Technische Universität München
Fakultät für Informatik

Doctoral Thesis

Autonomous Learning for Machine Perception

Ioannis Nektarios Chiotellis

Supervised by

Prof. Dr. Daniel Cremers

April 2021

TECHNISCHE UNIVERSITÄT MÜNCHEN

Fakultät für Informatik

Autonomous Learning for Machine Perception

Ioannis Nektarios Chiotellis

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:	Prof. Dr. Nassir Navab
Prüfer der Dissertation:	1. Prof. Dr. Daniel Cremers
	2. Prof. Emanuele Rodolà
	3. Priv.-Doz. Dr. Rudolph Triebel

Die Dissertation wurde am 12.05.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 03.12.2021 angenommen.

To my family

Love responsibility. Say: It is my duty, and mine alone, to save the earth. If it is not saved, then I alone am to blame.

– Nikos Kazantzakis

Abstract

In the last decade, major advancements in machine learning have had a major impact in various fields of science and engineering. Two of the most notable cases are the fields of computer vision and robotics. Due to the abundance of their applications in day to day tasks, these improvements have drastically changed the human experience and will likely have even more important consequences in the future.

Among machine learning approaches, deep learning has emerged as the most notable one, achieving impressive results in computer vision tasks such as image classification, segmentation, tracking, etc., thereby surpassing previous state-of-the-art approaches such as kernel methods. However, the success of deep learning models came along with a requirement for large amounts of labeled data. In fact, the success of deep neural networks has been attributed to their ability to *scale* better than previous methods with larger data sets.

However, autonomous systems are more likely to succeed if they are able to operate in a *small data* regime, where only a small subset of the data is labeled by humans. The underlying, possibly intricate relationships within the data as well as the major proportion of labels should be automatically inferred by the learning system.

In this thesis, we develop machine learning algorithms that aim to minimize the need for human supervision, utilizing techniques from Metric Learning, Semi-Supervised Learning, Active Learning and Reinforcement Learning. We explore problems from a diverse set of research areas including non-rigid shape analysis, autonomous driving, deep active learning and combinatorial optimization.

Zusammenfassung

In den letzten zehn Jahren hatten große Fortschritte beim maschinellen Lernen große Auswirkungen auf verschiedene Bereiche der Wissenschaft und Technik. Zwei der bemerkenswertesten Fälle sind die Bereiche Computer Vision und Robotik. Aufgrund der Fülle ihrer Anwendungen bei alltäglichen Aufgaben haben diese Verbesserungen die menschliche Erfahrung drastisch verändert und werden wahrscheinlich in Zukunft noch wichtigere Konsequenzen haben.

Unter den Ansätzen des maschinellen Lernens hat sich das deep learning als das bemerkenswerteste herausgestellt. Es erzielt beeindruckende Ergebnisse bei Computer-Vision-Aufgaben wie Bildklassifizierung, Segmentierung, Verfolgung usw. und übertrifft damit frühere Ansätze des Standes der Technik wie Kernel-Methoden. Der Erfolg von Deep Learning Modellen ging jedoch mit der Anforderung großer Mengen gekennzeichneter Daten einher. Tatsächlich wurde der Erfolg tiefer neuronaler Netze auf ihre Fähigkeit zurückgeführt, besser als frühere Methoden mit größeren Datenmengen zu betonen.

Autonome Systeme sind jedoch eher erfolgreich, wenn sie in einem *small data*-Regime arbeiten können, in dem nur eine kleine Teilmenge der Daten von Menschen gekennzeichnet wird. Die zugrunde liegenden, möglicherweise komplizierten Beziehungen innerhalb der Daten sowie der Hauptanteil der Etiketten sollten vom Lernsystem automatisch abgeleitet werden.

In dieser Arbeit entwickeln wir Algorithmen für maschinelles Lernen,, die darauf abzielen, den Bedarf an menschlicher Überwachung zu minimieren. Dabei werden Techniken aus den Bereichen metrisches Lernen, halbüberwachtes Lernen, aktives Lernen und verstärkendes Lernen verwendet. Wir untersuchen Probleme aus einer Vielzahl von Forschungsbereichen, einschließlich nicht starrer Formanalyse, autonomem Fahren, tiefem aktivem Lernen und kombinatorischer Optimierung.

