) and the results deteriorate drastically.

Like SIFT, GRBM and spGRBM perform best when their latent representations are ℓ_1 normalized before compared under the L_1 distance. spGRBM performs considerably better than its non-sparse version. This is *not* necessarily expected: Unlike e.g. in classification [65] sparse representations are considered problematic with respect to evaluating distances directly. *Lifetime sparsity* may be after all beneficial in this setting compared to strictly enforced population sparsity. Both models perform very poorly under the Jensen-Shannon divergence similarity (overall error rates around 60%), which is therefore not reported in Table 5.2. I was not successful in finding a compact representation with any of the two models, that performed at least in the range of PCA.

spGRBM shows an unexpected behaviour. While Notre Dame seems to be the simplest of the three datasets (with respect to the matching problem), its worst performance is obtained when the representations are learned on the Notre Dame patches. Furthermore, Liberty benefits more from unsupervised training on Half Dome than on Notre Dame, even though Notre Dame is much more similar to it (self-teaching hypothesis). Altogether the best performance is achieved when trained jointly on all three scenes. These three observations may indicate effects due to overfitting in the first two cases. This might be alleviated by enhancing the dataset with patches around keypoints from arbitrary images.

I also briefly compared RMSprop training with standard minibatch gradient descent training. There were no differences in the final evaluation results, yet RMSprop needed at most half of the training time necessary for comparable results with standard gradient descent.

5.4.3 *mcRBM*

mcRBM training is performed using the code from the original paper [296]. The patches are resampled again to 16×16 pixels. Then the samples are preprocessed by subtracting their mean (patchwise), followed by PCA whitening, which retains 99% of the variance. The

Table 5.3: Error rates for the mcRBM models. The table shows the percent of incorrect matches when 95% of the true matches are found. All numbers are given within ±0.5%. $(L_1 \ell_2)$ indicates that descriptors are ℓ_2 normalized and compared under the L1 distance metric. JSD denotes the Jensen-Shannon divergence.

		Test set		
Method	**Training set**	LY	ND	HD
SIFT	–	28.1	20.9	24.7
mcRBM $(L_1 \ell_2)$	LY	31.3	25.1	34.5
	ND	34.0	25.6	33.0
	HD	31.2	22.3	25.7
	LY/ND/HD	30.8	24.8	33.3
mcRBM (JSD)	LY	34.7	24.2	38.6
	ND	33.3	24.8	44.9
	HD	29.9	22.7	37.6
	LY/ND/HD	30.0	23.1	39.8

overall training procedure (with stochastic gradient descent) is identical to the one described in [296, Section 4].

Two different mcRBM architectures are considered: The first has 256 mean units, 512 factors and 512 covariance units. **P** is not constrained by any fixed topography. The results for his architecture are presented in Table 5.3. The second architecture is concerned with learning more compact representations: It has 64 mean units, 576 factors and 64 covariance units. **P** is initialized with a two-dimensional topography that takes 5×5 neighbourhoods of factors with a stride equal to 3. The results for the binarized representations of this model is shown in Table 5.4. Both architectures are trained for a total of 100 epochs, however updating **P** is only started after epoch 50.

For both instances the main insight is that *only the latent covariance units should act as the representations* for a given patch. Compared to results using the complete latent representation, this form of descriptors perform much better, in particular for the compact architecture with 64 covariance units. Using the complete latent representation of the mcRBM gives performance results that are much worse then the RAW descriptors. This is in accordance with manually designed descriptors: Many of these rely on distributions (i.e. histograms) of intensity gradients or edge directions [239, 259, 14], structural information which is encoded by the covariance units of an mcRBM. The

Table 5.4: Error rates for compact real-valued (mcRBM with $L_1\ell_2$ distance/normalization scheme) and binarized descriptors. The table shows the percent of incorrect matches when 95% of the true matches are found. D-Brief [395] learns compact binary descriptors with supervision.

		Test set		
Method	**Training set**	LY	ND	HD
SIFT	–	31.7	22.8	25.6
BRIEF	–	59.1	54.5	54.9
BRISK	–	79.3	74.8	73.2
SURF	–	54.0	45.5	43.5
mcRBM ($L_1\ell_2$)	LY	33.5	37.2	55.8
	ND	37.3	37.2	50.6
	YM	34.0	32.4	44.7
	LY/ND/HD	33.6	32.2	52.3
D-Brief (4 bytes)	LY	–	43.1	47.2
	ND	46.2	–	51.3
	HD	53.3	43.9	–
mcRBM (8 bytes)	LY	32.0	35.1	56.5
	ND	41.4	30.1	51.4
	YM	37.9	31.5	47.3
	LY/ND/HD	36.1	27.2	46.1

function of the mean units is to factor away aspects like lighting conditions which are uninformative for the similarity task (see also [301, Section 2]).

The large mcRBM architecture with 512 covariance units performs best under the L_1 distance measure when the latent representation is ℓ_2 normalized. When Half Dome is used as training set the overall performance on all three datasets is comparable to normalized SIFT, albeit at the cost of a 4.5 times larger feature representation. Compared to spGRBM one sees a noticeable improvement for the results on the respective training set (the diagonal entries in the subtable). Overall the more complex mcRBM seems to be beneficial for modeling Liberty and Notre Dame. Evaluating under the JSD distance measure produces results as bad as with (sp)GRBM. However, if one *scales down* the factors of the trained model linearly (a value of 3 is

appropriate), the results with respect to JSD improve remarkably, see Table 5.3, the last entry. The positive effect of this heuristic scaling is not as pronounced for the Half Dome dataset. Noteworthy, the same observation with respect to training on HD holds as with the $(L_1\ell_2)$ evaluation scheme. The scaling heuristic is also beneficial if the input is differently preprocessed: Scaling down the factors after unsupervised training is necessary if one seeks comparable results for contrast normalized, ZCA'ed [66] image patches (the results for this preprocessing scheme are not shown here). Instead of manually adjusting this scale factor, it should be a better idea to learn a scaling shared by all factors [73], yet integrating this scale parameter and optimizing it with respect to the mcRBM training objective did not result in the predicted improvements.

Finding *compact* representations for any kind of input data should be done with multiple layers of nonlinearities [208]. But even with only two layers it is possible to learn relatively good compact descriptors, cf. to the mcRBM entry in Table 5.4. If features are *binarized*, the representation can be made even more compact. In order to find a suitable binarization threshold I employ the following simple heuristic: After training on a dataset all latent covariance activations (values between 0 and 1) *of the training set* are histogrammed. The *median* of this histogram is then used as the threshold. Table 5.4, last entry, shows that the resulting compact (8 bytes) descriptor performs remarkably well, comparable or even better than several state-of-the-art descriptors. It performs even comparable to D-Brief [395], which is a binary descriptor learned in a supervised way.

5.4.4 *Other models*

I also trained several other unsupervised feature learning models: GRBM with nonlinear rectified hidden units[2] [268], various kinds of Autoencoders (sparse [66] and denoising [408] Autoencoders) and two layer models (stacked RBMs, Autoencoders with two hidden layers, cRBM [300]). None of these models performed better than PCA and usually failed to supersede the RAW pixel descriptor, too.

In several publications [66, 67] *gain-shaped K-means* performed surprisingly well compared to RBM or Autoencoder based models. Compared to standard K-means, the important ingredient is that image patches are ZCA'ed. Due to its fast training time it is an attractive model for learning low-level descriptors. However, even after scaling down the image patches to 12×12 (no learning happened with large image patches, instead the learned centers were exemplars from the training set), no convincing results could be achieved, see Table 5.5.

2 My experiments indicate that RMSprop is in this case also beneficial with respect to the final results: It learns models that perform about 2-3% better than those trained with stochastic gradient descent.

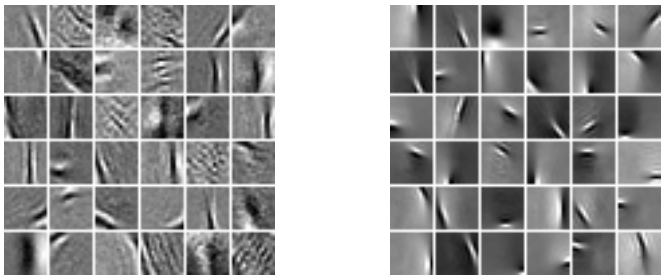
Table 5.5: Error rates for K-means, i.e. the percent of incorrect matches when 95% of the true matches are found. All numbers for K-means are given within ±0.5%.

		Test set		
Method	**Training set**	LY	ND	HD
SIFT	–	28.1	20.9	24.7
K-means	LY	40.9	38.9	44.3
	ND	45.0	39.7	44.1
	HD	49.1	37.3	47.9
	LY/ND/HD	42.7	35.3	43.2

DEEP AUTOENCODERS. It may be not surprising that models with only one or two layers of representations have difficulties with modeling the complex dataset at hand. SIFT itself can be represented as a model that has at least two layers. It therefore seems likely that a model with many layers performs better than either (sp)GRBM or mcRBM. In order to investigate this hypothesis, I trained a deep Autoencoder with six layers ($1024 - 512 - 256 - 128 - 256 - 512 - 1024$). One variant used GRBM (for the first layer) or standard binary-binary RBMs (for the subsequent layers) for layer initialization [208], a different variant simply trained the complete architecture in an end-to-end manner, which could be successfully trained if the layers got initialized properly [375]. The positive result of these models is that the reconstruction loss for a given image is an order of a magnitude better than the reconstruction loss one can achieve with PCA. Obviously, this is not very surprising given the amount of parameters the model posses. On the other hand, the 128 dimensional deep representations used for the matching task did not perform better than the representation learned with PCA. In the conclusion of this chapter I speculate why it is probably not enough in the case of unsupervised learning to simply scale the depth of ones models to achieve better results.

5.4.5 *Qualitative analysis*

When working with image data it is always instructive and interesting to visualize filters or features developed by a model. Unsurprisingly, both spGRBM (Figure 5.2a) and mcRBM (Figure 5.2b—these are columns from $\mathbf{C}$) learn Gabor like filters. Filters learned with GRBM (not shown) tend to be much more random and have overall a much lower number of such Gabor filters.



(a) (b)

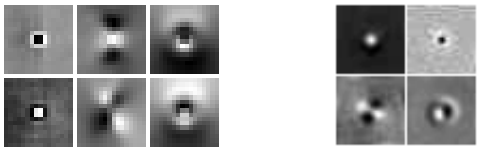
Figure 5.2: Patch models tend to learn Gabor filters. (a) Typical filters learned with spGRBM. (b) Filters from an mcRBM.

(a) (b)

Figure 5.3: (a) The pixelwise inverted standard deviations learned with a spGRBM plotted as a 2D image (darker gray intensities resemble lower numerical values). An input patch is elementwise multiplied with this image when computing the latent representation. This figure is generated by training on 32×32 patches for better visibility, but the same qualitative results appear with 16×16 patches. (b) The relative contribution of patch pixels learned by the large margin based convex optimization approach [351].

An interesting observation can be made when inspecting the diagonal elements of $\boldsymbol{\Lambda}^{1/2}$ from a spGRBM: Figure 5.3a shows these elements visualized as an image patch—when computing a latent representation, the input $\boldsymbol{v}$ is scaled (elementwise) by this matrix. This 2D image resembles a Gaussian that is dented at the center, the location of the keypoint of every image patch. This observation is even more interesting if compared to pooling configurations learned with a large margin convex optimization approach [351]: As Figure 5.3b shows, the weighting structure learned by the supervised algorithm is similar to the weighting matrix from spGRBM.

The mcRBM also places filters around the center (the location of the keypoint): Figure 5.4a shows some unusual filters from $\mathbf{C}$. They are centered around the keypoint and bear a strong resemblance to discriminative projections (Figure 5.4b) that are learned in a supervised way on this dataset [43, Figure 5]. Qualitatively, the filters in Figure 5.2d resemble log-polar filters that are used in several state-of-the-art feature designs [259]. While it may be surprising that these filters appear with the mcRBM, if one considers the energy function



(a) (b)

Figure 5.4: (a) The mcRBM also learns some variants of log-polar filters centered around the DoG keypoint. These are very similar to filters found when optimizing for the correspondence problem in a *supervised* setting. Several of such filters are shown in subfigure (b), taken from [43, Figure 5].

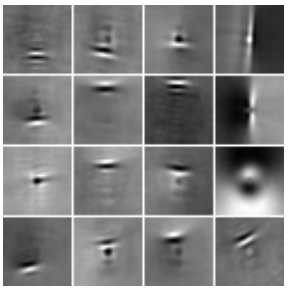


Figure 5.5: The basic keypoint filters are combined with Gabor filters, if these are placed close to the center. The Gabor filters get systematically arranged around the keypoint filters.

of the mcRBM, it matches the supervised criteria used for discriminative training for the DAISY [43] descriptors!

Finally, the very focused keypoint filters (first column in Figure 5.2) are often combined with Gabor filters *placed in the vicinity of the center*—the Gabor filters appear on their own, if they are too far from the center. If an mcRBM is trained with a fixed topography for $\mathbf{P}$, one sees that the Gabor filters get systematically arranged around the keypoint (Figure 5.5). This can be loosely interpreted as learning an orientation with respect to the keypoint at the center of the patch. But this is actually what happens in the algorithm of SIFT descriptor.

5.5 CONCLUSION

In summary the evaluation presented in this chapter leaves some positive and some negative impressions. On the negative side are definitely the large number of models that could not even outperform the most simple baselines of RAW and PCA descriptors. One might argue that this comes as no surprise, as the matching task requires methods that identify geometric invariances *between* patches but all employed models are never shown this kind of pairwise data. At least

outperforming a standard linear model like PCA should be possible, though.

Clearly, some of these models (even a plain deep autoencoder) are probably not expressive enough for the data. Of course it is possible that there are some methods that *apriori* have latent representations that are more useful for the kind of matching task while other methods choose representations more suitable for different tasks, but both classes of methods are equally expressive for the raw dataset. Overall this might require that unsupervised learning as a general methodology needs a more fine-grained definition. More specifically it means that the matching dataset should indeed serve as a general unsupervised benchmark, in particular because the most popular evaluation methods for unsupervised algorithms are inconsistent [385].

On the positive side is the result achieved by the mean-covariance RBM. It does not only show a good quantitative performance but also has interesting (and partially surprising) qualitative characteristics. The nice aspect of this model is that the interpretation of its two types of latent units (mean and covariance units) induces an appropriate usage in the matching task that eventually leads to the good performance: the covariance units are supposed to encode the image content and therefore are the representation that must be used for similarity problems.

An important question can be directly derived from this observation: Do methods that have different groups of latent variables (algorithmically and semantically different) perform better in general on this task (given of course that the specific group of latent units is chosen that supposedly is best suited for the matching task)? Or, more generally, should unsupervised methods always have factorized latent representations (also see the end of Section 2.7.4)?

6 TOUR DES CUL-DE-SACS

When summarizing research results it should be good practice to report approaches that did not work as expected, too. This is particularly true for methods that theoretically are a perfect fit for the problem or empirically have been verified with great success. Such negative research results can spur new research directions as they may highlight potential (overlooked) weaknesses of methods.

This chapter briefly gives an overview of failed attempts to tackle the problem of learning (compact) descriptors for local image patches (see Chapter 3) with supervised, semi-supervised and unsupervised methods. All proposed experiments use methods that are either theoretically or empirically well-motivated. Because the outcome of the various experiments was not as good as expected, I use one third of the evaluation setup: results are reported for the Liberty traininig set only (i.e. the test sets are Notre Dame and Half Dome). This reduced the computational burden when running the unsuccessful results several times to validate the findings. I describe fruitless explorations into

- semi-supervised models (Section 6.1): A type of gated undirected graphical model allows to describe *pairs* of input data and induces a semi-distance metric on the latent representation. The performance of this model on the matching dataset is poor. A simple extension of the learning rule using explicitly non-matching pairs (and thus turning the originally semi-supervised model into a fully supervised model) lead to a considerable improvement but the new model still performs worse than the baseline performance from SIFT.

- supervised extensions (Section 6.2): The supervised model from Chapter 4 has many possible options that could lead to considerable improvements: Models that take explicitly or implicitly scale into account, cost functions that are better tuned to the evaluation criteria or the magic-wand *dropout* [365]. Only a few of these ideas turned out to be good candidates for further investigations.

- unsupervised models (Section 6.3): Directed graphical models, relying on deep Neural Networks as fast approximators (see Section 2.7.4), seem to be a very good fit to the matching task. The reported failure on the dataset may hint at some missing ingrediants of these new approaches for tackeling more realistic datasets.

6.1 MODELING IMAGE PAIRS WITH A THREE-WAY RBM

Why did the unsupervised approaches in Chapter 5 perform so unconvincingly? One possible explanation is that for high-dimensional visual inputs the utilized models are simply not powerful enough (see Section 6.3 for one more attempt with supposedly better models). From a more abstract point of view it is questionable whether purely static unsupervised learning can actually be successful in the evaluation setting at hand: The matching task from Chapter 3 requires that a model identifies the invariances *between* patches. Probably these types of invariances are only detected to some degree by looking at single samples alone—most of the capacity of the models will be spent on describing the regularities found over the dataset as a whole. It is quite possible that the important invariances for describing pairs of samples can not be detected at all with such models.

So for an approach that operates closer to the actual task, the respective models should be able to know at least about the positive instances of matching pairs. Such instances are readily available in a phyiscally grounded data set: proximity in time or space is implicitly pairing samples. Of course it is still advantageous if these models can represent a single input, too.

With these considerations in mind, one possible approach is to use a set of binary latent variables $\mathbf{h} \in [0,1]^d$ that identify different kinds of basic relationships between two inputs $\mathbf{x} \in \mathbf{R}^{D_x}$,$\mathbf{y} \in \mathbf{R}^{D_y}$ and define the following *score* $S(\mathbf{x},\mathbf{y},\mathbf{h})$ *over triplets* $\mathbf{x},\mathbf{y},\mathbf{h}$ [372]:

$$S(\mathbf{x},\mathbf{y},\mathbf{h}) = \sum_{f=1}^{F} \mathbf{v_f}^T\mathbf{x} \times \mathbf{w_f}^T\mathbf{y} \times \mathbf{u_f}^T\mathbf{h}. \tag{6.1}$$

Hereby, $\mathbf{v_f} \in \mathbf{R}^{D_x}$, $\mathbf{w_f} \in \mathbf{R}^{D_y}$ and $\mathbf{u_f} \in \mathbf{R}^d$, for $f = 1,2,\dots f$ denote *filters* that will be learned from training data. For convenience, these filters are also put into matrices in a row-wise manner, i.e. $\mathbf{V} \in \mathbf{R}^{f\times D_x}$, $\mathbf{W} \in \mathbf{R}^{f\times D_y}$ and $\mathbf{U} \in \mathbf{R}^{f\times d}$, which allows to rewrite Eq. (6.1) more compactly:

$$S(\mathbf{x},\mathbf{y},\mathbf{h}) = \mathbf{x}^T\mathbf{V}^T\mathrm{diag}(\mathbf{U}\mathbf{h})\mathbf{W}\mathbf{y} \tag{6.2}$$

This score function is turned into an *energy function* $E(\mathbf{x},\mathbf{y},\mathbf{h},\theta)$ by extending it with bias terms:

$$\begin{aligned} E(\mathbf{x},\mathbf{y},\mathbf{h},\theta) = &-S(\mathbf{x},\mathbf{y},\mathbf{h}) - \mathbf{c}^T\mathbf{h} \\ &+ \frac{1}{2}(\mathbf{x}-\mathbf{a})^T(\mathbf{x}-\mathbf{a}) + \frac{1}{2}(\mathbf{y}-\mathbf{b}^T)(\mathbf{y}-\mathbf{b}) \end{aligned} \tag{6.3}$$

with $\theta = \{\mathbf{U}, \mathbf{V}, \mathbf{W}, \mathbf{a}, \mathbf{b}, \mathbf{c}\}$. The energy function defines an undirected graphical model (see Section 2.6.1) representing the probability distribution over the three variables $\mathbf{x}, \mathbf{y}, \mathbf{h}$:

$$\begin{aligned} p(\mathbf{x}, \mathbf{y}, \mathbf{h} \mid \theta) &= \frac{1}{Z} \exp(-E(\mathbf{x}, \mathbf{y}, \mathbf{h}, \theta)), \\ Z &= \sum_{\mathbf{h}} \int\int \exp(-E(\mathbf{x}, \mathbf{y}, \mathbf{h}, \theta)) d\mathbf{x} d\mathbf{y} \end{aligned} \tag{6.4}$$

The resulting model is similar to a basic RBM 2.236, but with multiplicative interactions. Without this type of interaction it would simply be an RBM that has $\mathbf{x}$ and $\mathbf{y}$ as a concatenated input.