Acknowledgments

This dissertation would not have been possible without the support and encouragement of multiple people. Firstly, I would like to thank my doctoral advisor Prof. Daniel Cremers for giving me complete freedom to pursue the research topics I was interested in and being available for a discussion whenever I asked for guidance. Further, for creating a friendly and diverse environment in the lab, filled with great, competent people. I feel lucky for becoming part of such an outstanding research group.

Secondly, I want to thank Dr. habil. Rudolph Triebel for his guidance and supervision, especially during the early stages of my PhD, his support during difficult periods and his valuable contributions in our collaborations.

I would like to express my gratitude to Sabine Wagner for taking care of any administrative issue that arised and to Quirin Lohr for his continuous technical support. Their work allowed me to focus exclusively on research and the group could not be as successful without them.

The papers published throughout this degree would not have been possible without the contributions of all my excellent collaborators: Daniel Cremers, Jiayu Liu, Sahand Sharifzadeh, Rudolph Triebel, Thomas Windheuser and Franziska Zimmermann. I am truly grateful for their hard work and all the fruitful discussions.

I want to thank my former office mates Zorah Lähner and Marvin Eisenberger for being great colleagues and excellent researchers. Our discussions are already missed. Further, I want to thank Thomas Möllenhoff, Tao Wu and Björn Häfner for their friendship, our diverging discussions and after-work dinners. I want to thank my colleagues from our newly created Deep Learning Focus Group: Christian Tomani, Qadeer Khan, Patrick Wenzel, Vladimir Golkov and Yueshong Shen. Our informal discussions about research and everything else have been a great addition to my experience in the group. I would also like to thank many colleagues met over the years at the Computer Vision Group at TUM for the amazing time, all the memorable moments and the nice discussions about research and completely unrelated topics: Florian Bernard, Niko Demmel, Csaba Domokos, Marvin Eisenberger, Thomas Frerix, Vladimir Golkov, Björn Häfner, Philip Häusser, Caner Hazirbas, Mariano Jaimez Tarifa, Qadeer Khan, Lukas Köstler, Zorah Lähner, Emanuel Laude, Benedikt Löwenhauser, Lingni Ma, Robert Maier, Thomas Möllenhoff, Michael

Möller, Yvain Queau, Emanuele Rodola, David Schubert, Christiane Sommer, Mohamed Souiai, Jan Stühmer, Christian Tomani, Rudolph Triebel, Vladyslav Usenko, Matthias Vestner, Rui Wang, Patrick Wenzel, Thomas Windheuser, Tao Wu, Nan Yang and Zhenzhang Ye. I will surely never forget the great moments we had; may it be conference trips, group retreats, Weißwurst breakfasts, coffee breaks, afternoon walks, table soccer.

I am very thankful to all my friends for always being by my side and enriching my life with joy and fun: Vyron, Anna H., Tasos, Vangelis, Nikos, Konstantina, Spyros, Fei, Alexandros T., Stellina, Anna M., Giorgos, Giota, Alfonso, Oriana, Marvin, Marlene, Barbara and Lars. And of course my friends who live far away but are always close: Alexandros A., Vasilis, Virginia, Giorgos M., Alex N., Corinna, Frank and Matthias. I dearly appreciate having all of you in my life.

Lastly, and most importantly, I want to especially thank my family for all their love, patience and unconditional support. My mother and my brothers have always had my back, believed in me and strongly supported me in whichever goal I pursued. My nephews always brighten up my day whenever I have the chance to visit. The journey of this PhD has been both challenging and exciting, paved with invaluable experiences and many lessons learned. I am equally excited about a promising future.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We may hope that machines will eventually compete with men in all purely intellectual fields. But which are the best ones to start with? Many people think that a very abstract activity, like the playing of chess, would be best. It can also be maintained that it is best to provide the machine with the best sense organs that money can buy, and then teach it to understand and speak English.

– Alan Turing

Machine learning is a very wide area of research. A long standing dream of science has been to build machines that could one day take on tasks that are tedious or dangerous for humans. In the 20th century, after the industrial revolution and as soon as automation became a reality, scientists envisioned that machines with computing abilities could also tackle more high-level tasks, such as playing games like chess or being able to describe what is depicted in an image. Thus, machine learning and computer vision have been coupled since the early days of computing.

As the interests of the author lie in the intersection of machine learning, computer vision and robotics, this thesis focuses on problems related to these topics. Problems in robotics are typically non-stationary and require online optimization techniques. Autonomous systems need to learn how to perform tasks with minimal human supervision and have to be able to operate while having incomplete information. Therefore, this thesis is mainly focused on two directions:

a) develop learning algorithms that can utilize small amounts of labels provided by humans, and

b) develop efficient algorithms that can operate in the online setting, aiming to close their performance gap with offline learning systems.

This thesis is based on the following publications:

- I. Chiotellis, R. Triebel, T. Windheuser, and D. Cremers (2016). "Non-Rigid 3D Shape Retrieval via Large Margin Nearest Neighbor Embedding". In: *European conference on computer vision (ECCV).*
- S. Sharifzadeh, I. Chiotellis, R. Triebel, and D. Cremers (2016). "Learning to Drive using Inverse Reinforcement Learning and Deep Q-Networks". In: *Workshop on "Deep Learning for Action and Interaction", Conference on Neural Information Processing Systems (NIPS).*
- I. Chiotellis*, F. Zimmermann*[1], D. Cremers, and R. Triebel (2018). "Incremental semi-supervised learning from streams for object classification". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*
- J. Liu, I. Chiotellis, R. Triebel, and D. Cremers (2020). "Effective Version Space Reduction for Convolutional Neural Networks". In: *Machine Learning and Knowledge Discovery in Databases - European Conference, (ECML-PKDD)*

and the preprint:

- I. Chiotellis, and D. Cremers (2020). "Neural Online Graph Exploration". (arXiv preprint arXiv:2012.03345).

[1]The symbol * denotes equal contribution.

These papers use techniques from Metric Learning, Semi-Supervised Learning, Active Learning and Reinforcement Learning. Thus, in this first chapter we briefly introduce the main ideas behind these categories of machine learning.

In the rest of Chapter 1, we introduce a few fundamental concepts of machine learning that are relevant and necessary to understand the works described in the following chapters. In Chapter 2, we describe an application of metric learning to non-rigid shape retrieval. In Chapter 3, we show a system that can learn to drive given a few demonstrations by a human using inverse reinforcement learning. In Chapter 4, we present an algorithm that can learn to classify objects in a stream of sparsely labeled data using a semi-supervised learning technique on a growing graph. In Chapter 5, we investigate active learning with convolutional neural networks through a principled theoretical framework and propose a novel query criterion. Finally, in Chapter 6, we study an important combinatorial optimization problem, called online graph exploration, by casting it as a reinforcement learning problem.

1.1 Supervised Learning

The most common regime in machine learning, that has also seen the most success, is the regime of supervised learning. Suppose that we observe a real-valued vector input variable $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^d$ and we want to use it to predict the value of a real-valued target variable $y \in \mathcal{Y} \subseteq \mathbb{R}$. We assume that we have a *training set* of N observations of the input variable $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)$ together with N corresponding observations of the *target* variable $(y_1, y_2, \dots, y_N)$.

The goal is to *learn* a mapping $f : \mathcal{X} \mapsto \mathcal{Y}$ from the given training data, such that given some unseen observation $\mathbf{x}_i^* \in \mathcal{X}$, we are able to predict their associated target or *label* $y_i^* \in \mathcal{Y}$. To do this, we represent the mapping f as a *differentiable* function parameterized by a set of parameters θ. Often this function is called the *model*:

$$y = f(\mathbf{x}; \theta). \tag{1.1}$$

The model parameters θ are to be optimized or in other words *trainable*. To train the parameters θ, we first need to define a real-valued *loss function* $\ell(y_i, f(\mathbf{x}_i; \theta)) \in \mathbb{R}$ that measures the discrepancy between a target label y_i and the model prediction $f(\mathbf{x}_i; \theta)$. Thus, our aim is to find parameters θ that minimize the loss function over all training examples:

$$\min_\theta L(\theta) = \min_\theta \sum_{i=1}^{N} \ell(y_i; f(\mathbf{x}_i; \theta)) \tag{1.2}$$

Loss functions are often chosen to be smooth convex functions, such that the gradient of the loss function is zero when the loss is minimal. Thus, an important ingredient to finding optimal parameters is to be able to compute the *gradient* of the loss function with respect to θ:

$$\nabla_\theta L(\theta) = \sum_{i=1}^{N} \nabla_\theta \ell(y_i; f(\mathbf{x}_i; \theta)) \tag{1.3}$$

When the prediction variable is a linear combination of the inputs (e.g. linear regression models), we can set the gradient to zero and find a closed-form solution for θ. However, most interesting problems involve more complicated input-output relationships and we need to learn *non-linear* mappings f. This brings the necessity for iterative optimization algorithms. A common algorithm to optimize for θ in this case is *gradient descent*. The idea is to make small steps in parameter space, each time taking the direction opposite to the gradient. In each iteration, we update the parameters as:

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta), \tag{1.4}$$

where $\eta > 0$ is the step size or the *learning rate*. When the loss function is minimized, we say that the model *fits* the training data well.