The bias terms in equation 6.3 for $\mathbf{x}$ and $\mathbf{y}$ are necessary in order to obtain a proper distribution: Both $\mathbf{x}$ and $\mathbf{y}$ represent continuous data, without the quadratic containment terms the energy function could be made arbitrarily negative. The exact form of the containment defines the actual conditional probability distribution. In this case it leads to conditional Gaussians:

$$\begin{aligned} p(\mathbf{x} \mid \mathbf{h}, \mathbf{y}, \theta) &= \mathcal{N}(\mathbf{x} \mid \mathbf{V}^T(\mathbf{W}\mathbf{y} \odot \mathbf{U}\mathbf{h}) + \mathbf{a}, \mathbf{I}) \\ p(\mathbf{y} \mid \mathbf{h}, \mathbf{x}, \theta) &= \mathcal{N}(\mathbf{y} \mid \mathbf{W}^T(\mathbf{V}\mathbf{x} \odot \mathbf{U}\mathbf{h}) + \mathbf{b}, \mathbf{I}) \end{aligned} \tag{6.5}$$

The conditional Gaussians are spherical—inputs $\mathbf{x}$ and $\mathbf{y}$ are not modeled, only the invariances with respect to each other. This is best seen when considering $p(\mathbf{x}, \mathbf{y} \mid \mathbf{h}, \theta)$:

$$p(\mathbf{x}, \mathbf{y} \mid \mathbf{h}, \theta) \sim \mathcal{N}\left(\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \middle| \begin{pmatrix} \mathbf{a} \\ \mathbf{b} \end{pmatrix}, \begin{pmatrix} \mathbf{I} & \mathbf{\Sigma} \\ \mathbf{\Sigma}^T & \mathbf{I} \end{pmatrix}^{-1} \right) \tag{6.6}$$

with $\mathbf{\Sigma} = \mathbf{V}^T \mathrm{diag}(\mathbf{U}\mathbf{h}) \mathbf{W}^T$, see Eq. (6.2).

Similar to the standard RBM the conditional distribution for the latent variable is a factorized Bernoulli distribution:

$$p(\mathbf{h} \mid \mathbf{x}, \mathbf{y}, \theta) = \prod_{i=1}^{d} \sigma\left(\left[\mathbf{U}^T(\mathbf{V}\mathbf{x} \odot \mathbf{W}\mathbf{y}) + \mathbf{c} \right]_i \right) \tag{6.7}$$

While the model is defined over the triplet $\mathbf{x}$, $\mathbf{y}$, $\mathbf{h}$, the important aspect is the relationship between pairs $(\mathbf{x}, \mathbf{y})$, described by the marginalized joint distribution $p(\mathbf{x}, \mathbf{y} \mid \theta)$:

$$\begin{aligned} p(\mathbf{x}, \mathbf{y}, \mid \theta) &= \sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{y}, \mathbf{h} \mid \theta) \\ &= \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{a})^T(\mathbf{x} - \mathbf{a}) - \frac{1}{2}(\mathbf{y} - \mathbf{b})^T(\mathbf{y} - \mathbf{b}) \right) \times \\ &\quad \left(\prod_{k=1}^{d} \left(1 + \exp\left(\mathbf{c}_k + \mathbf{u}_k{}^T(\mathbf{V}\mathbf{x} \odot \mathbf{W}\mathbf{y}) \right) \right) \right) / Z \end{aligned} \tag{6.8}$$

where $\mathbf{u}_k$ is the k-th *column* of $\mathbf{U}$ and Z is the normalizing constant defined in Eq. (6.4). Z is not tractable for reasonable large d which is

a problem when doing maximum likelihood learning for the parameters θ. With $\ell(\theta) = \sum_{n=1}^{N} \log p(x_n, y_n \mid \theta)$ being the log-likelihood over a dataset with N observed input pairs, the derivative $\nabla_\theta \ell(\theta)$ of the log-likelihood with respect to the parameters corresponds to the standard derivative-expression for undirected graphical models (see Eq. (2.234)):

$$\begin{aligned}\nabla_\theta \ell(\theta) = \sum_n \Big(&- \mathbb{E}_{p(h|x,y,\theta)} \left[\nabla_\theta \left(E(x, y, h, \theta) \right) \right] \\ &+ \mathbb{E}_{p(h,x,y|\theta)} \left[\nabla_\theta \left(E(x, y, h, \theta) \right) \right] \Big)\end{aligned} \tag{6.9}$$

Optimizing the paramters is usually done with stochastic gradient descent, using mini-batches of samples. The specific gradients for U, V, W, a, b, c can be derived using the Matrix Calculus rules from Section 2.2 (among others, the trace-derivative trick) and are given by the following expressions (these are incorrectly stated in the original paper [372]):

$$\begin{aligned}\nabla_U E(x, y, h, \theta) &= -(Vx \odot Wy)h^T \\ \nabla_V E(x, y, h, \theta) &= -(Wy \odot Uh)x^T \\ \nabla_W E(x, y, h, \theta) &= -(Vx \odot Uh)y^T \\ \nabla_a E(x, y, h, \theta) &= -(x - a) \\ \nabla_b E(x, y, h, \theta) &= -(y - b) \\ \nabla_c E(x, y, h, \theta) &= -h\end{aligned} \tag{6.10}$$

Due to equation Eq. (6.7) sampling from $p(h \mid x, y, \theta)$ is computationally cheap, so the first term of Eq. (6.9) (the *positive phase*) can be approximated with the Monte Carlo principle. However the second term (the *negative phase*) is an average with respect to the *model distribution* over x, y and h. One possible remedy is to apply Contrastive Divergence (CD) (see Eq. (2.242)) where the two groups of variables that get alternatedly sampled in the Gibbs Chain are the pair (x, y) and h. $p(x, y \mid h, \theta)$ is Gaussian (Eq. (6.6)), but its covariance matrix is the inverse of a larger matrix formed by some of the parameters. So every Gibbs step involves a costly matrix inversion which makes standard Contrastive Divergence not applicable here.

Because of the special conditional independence relationship between the three random variables, a three-way Contrastive Divergence approximation [372] can be utilized to generate a Monte Carlo estimate of the second phase: using Gibbs sampling, the distributions $p(h \mid x, y, \theta)$, $p(x \mid h, y, \theta)$ and $p(y \mid h, x, \theta)$ are circularly sampled. The structure between x and y does not imply any sampling order and therefore the ordering for $p(x \mid h, y, \theta)$ and $p(y \mid h, x, \theta)$ should be choosen anew (e.g. randomly) every time h got sampled. In order to avoid overfitting it might be a good idea to regularize the log-likelihood function, e.g. with the Frobenius norm (Eq. (2.30)) of the matrices. If x and y are form the same domain and $D_x \equiv D_y$ then it

might also be a good regularization approach to set $\mathbf{V} \equiv \mathbf{W}$. In this case the model can also be used to determine a latent representation for a single input $\mathbf{x}$ alone (representing a covariance-RBM [300]).

The log-probability of an image pair $(\mathbf{x}, \mathbf{y})$ is given by

$$\begin{aligned}\log p(\mathbf{x}, \mathbf{y} \mid \theta) = &-\frac{1}{2}(\mathbf{x} - \mathbf{a})^T(\mathbf{x} - \mathbf{a}) - \frac{1}{2}(\mathbf{y} - \mathbf{b})^T(\mathbf{y} - \mathbf{b}) \\ &+ \sum_{k=1}^{d} \log\left(1 + \exp\left(\mathbf{c}_k + \mathbf{u_k}^T(\mathbf{V}\mathbf{x} \odot \mathbf{W}\mathbf{y})\right)\right) \\ &- \log Z\end{aligned} \tag{6.11}$$

$\log Z$ can not be computed, but when *comparing (i.e. subtracting)* the log-probabilities of two image pairs it cancels. Hence a *semi-metric* $d(\mathbf{x}, \mathbf{y})$ for an image pair can be defined as

$$\begin{aligned}d(\mathbf{x}, \mathbf{y}) = &-\log p(\mathbf{x}, \mathbf{y} \mid \theta) - \log p(\mathbf{y}, \mathbf{x} \mid \theta) \\ &+ \log p(\mathbf{x}, \mathbf{x} \mid \theta) + \log p(\mathbf{y}, \mathbf{y} \mid \theta)\end{aligned} \tag{6.12}$$

This definition not only removes the sensitivity to how well images themselves are modelled (the last two terms) but also ensures symmetry ($d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$. An alternative way to obtain a symmetric semi-metric is by weight sharing the filters $\mathbf{V}$ and $\mathbf{W}$.

This model is theoretically very nice and in combination with the semi-metric from eq. 6.12 fits exactly the matching problem. However, even with a large amount of different hyperparameter settings I was not able to achieve false positive rates that were close to the SIFT baseline reported in Chapter 3: Overall error rates stayed above 60% using positive pairs from the Liberty training set (this was also true for the other two training sets).

One possible reason for the poor performance might be that the model is overfitting on the training data. This happens easily with standard Contrastive Divergence training, as the probability space outside a small viccinity of the training samples is not explored. For pairs of images this idiosyncrasy is even more problematic. Persistent Contrastive Divergence [387] is an extension of CD to circumvent this problem—but it did not help in the case at hand. The performance did not increase. There also exists an explanation for this observation: The persistent samples are only helpful when they can tackle the difficult space of the probability landscape induced by the model parameters. But this means that the persistent samples should consist of *good false pairs* which is difficult to ensure in the absence of a valid model for a single image—the persistent chain can easily lose track and start producing negative image pairs that don't help with learning at all.

The model performance can be increased considerably if the persistent samples are explicitly paired with valid negative pairs. The resulting model is then no longer semi-supervised, though the negative pairs can simply consist of arbitrary image pairs. More specifically, given a set $\mathcal{N}$ of negative pairs, *Explicit Negative Contrasting*

uses negative pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{N}$ in order to compute a better gradient $\nabla_\theta \ell(\theta)$:

$$\begin{aligned}\nabla_\theta \ell(\theta) = \sum_{m \in \mathcal{N}} \sum_{n} \Big(& -\mathbb{E}_{p(\mathbf{h}|\mathbf{x},\mathbf{y},\theta)} \left[\nabla_\theta \left(E(\mathbf{x}, \mathbf{y}, \mathbf{h}, \theta)\right)\right] \\ & + \frac{1}{2}\big(\mathbb{E}_{p(\mathbf{h},\mathbf{x},\mathbf{y}|\theta)} \left[\nabla_\theta \left(E(\mathbf{x}, \mathbf{y}, \mathbf{h}, \theta)\right)\right] \\ & + \mathbb{E}_{p(\mathbf{h}|\mathbf{x}_m,\mathbf{y}_m,\theta)} \left[\nabla_\theta \left(E(\mathbf{x}, \mathbf{y}, \mathbf{h}, \theta)\right)\right] \big)\Big)\end{aligned} \tag{6.13}$$

The Monte Carlo approximation of the expectation over the model can be done with Persistent Constrastive Divergence and optimization uses mini-batched stochastic gradient descent (i.e. a small subset of positive and negative image pairs is used for training). Note that without the term representing the negative phase the results reported in the next paragraph could not be obtained.

Training on positive and negative image pairs from the Liberty training set resulted in a substantial reduction of error rates. Using 784 factors and 200 latent units, the false positive rate at 95% recall was 29% on the Notre Dame evaluation set and 40% on the Halfe Dome evaluation set. Compared to the baseline models and state-of-the art results for supervised models (see Chapter 3) these results are still far from impressive. The learned filters (Figure 6.1) however show that the involved transformations between image pairs are detected very well—this indicates that better performance could be possible if this new model is tweaked into the right direction. These filters resemble to a large degree the well-known log-polar filters [430], which are best suited to detect rotations. Apart from rotatations the other dominating geometric transformation between image pairs is a scale transformation. This is represented by filters having a log-polar structure at different scales and by filters depicting vortex-like structures. The involved image transformations are heavily non-linear and therefore the learned filters do not look perfectly smooth.

While this new model is not competitive with respect to the matching task, it is nonetheless an intersting option when a more general probabilistic model should be learned. The supervised three-way factored RBM model is much more generally useful, for example it can also be used for denoising or impainting tasks, if one of the inputs is of lower quality (e.g. corrupted).

6.2 EXPLORING EXTENSIONS TO THE SUPERVISED MODEL

For the basic supervised model described in Chapter 4 a set of possible improvements exists:

- Architecture: The dataset encompasses image patches covering a wide range of different scales. While the max-pooling operator seems to be able to resolve some of these scale issues it is

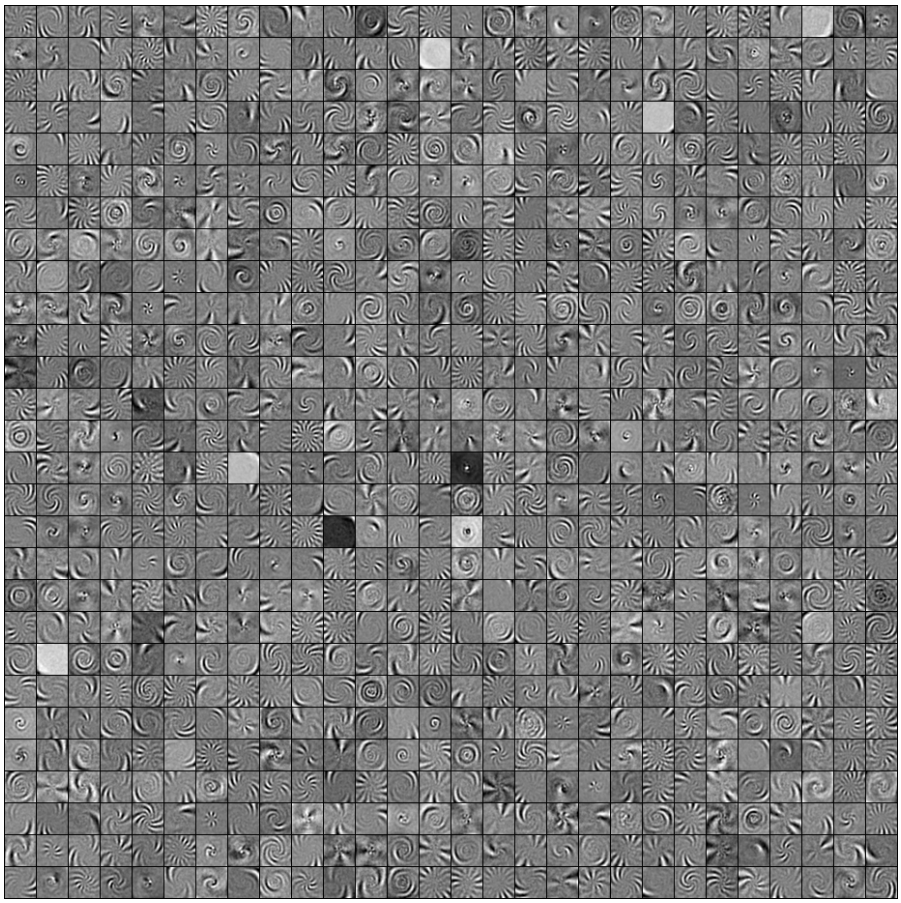


Figure 6.1: Filters for a factorized three-way RBM learned with *explicit contrastive Divergence*. The model has 200 latent binary units, 784 factors and 32×32 inputs. The two projection matrices for the input pairs ($\boldsymbol{V}$, $\boldsymbol{W}$) are shared. Positive and negative image pairs are from the Liberty training set. Optimization was done with mini-batched stochastic gradient descent with momentum, using 50 positive and 50 negative samples each. The persistent chain encompasses also 50 samples.

not perfectly suited for this task. One possible approach is to explicitly model scale with several *complete multi-layer pathways*. A recurrsive attention model on the other hand can handle different scales in an implict way.

- Cost functions: While DrLim is a well-founded heuristic to determine similarity embeddings it has at least two shortcomings. The DrLim objective does too much work: matching descriptors only need to be closer to each other than non-matching descriptors. So ranking-like constraints might be more suitable than DrLim-based cost functions. Furthermore, the resulting embeddings are point estimates which can not capture the inherenct uncertainty of the input data.

6.2.1 *Scale-aware architecture*

If it is possible to decompose the input into different entities that resemble information at different scales than a scale-aware architecture can be built in an explicit manner: Every entity is processed by a deep network and the resulting outputs are combined into one global representation for the original input.

For images a widely used decomposition that was shown to capture scale information well [232] is the Laplacian Pyramid [48]. The Laplacian Pyramid is constructed out of the so-called Gaussian Pyramid. For a given image $\mathbf{x}$ a Gaussian Pyramid is built by subsampling the original image repeatedly. More specifically, let R_k be an operator that subsamples an image by a facotr of k (such a subsampling operator usually has a Gaussian smoothing operator as a first step in order to avoid aliasing effects). The first element P_0 of the Gaussian pyramid is the original image $\mathbf{x}$. The second element is $P_1 = R_k(\mathbf{x})$, the third element is $P_2 = R_k(P_1)$ and so on.

In order to construct elements for the Laplacian Pyramid, an expand operator E_k is necessary, which upsamples an image by a factor of k (this upsampling operator usually has a Gaussian smoothing operator as a final step). The level k of a Laplacian Pyramid is then built by the difference of two consecutive images from the Gaussian Pyramid:

$$L_l = P_l - E_k(P_{l+1}) \tag{6.14}$$

The last element of the Laplacian Pyramid is the last image of the Gaussian Pyramid.

A Laplacian Pyramid with k levels is processed by k deep Convolutional Networks [94]. In the experiments I decided to have a pyramid of depth 3. The network applied to the first level of the pyramid has the same architecture as the baseline network from Chapter 4:

- A convolutional layer with 40 5×5 filters, followed by a standard 2x2 max pooling operation. The activation function is tanh.
- A convolutional layer with 40 5×5 filters, followed by a standard 2x2 max pooling operation. The activation function is tanh.
- A convolutional layer with 40 4×4 filters, followed by a standard 2x2 max pooling operation. The activation function is tanh.
- A convolutional layer with 64 5×5 filters, which reduces every feature layer to a scalar. The activation function is tanh.
- Finally a fully connected linear layer of size 64 x 32.

The second-level image is 32×32, hence the deep network has one layer less:

- A convolutional layer with 40 5×5 filters, followed by a standard 2x2 max pooling operation. The activation function is tanh.
- A convolutional layer with 40 5×5 filters, followed by a standard 2x2 max pooling operation. The activation function is tanh.
- A convolutional layer with 64 5×5 filters, which reduces every feature layer to a scalar. The activation function is tanh.
- Finally a fully connected linear layer of size 64 x 32.

Finally, the 16×16 downsampled original image is processed by a two layer network:

- A convolutional layer with 40 7×7 filters, followed by a standard 2x2 max pooling operation. The activation function is tanh.
- A convolutional layer with 64 5×5 filters, which reduces every feature layer to a scalar. The activation function is tanh.
- Finally a fully connected linear layer of size 64 x 32.

The three 32-dimensional representations are averaged in order to get the final representation[1]. The cost function is $\ell_{\text{DrLim}^{++}}(\theta)$, with $m_{\text{pull}} = 1.5$ and $m_{\text{push}} = 5$. Using stochastic gradient descent (lr=0.01) with minibatches of size 100, the above configuration achieved a false positive rate of 9.2% on the Notre Dame evaluation set and 16.8% on the Half Dome evaluation set.

A methodologically more interesting approach is to resolve the problem of different scales with a recursive attention model [263, 368]. This means that the convolutional model from Chapter 4 is applied several times to the same patch and previous computations are then used to adaptivly change this patch. Specifically the representation computed at iteration k is used to determine a multiplicative filter for the input at iteration $k+1$:

$$\begin{aligned} \mathbf{r}_k &= f(\mathbf{x}_k, \theta) \\ \mathbf{x}_0 &= \mathbf{x} \\ \mathbf{x}_k &= g(\mathbf{r}_k) \odot \mathbf{x} \end{aligned} \tag{6.15}$$

$f : \mathbf{R}^{64\times64} \to \mathbf{R}^{32}$ is the Convolutional Network with four layers from Chapter 4, $\mathbf{x}$ is some input patch and $g : \mathbf{R}^{32} \to \mathbf{R}^{64\times64}$ is a function that maps a representation to an attention patch. I implement g with a deep Convolutional Network (four layers, using perforated upsampling [285] to extend spatial scale) with a sigmoid output layer. For

1 I also tried to learn the weightings of the three subrepresenations with an additional deep gating network [175] which was applied to the original image. This gating network is also a Convolutional Network, but severly limited in capacity in order to keep the number of parameters small. This might be a reason why the approach did not work.

an attention model of length K the representation after K iterations is used for the matching task. The cost function is again $\ell_{\text{DrLim}^{++}}(\theta)$.

All involved operations in Eq. (6.15) are differentiable with respect to parameters, so backpropagation can be applied. However, training this model proved to be quite challening. For $K > 2$ training did not converge (unless a very small learning rate was used, but then no learning did happen). The best results obtained with $K = 2$ were 27.3% for the Notre Dame evaluation set and 39.1% for the Half Dome evaluation set.

6.2.2 *Cost functions*

> One should solve the [classification] problem directly and never solve a more general problem as an intermediate step [...], or, when solving a problem, one should avoid solving a more general problem as an intermediate step. Vladimir Vapnik [402].