1.1.1 Regression and Classification

The two most well known and studied supervised learning problems are regression and classification. Their main difference lies in the space $\mathcal{Y}$ of the target labels. In regression, the labels are continuous: $\mathcal{Y} \subseteq \mathbb{R}$. In classification the labels are categorical, e.g. $\mathcal{Y} = \{1, 2, \ldots, K\}$ for a problem with K classes.

A typical loss function for regression is the mean squared error (MSE):

$$L_{MSE}(\theta) = \frac{1}{N} \sum_{i=1}^{N} ||y_i - f(\mathbf{x}_i; \theta)||^2. \tag{1.5}$$

A typical loss function for classification is the cross-entropy loss (CE):

$$L_{CE}(\theta) = \frac{1}{N} \sum_{i=1}^{N} -\log p(y_i|\mathbf{x}_i; \theta), \tag{1.6}$$

where $p(y_i|\mathbf{x}_i; \theta)$ is the conditional probability of class y_i given sample $\mathbf{x}_i$, estimated by the model. A common way to infer probabilities from a model's predictions is to let the model output a K-dimensional vector and use the softmax function:

$$p(j|\mathbf{x}_i; \theta) = \frac{\exp(f_j(\mathbf{x}_i; \theta))}{\sum_{k=1}^{K} \exp(f_k(\mathbf{x}_i; \theta))} \qquad \forall j \in \{1, 2, \ldots K\} \tag{1.7}$$

where by $f_j(\mathbf{x}_i; \theta)$ we denote the model's prediction associated with class j, given that the model outputs K predictions, one for each class.

1.1.2 Metric Learning

Metric learning is a beautiful idea complementary to many well known machine learning algorithms that has pushed the state of the art in problems ranging from classification and clustering to self-supervised representation learning and reinforcement learning.

The idea is that instead of directly learning a mapping $f : \mathcal{X} \mapsto \mathcal{Y}$, we can learn a metric $d : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}_+$ that reflects the *desired distance* or (dis-)similarity between training examples. This notion of similarity is given by the labels for classification and regression problems, but it can also be based on any other source of information, making metric learning applicable in many domains. This makes metric learning a particularly attractive learning paradigm since we are interested in reducing human supervision.

Formally, the idea in metric learning is to to learn a mapping $h : \mathcal{X} \mapsto \mathcal{Z}$, such that the metric $d : \mathcal{Z} \times \mathcal{Z} \mapsto \mathbb{R}_+$ reflects the desired dissimilarities. Usually $\mathcal{Z} \subseteq \mathbb{R}^M$ is called the embedding space. For classification problems, we can then use a simple k-nearest neighbor classifier acting on the embedding space $\mathcal{Z}$. If the target output space is explicitly needed for inference, we can additionally learn a simple mapping $f : \mathcal{Z} \mapsto \mathcal{Y}$, either separately or jointly with h. Imposing a loss on the embedding space $\mathcal{Z}$ rather than the output space $\mathcal{Y}$ regularizes and smooths the learning process. Moreover, metric learning allows for some interpretability of the model.

Let us assume we are given binary labels $y_{ij} \in \{0, 1\}$ indicating whether training samples $\mathbf{x}_i$ and $\mathbf{x}_j$ are similar or not. A common measure of similarity between vectors is the cosine similarity:

$$\cos(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i}{||\mathbf{x}_i||} \cdot \frac{\mathbf{x}_j}{||\mathbf{x}_j||}, \tag{1.8}$$

which lies in $[-1, +1]$ and intuitively measures the angle between the vectors. One loss often used in metric learning is a pairwise loss based on the cosine similarity:

$$\ell_{cos}(\mathbf{z}_i, \mathbf{z}_j, y_{ij}) = \begin{cases} 1 - \cos(\mathbf{z}_i, \mathbf{z}_j), & \text{if } y_{ij} = 1 \\ \max(0, \cos(\mathbf{z}_i, \mathbf{z}_j)), & \text{if } y_{ij} = 0, \end{cases} \tag{1.9}$$

where we have used $\mathbf{z}_i = h(\mathbf{x}_i; \theta)$ to simplify notation. The training loss is computed with respect to all pairs of training inputs:

$$L_{cos}(\theta) = \sum_{i=1}^{N} \sum_{j>i}^{N} \ell_{cos}(\mathbf{z}_i, \mathbf{z}_j, y_{ij}). \tag{1.10}$$

1.2 Semi-Supervised Learning

A well studied scenario in machine learning is that of semi-supervised learning (SSL). In this case, we are only given labels for a small subset of the training samples.

In particular we are given a training set $X = X_L \cup X_U$ that consists of labeled samples X_L and unlabeled samples X_U. We are only given labels Y_L that correspond to the samples in X_L. A naive approach would be to completely ignore the unlabeled samples X_U and train a model with supervised learning on the labeled training set (X_L, Y_L). However, SSL algorithms aim to utilize the unlabeled data as much as possible.

There are two general approaches to this problem, the *transductive* and the *inductive* approach. In the *transductive* setting (V. Vapnik, 1998), we are only interested in specifically estimating the labels Y_U of the unlabeled set X_U and we do not care about any other potentially unseen data.

The second approach is the *inductive* setting, where as usual we learn a general model $f : \mathcal{X} \mapsto \mathcal{Y}$. We can then apply f to infer the labels Y_U but we can use f to predict the labels of unseen data as well.

1.3 Active Learning

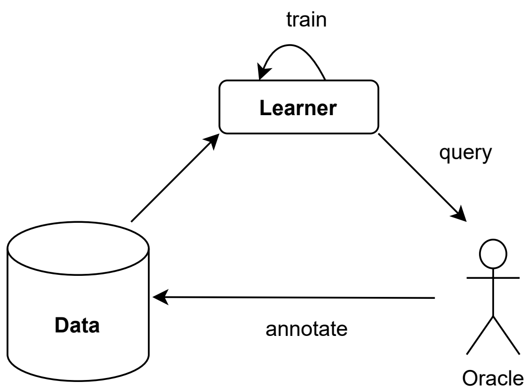


Figure 1.1: The active learning loop.

Another area of research that is particularly relevant to industrial applications is the field of Active Learning (AL). As in semi-supervised learning, the learning algorithm is initially given access to the labels of a small subset of training samples. However, in between learning rounds the algorithm can *query* an oracle (e.g. a human annotator) for a small number of labels that are then added to the training set. In the next training round, the model can be trained with the extended training set. The aim of an active learning system is to learn a good mapping f while making as few queries as possible. Therefore, it is important that the learning system has a good way of estimating its own uncertainty about a sample's label and use this as a criterion to query the oracle only for the most informative labels. Thus, the main focus of research in AL lies in finding good measures of uncertainty and deriving query criteria.

1.4 Reinforcement Learning

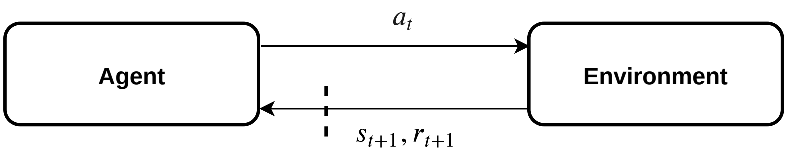


Figure 1.2: The agent-environment interaction loop.

Autonomous systems must be able to operate online, namely to quickly adapt to new situations and react appropriately. Further, *truly* autonomous systems must be able to influence their environment, namely to be able to *act*. Reinforcement learning (RL) is a very general framework that captures this setting and can be applied to a vast number of problems (e.g. board games, navigation, recommendation systems etc.). As such it also carries many hopes for the realization of *truly* intelligent systems.