While this quote is often used *against* generative modeling, it can also be applied to select between different supervised cost functions (if the problem has a more flexible structure than standard classification). When the goal is to ensure that a threshold for distances can be found such that matching pairs are closer than non-matching pairs then DrLim solves this problem only indirectly: By ensuring that matching pairs have distances smaller than m_{push} and non-matching pairs have distances greater than m_{pull} and with $m_{\text{pull}} < m_{\text{push}}$ the above goal is achieved. However, the task can be represented in a direct way, requiring that the distance $d_\theta(\cdot,\cdot)$ (see Eq. (4.2)) associated with a matching pair (the set of all matching pairs is denoted by $\mathcal{M}$) is well below the distance of any negative pair (the set $\mathcal{N}$):

$$d_\theta(x_1^m, x_2^m) < d_\theta(x_1^n, x_2^n) - c, \quad \forall m \in \mathcal{M}, n \in \mathcal{N} \tag{6.16}$$

with c acting as an additional margin. The corresponding loss function is a ranking constraint:

$$\ell_{\text{rank}}(\theta) = \sum_{m \in \mathcal{M}, n \in \mathcal{N}} \max(d_\theta(x_1^m, x_2^m) - d_\theta(x_1^n, x_2^n) + c, 0) \tag{6.17}$$

In order to ensure that matching pairs form a manifold, this loss function can be extended by a DrLim-like term that pulls matching pairs together:

$$\begin{aligned} \ell_{\text{rank}+}(\theta) = & \sum_{m \in \mathcal{M}} \|d_\theta(x_1^m, x_2^m) - m_{\text{pull}}\|^2 \\ & + \sum_{m \in \mathcal{M}, n \in \mathcal{N}} \max(d_\theta(x_1^m, x_2^m) - d_\theta(x_1^n, x_2^n) + c, 0) \end{aligned} \tag{6.18}$$

While the above two loss functions model the matching task in a direct way it turns out that the underlying optimization is difficult: Standard stoachstic gradient descent fluctuates heavily on the training set and the validation set. A more stable optimization performance can be achieved with AdAM [199]. While for both cost functions the optimization shows very good results for the first few epochs on the respective validation sets, the overall performance is much worse than the baseline cost-function $\ell_{\text{DrLim}^{++}}(\theta)$ from Section 4.3: The standard four-layer Convolutional Network with a 32-dimensional embedding achieves 10.2% and 17.5% on the Notre Dame and Half Dome evaluation sets respectively when trained with $\ell_{\text{rank}}(\theta)$ on the Liberty dataset. For $\ell_{\text{rank}+}(\theta)$ it results in 16.8% and 26.1% false positive rates respectively.

So DrLim seems to be indeed not such a bad idea as a cost function for the matching task. A limitation of the approach might be the fact that the low-dimensional representations are describing point estimates. A point estimate can never express uncertainties with which an input is associated. One possible solution to this aspect is to learn *latent density embeddings* [4, 406].

In order to model a density with a parametric model it needs to be represented in a suitable form. One possible way is to use the sufficient statistics of a density. For example a Gaussian is represented by its mean vector and its covariance matrix.

In this work Gaussians with diagonal covariance matrices are considered, so in order to encode a d-dimensional Gaussian, 2d numbers are necessary. These are the outputs of a linear layer atop a deep Convolutional Network $f(\cdot, \theta)$:

$$\mu(x), \sigma(x) = f(x, \theta) \tag{6.19}$$

The Kullback-Leibler divergence between two distributions can be used to induce a valid distance measure between two inputs x_1 and x_2:

$$\begin{aligned} d(x_1, x_2) = \frac{1}{2}\big(&\mathcal{KL}[\mathcal{N}(x \mid f(x_1, \theta)) || \mathcal{N}(x \mid f(x_2, \theta))] \\ &+ \mathcal{KL}[\mathcal{N}(x \mid f(x_2, \theta)) || \mathcal{N}(x \mid f(x_1, \theta))]\big) \end{aligned} \tag{6.20}$$

where $\mathcal{N}(x \mid f(\cdot, \theta)$ is a Gaussian with parameters μ and Σ determined by the deep Convoluational Network $f(\cdot, \theta)$. For two d-dimensional Gaussians $\mathcal{N}(x \mid \mu_1, \sigma_1^2)$ and $\mathcal{N}(x \mid \mu_2, \sigma_2^2)$ the Kullback-Leibler divergence is given by:

$$\frac{1}{2}\Big(\sum_i \frac{\sigma_{1i}^2}{\sigma_{2i}^2} + \sum_i \frac{(\mu_{1i} - \mu_{2i})^2}{\sigma_{2i}^2} - d + 2\sum_i \log \sigma_{2i} - 2\sum_i \log \sigma_{1i}\Big) \tag{6.21}$$

Given this distance measure the standard approach from Chapter 4 can be applied. More specifically the four-layer Convoluation Network (with tanh activation function and a 32-dimensional output) is

trained using the $\ell_{\text{DrLim}^{++}}(\theta)$ objective. It achieves a false positive rate of 9.0% for Notre Dame and 15.5% for Half Dome when trained on image pairs from the Liberty set. Note that this performance is achieved with a 16-dimensional Gaussian. If a 32-dimensional Gaussian is used (i.e. the Convolutional Network has 64 outputs) the false positive rate improves to 8.2% and 14.8% respectively. In both cases standard minibatched gradient descent with a learning rate of 0.01 is used.

6.2.3 *Dropout*

Sine its introduction in 2012 Dropout [365] has a reputation as an indispensible tool for training good deep supervised Neural Networks. This might be a somewhat myopic viewpoint neglecting the fact that most of the successful applications of Dropout are only classification tasks, mostly limited to object classification for image datasets. And even in this domain, several state-of-the-art results have been achieved without dropout [64, 60].

The basic architecture from Chapter 4 and the embedding task for the matching problem are therefore a good setting to validate the usability of Dropout. Dropout is usually applied to fully-connected layers only. In order to apply it to a convolutional layer I decided to consider complete feature maps as the basic entity that should be masked. However, no improvements could be achieved, neither with tanh nor with ReLU activations. Instead, the performance decreased at least by 10% compared to results reported in Section 4.3.

One hypothesis for these results is that Dropout is detrimental when the model-size is not large enough. Furthermore, a very recent paper [104] indicates that the standard approach of scaling weights with the Dropout-rate during inference is particularly troublesome for convolutional layers. Instead, in this case inference should be done with a proper Monte Carlo approximation.

6.3 IMAGE SIMILARITIES WITH VARIATIONAL AUTOENCODERS

Directed graphical models showed impressive results modeling image patches [200, 262, 307, 132]. Of course, the evaluation of the models used in these reports differs significantly from the matching task. Nevertheless, their ability to model images beyond 16×16 pixels make these approaches an interesting sandbox for experiments with the matching data. Specifically the variational Autoencoder framework (see Section 2.7.4) is appealing as its encoder and decoder architecture allow Convolutional Networks (relying on upsampling techniques [285] for the decoder) and the latent representations can be described as Gaussian embeddings which showed very good performance previously (see Section 6.2).

To make a long experimentation phase of about 12 months short also the variational Autoencoding framework did not lead to better matching performance as a standard deep Autoencoder (see Section 5.4.4). The performance always stayed below the performance of PCA (see Table 5.1). Different input sizes (also down to 16×16 pixels), different types of networks (fully-connected and convolutional), different optimizers (RMSprop, AdaDelta, AdAM) and different variational distributions (Gaussian, Laplacian, log-Gaussian) did not have any appreciable impact on the matching performance.

I observed that the quality of the generated samples was not good and therefore changed the log-likelihood model to a multiscale formulation: The decoder generates three images from a shared latent representation. These images are then used to compose the final (32×32) image. The first image is 8×8 pixels. It is upsampled and added to the second generated image (16×16). This aggregated image is again upsampled and added to the third image, giving the final image. While other methods reported good results with such a multiscale approach [82] the matching performance did not improve.

7 CONCLUSION

From a distance most of the work presented in this thesis follows a well-known pattern over the last years in connectionist research within the field of Computer Vision: Deep supervised networks outperform competing approaches on some benchmark while deep unsupervised models are *not there yet*. However, a closer look reveals a much more nuanced picture.

One of the most important differences to other work involving deep Convolutional Networks is the dataset (Chapter 3). It is deliberately chosen to be not a high-level Computer Vision task but to have labels that only resemble a similarity relation based on principled physical properties of the pictured world. This is an important aspect when one is concerned with evaluating unsupervised methods as the current approaches to do so are insufficient [385]. With a similarity task that is as free from overlayed human interpretation (e.g. the labels of high-level object classification) as possible the hope was that unsupervised learning algorithms can truly show their potential and a better assessment of these kinds of algorithms is possible.

However, this hope was only partially fulfilled: most investigated methods failed. It is not completely clear if this is due to the limited capacity of some of the models or due to the fact that at least for the matching task unsupervised models also need access to some of the mechanism that generates valid pairs (i.e. multi-view information must be available to a method)[1]. In a certain way unsupervised learning also needs to integrate some aspects of the target space, an idea that was recently put forward in a more concrete instantiation [374]. What this means in a very abstract sense is that the definition of unsupervised learning needs to be refined, and this refinement then might bring also better methods to evaluate such models in a consistent way [385].

The supervised model seems to be a boring repetition of the yet-another-deep-convolutional-network showcase. It is not! The utilized architecture has a particularly small memory footprint both with respect to the number of parameters and the final representation. These aspects are widely neglected in typical research efforts that try to improve on some state-of-the-art results. However they are crucial for real-world applications, e.g. in cars or on mobile phones. Nonetheless it is quite possible to achieve competitive performance with a

1 One might argue that most learning by humans is unsupervised. However, humans have a really good implicit teacher—physics. For example, our vision system does not observe a sequences of arbitrary images but instead a never ending movie whose frames adhere strongly to temporal and spatial constraints.

constrained model, too. A powerful optimizer seems to be most important in such a case. Additionally, first experiments also indicate that a more expressive way to represent an entity can lead to unexpected improvements (see Section 6.2).

I also presented some general algorithmic ideas independent of the empirical evaluation for both supervised as well as unsupervised Neural Network-based models. *Explicit negative Contrastive divergence* was evaluated with respect to the matching task, but needs to be validated on more multi-view datasets. Similarly, variational autoencoded clustering (see end of Section 2.7.4) needs to be evaluated on a wider range of datasets. Finally, *Hobbesian Networks*, also introduced in Section 2.7.4 where only presented together with a qualitative assessment and are in need of the same comprehensive quantitative evaluation.

FUTURE WORK

One of the main motivations for training models on low-level image patches is the goal to build a general Computer Vision system in a bottom-up manner. Clearly, this has not been attempted at all and is therefore one of the major tasks for the future.

Apart from this more general point of view a large set of open questions and undone experiments exist with the presented line of work. For the supervised modeling task one might be primarily interested on the existing four-layer deep convolutional architecture. It seems that currently the following investigations seem to be most promising in order to improve the performance:

- Batch Normalization [171]: In the evaluations it was empirically shown, a better optimizer indeed leads to a much better performance. Batch-Normalziation is an orthogonal component to any optimization procedure, improving the learning capacity of existing networks. Actually, if the results from the Batch-Normalization paper are transferable to the matching setting, than this might be the easiest way to considerably improve the matching performance.

- Fracking [348]: After the model is trained for some time most of the training pairs, positive as well as negative no longer carry useful information. Instead, *informative* training samples must be selected. One approach is *fracking* which improved the performance of the trained models by 25%. In addition to fracking, adversarial training [122] on input pairs might also improve the performance.

- Student-Teacher training: A recent trend is to *distill* [47, 148, 206] a large teacher network into a more compact student network. This would allow to first learn a very powerful (i.e. having many

parameters) teacher network mapping patches to a compact representation and then use these representations as targets for a student network that is also limited in the number of parameters. The teacher network can also be used to produce additional training data, mapping arbitrary patches to representations. In this case the student network might also be trained with an additional DrLim cost for the patches from the original training set (however the contrastive term can be left out, as no degenerate solutions are possible).

- Convolutional Dropout for regression [104]: I explained in Section 6.2 that Dropout did not help at all to improve the matching performance which might be due to the small model-size. A more probably hypothesis is that for Convolutional Layers inference must be done relying on Monte Carlo approximations [104] which I did not do. Even then it is not clear if any improvement can be achieved as there are no reports for successful applications of Dropout to regression problems. A Monte Carlo inference scheme induces a different way to compute distances between representation, for example as is demonstrated in Section 6.2.

Instead of the standard deep convolutional architecture, different model classes can also be used. One possible idea is to add an additional attention module to the baseline architecture in the form of a Spatial Transformer Network [177]. This module should particularly help in dealing with different scales of patches, relieving the max-pooling operation from this task. Similar to the recursive attention model described in Section 6.2 this approach can be extended by iteratively applying the complete model to a given patch. The recurrent (feedback) signal for the next iteration is again based on the computed representation of the current iteration and will be fed into the Spatial Transformer module.

Recently [384, 369] Multidimensional Recurrent Neural Networks [127] (MDRNNs) emerged as strong alternatives to Convolutional Networks for Computer Vision tasks. Previous work already demonstrated how standard RNNs can be used to learn fixed representations [17] so it is straightforward to adapt MDRNNs to the matching task of this work.

The results attained with the unsupervised models (Chapter 5) were not convincing. One hypothesis is that the utilized models were not powerful enough for the type of data. It therefore is a valid attempt to have another round of experiments if the models in question appear powerful enough. Recently two candidate models, DRAW [132] and LAPGAN [82] have been introduced, outperforming other approaches on various modeling tasks related to images. Of course, both models should also be extended to multi-view data and then

evaluated on the matching task, similar to the three-way RBM in Section 6.1.

Generative Adversarial Networks (GAN) [121] do not infer latent representations for an input sample. One remedy might be to utilize some hidden representation of its discriminative network, another possible solution is to use gradient descent in the input space of the generative network for a given input.

While these two models represent reasonable and valid candidates for further experiments they do not resolve the fundamental problem that is associated with latent representations of unsupervised methods: the representations are not discriminative enough. In Natural Language Processing the problem of finding discriminative representations of the basic entities (*words*) was recently successfully tackled with the simple and elegant `word2vec` approach [261]. While it might be debatable if this method can be called *unsupervised* (see my comments at the beginning of this chapter), the word representations are learned relying only on structural information of the problem domain. For Computer Vision a similar method `patch2vec` can be defined which only relies on structural information contained in images (or videos). For the case of images this structural information might be given by the patches at keypoints (a well-defined mathematical concept [232, 239]). Patches at keypoints should then have a representation that has predictive capacity. That is, given a patch and the set of its neighboring patches, the representation of the patch should be predictable by the representation of the other patches. Representations are computed by a Deep Convolutional Network which might also take into account the wider context of the respective patch (i.e. a larger part of the overall image). Patches with different scales can be handled in an unifying way using Spatial Pyramid Pooling [145].

Part II

APPENDIX

A

EM

Expectation Maximization is a method for finding maximum likelihood solutions for probability models having *latent* (unobserved) variables. This short exposure is mainly based on Neal and Hinton [273]. It presents EM as an algorithm for maximizing a joint function of the parameters and of the distribution over the unobserverd variables (this differs from the standard presentation (e.g. [254]). In particular this viewpoint justifies incremental versions of the algorithm (partial M-steps **and** partial E-steps).

I denote the set of observed variables by $\mathcal{X}$, the set of latent variables by $\mathcal{Z}$ and the parameters of the probability model by θ. The goal is to find the maximum likelihood estimates for θ given $\mathcal{X}$. However, suppose that direct optimization of $\log p(\mathcal{X} \mid \theta) := \ell(\theta)$ is difficult, optimizing the *complete-data* log likelihood $\log p(\mathcal{X}, \mathcal{Z} \mid \theta)$ on the other hand is significantly easier (i.e. computationally tractable).

A.1 DERIVATION

After introducing an arbitrary probability distribution $q(\mathcal{Z})$ defined over the latent variables, one obtains a lower bound on the log likelihood $\ell(\theta)$ (using Jensen's inequality, which can be applied because log is concave and $q(\mathcal{Z})$ is a distribution).

$$\ell(\theta) = \log \sum_{\mathcal{Z}} q(\mathcal{Z}) \frac{p(\mathcal{X}, \mathcal{Z})}{q(\mathcal{Z})} \geqslant \sum_{\mathcal{Z}} q(\mathcal{Z}) \log \frac{p(\mathcal{X}, \mathcal{Z} \mid \theta)}{q(\mathcal{Z})} := \mathcal{F}(q, \theta) \quad \text{(A.1)}$$

This assumes *discrete* latent variables (otherwise use $\int$ instead of $\sum$, with some additional constraints on $q(\mathcal{Z})$). $\mathcal{F}(q, \theta)$ is actually a term common in statistical physics and usually called *free energy*. It can be decomposed as follows:

$$\begin{aligned} \mathcal{F}(q, \theta) &= \int q(\mathcal{Z}) \log \frac{p(\mathcal{X}, \mathcal{Z} \mid \theta)}{q(\mathcal{Z})} d\mathcal{Z} = \\ &= \int q(\mathcal{Z}) \log p(\mathcal{X}, \mathcal{Z} \mid \theta) d\mathcal{Z} - \int q(\mathcal{Z}) \log q(\mathcal{Z}) d\mathcal{Z} = \\ &= \langle \log p(\mathcal{X}, \mathcal{Z} \mid \theta) \rangle_{q(\mathcal{Z})} + \mathcal{H}[q] \end{aligned} \quad \text{(A.2)}$$

$\langle \cdot \rangle_{q(\mathcal{Z})}$ denotes hereby the expectation under the distribution $q(\mathcal{Z})$, $\mathcal{H}[\cdot]$ the entropy.

Expectation Maximization is an iterative algorithm, alternating between the following two steps:

- **E-Step:** Maximize $\mathcal{F}(q, \theta)$ with respect to the distribution over the hidden variables, holding the parameters θ fixed:

$$q^k(\mathcal{Z}) := \arg\max_{q(\mathcal{Z})} \mathcal{F}(q(\mathcal{Z}), \theta^{(k-1)})) \tag{A.3}$$

- **M-Step:** Maximize $\mathcal{F}(q, \theta)$ with respect to the parameters θ, keeping the distribution $q(\mathcal{Z})$ fixed (the entropy term does not depend on θ):

$$\theta^k := \arg\max_{\theta} \mathcal{F}(q^k(\mathcal{Z}), \theta)) = \arg\max_{\theta} \langle \log p(\mathcal{X}, \mathcal{Z} \mid \theta) \rangle_{q^k(\mathcal{Z})} \tag{A.4}$$

So EM is basically a coordinate ascent in $\mathcal{F}$. What's happening in the E-Step becomes much clearer when we rewrite the free energy:

$$\begin{aligned}
\mathcal{F}(q, \theta) &= \int q(\mathcal{Z}) \log \frac{p(\mathcal{X}, \mathcal{Z} \mid \theta)}{q(\mathcal{Z})} d\mathcal{Z} \\
&= \int q(\mathcal{Z}) \log \frac{p(\mathcal{Z} \mid \mathcal{X}, \theta) p(\mathcal{X} \mid \theta)}{q(\mathcal{Z})} d\mathcal{Z} \\
&= \int q(\mathcal{Z}) \log p(\mathcal{X} \mid \theta) d\mathcal{Z} + \int q(\mathcal{Z}) \log \frac{p(\mathcal{Z} \mid \mathcal{X}, \theta)}{q(\mathcal{Z})} d\mathcal{Z} \\
&= \log p(\mathcal{X} \mid \theta) \int q(\mathcal{Z}) d\mathcal{Z} - \mathcal{KL}[q(\mathcal{Z}) \| p(\mathcal{Z} \mid \mathcal{X}, \theta)] \\
&= \ell(\theta) - \mathcal{KL}[q(\mathcal{Z}) \| p(\mathcal{Z} \mid \mathcal{X}, \theta)]
\end{aligned} \tag{A.5}$$

$\mathcal{KL}[\cdot\|\cdot]$ denotes hereby the *Kullback-Leibler divergence* between two probability distributions. Eq. (A.5) says that for fixed θ, $\mathcal{F}$ is bounded from above by ℓ, illustrated in Figure A.1 graphically, and this bound is achieved with $\mathcal{KL}[q(\mathcal{Z}) \| p(\mathcal{Z} \mid \mathcal{X}, \theta)] = 0$. But for two distributions p, q, $\mathcal{KL}[p\|q] = 0$ if and only if $p = q$. So the E-Step simply sets

$$q^k(\mathcal{Z}) = p(\mathcal{Z} \mid \mathcal{X}, \theta^k) \tag{A.6}$$

and after the E-Step, the free energy equals the likelihood (for the given θ). Still, $p(\mathcal{Z} \mid \mathcal{X}, \theta)$ may be very complex. However, in standard applications our data items are independently and identical distributed. From this independence assumption we have $p(\mathcal{X}, \mathcal{Z} \mid \theta) = \prod_n p(x_n, z_n \mid \theta)$ (we assumed here that $\mathcal{X}$ contains n elements x_n and $\mathcal{Z}$ respectively n elements z_n) and by marginalizing over $\mathcal{Z}$ we have

$$\begin{aligned}
p(\mathcal{X} \mid \theta) = \sum_{\mathcal{Z}} p(\mathcal{X}, \mathcal{Z} \mid \theta) = \sum_{\mathcal{Z}} \prod_n p(x_n, z_n \mid \theta) = \\
= \prod_n \sum_{z_n} p(x_n, z_n \mid \theta) = \prod_n p(x_n \mid \theta)
\end{aligned} \tag{A.7}$$

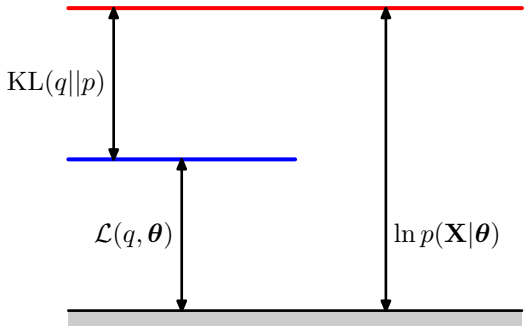


Figure A.1: The decomposition given by Eq. (A.5). The free energy $\mathcal{F}(q, \theta)$ is a lower bound on $\ell(\theta)$, because the Kullback-Leibler divergence is always nonnegative. The figure is taken from [35].

Therefore, the posterior probability $p(\mathcal{Z} \mid \mathcal{X}, \theta)$ factorizes with respect to n:

$$\begin{aligned} p(\mathcal{Z} \mid \mathcal{X}, \theta) &= \frac{p(\mathcal{X}, \mathcal{Z} \mid \theta)}{\sum_{\mathcal{Z}} p(\mathcal{X}, \mathcal{Z} \mid \theta)} = \frac{\prod_{n=1}^{N} p(\mathbf{x}_n, \mathbf{z}_n \mid \theta)}{\prod_{n=1}^{N} p(\mathbf{x}_n \mid \theta)} = \\ &= \frac{\prod_{n=1}^{N} p(\mathbf{z}_n \mid \mathbf{x}_n, \theta) p(\mathbf{x}_n \mid \theta)}{\prod_{n=1}^{N} p(\mathbf{x}_n \mid \theta)} = \prod_{n=1}^{N} p(\mathbf{z}_n \mid \mathbf{x}_n, \theta) \end{aligned} \tag{A.8}$$

It is now straighforward to see that the E-Step and M-Step together *never* decrease the likelihood:

- The E-Step brings the free energy to the likelihood.

$$\ell(\theta^{k-1}) = \mathcal{F}(q^k, \theta^{k-1}) \tag{A.9}$$

- The M-Step optimizes the free energy with respect to θ.

$$\mathcal{F}(q^k, \theta^{k-1}) \leqslant \mathcal{F}(q^k, \theta^k) \tag{A.10}$$

- As already shown in Eq. (A.1), the free energy is upper bounded by the likelihood.

$$\mathcal{F}(q^k, \theta^k) \leqslant \ell(\theta^k) \tag{A.11}$$

Note that even though I describe both the E-step and the M-step as maximization operations, there is no need for an actual maximization operation. The M-step of the algorithm may be only *partially* implemented, as long as it results in an improvement of $\mathcal{F}(q, \theta)$ which always leads to the true likelihood improving as well. [254] refer to such variants as *generalized EM* algorithms. There is clearly also no need to use the optimal distribution over hidden configurations. One can use any distribution that is convenient so long as the distribution is always updated in a way that improves $\mathcal{F}$. This is usally referred as a *partial E-step*. For example, considering Eq. (A.8), a partial E-step

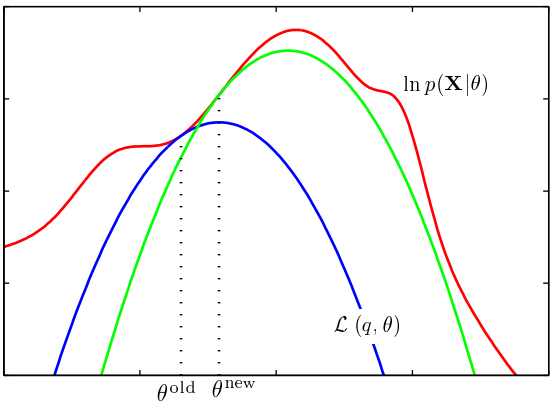


Figure A.2: Given some initial parameter value θ^{old}, the E-Step (evaluating the posterior distribution over latent variables) brings the value of $\mathcal{F}(q^k, \theta^{\text{old}})$ to $\ell(\theta^{\text{old}})$, as shown by the blue curve (the bound makes a tangential contact). Note that the lower bound need not be a concave function with a unique maximum, as depicted here. In the M-Step, the bound is optimized giving the value θ^{new}. The procedure is then repeated. Figure is taken from [35].

would be to update $p(z_i \mid x_i, \theta)$ for one particular i and then apply an M-step.

The operation of the EM algorithm can also be viewed in the space of the parameters θ, as illustrated schematically in Figure A.2.

It is still necesary to show that maxima in $\mathcal{F}$ correspond to maxima in ℓ. Consider a fix point $(q^*(\mathcal{Z}), \theta^*)$ of the EM algorithm. Then we have:

$$\frac{\partial}{\partial\theta}\langle \log p(\mathcal{X}, \mathcal{Z} \mid \theta)\rangle_{q^*(\mathcal{Z})} \mid_{\theta^*} = 0 \tag{A.12}$$

Thus:

$$\begin{aligned}\ell(\theta) = \log p(\mathcal{X} \mid \theta) &= \langle \log p(\mathcal{X} \mid \theta)\rangle_{q^*(\mathcal{Z})} = \\ &= \left\langle \log \frac{p(\mathcal{X}, \mathcal{Z} \mid \theta)}{p(\mathcal{Z} \mid \mathcal{X}, \theta)} \right\rangle_{q^*(\mathcal{Z})} = \\ &= \langle \log p(\mathcal{X}, \mathcal{Z} \mid \theta)\rangle_{q^*(\mathcal{Z})} - \langle \log p(\mathcal{Z} \mid \mathcal{X}, \theta)\rangle_{q^*(\mathcal{Z})}\end{aligned} \tag{A.13}$$

and so:

$$\frac{d}{d\theta}\ell(\theta) = \frac{d}{d\theta}\langle \log p(\mathcal{X}, \mathcal{Z} \mid \theta)\rangle_{q^*(\mathcal{Z})} - \frac{d}{d\theta}\langle \log p(\mathcal{Z} \mid \mathcal{X}, \theta)\rangle_{q^*(\mathcal{Z})} \tag{A.14}$$

Finally,

$$\frac{d}{d\theta}\ell(\theta) \mid_{\theta^*} = 0 \tag{A.15}$$

so the EM algorithm converges to a stationary point of $\ell(\theta)$.

Additionally, the type of such a stationary point must be a maximum in ℓ: Consider $\frac{d^2}{d\theta^2}\ell(\theta)$:

$$\frac{d^2}{d\theta^2}\ell(\theta) = \frac{d^2}{d\theta^2}\langle \log p(\mathcal{X}, \mathcal{Z} \mid \theta)\rangle_{q^*(\mathcal{Z})} - \frac{d^2}{d\theta^2}\langle \log p(\mathcal{Z} \mid \mathcal{X}, \theta)\rangle_{q^*(\mathcal{Z})}$$

(A.16)

With $\theta = \theta^*$, the first term on the right is negative (a maximum) and (given certain assumptions[1], see [422, 254] for more details) the second term is positive. Thus the curvature of the likelihood is negative and θ^* is a maximum of ℓ.

RATE OF CONVERGENCE. In general, the rate of convergence of the EM algorithm is reported to be slow (linear, or even sublinear). See [422, 254] for a detailed analysis.

A.2 APPLICATION: MIXTURE OF GAUSSIANS

Consider a superposition of K Gaussian densities of the form

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{A.17}$$

with the *mixing coefficients* π_k being normalized: $\sum_k \pi_k = 1$. Also, with $p(\mathbf{x}) \geqslant 0$ (it is a probability distribution) and $\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \geqslant 0$, we get $\pi_k \geqslant 0$ for $k = 1 \ldots K$. Note that from a generative view point, π_k is he prior probability of picking the k^{th} component and the density $\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is the probability of $\mathbf{x}$ conditioned on k. Given a set of n observations $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, one way to set the values of the parameters ($\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ and π_k (for all k)) is to use maximum likelihood. The loglikelihood function is

$$\log p(\mathcal{X} \mid \theta) = \sum_{n=1}^{n} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right) \tag{A.18}$$

with $\theta = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_k, \pi_1, \ldots, \pi_k)$. As a result of the presence of the summation inside the logarithm the maximum likelihood solution for the parameters no longer has a closed form analytical solution. One could use an iterative numeric optimization technique but instead, consider EM.

For a given datapoint $\mathbf{x}$ the associated latent variable z indicates which of the K Gaussians generated the data point. Thus $p(z = k \mid \mathbf{x}, \theta)$ denotes the probability that $\mathbf{x}$ was generated by the k^{th} component (these probabilities are also known as *responsibilities*). Bayes' theorem results in

$$p(z_i = k \mid \mathbf{x}_i, \theta) = \frac{\pi_k \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{c=1}^{K} \pi_c \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)} =: \gamma_{ik}. \tag{A.19}$$

Assuming i.i.d data Eq. (A.8) can be applied, so the E-step is straightforward.

1 θ^* can also be a saddle point

For the M-step the functional form of the expected complete log-likelihood $\langle \log p(\mathcal{X}, \mathcal{Z} \mid \theta) \rangle_{p(\mathcal{Z}|\mathcal{X},\theta)}$ is necessary:

$$\begin{aligned}\langle \log p(\mathcal{X}, \mathcal{Z} \mid \theta) \rangle_{p(\mathcal{Z}|\mathcal{X},\theta)} &= \sum_{\mathcal{Z}} p(\mathcal{Z} \mid \mathcal{X}, \theta) \left(\sum_{i=1}^{n} \log p(x_i, z_i \mid \theta) \right) = \\ &= \sum_{z_1=1}^{K} \sum_{z_2=1}^{K} \cdots \sum_{z_n=1}^{K} \left(\prod_{j=1}^{n} p(z_j \mid x_j, \theta) \times \right. \\ &\quad \left. \left(\sum_{i=1}^{n} \log p(x_i, z_i \mid \theta) \right) \right)\end{aligned} \tag{A.20}$$

Here, I have used the definition of the expectation and the result of Eq. (A.8). This formula must be restructured in order to be able to proceed. The idea is to find a reformulation only for one fixed data point x_i. To get to an expression that is equivalent to the previous one, a sum over all data points is necessary:

$$\sum_{i=1}^{n} \sum_{k=1}^{K} \log p(x_i, z_i = k \mid \theta) \left(\sum_{z_1=1}^{K} \sum_{z_2=1}^{K} \cdots \sum_{z_n=1}^{K} \delta_{k,z_i} \prod_{j=1}^{n} p(z_j \mid x_j, \theta) \right) \tag{A.21}$$

with δ_{k,z_i} the Kronecker delta. The second part of this expression can be greatly simplified because for a given i and a given k one can write:

$$\begin{aligned}&\sum_{z_1=1}^{K} \sum_{z_2=1}^{K} \cdots \sum_{z_n=1}^{K} \delta_{k,z_i} \prod_{j=1}^{n} p(z_j \mid x_j, \theta) \\ &= \left(\sum_{z_1=1}^{K} \cdots \sum_{z_{i-1}=1}^{K} \sum_{z_{i+1}=1}^{K} \cdots \sum_{z_n=1}^{K} \prod_{j=1, j\neq i}^{n} p(z_j \mid x_j, \theta) \right) p(z_i = k \mid x_i, \theta) \\ &= \left(\prod_{j=1, j\neq i}^{n} \left(\sum_{z_j=1}^{K} p(z_j \mid x_j, \theta) \right) \right) p(z_i = k \mid x_i, \theta) = 1 \cdot \gamma_{ik}\end{aligned}$$

This results in a simple expression for the expected complete log-likelihood $\langle \log p(\mathcal{X}, \mathcal{Z} \mid \theta) \rangle_{p(\mathcal{Z}|\mathcal{X},\theta)}$ of *any finite mixture model*:

$$\sum_{i=1}^{n} \sum_{k=1}^{K} \gamma_{ik} \left(\log p(x_n \mid z_i = k, \theta) + \log p(x_i = k) \right) \tag{A.22}$$

Note that the previous derivations (for the M-step) did not take into account that a mixture of *gaussians* (i.e. $p(x_i \mid z_i = k, \theta) = \mathcal{N}(x_i \mid \mu_k, \Sigma_k)$) is used. Optimizing Eq. (A.22) with respect to the parameter

set $\theta = (\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \pi_1, \ldots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K, \pi_K)$ results in the following equations (the exact derivations of these are not shown here):

$$\begin{aligned} \boldsymbol{\mu}_k{}^{\text{new}} &= \frac{\sum_{i=1}^{n} \gamma_{ik} \mathbf{x}_i}{\sum_{i=1}^{n} \gamma_{ik}} \\ \boldsymbol{\Sigma}_k{}^{\text{new}} &= \frac{\sum_{i=1}^{n} \gamma_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k{}^{\text{new}})(\mathbf{x}_i - \boldsymbol{\mu}_k{}^{\text{new}})^T)}{\sum_{i=1}^{n} \gamma_{ik}} \\ \pi_k^{\text{new}} &= \sum_{i=1}^{n} \frac{\gamma_{ik}}{n} \end{aligned} \tag{A.23}$$

K-Means

Probabilistic models can often be transformed to non-probabilistic variants by considering some limit-behaviour. Consider a mixture of K isotropic Gaussians (MoG), each with the same covariance $\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}$. In the limit $\sigma^2 \to 0$ the EM algorithm for MoG converges to the K-Means algorithm [35]. The K-Means algorithm is a very simple clustering algorithm, formed by an iterative two-step procedure:

- Partition the dataset according to the closest distance of samples to the respective cluster centers.
- (Re-)Compute the cluster center for every partition by averaging the respective samples in a partition.

The second step corresponds to the M-step from the previous derivation with responsibilities fixed to the uniform distribution over a given partition. In the general setting of the MoG model, The E-step for the MoG model needs to take the limiting behaviour of σ into account. For some data point $\mathbf{x}_i$ it is:

$$\begin{aligned} p(z_i = k \mid \mathbf{x}_i) &= \frac{\pi_k \exp\left(\frac{-||\mathbf{x}_i - \boldsymbol{\mu}_k||^2}{2\sigma^2}\right)}{\sum_l \pi_l \exp\left(\frac{-||\mathbf{x}_i - \boldsymbol{\mu}_l||^2}{2\sigma^2}\right)} \\ &= \frac{1}{\sum_l \frac{\pi_l}{\pi_k} \exp\left(\frac{-||\mathbf{x}_i - \boldsymbol{\mu}_l||^2 + ||\mathbf{x}_i - \boldsymbol{\mu}_k||^2}{2\sigma^2}\right)} \end{aligned} \tag{A.24}$$

If k denotes the component that is closest to $\mathbf{x}_i$, then $||\mathbf{x}_i - \boldsymbol{\mu}_l||^2 \geqslant ||\mathbf{x}_i - \boldsymbol{\mu}_k||^2$ for all l, then $-||\mathbf{x}_i - \boldsymbol{\mu}_l||^2 + ||\mathbf{x}_i - \boldsymbol{\mu}_k||^2 \leqslant 0$ for all l and thus the denominator converges to 1 if $\sigma^2 \to 0$ (because if $l = k$, this part of the sum in the denomiator is always 1, and all other summands converge to 0 because $\exp(-\infty)$ does so).

On the other hand, if k is not resembling the closest component, then $-||\mathbf{x}_i - \boldsymbol{\mu}_l||^2 + ||\mathbf{x}_i - \boldsymbol{\mu}_k||^2 > 0$ for l denoting the closest component, and whith $\sigma^2 \to 0$ the exponent of this component is

$$\frac{-||\mathbf{x}_i - \boldsymbol{\mu}_l||^2 + ||\mathbf{x}_i - \boldsymbol{\mu}_k||^2}{2\sigma^2} \to +\infty \tag{A.25}$$

and thus the denominator converges to ∞. In total, this results in the hard assignment step of K-Means.

A.3 FACTOR ANALYSIS

Factor Analysis is a directed graphical model that represents a linear Gaussian system (Eq. (2.137)):

$$\begin{aligned} p(\mathbf{h}) &= \mathcal{N}(\mathbf{h} \mid \mathbf{0}, \mathbf{I}) \\ p(\mathbf{x} \mid \mathbf{h}, \theta) &= \mathcal{N}(\mathbf{x} \mid \mathbf{W}\mathbf{h} + \boldsymbol{\mu}, \boldsymbol{\Psi}) \end{aligned} \tag{A.26}$$

with $\theta \equiv \{\boldsymbol{\mu}, \mathbf{W}, \boldsymbol{\Psi}\}$. Inverting the system is therefore straightforward (Eq. (2.143)):

$$p(\mathbf{h} \mid \mathbf{x}, \theta) = \mathcal{N}(\mathbf{h} \mid \mathbf{m}, \boldsymbol{\Sigma}) \tag{A.27}$$

with

$$\boldsymbol{\Sigma} = (\mathbf{I} + \mathbf{W}^T \boldsymbol{\Psi}^{-1} \mathbf{W})^{-1} \tag{A.28}$$

and

$$\mathbf{m} = \boldsymbol{\Sigma}(\mathbf{W}^T \boldsymbol{\Psi}^{-1}(\mathbf{x} - \boldsymbol{\mu})) \tag{A.29}$$

In statistics $\mathbf{m}$ is often denoted the *score* of a sample $\mathbf{x}$. Note that the posterior covariance $\boldsymbol{\Sigma}$ is independent of $\mathbf{x}$, i.e. the posterior uncertainty is independent of the observation (an obvious weakness of the model).

Fitting the parameters θ using Maximum Likelihood Estimation might appear to be simple, because $p(\mathbf{x} \mid \boldsymbol{\mu}, \mathbf{W}, \boldsymbol{\Psi})$ is a Gaussian (Eq. (2.140)):

$$p(\mathbf{x} \mid \boldsymbol{\mu}, \mathbf{W}, \boldsymbol{\Psi}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi}) \tag{A.30}$$

The Maximum Likelihood Estimation of $\boldsymbol{\mu}$ is therefore (see Eq. (2.225), assuming n observations $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$):

$$\boldsymbol{\mu} = \frac{1}{n} \sum_i \mathbf{x}_i \tag{A.31}$$

However, there are no closed-form solutions possible for $\mathbf{W}$ and $\boldsymbol{\Psi}$, because the covariance matrix $\mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi}$ appears both in an inverse as well as a determinant in the log-likelihood $\log p(\mathbf{x} \mid \theta)$. One possible solution is therefore gradient based optimization. Because the posteriors are Gaussians, the EM algorithm is a tractable alternative approach to fit $\mathbf{W}$ and $\boldsymbol{\Psi}$. Eq. (A.27) realizes already the E-step.

For the M-step the free energy $\mathcal{F}(p(\mathbf{h} \mid \cdot), \theta) = \mathbb{E}_{p(\mathbf{h}|\mathbf{x},\theta)}[p(\mathbf{x}, \mathbf{h} \mid \theta)] + \mathcal{H}[q]$ must be optimized with respect to θ. First, write only those terms of $\mathcal{F}(p(\mathbf{h} \mid \cdot)$ that involve θ:

$$\begin{aligned}
\mathcal{F}(\cdot) &\propto \sum_i \mathbb{E}_{p(\mathbf{h}|\cdot)} \left[\log \left(\frac{1}{\sqrt{|2\pi\Psi|}} \times \right. \right. \\
&\quad \left. \left. \exp\left(-\frac{1}{2} (\mathbf{x}_i - \mu - W\mathbf{h}_i)^\top \Psi^{-1} (\mathbf{x}_i - \mu - W\mathbf{h}_i) \right) \right) \right] \\
&\propto \sum_i \mathbb{E}_{p(\mathbf{h}|\cdot)} \left[-\frac{1}{2} \log|\Psi| - \frac{n}{2} \log(2\pi) - \right. \\
&\quad \left. \frac{1}{2} (\mathbf{x}_i - \mu - W\mathbf{h}_i)^\top \Psi^{-1} (\mathbf{x}_i - \mu - W\mathbf{h}_i) \right] \\
&\propto \sum_i \mathbb{E}_{p(\mathbf{h}|\cdot)} \left[-\frac{1}{2} (\mathbf{x}_i - \mu - W\mathbf{h}_i)^\top \Psi^{-1} (\mathbf{x}_i - \mu - W\mathbf{h}_i) \right] \\
&\propto \sum_i \mathbb{E}_{p(\mathbf{h}|\cdot)} \left[(\mathbf{x}_i - \mu)^\top \Psi^{-1} W\mathbf{h}_i - \frac{1}{2} \mathbf{h}_i W^\top \Psi^{-1} W\mathbf{h}_i \right]
\end{aligned} \tag{A.32}$$

Because of the pathwise derivative estimator (Eq. (2.208), that is, derivative and expectation operator can be swapped) it is now simple to continue:

$$\begin{aligned}
\nabla_W \mathcal{F}(\cdot) &\propto \nabla_W \sum_i \mathbb{E}_{p(\mathbf{h}|\cdot)} \left[(\mathbf{x}_i - \mu)^\top \Psi^{-1} W\mathbf{h}_i - \frac{1}{2} \mathbf{h}_i W^\top \Psi^{-1} W\mathbf{h}_i \right] \\
&= \sum_i \mathbb{E}_{p(\mathbf{h}|\cdot)} \left[\nabla_W \left(\operatorname{tr}\left((\mathbf{x}_i - \mu)^\top \Psi^{-1} W\mathbf{h}_i \right) - \right. \right. \\
&\quad \left. \left. \frac{1}{2} \operatorname{tr}\left(\mathbf{h}_i W^\top \Psi^{-1} W\mathbf{h}_i \right) \right) \right] \\
&= \sum_i \mathbb{E}_{p(\mathbf{h}|\cdot)} \left[\mathbf{h}_i (\mathbf{x}_i - \mu)^\top \Psi^{-1} - \mathbf{h}_i \mathbf{h}_i^\top W^\top \Psi^{-1} \right] \\
&= \sum_i \left(\mathbb{E}_{p(\mathbf{h}|\cdot)}[\mathbf{h}_i] (\mathbf{x}_i - \mu)^\top \Psi^{-1} - \mathbb{E}_{p(\mathbf{h}|\cdot)}[\mathbf{h}_i \mathbf{h}_i^\top] W^\top \Psi^{-1} \right) \\
&\stackrel{!}{=} 0
\end{aligned} \tag{A.33}$$

Hereby, the well-known trace-trick was applied twice, allowing to utilize the rules for trace derivatives (Eq. (2.76) and Eq. (2.81)). The above equation can be solved in closed form:

$$W = \left[\sum_i (\mathbf{x}_i - \mu) \mathbb{E}_{p(\mathbf{h}|\cdot)}[\mathbf{h}_i]^\top \right] \left[\sum_i \mathbb{E}_{p(\mathbf{h}|\cdot)}[\mathbf{h}_i \mathbf{h}_i^\top] \right]^{-1} \tag{A.34}$$

with $\mathbb{E}_{p(\mathbf{h}|\cdot)}[\mathbf{h}_i \mathbf{h}_i^\top] = \Sigma + \mathbf{m}_i \mathbf{m}_i^\top$. The functional relation to standard linear regression (Eq. (2.249)) is evident in the above solution. A similarly involved derivation gives the M-step for Ψ:

$$\Psi = \frac{1}{n} \operatorname{diag}(G(W)) \tag{A.35}$$

where $G(W)$ is defined as

$$\begin{aligned}
G(W) = \sum_i \Big(&(\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^\top + W \mathbb{E}_{p(\mathbf{h}|\cdot)}[\mathbf{h}_i \mathbf{h}_i^\top] W^\top \\
&- 2W \mathbb{E}_{p(\mathbf{h}|\cdot)}[\mathbf{h}_i] (\mathbf{x}_i - \mu)^\top \Big)
\end{aligned} \tag{A.36}$$

PCA

Similar to the Mixture-of-Gaussian case it is possible to derive a well-known non-probabilistic algorithm from Factor Analysis by considering a limit-behaviour of the model. First, consider the special case of probabilistic PCA (pPCA), i.e. $\boldsymbol{\Psi} = \sigma^2\mathbf{I}$. In the limit $\sigma^2 \to 0$ the posterior mean for the probabilistic PCA model becomes an orthogonal projection onto the same principal subspace as in PCA. The posterior mean for pPCA is (see Eq. (A.27)):

$$\mathbf{m} = \boldsymbol{\Sigma}(\boldsymbol{W}^T\sigma^{-2}\mathbf{I}(\mathbf{x} - \boldsymbol{\mu})) \tag{A.37}$$

with

$$\boldsymbol{\Sigma} = (\mathbf{I} + \boldsymbol{W}^T\sigma^{-2}\mathbf{I}\boldsymbol{W})^{-1} = \sigma^2(\sigma^2\mathbf{I} + \boldsymbol{W}^T\boldsymbol{W})^{-1}. \tag{A.38}$$

Hence,

$$\mathbf{m} = (\sigma^2\mathbf{I} + \boldsymbol{W}^T\boldsymbol{W})^{-1}(\boldsymbol{W}^T(\mathbf{x}_i - \boldsymbol{\mu})). \tag{A.39}$$

In the case of pPCA the maximum-likelihood solution can be given in closed form (Eq. (2.256)):

$$\boldsymbol{W}_{\text{MLE}} = \mathbf{L}_l(\boldsymbol{\Lambda}_l - \sigma^2\mathbf{I})^{\frac{1}{2}}\mathbf{R} \tag{A.40}$$

with $\mathbf{L}_l$ being the $d \times l$ matrix whose columns are the first l eigenvectors of the empirical covariance matrix $\boldsymbol{S}$ (Eq. (2.227)) and $\boldsymbol{\Lambda}_l$ the diagonal matrix of corresponding eigenvalues ($\mathbf{R}$ is an arbitrary orthogonal matrix).

If $\sigma^2 \to 0$ the maximum-likelihood solution $\boldsymbol{W}_{\text{MLE}}$ converges to $\boldsymbol{V}_l\boldsymbol{\Lambda}_l^{1/2}$. So $(\sigma^2\mathbf{I} + \boldsymbol{W}^T\boldsymbol{W})^{-1} \to \boldsymbol{\Lambda}_l^{-1}$, and thus

$$\mathbf{m} = \boldsymbol{\Lambda}_l^{-1/2}\boldsymbol{V}_l^T(\mathbf{x} - \boldsymbol{\mu}) \tag{A.41}$$

which is the same subspace that PCA also projects to (Eq. (2.260)).

B

PREDICTABILITY MINIMIZATION FOR DEEP LEARNING

Deep multi-layer Neural Networks have many levels of non-linearities, which allow them to potentially represent very compactly highly non-linear and highly-varying functions. This belief is not new [318, 157, 400] and was strengthened by recent theoretical studies [25, 24].

Unfortunately, training deep networks is difficult with standard gradient-based approaches. Indeed, until recently deep networks were generally found to perform no better, and often worse, than Neural Networks with one or two hidden layers, [383]. However the introduction of new learning algorithms [155, 151, 28, 298, 225, 407] has made it possible to train deep network structures such as Deep Belief Networks [155] and stacked autoencoders [28].

The common theme among these approaches is the use of an unsupervised training phase that performs a layer by layer initialization, thereby allowing a representation of the data to be built up hierarchically. However there is less agreement on what properties the intermediate stages of this representation should have. One approach is to train each level to reconstruct its own input (i.e. the representation given by the level below) as accurately as possible, which leads naturally to stacked autoencoders [28]. If the hidden layers are further required to identify the dependencies and regularities of the unknown distribution from which the inputs were drawn, one arrives at denoising autoencoders [407]. Yet another guiding principle is sparseness for the representations [298, 225]. However, there is no well-grounded theory and the above mentioned requirements feel rather arbitrary.

Looking for criteria that such intermediate representations should fulfill, we take a step back and consider the goal of unsupervised learning in general. Informally, unsupervised learning is driven by finding structure in data (or, likewise, to identify redundancy in it). Additionally, the identified structures should be useful to extract information from the data. In [13] several ways are proposed how these goals can be achieved. In particular the author argues that unsupervised learning should yield a representation, whose elements are independent (such representations are often called *factorial codes*). Predictability minimization (PM) [330] is an unsupervised learning principle that aims at finding such factorial codes. In this paper we argue that PM is well suited as an unsupervised learning mechanism for initializing the layers of deep Neural Networks. We will further discuss whether reconstruction is a necessity towards factorial codes and

Joint work with Thomas Rückstieß.

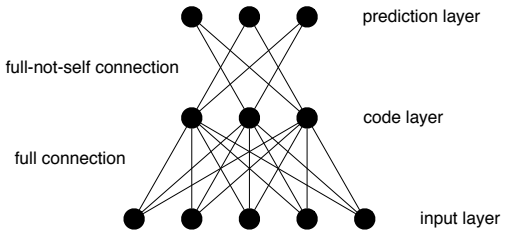


Figure B.1: Predictability Minimization architecture: The input patterns are presented to the network at the bottom layer and fed forward to the code layer which is fully connected to the inputs and has a non-linear activation. The predictor sits on top of the code layer, trying to predict each of the n code units' activations from the remaining $n-1$ code units.

suggest that independence of the code units should be considered a suitable alternative criterion.

Section 2 briefly describes the general idea behind PM and also the proposed implementation (following [330]) Thea application of PM to the layers of deep Neural Networks is discussed in Section 3. Experiments are covered in Section 4 and concluded in Section 5.

B.1 PREDICTABILITY MINIMIZATION

The principle of predictability minimization [330] is at the core of our method. In general, PM is an unsupervised method for learning non-redundant representations of input data. In particular it aims at finding binary (or at least quasi-binary) factorial codes. A factorial code must fulfill (i) the *binary criterion*, that is, each code-symbol should be either 1 or 0 and, (ii) the *invertibility criterion*, i.e. it must be possible to reconstruct the input from the code and (iii) the *independence criterion*, i.e. the occurence of each code symbol should be independent from all other code symbols.

A simple architecture is proposed to achieve the independence criterion: A one layer feedforward network (*code network*) with c output units (*code units*) has an n dimensional input. Assuming that a given input to this network carries redundant information then the network's goal is to respond with less redundant (but still *informative*) output (ideally, it would create a factorial code of the input [13]). In order to achieve this, another component is introduced: there is an adaptive *predictor* for each code unit that tries to predict the output of this unit given the remaining $n-1$ code units. However, each code unit tries to become as unpredictable as possible by representing an aspect of the input that is not represented by the other code units. Thus, the predictors and the code network are tightly interwoven and will co evolve during learning. Figure B.1 depicts this interrelation graphically.

Interestingly [330] states that conditions (i) and (ii) can be ignored and PM should be implemented instead by focusing on the independence criterion alone. From a mathematical point of view, the independence criterion can be realised in a simple way: Given the i-th input vector x_i, the code network produces a c dimensional output $y_i = (y_i^1, y_i^2, \ldots, y_i^c)$ with $y_i^j \in [0,1], j = 1, \ldots, c$. The predictor for code unit k is denoted by p^k, its output in response to $\{y_i^j : \ j \neq k\}$ is denoted by p_i^k. p^k is trained to *minimize*

$$\sum_i (p_i^k - y_i^k)^2 \tag{B.1}$$

that is, p^k learns to approximate the conditional expectation $E(y^k \mid \{y^j : j \neq k\})$ of y^k. As already indicated in the paragraph before, the code network and the predictors resemble opposing forces and this is even more highlighted by the following striking symmetry: the code network is trained to *maximize* exactly this same objective function [330]. It is beyond the scope of this paper to discuss in detail, why it is actually sufficient just to consider eq. (B.1). More information on this aspect is provided in [332].

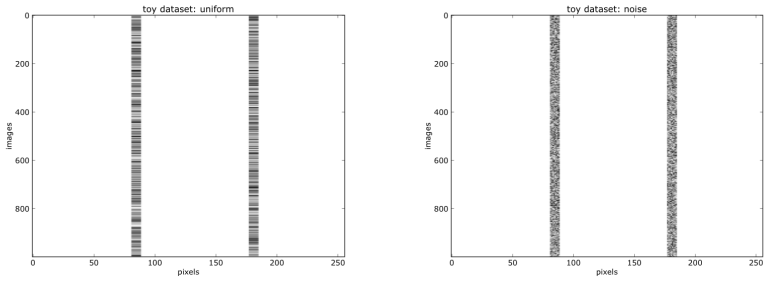
B.2 PM FOR DEEP NEURAL NETWORKS

As it was pointed out previously [90], a single layer of binary features is not the best way to capture the structure of (possibly high dimensional) input data. [335] already implemented a hierarchy of PM modules, where each layer computes the input to the next layer. However this hierarchy was not considered as an initialization for a deep Neural network, which then is globally fine tuned using another training criterion for the task at hand.

B.2.1 *Greedy Layerwise Pretraining And Fine Tuning*

Similar to Deep Belief Networks and stacked autoencoders, we use several layers of PM modules to build a deep network. The output of the code units in the kth module will be used as the input for the $(k+1)$th module, and the kth module will be trained before the one at layer $k+1$. After all layers are pre trained, a *fine tuning* phase follows which aims at globally adjusting all parameters of the deep network according to some (supervised) criterion.

Preliminary experiments on small toy data sets prompted us to reconsider the reconstruction criterion already mentioned in Section B.1. We found that adding the reconstruction criterion to the objective function (B.1) for the code network helped to improve performance of the final deep network significantly. In this case, one gets a two layer feed forward network whose first layer (representing the original code network) is trained by PM and a traditional autoencoder



(a) uniform toydata (b) noisy toydata

Figure B.2: Toy datasets. Each horizontal line represents one of 1000 input patterns. The images are mostly white, only two patches of 8 pixels each have a random value between 0.0 and 1.0. In (a) the value is kept constant for each patch, in (b) each pixel value in both patches is random. Please note that the inputs are plotted as horizontal lines in this graph but represent square images of dimension 16x16 that match the feature plots in Fig. B.3.

together, while the second layer is trained by the reconstruction criterion alone. Note that from the point of view of a traditional autoencoder the combiniation with PM now offers the possibility to train a code layer that is bigger in size than the input dimension (an important aspect when trying to build good deep networks [90]). Because of its striking advantage we did not use the traditional autoencoder to implement the reconstruction criterion but the denoising version [407] instead.

B.3 EXPERIMENTS

Two different sets of experiments were conducted: Experiments on artificially created toy data and on a challenging version of the well-known MNIST hand-written digits benchmark. The toy datasets were used to shed light onto the internal mechanisms of PM. We discuss the results of our toy experiments in Section B.3.1 and the MNIST benchmark in section B.3.2.

All experiments use the following PM architecture: A non-linear predictor with sigmoid activation sits on top of a one layer sigmoid code network. Both layers are seperately trained in batch mode with gradient ascent and descent, respectively. For performance comparisons we look at both traditional and denoising autoencoders (AE and dAE). Furthermore, we conducted some experiments with a hybrid version: a predictability minimizing denoising autoencoder (PM-dAE). Here we alternated PM and dAE training repeatedly on the same weights.

B.3.1 *PM Analysis with Artificial Data*

Having the predictor and code layer fight each other by minimizing (predictor) vs. maximizing (codes) the prediction error forces the code units to become less redundant. Ideally, the code units become completely independent but this independence cannot be *created* within the deterministic network. Instead, the units have to draw their independence from the input patterns, concentrating on different areas of the image.

With artificially created toy data (shown in Fig. B.2) the mechanisms of PM can be further investigated. We use two data sets, each set consists of 1000 images of 256 pixels each, with most pixels having a value of 1.0 (white). Only two patches, located at the same positions in each image and 8 pixels wide, are independently assigned a random value between 0.0 and 1.0. While in the uniform dataset (Fig. B.2a) the pixels in each patch have the same color, in the noisy dataset (Fig. B.2b) the pixels vary inside a patch as well.

B.3.1.1 *Uniform Toy Data*

Since there are only two independent sources in the input data, we chose to have only two units in the code layer. In order to become independent, they each have to concentrate on a different patch. As a matter of fact, it would be sufficient for the units to have their receptive field focus on just a single pixel rather than the whole 8-pixel patch. In our experiments, however, this was never the case. Instead, the resulting features, depicted in Fig. B.3, always cover a patch as a whole.

Figure B.3 shows that each PM code unit's receptive field converges to one single patch for maximal unpredictability. In contrast, the AE units draw their inputs from both patches (same results are obtained for dAE). Their objective function ensures that enough information is stored to reconstruct the input patterns, but does not force the units to become independent. Looking at the prediction errors (Fig. B.4), however, we discover that both code layers minimize their predictability. While this is expected for PM, it seems that in order to reconstruct the image, the AE units necessarily have to become more independent from each other as well. This result also corroborates the recent findings in [213].

It is not clear, if the ability to reconstruct input patterns inevitably leads to better intermediate representations and, later on, better performance for a deep network whose layers are trained in this way.

B.3.1.2 *Noisy Toy Data*

The noisy dataset is more complex than the uniform version from section B.3.1.1 because its information content is much higher. While

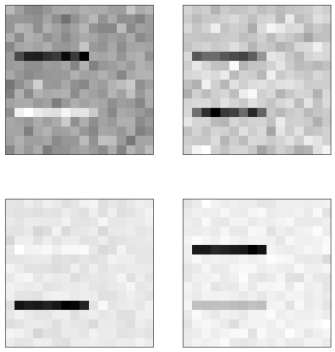


Figure B.3: AE (top) and PM (bottom) features on the toy dataset. Each of the two PM units concentrates on one patch for maximal unpredictability. The AE features contain enough information to almost completely reconstruct the original images, but the units are not as independent.

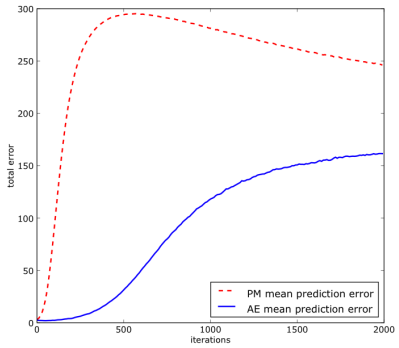


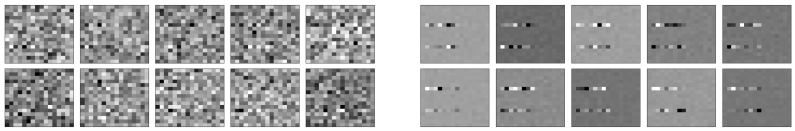
Figure B.4: Prediction error of both AE and PM network, averaged over 20 runs. While the objective function for AE does not force the units to become unpredictable, their prediction error still increases (although much slower than for the PM network), indicating that independent features help the AE in reconstructing the input patterns.

structure is clearly present in the data (and can easily be identified by inspection in Fig. B.2b) it is difficult for (d)AE architectures to reconstruct the inputs as it can only perform well by memorizing the patterns.

We increased the code layer size to 10 units and let a PM network compete against a hybrid version: a predictability minimizing autoencoder (PMAE). For PMAE, two different learning rates for the PM module were chosen. After 2000 epochs, both PMAE networks were able to isolate the noisy patches and tune their features to the location of the patches, while the AE features still looked random. Figure B.5b shows the 10 features for AE and one of the PMAEs, the other features were ommited in this plot but were very similar.

At 2000 epochs, AE had improved only marginally with an error of 310.23, while PMAE's reconstruction errors were 202.71 and 175.50, respectively. Because we could see slight improvement in AE's error, we let the experiment continue for another 10000 epochs and found that all three methods evenutally converged to almost the same error, but it took pure AE unusually long to do so.

While we looked at the prediction errors in section B.3.1.1, here we used PM to guide AE training and were interested in its convergence speed. The results for the noisy toy dataset imply that AE training can be accelerated significantly if combined with PM training. Differ-



(a) AE features after 2000 epochs (b) PMAE features after 2000 epochs

Figure B.5: Plots of the weights for each of the 10 code units after training for 2000 epochs.

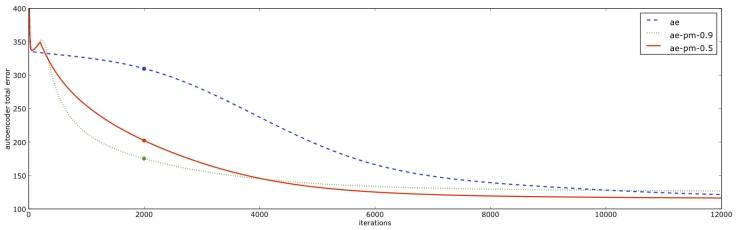


Figure B.6: Reconstruction error for pure AE (blue, dashed line) and two PMAE networks with different learning rates. Combining PM and AE leads to significantly faster convergence and similar results.

ent PM learning rates led to different convergence speed, but both outperformed pure AE by far. It remains to be seen if this result is valid for larger and more complex datasets as well.

B.3.2 *Digit Classification*

We performed experiments on a subset of the original MNIST problem which is part of a bigger benchmark suite [212]. The problem is divided into a training, validation and test set (10000, 2000, 50000 respectively). The input dimensionality is $d = 28 \times 28 = 768$.

Due to our current restricted hardware equipment we were only able to experiment with Neural Networks with two hidden layers. We also had to fix the size of each layer: The first layer contains 1000 code units, the second 2000. Classification was done with an additional softmax layer. After the unsupervised pre training was finished, we trained the softmax layer seperately for 30 epochs, using standard backpropagation. This was followed by a global fine-tuning phase, again using standard backpropagation. To speed up training (both unsupervised and supervised), we subdivided the training set into minibatches each containing 1000 training cases and updated the weights after each minibatch. We tried several values for the hyperparameters (learning rates, training epochs during unsupervised pre training) and used early stopping in the fine tuning phase. We se-

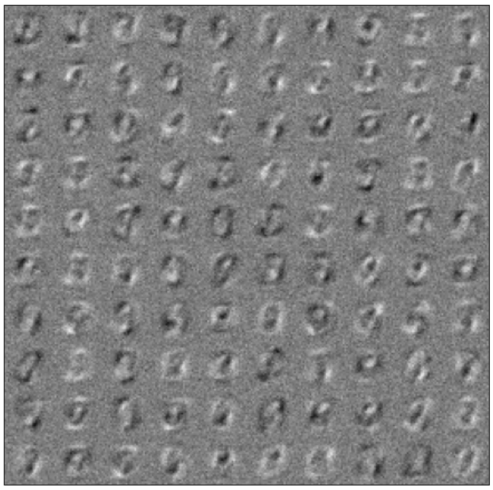


Figure B.7: PM Features. Depicted are the weight matrices of 100 code units, trained with pure PM. Some features resemble silhouettes of digits or parts of digits that occur frequently.

lected the best setting for each model according to its classification performance on the validation set.

In total, we trained 4 models:

SDAE A stacked denoising autoencoder [407] achieved a classification error of 3.764%.

SPM Both layers were initialized with PM as described in Section B.1, (i.e. the reconstruction criterion is ignored). The classification error is 3.968%.

SPMAE Each layer is pre trained with PM and a traditional autoencoder. The classification error is 3.782%.

SPMDAE Both layers are pre trained with PM combined with a denoising autoencoder. The classification error is 3.734%.

B.4 CONCLUSION AND FUTURE WORK

We draw several conclusions from our results. First, the combination of a stacked (denoising) PMAE delivered the best results, beating sdAE marginally. This supports the findings of the noisy toy data experiment (section B.3.1.2) and demonstrates that autoencoders can be improved by PM.

Secondly, using PM, it is possible to get comparable results with a traditional AE (which is not denoising) even if the code layer size is bigger than the input dimension. This suggests that the denoising

part of AE can be replaced by PM training, preventing the AE to simply learn the identity function and forcing it into a direction of *useful* intermediate representations.

And lastly, PM alone without enforcing reconstruction can compete with the above methods and yields similar classification results. This shows that the AE's reconstrucing property is not necessarily the key incredient for good classification. Other criteria, like minimized predictability, are valid candidates for creating good codes and should be considered in future analysis of deep learning architectures. A selection of features created by the PM network are shown in Figure B.7.

Immediate future work is obvious: We want to perform all experiments with our proposed method (and its variants) on the benchmark of classification problems used in [212]. In particular we are interested how the performance evolves when using more code units per layer and more layers.

PM was originally introduced for finding good (quasi-) binary codes in an unsupervised manner. Thus it is a straight forward idea to combine PM with semantic hasing [322].

Finally, another interesting direction will be to add neighborhood relationships, or general lateral connections, between the code units similar to the idea presented in [213].

BIBLIOGRAPHY

[1] O. Abdel-Hamid, A. Mohamed, H. Jiang, and G. Penn. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4277–4280, 2012. (Cited on page 133.)

[2] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. A learning algorithm for boltzmann machines*. *Cognitive science*, 9(1):147–169, 1985. (Cited on page 48.)

[3] F. Agostinelli, M. Hoffman, P. J. Sadowski, and P. Baldi. Learning activation functions to improve deep neural networks. In *International Conference on Learning Representations (ICLR), Workshop Track*, 2015. (Cited on page 70.)

[4] M. A. Aizerman, E. A. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. In *Automation and Remote Control*, pages 821–837, 1964. (Cited on page 207.)

[5] G. An. The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation*, 8(3):643–674, 1996. (Cited on page 94.)

[6] A. Anandkumar, R. Ge, D. Hsu, S. M. Kakade, and M. Telgarsky. Tensor decompositions for learning latent variable models. *The Journal of Machine Learning Research*, 15(1):2773–2832, 2014. (Cited on page 13.)

[7] A. Auffinger and G. B. Arous. Complexity of random smooth functions on the high-dimensional sphere. *The Annals of Probability*, 41(6):4214–4247, 2013. (Cited on page 89.)

[8] A. Auffinger, G. B. Arous, and J. Černỳ. Random matrices and complexity of spin glasses. *Communications on Pure and Applied Mathematics*, 66(2):165–201, 2013. (Cited on page 89.)

[9] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, 2014. (Cited on pages 99 and 100.)

[10] D. H. Ballard. Modular learning in neural networks. In *Proc. AAAI*, pages 279–284, 1987. (Cited on pages 93 and 94.)

[11] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012. (Cited on page 41.)

[12] D. Barber and C. M. Bishop. Ensemble learning in bayesian neural networks. In *Generalization in Neural Networks and Machine Learning*, 1998. (Cited on page 111.)

[13] H. B. Barlow. Unsupervised learning. *Neural Computation*, 1(3): 295–311, 1989. (Cited on pages 227 and 228.)

[14] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*, pages 404–417. Springer, 2006. (Cited on pages 131, 187, and 190.)

[15] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *CoRR*, abs/1502.05767, 2015. (Cited on page 72.)

[16] J. Bayer and C. Osendorfer. Learning stochastic recurrent networks. *CoRR*, abs/1411.7610, 2014. (Cited on pages 111 and 115.)

[17] J. Bayer, C. Osendorfer, and P. van der Smagt. Learning sequence neighbourhood metrics. In *International Conference on Artificial Neural Networks (ICANN)*, 2012. (Cited on pages 141, 176, and 213.)

[18] J. Bayer, C. Osendorfer, C. Chen, S. Urban, and P. van der Smagt. On fast dropout and its applicability to recurrent networks. *CoRR*, abs/1311.0701, 2013. (Cited on page 100.)

[19] J. Bayer, C. Osendorfer, S. Urban, and P. van der Smagt. Training neural networks with implicit variance. In *Neural Information Processing ICONIP*, pages 132–139, 2013. (Cited on page 100.)

[20] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009. (Cited on page 61.)

[21] S Becker and Y. LeCun. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school. San Matteo, CA: Morgan Kaufmann*, 1988. (Cited on page 155.)

[22] S. Becker and Y. LeCun. Improving the convergence of back-propagation learning with second-order methods. In D. Touretzky, G. E. Hinton, and T. Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School*, pages 29–37. Morgan Kaufman, 1989. (Cited on pages 90 and 155.)

[23] A. J. Bell and T. J. Sejnowski. An information-maximization approach to blind separation and blind deconvolution. *Neural Computation*, 7(6):1129–1159, 1995. (Cited on page 95.)

[24] Y. Bengio. *Learning deep architectures for AI*. Now Publishers Inc, 2009. (Cited on page 227.)

[25] Y. Bengio and Y. Le Cun. Scaling learning algorithms towards ai. In L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, editors, *Large-Scale Kernel Machines*. MIT Press, 2007. (Cited on pages 91, 93, and 227.)

[26] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks*, 5(2):157–166, 1994. (Cited on page 75.)

[27] Y. Bengio, O. Delalleau, and N. L. Roux. The curse of highly variable functions for local kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, 2005. (Cited on pages 67 and 91.)

[28] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2006. (Cited on pages 93, 94, and 227.)

[29] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. Advances in optimizing recurrent networks. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8624–8628. IEEE, 2013. (Cited on page 155.)

[30] Y. Bengio, L. Yao, G. Alain, and P. Vincent. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems (NIPS)*, 2013. (Cited on page 101.)

[31] J. Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 192–236, 1974. (Cited on page 44.)

[32] M. J. Betancourt, S. Byrne, S. Livingstone, and M. Girolami. The geometric foundations of hamiltonian monte carlo. *arXiv preprint arXiv:1410.5110*, 2014. (Cited on page 40.)

[33] E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. In *International conference on Knowledge Discovery and Data Mining (KDD)*, pages 245–250. ACM, 2001. (Cited on page 128.)

[34] C. M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995. (Cited on pages 66, 68, 86, 91, and 96.)

[35] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. (Cited on pages 4, 31, 36, 40, 41, 43, 56, 57, 59, 77, 97, 109, 133, 146, 219, 220, and 223.)

[36] A. L. Blum and R. L. Rivest. Training a 3-node neural network is NP-complete. *Neural Networks*, 5(1):117–127, 1992. (Cited on page 71.)

[37] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010. (Cited on pages 88 and 154.)

[38] L. Bottou. Stochastic gradient tricks. In G. Montavon, G. B. Orr, and K. R. Müller, editors, *Neural Networks, Tricks of the Trade, Reloaded*, Lecture Notes in Computer Science (LNCS 7700), pages 430–445. Springer, 2012. (Cited on page 88.)

[39] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 1988. (Cited on pages 94 and 100.)

[40] J. S. Bridle, Anthony J. R. Heading, and J. C. D. MacKay. Unsupervised classifiers, mutual information and 'phantom targets'. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1096–1101, 1991. (Cited on page 115.)

[41] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah. Signature verification using siamese time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):669–688, 1993. (Cited on page 176.)

[42] M. Brown and D. G. Lowe. Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision (IJCV)*, 74(1):59–73, 2007. (Cited on page 132.)

[43] M. Brown, G. Hua, and S. Winder. Discriminative learning of local image descriptors. *Pattern Analysis and Machine Intelligence (PAMI)*, 2010. (Cited on pages 122, 123, 127, 135, 137, 139, 176, 180, 181, 182, 185, 186, 194, and 195.)

[44] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR)*, 2013. (Cited on page 140.)

[45] A. E. Bryson. A gradient method for optimizing multi-stage allocation processes. In *Proc. Harvard Univ. Symposium on digital computers and their applications*, 1961. (Cited on page 72.)

[46] A. E. Bryson and Y. C. Ho. *Applied optimal control: optimization, estimation, and control*. Blaisdell Pub. Co., 1969. (Cited on page 72.)

[47] C. Bucilua, R. Caruana, and A. Niculescu-Mizil. Model compression. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 2006. (Cited on page 212.)

[48] P. J. Burt and E. H. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4): 532–540, 1983. (Cited on page 204.)

[49] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua. Brief: Computing a local binary descriptor very fast. *Pattern Analysis and Machine Intelligence (PAMI)*, 34(7):1281–1298, 2012. (Cited on page 187.)

[50] M. A. Carreira-Perpinán and W. Wang. Distributed optimization of deeply nested systems. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2014. (Cited on pages 100, 134, and 169.)

[51] A. Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *Compte Rendu à l'Académie des Sciences, Paris*, 25:536–538, 1847. (Cited on page 14.)

[52] P. Chang and J. Krumm. Object recognition with color cooccurrence histograms. In *Computer Vision and Pattern Recognition (CVPR)*, volume 2, 1999. (Cited on page 131.)

[53] K. Chatfield, V. S. Lempitsky, A. Vedaldi, and A. Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *British Machine Vision Conference (BMVC)*, 2011. (Cited on pages 170 and 171.)

[54] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference (BMVC)*, 2014. (Cited on pages 3, 169, 170, and 171.)

[55] X. Chen and A. L. Yuille. Articulated pose estimation by a graphical model with image dependent pairwise relations. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1736–1744, 2014. (Cited on page 100.)

[56] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Empiricial Methods in Natural Language Processing (EMNLP)*, 2014. (Cited on page 107.)

[57] A. Choromanska, M. Henaff, M. Mathieu, G. Arous, and Y. LeCun. The loss surface of multilayer networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015. (Cited on pages 71, 86, 89, and 177.)

[58] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio. A recurrent latent variable model for sequential data. In *NIPS*, 2015. (Cited on pages 111 and 115.)

[59] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber. A committee of neural networks for traffic sign classification. In *International Joint Conference on Neural Networks (IJNN)*, pages 1918–1921, 2011. (Cited on pages 3 and 99.)

[60] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012. (Cited on pages 99, 121, 133, 158, and 208.)

[61] D. Cireşan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. (Cited on pages 3 and 121.)

[62] D. Cireşan, A. Giusti, L. M. Gambardella, and J. Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In *Medical Imaging Computing and Computer Assisted Interventions (MICCAI)*, volume 2, pages 411–418, 2013. (Cited on pages 3, 99, 121, and 133.)

[63] D. C. Cireşan, U. Meier, L.M. Gambardella, and J. Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12):3207–3220, 2010. (Cited on pages 3, 98, and 158.)

[64] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1237–1242, 2011. (Cited on pages 133, 152, 158, and 208.)

[65] A. Coates and A. Ng. The importance of encoding versus training with sparse coding and vector quantization. In *International Conference on Machine Learning (ICML)*, 2011. (Cited on pages 183 and 189.)

[66] A. Coates, H. Lee, and A. Ng. An analysis of single layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011. (Cited on pages 169, 174, and 192.)

[67] A. Coates, A. Karpathy, and A. Ng. Emergence of object-selective features in unsupervised feature learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2012. (Cited on pages 180 and 192.)

[68] P. Comon. Independent component analysis—a new concept? *Signal Processing*, 36(3):287–314, 1994. (Cited on page 95.)

[69] T. H. Cormen, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001. (Cited on page 72.)

[70] T.M. Cover and J.A. Thomas. *Elements of information theory*. Wiley-interscience, 2006. (Cited on page 186.)

[71] B.J. Culpepper, J. Sohl-Dickstein, and B.A. Olshausen. Building a better probabilistic model of images by factorization. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. (Cited on pages 111 and 112.)

[72] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. (Cited on page 71.)

[73] G. E. Dahl, M. A. Ranzato, A. Mohamed, and G. E. Hinton. Phone recognition with the mean-covariance restricted Boltzmann machine. In *Advances in Neural Information Processing Systems (NIPS)*, pages 469–477, 2010. (Cited on pages 133 and 192.)

[74] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893. IEEE, 2005. (Cited on page 131.)

[75] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 57 (11):1413–1457, 2004. (Cited on pages 61 and 101.)

[76] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2014. (Cited on pages 71, 89, 103, and 154.)

[77] G. M. Davis, S. G. Mallat, and Z. Zhang. Adaptive time-frequency decompositions. *Optical Engineering*, 33(7):2183–2191, 1994. (Cited on page 61.)

[78] Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *Acoustics, Speech and Signal Processing*, 28(4):357–366, 1980. (Cited on page 133.)

[79] P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel. The helmholtz machine. *Neural Computation*, 7(5):889–904, 1995. (Cited on pages 109 and 110.)

[80] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. (Cited on pages 98, 121, 135, and 174.)

[81] M. Denil, B. Shakibi, L. Dinh, and N. de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2148–2156, 2013. (Cited on page 100.)

[82] E. L Denton, C. Chintala, A. Szlam, and R. Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015. (Cited on pages 209 and 213.)

[83] D. L. Donoho and X. Huo. Uncertainty principles and ideal atomic decomposition. *Information Theory*, 47(7):2845–2862, 2001. (Cited on page 61.)

[84] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2014. (Cited on pages 3 and 98.)

[85] S. E. Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45, 1962. (Cited on page 72.)

[86] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011. (Cited on page 89.)

[87] R. Durbin and D. E. Rumelhart. Product units: A computationally powerful and biologically plausible extension to backpropagation networks. *Neural Computation*, 1(1):133–142, March 1989. (Cited on page 69.)

[88] S. Edelman, N. Intrator, and T. Poggio. Complex cells and object recognition. *Unpublished manuscript: http://kybele. psych. cornell. edu/ edelman/archive. html*, 1997. (Cited on page 131.)

[89] J. L. Elman and D. Zipser. Learning the hidden structure of speech. *The Journal of the Acoustical Society of America*, 83(4): 1615–1626, 1988. (Cited on page 94.)

[90] D. Erhan, P. A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent. The difficulty of training deep architectures and the effect of unsupervised pre-training. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009. (Cited on pages 229 and 230.)

[91] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes challenge 2007 (voc2007) results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html, 2007. (Cited on page 169.)

[92] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes (VOC) challenge. *International Journal of Computer Vision (IJCV)*, 80(2):303–338, 2010. (Cited on page 169.)

[93] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems (NIPS)*, 1990. (Cited on page 93.)

[94] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *Pattern Analysis and Machine Intelligence (PAMI)*, 35(8):1915–1929, 2013. (Cited on pages 99, 121, 133, and 204.)

[95] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 2007. (Cited on pages 174 and 179.)

[96] M. A. T. Figueiredo, R. D. Nowak, and S. J. Wright. Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems. *Selected Topics in Signal Processing, IEEE Journal of*, 1(4):586–597, 2007. (Cited on page 61.)

[97] P. Fischer, A. Dosovitskiy, and T. Brox. Descriptor matching with convolutional neural networks: a comparison to sift. *arXiv preprint arXiv:1405.5769*, 2014. (Cited on page 176.)

[98] Peter Földiak. Forming sparse representations by local anti-hebbian learning. *Biological cybernetics*, 64(2):165–170, 1990. (Cited on pages 103 and 104.)

[99] W. T. Freeman and E. H. Adelson. The design and use of steerable filters. *Pattern Analysis and Machine Intelligence (PAMI)*, 13 (9):891–906, 1991. (Cited on page 131.)

[100] Y. Freund and D. Haussler. Unsupervised learning of distributions on binary vectors using two layer networks. Technical report, University of California, Santa Cruz, 1994. (Cited on page 182.)

[101] M. C. Fu. Gradient estimation. *Handbooks in operations research and management science*, 13:575–616, 2006. (Cited on pages 41 and 42.)

[102] K. Fukunaga. *Introduction to statistical pattern recognition*. Academic press, 1990. (Cited on page 57.)

[103] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980. (Cited on page 83.)

[104] Y. Gal and Z. Ghahramani. Bayesian convolutional neural networks with bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*, 2015. (Cited on pages 97, 208, and 213.)

[105] Z. Gan, R. Henao, D. Carlson, and L. Carin. Learning deep sigmoid belief networks with data augmentation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015. (Cited on page 64.)

[106] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR)*, 2012. (Cited on page 124.)

[107] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence (PAMI)*, 1984. (Cited on page 38.)

[108] A. P. George and W. B. Powell. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine Learning*, 65(1):167–198, 2006. (Cited on page 89.)

[109] F. A Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10): 2451–2471, 2000. (Cited on pages 87, 91, and 97.)

[110] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber. Learning precise timing with LSTM recurrent networks. *The Journal of Machine Learning Research*, 3:115–143, 2003. (Cited on page 91.)

[111] Z. Ghahramani and G. E. Hinton. The EM algorithm for mixtures of factor analyzers. *University of Toronto Technical Report CRG-TR-96-1*, 1997. (Cited on page 60.)

[112] R. Girshick, F. Iandola, T. Darrell, and J. Malik. Deformable part models are convolutional neural networks. In *CVPR*, 2015. (Cited on page 83.)

[113] Ross Girshick. Fast R-CNN. In *International Conference on Computer Vision (ICCV)*, 2015. (Cited on page 121.)

[114] P. Glasserman. *Monte Carlo methods in financial engineering*, volume 53. Springer Science & Business Media, 2003. (Cited on page 42.)

[115] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256, 2010. (Cited on page 92.)

[116] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011. (Cited on pages 70 and 150.)

[117] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems (NIPS)*, 2005. (Cited on page 141.)

[118] G. H. Golub and C. F. Van Loan. *Matrix computations*, volume 3. The Johns Hopkins University Press, 2012. (Cited on pages 12 and 140.)

[119] R. Gomes, A. Krause, and P. Perona. Discriminative clustering by regularized information maximization. In *Advances in Neural Information Processing Systems (NIPS)*, 2010. (Cited on page 115.)

[120] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *International Conference on Machine Learning (ICML)*, 2013. (Cited on pages 70 and 150.)

[121] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, 2014. (Cited on page 214.)

[122] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014. (Cited on page 212.)

[123] I. J. Goodfellow, O. Vinyals, and A. M. Saxe. Qualitatively characterizing neural network optimization problems. *CoRR*, abs/1412.6544, 2014. (Cited on pages 71, 89, 99, and 177.)

[124] A. Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2011. (Cited on page 111.)

[125] A. Graves. Offline arabic handwriting recognition with multidimensional recurrent neural networks. In *Guide to OCR for Arabic Scripts*, pages 297–313. Springer, 2012. (Cited on page 133.)

[126] A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. *Advances in Neural Information Processing Systems (NIPS)*, 21, 2009. (Cited on page 133.)

[127] A. Graves, S. Fernandez, and J. Schmidhuber. Multi-dimensional recurrent neural networks. In *International Conference on Artificial Neural Networks (ICANN)*, 2007. (Cited on page 213.)

[128] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *Pattern Analysis and Machine Intelligence (PAMI)*, 31(5):855–868, 2009. (Cited on page 99.)

[129] A. Graves, A. Mohamed, and G. E. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP)*, pages 6645–6649, 2013. (Cited on page 99.)

[130] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In *International Conference on Machine Learning (ICML)*, 2010. (Cited on pages 61, 102, and 107.)

[131] K. Gregor, A. Szlam, and Y. LeCun. Structured sparse coding via lateral inhibition. In *Advances in Neural Information Processing Systems (NIPS)*, 2011. (Cited on pages 106 and 107.)

[132] K. Gregor, I. Danihelka, A. Graves, and D. Wierstra. DRAW: A recurrent neural network for image generation. In *International Conference on Machine Learning (ICML)*, 2015. (Cited on pages 114, 208, and 213.)

[133] A. Griewank. A mathematical view of automatic differentiation. *Acta Numerica*, 12:321–398, 2003. (Cited on page 72.)

[134] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008. (Cited on page 72.)

[135] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical report, California Institute of Technology, 2007. (Cited on page 174.)

[136] G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, USA, 2001. (Cited on page 19.)

[137] C. Gulcehre, K. Cho, R. Pascanu, and Y. Bengio. Learned-norm pooling for deep feedforward and recurrent neural networks. In *Machine Learning and Knowledge Discovery in Databases*, pages 530–546. Springer, 2014. (Cited on page 84.)

[138] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006. (Cited on pages 143, 144, and 176.)

[139] E. Hairer, C. Lubich, and G. Wanner. Geometric numerical integration illustrated by the Störmer–Verlet method. *Acta numerica*, 12:399–450, 2003. (Cited on page 40.)

[140] X. Han, T. Leung, Y. Jia, R. Sukthankar, and A. C. Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. (Cited on page 176.)

[141] T. D. Harrison. *A Connectionist framework for continuous speech recognition*. PhD thesis, Cambridge University, 1987. (Cited on page 94.)

[142] J. Håstad. *Computational Limitations of Small-depth Circuits*. MIT Press, Cambridge, MA, USA, 1987. (Cited on page 91.)

[143] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. (Cited on page 39.)

[144] M. Havlena and K. Schindler. Vocmatch: Efficient multiview correspondence for structure from motion. In *European Conference on Computer Vision (ECCV)*, pages 46–60, 2014. (Cited on page 132.)

[145] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision (ECCV)*, 2014. (Cited on pages 169 and 214.)

[146] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*, 2015. (Cited on pages 70, 87, and 93.)

[147] H. Heuser. *Lehrbuch der Analysis, Teil 1*. Vieweg+Teubner Verlag, 2003. (Cited on pages 13, 61, 103, and 106.)

[148] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. (Cited on page 212.)

[149] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002. (Cited on pages 50, 183, and 188.)

[150] G. E. Hinton. A practical guide to training restricted boltzmann machines. Technical report, University of Toronto, 2010. (Cited on pages 183 and 188.)

[151] G. E. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. (Cited on pages 182 and 227.)

[152] G. E. Hinton and T. J. Sejnowski. Learning and relearning in boltzmann machines. In D. E. Rumelhart, J. L. McClelland, and PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, pages 282–317. MIT Press, 1986. (Cited on pages 47 and 57.)

[153] G. E. Hinton and D. Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Conference on Computational Learning Theory*, 1993. (Cited on pages 109 and 111.)

[154] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268 (5214):1158–1161, 1995. (Cited on page 65.)

[155] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006. (Cited on pages 64, 93, and 227.)

[156] G. E. Hinton, S. Osindero, M. Welling, and Y. W. Teh. Unsupervised discovery of nonlinear structure using contrastive backpropagation. *Cognitive science*, 30(4):725–731, 2006. (Cited on pages 99 and 100.)

[157] G.E. Hinton. Connectionist learning procedures. *Artificial intelligence*, 40(1-3):185–234, 1989. (Cited on page 227.)

[158] T. Hobbes. *Leviathan or The Matter, Forme and Power of a Common Wealth Ecclesiasticall and Civil*. 1651. (Cited on page 104.)

[159] S. Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. diploma thesis, 1991. Advisor:J. Schmidhuber. (Cited on page 75.)

[160] S. Hochreiter and J. Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997. (Cited on page 99.)

[161] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. (Cited on pages 76 and 91.)

[162] S. Hochreiter and J. Schmidhuber. LOCOCODE performs nonlinear ICA without knowing the number of sources. In J.-F. Cardoso, C. Jutten, and P. Loubaton, editors, *First International Workshop on Independent Component Analysis and Signal Separation, Aussois, France*, pages 149–154, 1999. (Cited on page 95.)

[163] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001. (Cited on page 75.)

[164] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013. (Cited on page 109.)

[165] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982. (Cited on page 103.)

[166] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge university press, 2012. (Cited on page 9.)

[167] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. (Cited on page 71.)

[168] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6): 417, 1933. (Cited on page 57.)

[169] M. J. Huiskes, B. Thomee, and M. S. Lew. New trends and ideas in visual concept detection: The mir flickr retrieval evaluation initiative. In *International Conference on Multimedia Information Retrieval*, pages 527–536. ACM, 2010. (Cited on page 170.)

[170] A. Hyvärinen and E. Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13(4):411–430, 2000. (Cited on page 95.)

[171] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015. (Cited on pages 95, 96, 97, and 212.)

[172] A. G. Ivakhnenko. Polynomial theory of complex systems. *IEEE Transactions on Systems, Man and Cybernetics*, (4):364–378, 1971. (Cited on page 72.)

[173] A. G. Ivakhnenko and V. G. Lapa. *Cybernetic Predicting Devices*. CCM Information Corporation, 1965. (Cited on page 72.)

[174] T. Iwata, D. Duvenaud, and Z. Ghahramani. Warped mixtures for nonparametric cluster shapes. In *Uncertainty in Artificial Intelligence*, 2013. (Cited on page 119.)

[175] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. (Cited on page 205.)

[176] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference (BMVC)*, 2014. (Cited on page 100.)

[177] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015. (Cited on page 213.)

[178] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology, 2001. (Cited on page 93.)

[179] H. Jaeger. Adaptive nonlinear system identification with echo state networks. In *Advances in Neural Information Processing Systems*, 2002. (Cited on page 93.)

[180] M. Jahrer, M. Grabner, and H. Bischof. Learned local descriptors for recognition and matching. In *Computer Vision Winter Workshop*, 2008. (Cited on page 176.)

[181] A. K. Jain, N. K. Ratha, and S. Lakshmanan. Object detection using gabor filters. *Pattern Recognition*, 30(2):295–309, 1997. (Cited on page 131.)

[182] K. Jarrett, K. Kavukcuoglu, M.A. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *International Conference on Computer Vision (ICCV)*, 2009. (Cited on pages 140 and 152.)

[183] A Jazwinski. Filtering for nonlinear dynamical systems. *IEEE Transactions on Automatic Control*, 11(4):765–766, 1966. (Cited on page 28.)

[184] H. Jégou, T. Furon, and J. Fuchs. Anti-sparse coding for approximate nearest neighbor search. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2029–2032, 2012. (Cited on page 159.)

[185] J. L. W. V. Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30(1):175–193, 1906. (Cited on page 20.)

[186] Y. Jia and T. Darrell. Heavy-tailed distances for gradient based image descriptors. In *Advances in Neural Information Processing Systems (NIPS)*, 2011. (Cited on page 185.)

[187] Y. Jia, M. Salzmann, and T. Darrell. Factorized latent spaces with structured sparsity. In *Advances in Neural Information Processing Systems (NIPS)*, 2010. (Cited on page 111.)

[188] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014. (Cited on pages 86 and 140.)

[189] I. Jolliffe. *Principal component analysis*. Wiley Online Library, 2002. (Cited on page 57.)

[190] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999. (Cited on pages 60 and 109.)

[191] R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning (ICML)*, 2015. (Cited on page 87.)

[192] J. S. Judd. *Neural network design and the complexity of learning*. MIT Press, 1990. (Cited on page 71.)

[193] C. Jutten and J. Herault. Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture. *Signal Processing*, 24(1):1–10, 1991. (Cited on page 95.)

[194] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. *Association for Computational Linguistics (ACL)*, 2014. (Cited on page 133.)

[195] N. Kalchbrenner, I. Danihelka, and A. Graves. Grid long short-term memory. *arXiv preprint arXiv:1507.01526*, 2015. (Cited on pages 96 and 106.)

[196] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering*, 82(1):35–45, 1960. (Cited on page 28.)

[197] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. *arXiv preprint arXiv:1412.2306*, 2014. (Cited on page 99.)

[198] H. J. Kelley. Gradient theory of optimal flight paths. *ARS Journal*, 30(10):947–954, 1960. (Cited on page 72.)

[199] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2014. (Cited on page 207.)

[200] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations (ICLR)*, 2013. (Cited on pages 42, 65, 109, 110, and 208.)

[201] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3581–3589, 2014. (Cited on pages 111, 114, 116, and 118.)

[202] J. F. C. Kingman and S. J. Taylor. *Introduction to Measure and Probability*. Cambridge University Press, 1966. (Cited on page 26.)

[203] D. E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., 1997. (Cited on page 90.)

[204] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009. (Cited on pages 43 and 52.)

[205] K. Königsberger. *Analysis 1 (Springer Lehrbuch)*. Springer, 2004. (Cited on page 13.)

[206] A. Korattikara, V. Rathod, K. Murphy, and M. Welling. Bayesian dark knowledge. In *Advances in Neural Information Processing Systems (NIPS)*, 2015. (Cited on pages 100 and 212.)

[207] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. (Cited on pages 104, 150, 152, 174, and 179.)

[208] A. Krizhevsky and G.E. Hinton. Using very deep autoencoders for content-based image retrieval. In *European Symposium on Artificial Neural Network (ESANN)*, 2011. (Cited on pages 84, 180, 192, and 193.)

[209] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012. (Cited on pages 3, 70, 98, 99, 121, 127, 140, 158, and 182.)

[210] B. Kulis. Metric learning: A survey. *Foundations and Trends in Machine Learning*, 5(4):287–364, 2012. (Cited on pages 143 and 176.)

[211] H. Larochelle and I. Murray. The neural autoregressive distribution estimator. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011. (Cited on page 112.)

[212] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *International Conference on Machine Learning (ICML)*, 2007. (Cited on pages 233 and 235.)

[213] H. Larochelle, D. Erhan, and P. Vincent. Deep learning using robust interdependent codes. In *Artificial Intelligence and Statistics*, 2009. (Cited on pages 231 and 235.)

[214] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2169–2178, 2006. (Cited on pages 132 and 171.)

[215] Q. V. Le, A. Karpenko, J. Ngiam, and A. Y. Ng. ICA with reconstruction cost for efficient overcomplete feature learning. In *Advances in Neural Information Processing Systems*, 2011. (Cited on page 95.)

[216] Q. V. Le, R. Monga, M. Devin, G. Corrado, K. Chen, M. A. Ranzato, J. Dean, and A. Y. Ng. Building high-level features using large scale unsupervised learning. In *International Conference on Machine Learning (ICML)*, 2012. (Cited on page 180.)

[217] Q. V. Le, N. Jaitly, and G. E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *CoRR*, abs/1504.00941, 2015. (Cited on page 93.)

[218] Y. LeCun. Une procédure d'apprentissage pour réseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks). In *Cognitiva 85*, pages 599–604, 1985. (Cited on page 72.)

[219] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. (Cited on page 83.)

[220] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. *Advances in Neural Information Processing Systems (NIPS)*, 1990. (Cited on page 90.)

[221] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. (Cited on pages 83 and 152.)

[222] Y. LeCun, C. Cortes, and C. J. C. Burges. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist, 1998. Accessed: 2015-10-01. (Cited on page 105.)

[223] Y. LeCun, F. J. Huang, and L. Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004. (Cited on page 179.)

[224] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller. Efficient backprop. In *Neural Networks: Tricks of the trade*, pages 9–48. Springer, 2012. (Cited on pages 70, 87, and 95.)

[225] H. Lee, C. Ekanadham, and A. Ng. Sparse deep belief net model for visual area v2. In *Advances in Neural Information Processing Systems (NIPS)*, 2008. (Cited on page 227.)

[226] R. Lengellé and T. Denoeux. Training MLPs layer by layer using an objective function for internal representations. *Neural Networks*, 9(1):83–97, 1996. (Cited on page 93.)

[227] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *Pattern Analysis and Machine Intelligence (PAMI)*, 28(9): 1465–1479, 2006. (Cited on pages 137 and 181.)

[228] S. Leutenegger, M. Chli, and R.Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *International Conference on Computer Vision (ICCV)*, 2011. (Cited on page 187.)

[229] J. Lin. Divergence measures based on the shannon entropy. *Information Theory*, 37(1):145–151, 1991. (Cited on page 186.)

[230] L. Lin, O. Morère, V. Chandrasekhar, A. Veillard, and H. Goh. DeepHash: Getting regularization, depth and fine-tuning right. *CoRR*, abs/1501.04711, 2015. (Cited on pages 159 and 178.)

[231] M. Lin, Q. Chen, and S. Yan. Network in network. In *International Conference on Learning Representations (ICLR)*, 2014. (Cited on page 106.)

[232] T. Lindeberg. Scale-space theory: A basic tool for analyzing structures at different scales. *Journal of applied statistics*, 1994. (Cited on pages 122, 123, 204, and 214.)

[233] S. Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's thesis, University Helsinki, 1970. (Cited on page 72.)

[234] A. Livnat, C. Papadimitriou, N. Pippenger, and M. W. Feldman. Sex, mixability, and modularity. *Proceedings of the National Academy of Sciences*, 107(4):1452–1457, 2010. (Cited on page 97.)

[235] N. K. Logothetis, J. Pauls, and T. Poggio. Shape representation in the inferior temporal cortex of monkeys. *Current Biology*, 5 (5):552–563, 1995. (Cited on page 131.)

[236] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. *arXiv preprint arXiv:1411.4038*, 2014. (Cited on page 86.)

[237] J. L. Long, N. Zhang, and T. Darrell. Do convnets learn correspondence? In *Advances in Neural Information Processing Systems (NIPS)*, 2014. (Cited on page 176.)

[238] D. G. Lowe. Object recognition from local scale-invariant features. *IJCV*, 2:1150–1157, 1999. (Cited on pages 126 and 139.)

[239] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. (Cited on pages 131, 137, 181, 182, 186, 190, and 214.)

[240] M. LukoševičIus and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009. (Cited on page 93.)

[241] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013. (Cited on page 70.)

[242] D. J. C. MacKay. *Information theory, Inference and Learning Algorithms*. Cambridge University Press, 2003. (Cited on pages 34, 39, 40, 47, and 90.)

[243] J. R. Magnus and H. Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Wiley, 3rd edition, 2007. (Cited on pages 13, 15, and 16.)

[244] A. Makhzani and B. J. Frey. k-sparse autoencoders. *CoRR*, abs/1312.5663, 2013. (Cited on page 95.)

[245] J. Malik and P. Perona. Preattentive texture discrimination with early vision mechanisms. *Journal of the Optical Society of America A*, 7(5):923–932, 1990. (Cited on pages 70 and 150.)

[246] J Martens. Deep learning via hessian-free optimization. In *International Conference on Machine Learning (ICML)*, 2010. (Cited on page 89.)

[247] J Martens. Learning the linear dynamical system with asos. In *International Conference on Machine Learning (ICML)*, 2010. (Cited on page 153.)

[248] J. Masci, U. Meier, G. Fricout, and J. Schmidhuber. Multi-scale pyramidal pooling network for generic steel defect classification. In *International Joint Conference on Neural Networks (IJCNN)*, 2013. (Cited on page 3.)

[249] J. Masci, M. M Bronstein, A. M. Bronstein, and J. Schmidhuber. Multimodal similarity-preserving hashing. *Pattern Analysis and Machine Intelligence (PAMI)*, 36(4):824–830, 2014. (Cited on pages 176 and 178.)

[250] J. Masci, D. Boscaini, M. M Bronstein, and P. Vandergheynst. Shapenet: Convolutional neural networks on non-euclidean manifolds. *CoRR*, abs/1501.06297, 2015. (Cited on page 140.)

[251] M. Mathieu, M. Henaff, and Y. LeCun. Fast training of convolutional networks through ffts. *CoRR*, abs/1312.5851, 2013. (Cited on page 86.)

[252] C. E. McCulloch. Maximum likelihood variance components estimation for binary data. *Journal of the American Statistical Association*, 89(425):330–335, 1994. (Cited on page 60.)

[253] C. E. McCulloch. Maximum likelihood algorithms for generalized linear mixed models. *Journal of the American statistical Association*, 92(437):162–170, 1997. (Cited on page 60.)

[254] G. J. McLachlan and T. Krishnan. *The EM algorithm and extensions*. Wiley New York, 1997. (Cited on pages 58, 217, 219, and 221.)

[255] David A Medler. A brief history of connectionism. *Neural Computing Survey*, 1998. (Cited on page 4.)

[256] R. Memisevic. Gradient-based learning of higher-order image features. In *International Conference on Computer Vision*, 2011. (Cited on pages 111 and 113.)

[257] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953. (Cited on page 39.)

[258] A. S. Mian, M. Bennamoun, and R. Owens. Three-dimensional model-based object recognition and segmentation in cluttered scenes. *Pattern Analysis and Machine Intelligence (PAMI)*, 28(10): 1584–1601, 2006. (Cited on page 131.)

[259] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence (PAMI)*, 2005. (Cited on pages 137, 179, 181, 190, and 194.)

[260] T. Mikolov. *Statistical Language Models Based on Neural Networks*. PhD thesis, Brno University of Technology, 2012. (Cited on page 98.)

[261] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations (ICLR)*, 2013. (Cited on pages 5 and 214.)

[262] A. Mnih and K. Gregor. Neural variational inference and learning in belief networks. In *International Conference on Machine Learning (ICML)*, 2014. (Cited on pages 109 and 208.)

[263] V. Mnih, N. Heess, and A. Graves. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems (NIPS)*, 2014. (Cited on page 205.)

[264] H. Mobahi, R. Collobert, and J. Weston. Deep learning from temporal coherence in video. In *International Conference on Machine Learning (ICML)*, 2009. (Cited on page 126.)

[265] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2924–2932, 2014. (Cited on pages 70 and 150.)

[266] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. (Cited on pages 55 and 56.)

[267] A. F. Murray and P. J. Edwards. Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training. *Neural Networks*, 5(5):792–802, 1994. (Cited on page 94.)

[268] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2010. (Cited on pages 70, 150, and 192.)

[269] V. Nair and G.E. Hinton. 3-d object recognition with deep belief nets. In *Advances in Neural Information Processing Systems (NIPS)*, 2009. (Cited on page 183.)

[270] B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM journal on computing*, 24(2):227–234, 1995. (Cited on page 61.)

[271] R. M. Neal. Probabilistic inference using markov chain monte carlo methods. Technical report, University of Toronto, 1993. (Cited on pages 37 and 185.)

[272] R. M. Neal. MCMC using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2010. (Cited on page 40.)

[273] R. M. Neal and G. E. Hinton. A new view of the EM algorithm that justifies incremental and other variants. *Learning in Graphical Models*, 1998. (Cited on pages 59, 60, and 217.)

[274] R.M. Neal. Connectionist learning of belief networks. *Artificial Intelligence*, 56(1):71–113, 1992. (Cited on pages 62 and 63.)

[275] Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983. (Cited on page 154.)

[276] R. Neuneier and H. G. Zimmermann. How to train neural networks. In *Neural Networks: Tricks of the Trade*, pages 373–423. Springer, 2012. (Cited on page 93.)

[277] M. Nickel, V. Tresp, and H. P. Kriegel. A three-way model for collective learning on multi-relational data. In *International Conference on Machine Learning (ICML)*, pages 809–816, 2011. (Cited on page 13.)

[278] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer verlag, 2006. (Cited on pages 14 and 71.)

[279] S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992. (Cited on page 77.)

[280] M. Oberweger, P. Wohlhart, and V. Lepetit. Hands deep in deep learning for hand pose estimation. In *Computer Vision Winter Workshop (CVWW)*, 2015. (Cited on page 100.)

[281] K. S. Oh and K. Jung. GPU implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004. (Cited on page 98.)

[282] B.A. Olshausen and D.J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996. (Cited on page 61.)

[283] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Is object localization for free?–weakly-supervised learning with convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. (Cited on page 121.)

[284] C. Osendorfer, J. Bayer, S. Urban, and P. van der Smagt. Convolutional neural networks learn compact local image descriptors. In *Neural Information Processing (ICONIP)*, pages 624–630, 2013. (Cited on page 176.)

[285] C. Osendorfer, H. Hubert Soyer, and P. van der Smagt. Image super-resolution with fast approximate convolutional sparse coding. In *Neural Information Processing (ICONIP)*, 2014. (Cited on pages 86, 205, and 208.)

[286] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):127–239, 2014. (Cited on page 61.)

[287] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013. (Cited on page 98.)

[288] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988. (Cited on page 62.)

[289] K. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. (Cited on page 57.)

[290] B. Pepik, R. Benenson, T. Ritschel, and B. Schiele. What is holding back convnets for detection? In *German Conference on Pattern Recognition (GCPR)*, pages 517–528. Springer, 2015. (Cited on page 122.)

[291] P. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. In *International Conference on Machine Learning (ICML)*, pages 82–90, 2014. (Cited on page 121.)

[292] B. Poole, J. Sohl-Dickstein, and S. Ganguli. Analyzing noise in autoencoders and deep networks. *CoRR*, abs/1406.1831, 2014. (Cited on pages 94 and 95.)

[293] L. Prechelt. Early stopping—but when? In *Neural Networks: Tricks of the Trade*, pages 53–67. Springer, 2012. (Cited on page 97.)

[294] T. Raiko, H. Valpola, and Y. LeCun. Deep learning made easier by linear transformations in perceptrons. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 924–932, 2012. (Cited on page 96.)

[295] R. Raina, A. Battle, H. Lee, B. Packer, and A.Y. Ng. Self-taught learning: Transfer learning from unlabeled data. In *International Conference on Machine Learning (ICML)*, 2007. (Cited on pages 182 and 185.)

[296] M. A. Ranzato and G. E. Hinton. Modeling pixel means and covariances using factorized third-order boltzmann machines. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. (Cited on pages 111, 180, 185, 189, and 190.)

[297] M. A. Ranzato and M. Szummer. Semi-supervised learning of compact document representations with deep networks. In *International conference on Machine learning (ICML)*, 2008. (Cited on pages 93, 94, and 116.)

[298] M. A. Ranzato, C. S. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems (NIPS)*, 2006. (Cited on page 227.)

[299] M. A. Ranzato, L. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2007. (Cited on page 83.)

[300] M. A. Ranzato, A. Krizhevsky, and G. E. Hinton. Factored 3-way restricted boltzmann machines for modeling natural images. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010. (Cited on pages 184, 192, and 201.)

[301] M. A. Ranzato, V. Mnih, and G.E. Hinton. Generating more realistic images using gated mrf's. In *Advances in Neural Information Processing Systems (NIPS)*, 2010. (Cited on page 191.)

[302] M. A. Ranzato, J. Susskind, V. Mnih, and G. E. Hinton. On deep generative models with applications to recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. (Cited on page 113.)

[303] A. Rasmus, H. Valpola, M. Honkala, M. Berglund, and T. Raiko. Semi-supervised learning with ladder network. In *Advances in Neural Information Processing Systems (NIPS)*, 2015. (Cited on page 99.)

[304] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Converence on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 512–519, 2014. (Cited on pages 99 and 121.)

[305] B. Recht. Projected gradient methods, 2012. (Cited on page 58.)

[306] D. J. Rezende and S. Mohamed. Variational Inference with Normalizing Flows. In *International Conference on Machine Learning (ICML)*, 2015. (Cited on pages 109, 111, and 112.)

[307] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning (ICML)*, pages 1278–1286, 2014. (Cited on pages 42, 65, 109, 110, and 208.)

[308] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *International Conference on Neural Networks*, pages 586–591. IEEE, 1993. (Cited on page 155.)

[309] M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature neuroscience*, 2(11):1019–1025, 1999. (Cited on page 152.)

[310] S. Rifai, G. Mesnil, P. Vincent, X. Muller, Y. Bengio, Y. N. Dauphin, and X. Glorot. Higher order contractive auto-encoder. In *European Conference on Machine Learning (ECML)*, 2011. (Cited on page 95.)

[311] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contracting auto-encoders. In *International Conference on Machine Learning (ICML)*, 2011. (Cited on page 95.)

[312] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951. (Cited on pages 86, 87, and 154.)

[313] A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department, 1987. (Cited on page 68.)

[314] J. T. Rolfe and Y. LeCun. Discriminative recurrent sparse auto-encoders. In *International Conference on Learning Representations (ICLR)*, 2013. (Cited on page 107.)

[315] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *CoRR*, abs/1412.6550, 2014. (Cited on page 93.)

[316] P. M. Roth and M. Winter. Survey of appearance-based methods for object recognition. Technical report, Institute for Computer Graphics and Vision, Graz University of Technology, Austria, 2008. (Cited on page 131.)

[317] S. Roweis. EM algorithms for PCA and SPCA. In *Advances in Neural Information Processing Systems (NIPS) 27*, 1998. (Cited on page 60.)

[318] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 2: Psychological and Biological Models*. MIT Press Cambridge, MA, USA, 1986. (Cited on pages 4, 94, and 227.)

[319] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. (Cited on pages 72 and 87.)

[320] H. Sak, A. Senior, and F. Beaufays. Long Short-Term Memory recurrent neural network architectures for large scale acoustic modeling. In *Proc. INTERSPEECH*, 2014. (Cited on pages 92 and 99.)

[321] R. Salakhutdinov and G. E. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 412–419, 2007. (Cited on pages 101 and 116.)

[322] R. Salakhutdinov and G. E. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 2008. (Cited on pages 180 and 235.)

[323] R. Salakhutdinov and G. E. Hinton. An efficient learning procedure for deep boltzmann machines. *Neural Computation*, 24(8): 1967–2006, 2012. (Cited on page 51.)

[324] R. Salakhutdinov and H. Larochelle. Efficient learning of deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 693–700, 2010. (Cited on page 65.)

[325] M. Salzmann, C. H. Ek, R. Urtasun, and T. Darrell. Factorized orthogonal latent spaces. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010. (Cited on page 111.)

[326] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations (ICLR)*, 2014. (Cited on pages 70, 71, 89, 93, and 94.)

[327] T. Schaul, S. Zhang, and Y. LeCun. No more pesky learning rates. In *International Conference on Machine Learning (ICML)*, 2013. (Cited on pages 87 and 89.)

[328] D. Scherer, A. Müller, and S. Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International Conference on Artificial Neural Networks (ICANN)*, pages 92–101. Springer, 2010. (Cited on pages 152 and 153.)

[329] C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. *Pattern Analysis and Machine Intelligence (PAMI)*, 19(5): 530–534, 1997. (Cited on page 131.)

[330] J. Schmidhuber. Learning factorial codes by predictability minimization. *Neural Computation*, 4(6):863–879, 1992. (Cited on pages 103, 227, 228, and 229.)

[331] J. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2): 234–242, 1992. (Cited on page 93.)

[332] J. Schmidhuber. *Netzwerkarchitekturen, Zielfunktionen und Kettenregel*. Habilitation, Technische Universitat München, 1993. (Cited on page 229.)

[333] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. (Cited on pages 3, 4, 66, 72, and 73.)

[334] J. Schmidhuber and D. Prelinger. Discovering predictable classifications. *Neural Computation*, 5(4):625–635, 1993. (Cited on page 176.)

[335] J. Schmidhuber, M. Eldracher, and B. Foltin. Semilinear predictability minimization produces well-known feature detectors. *Neural Computation*, 8(4):773–786, 1996. (Cited on pages 93 and 229.)

[336] N. N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 2002. (Cited on pages 66, 89, and 153.)

[337] N. N. Schraudolph, M. Eldracher, and J. Schmidhuber. Processing images by semi-linear predictability minimization. *Network: Computation in Neural Systems*, 1999. (Cited on page 103.)

[338] F. Schroff, D. Kalenichenko, and J. Philbin. FaceNet: A unified embedding for face recognition and clustering. *arXiv preprint arXiv:1503.03832*, 2015. (Cited on page 176.)

[339] J. Schulman, N. Heess, T. Weber, and P. Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems (NIPS)*, 2015. (Cited on pages 41 and 42.)

[340] H. Schulz and S. Behnke. Learning object-class segmentation with convolutional neural networks. In *European Symposium on Artificial Neural Networks (ESANN)*, 2012. (Cited on pages 121 and 133.)

[341] P. Sermanet, S. Chintala, and Y. LeCun. Convolutional neural networks applied to house numbers digit classification. In *International Conference on Pattern Recognition (ICPR)*, pages 3288–3291, 2012. (Cited on page 83.)

[342] N. E. Sharkey. Image compression by back propagation: a demonstration of extensional programming. In *Advances in Cognitive Science*, volume 2. Ablex Publishing, 1987. (Cited on page 94.)

[343] H. Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000. (Cited on page 95.)

[344] J. Shotton, M. Johnson, and R. Cipolla. Semantic texton forests for image categorization and segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008. (Cited on page 135.)

[345] H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150, 1995. (Cited on page 71.)

[346] J. Sietsma and R. J. F. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67–79, 1991. (Cited on page 94.)

[347] J. Šíma. Training a single sigmoidal neuron is hard. *Neural Computation*, 14(11):2709–2728, 2002. (Cited on page 71.)

[348] E. Simo-Serra, E. Trulls, L. Ferraz, I. Kokkinos, P. Fua, and F. Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *International Conference on Computer Vision (ICCV)*, 2015. (Cited on pages 176 and 212.)

[349] K. Simonyan. *Large-Scale Learning of Discriminative Image Representations*. PhD thesis, University of Oxford, 2013. (Cited on page 176.)

[350] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. (Cited on pages 3, 93, 99, and 121.)

[351] K. Simonyan, A. Vedaldi, and A. Zisserman. Descriptor learning using convex optimisation. In *European Conference on Computer Vision (ECCV)*, 2012. (Cited on pages 127, 135, 139, 159, 176, and 194.)

[352] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman. Discovering objects and their location in images. In *International Conference on Computer Vision (ICCV)*, volume 1, pages 370–377, 2005. (Cited on page 132.)

[353] J. Sivic, B.C. Russell, A. Zisserman, W.T. Freeman, and A.A. Efros. Unsupervised discovery of visual object class hierarchies. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008. (Cited on page 132.)

[354] A. Smith, A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2013. (Cited on page 35.)

[355] P. Smolensky. Information processing in dynamical systems: foundations of harmony theory. *MIT Press Computational Models Of Cognition And Perception Series*, pages 194–281, 1986. (Cited on page 49.)

[356] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, 2012. (Cited on page 146.)

[357] R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng. Parsing with compositional vector grammars. In *Proceedings of the ACL conference*, 2013. (Cited on page 93.)

[358] J. Sohl-Dickstein, M. Mudigonda, and M. Deweese. Hamiltonian monte carlo without detailed balance. In *International Conference on Machine Learning (ICML)*, pages 719–726, 2014. (Cited on page 40.)

[359] C. Spearman. "General Intelligence", objectively determined and measured. *The American Journal of Psychology*, 15(2):201–292, 1904. (Cited on page 59.)

[360] P. Sprechmann, A. Bronstein, and G. Sapiro. Learning efficient structured sparse models. In *International Conference on Machine Learning (ICML)*, 2012. (Cited on pages 102 and 104.)

[361] P. Sprechmann, R. Litman, T. B. Yakar, A. M. Bronstein, and G. Sapiro. Supervised sparse analysis and synthesis operators. In *Advances in Neural Information Processing Systems (NIPS)*, pages 908–916, 2013. (Cited on page 107.)

[362] P. Sprechmann, A. Bronstein, and G. Sapiro. Learning efficient sparse and low rank models. *Pattern Analysis and Machine Intelligence (PAMI)*, 2015. (Cited on page 102.)

[363] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. Striving for simplicity: The all convolutional net. In *International Conference on Learning Representations (ICLR)*, 2015. (Cited on pages 84, 140, 151, 152, and 153.)

[364] N. Srivastava and R. Salakhutdinov. Multimodal learning with deep boltzmann. In *Advances in Neural Information Processing Systems (NIPS)*, 2013. (Cited on pages 70 and 97.)

[365] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. (Cited on pages 97, 197, and 208.)

[366] R. K. Srivastava, J. Masci, S. Kazerounian, F. Gomez, and J. Schmidhuber. Compete to compute. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2310–2318, 2013. (Cited on page 150.)

[367] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015. (Cited on pages 67, 96, 97, and 106.)

[368] M. Stollenga, J. Masci, F. J. Gomez, and J. Schmidhuber. Deep networks with internal selective attention through feedback connections. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3545–3553, 2014. (Cited on pages 100 and 205.)

[369] M. Stollenga, W. Wonmin Byeon, M. Liwicki, and J. Schmidhuber. Parallel multi-dimensional LSTM, with application to fast biomedical volumetric image segmentation. In *Advances in Neural Information Processing Systems (NIPS)*, 2015. (Cited on pages 100, 133, and 213.)

[370] R. L. Stratonovich. Conditional markov processes and their application to the theory of optimal control. 1968. (Cited on page 28.)

[371] A. Stuhlmüller, J. Taylor, and N. D. Goodman. Learning stochastic inverses. In *Advances in Neural Information Processing Systems (NIPS)*, 2013. (Cited on page 65.)

[372] J. Susskind, R. Memisevic, G. E. Hinton, and M. Pollefeys. Modeling the joint density of two images under a variety of transformations. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. (Cited on pages 198 and 200.)

[373] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112, 2014. (Cited on page 99.)

[374] I. Sutskever, R. Jozefowicz, K. Gregor, D. Rezende, T. Lillicrap, and O. Vinyals. Towards principled unsupervised learning. *arXiv preprint arXiv:1511.06440*, 2015. (Cited on page 211.)

[375] Ilya Sutskever. *Training recurrent neural networks*. PhD thesis, University of Toronto, 2013. (Cited on pages 89, 93, 154, and 193.)

[376] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 28. MIT press, 1998. (Cited on page 28.)

[377] K. Swersky, D. Tarlow, I. Sutskever, R. Salakhutdinov, R. Zemel, and R. Adams. Cardinality restricted boltzmann machines. In *Advances in Neural Information Processing Systems (NIPS)*, 2012. (Cited on page 189.)

[378] K. Swersky, J. Snoek, and R. P. Adams. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014. (Cited on page 146.)

[379] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2553–2561, 2013. (Cited on page 121.)

[380] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. (Cited on pages 3, 99, and 121.)

[381] Y. Tang and A. Mohamed. Multiresolution deep belief networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012. (Cited on pages 180, 182, and 188.)

[382] G. W. Taylor, I. Spiro, C. Bregler, and R. Fergus. Learning invariance through imitation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. (Cited on page 176.)

[383] G. Tesauro. Practical issues in temporal difference learning. *Machine learning*, 8(3):257–277, 1992. (Cited on pages 91 and 227.)

[384] L. Theis and M. Bethge. Generative image modeling using spatial LSTMs. In *Advances in Neural Information Processing Systems (NIPS)*, 2015. (Cited on pages 133 and 213.)

[385] L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015. (Cited on pages 4, 124, 196, and 211.)

[386] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996. (Cited on page 61.)

[387] T. Tieleman. Training restricted boltzmann machines using approximations to the likelihood gradient. In *International Conference on Machine Learning (ICML)*, 2008. (Cited on pages 51, 183, and 201.)

[388] T. Tieleman and G.E. Hinton. Lecture 6.5 - rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012. (Cited on pages 89, 90, 155, and 188.)

[389] M. E. Tipping and C. M. Bishop. Mixtures of probabilistic principal component analyzers. *Neural Computation*, 11(2):443–482, 1999. (Cited on page 56.)

[390] M. Titsias and M. Lázaro-Gredilla. Doubly stochastic variational bayes for non-conjugate inference. In *International Conference on Machine Learning (ICML)*, 2014. (Cited on page 110.)

[391] J. Tobin. Estimation of relationships for limited dependent variables. *Econometrica: journal of the Econometric Society*, pages 24–36, 1958. (Cited on page 71.)

[392] E. Tola, V. Lepetit, and P. Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *Pattern Analysis and Machine Intelligence (PAMI)*, 32(5):815–830, 2010. (Cited on page 132.)

[393] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1653–1660, 2014. (Cited on pages 100 and 121.)

[394] J. A. Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10): 2231–2242, 2004. (Cited on page 61.)

[395] T. Trzcinski and V. Lepetit. Efficient discriminative projections for compact binary descriptors. In *European Conference on Computer Vision (ECCV)*, 2012. (Cited on pages 176, 191, and 192.)

[396] T. Trzcinski, M. Christoudias, V. Lepetit, and P. Fua. Learning image descriptors with the boosting-trick. In *Advances in Neural Information Processing Systems (NIPS)*, 2012. (Cited on pages 127, 135, 139, 146, and 176.)

[397] T. Trzcinski, M. Christoudias, P. Fua, and V. Lepetit. Boosting binary image descriptors. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013. (Cited on page 160.)

[398] S. Urban and P. van der Smagt. A neural transfer function for a smooth and differentiable transition between additive and multiplicative interactions. *CoRR*, abs/1503.05724, 2015. (Cited on page 69.)

[399] B. Uria, I. Murray, and H. Larochelle. A deep and tractable density estimator. In *International Conference on Machine Learning (ICML)*, 2014. (Cited on page 113.)

[400] P. E. Utgoff and D. J. Stracuzzi. Many-layered learning. *Neural Computation*, 2002. (Cited on page 227.)

[401] P. van der Smagt and G. Hirzinger. Solving the ill-conditioning in neural network learning. In G. Montavon, G. B. Orr, and K. R. Müller, editors, *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*, pages 191–203. Springer Berlin Heidelberg, 2012. (Cited on page 96.)

[402] V. Vapnik. *Statistical Learning Theory*. Wiley, 1998. (Cited on page 206.)

[403] V. Vapnik. *The nature of statistical learning theory*. Springer Verlag, 2000. (Cited on page 33.)

[404] A. Vedaldi and B. Fulkerson. Vlfeat: An open and portable library of computer vision algorithms. In *International Conference on Multimedia*, 2010. (Cited on page 187.)

[405] M. Vetterli, J. Kovačević, and V. K. Goyal. *Foundations of signal processing*. Cambridge University Press, 2014. (Cited on pages 79, 81, and 86.)

[406] L. Vilnis and A. McCallum. Word representations via Gaussian Embedding. In *International Conference on Learning Representations (ICLR)*, 2015. (Cited on pages 5 and 207.)

[407] P. Vincent, H. Larochelle, Y. Bengio, and P.A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning (ICML)*, 2008. (Cited on pages 94, 227, 230, and 234.)

[408] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, 2010. (Cited on pages 101 and 192.)

[409] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. *arXiv preprint arXiv:1411.4555*, 2014. (Cited on page 99.)

[410] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008. (Cited on page 43.)

[411] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning (ICML)*, pages 1058–1066, 2013. (Cited on pages 97 and 98.)

[412] Q. Wang and J. JaJa. From maxout to channel-out: Encoding information on sparse pathways. In *International Conference on Artificial Neural Networks (ICANN)*, pages 273–280. Springer, 2014. (Cited on page 70.)

[413] S. Wang and C. Manning. Fast dropout training. In *International Conference on Machine Learning (ICML)*, pages 118–126, 2013. (Cited on page 100.)

[414] J. Weng, N. Ahuja, and T. S. Huang. Cresceptron: a self-organizing neural network which grows adaptively. In *International Joint Conference on Neural Networks (IJCNN)*, volume 1, pages 576–581, 1992. (Cited on pages 83 and 152.)

[415] P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4): 339–356, 1988. (Cited on page 68.)

[416] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990. (Cited on page 77.)

[417] J. Weston, F. Ratle, H. Mobahi, and R. Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012. (Cited on pages 93 and 94.)

[418] S. Wiesler and H. Ney. A convergence analysis of log-linear training. In *Advances in Neural Information Processing Systems (NIPS)*, pages 657–665, 2011. (Cited on page 95.)

[419] R. J. Williams. Complexity of exact gradient computation algorithms for recurrent neural networks. Technical Report Technical Report NU-CCS-89-27, Boston: Northeastern University, College of Computer Science, 1989. (Cited on page 68.)

[420] S. Winder and M. Brown. Learning local image descriptors. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007. (Cited on pages 137, 181, and 182.)

[421] P. Wohlhart and V. Lepetit. Learning descriptors for object recognition and 3d pose estimation. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. (Cited on page 100.)

[422] C. F. J. Wu. On the convergence properties of the EM algorithm. *The Annals of Statistics*, 1983. (Cited on page 221.)

[423] K. M. Yi, Y. Verdie, P. Fua, and V. Lepetit. Learning to assign orientations to feature points. *arXiv preprint arXiv:1511.04273*, 2015. (Cited on page 176.)

[424] L. Younes. On the convergence of markovian stochastic algorithms with rapidly decreasing ergodicity rates. *Stochastics: An International Journal of Probability and Stochastic Processes*, 65(3-4): 177–228, 1999. (Cited on page 51.)

[425] A. L. Yuille. The convergence of contrastive divergences. In *Advances in Neural Information Processing Systems (NIPS)*, 2005. (Cited on page 51.)

[426] S. Zagoruyko and N. Komodakis. Learning to compare image patches via convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. (Cited on page 176.)

[427] M. D. Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012. (Cited on pages 89, 90, 117, 155, and 156.)

[428] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. In *International Conference on Learning Representations (ICLR)*, 2013. (Cited on page 152.)

[429] H. Zhou, Y. Yuan, and C. Shi. Object tracking using sift features and mean shift. *Computer Vision and Image Understanding (CVIU)*, 113(3):345–352, 2009. (Cited on page 132.)

[430] S. Zokai and G. Wolberg. Image registration using log-polar mappings for recovery of large-scale similarity and projective transformations. *Image Processing*, 14(10):1422–1434, 2005. (Cited on page 202.)