In the RL framework, there is a continual interaction between an agent and their environment (Figure 1.2). The agent follows a *policy* $\pi : \mathcal{S} \mapsto \mathcal{A}$, that maps states $s_t \in \mathcal{S}$ to actions $a_t \in \mathcal{A}$. From a given state s_t, the agent performs an action a_t and the environment "responds" with a new state s_{t+1} and a reward signal r_{t+1}. The goal of the agent is to choose actions such that the cumulative reward is maximized. Thus the notion of a *value function* of a state s, given a policy π is introduced:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} | s_t = s], \tag{1.11}$$

where $\gamma \in [0, 1]$ is a discount factor that weighs distant future rewards less than imminent rewards. The discount factor is especially important in continual (infinite horizon) problems to avoid infinite sums. Further to describe the influence of a particular action, the notion of an *action-value function* is used, assuming that action a is performed and policy pi is followed thereafter:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} | s_t = s, a_t = a]. \tag{1.12}$$

RL problems are formally described as Markov Decision Processes (MDPs). An MDP is defined as a 5-tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, namely a state space $\mathcal{S}$, an action space $\mathcal{A}$, a state transition probability function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, a reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ and a discount factor $\gamma \in [0, 1]$.

RL algorithms either directly parameterize and learn a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$ or learn the action value function $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ and obtain a policy by taking the action with the maximal value:

$$a_t = \arg\max_a Q(s_t, a). \tag{1.13}$$

1.4.1 Inverse Reinforcement Learning

Inverse Reinforcement Learning (IRL) is the process of learning a policy by observing the behavior of another agent - that is an expert. In *imitation learning*, the algorithm simply learns to *copy* the expert's behavior. A more subtle approach is to estimate the reward or the value function that the expert is trying to maximize. This way, a policy different than the one of the expert may be learned, and thus the expert's performance may be even surpassed.

1.5 Deep Learning

The term “deep learning” describes a wide range of machine learning systems which primarily use deep neural networks as models. In the last decade, deep learning systems have improved the state-of-the-art in all previously described categories of learning. Moreover, recently *self-supervised* deep learning systems have reached results that are competitive even to supervised models.

1.5.1 Neural Networks

Artificial neural networks are machine learning models that originally aimed to resemble biological systems in the way they process information. Research in such models began already in the 1940s (McCulloch and Pitts, 1943). However, interest in neural networks was abandoned at least three times until today, each time because a different seemingly unsolvable challenge was encountered. And yet, every time, a breakthrough idea would bring them back to the forefront of research. The latest instance of this was in 2012, when a team of researchers won a popular image recognition competition (ILSVRC-2012) by a large margin, using a deep convolutional neural network (Krizhevsky et al., 2012).

Neural networks consist of *layers* of neurons. A *neuron* is the smallest building block of a neural network. Given a set of inputs, it is meant to perform a fast, simple computation and output a scalar response, imitating a biological neuron that responds to a set of incoming electrical signals. First, each neuron j computes a linear combination of their inputs x_i:

$$a_j = \sum_{i=1}^{D} w_{ji} x_i + b_j, \tag{1.14}$$

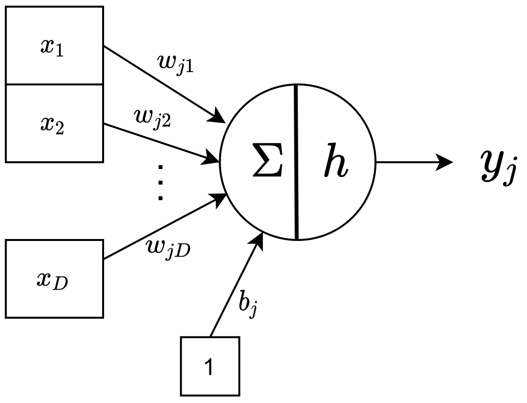


Figure 1.3: Illustration of a single artificial neuron.

where w_{ji} are called the *weights* and b_j the *bias* of neuron j. To achieve a nonlinear mapping, each neuron implements a differentiable, nonlinear *activation function* h such that its output is:

$$y_j = h(a_j). \tag{1.15}$$

A layer consists of a stack of neurons implementing the same activation function h, while each neuron has their own weights and biases. Therefore, each neuron in a layer can have a different response to the same set of input signals. Thus, we can compactly write the computation of a layer of M neurons in matrix-vector notation:

$$\mathbf{y} = h(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{1.16}$$

where $\mathbf{x} \in \mathbb{R}^D$ is the layer's input, $\mathbf{y} \in \mathbb{R}^M$ is the layer's output and $h(\cdot)$ is applied elementwise. The weight matrix $\mathbf{W} \in \mathbb{R}^{M \times D}$ and the bias vector $\mathbf{b} \in \mathbb{R}^M$ are the parameters of this layer that are to be learned.

Originally, typical activation functions for neural networks were the logistic sigmoid function and the $\tanh(\cdot)$ function. This way, the output of a neuron could be interpreted as a binary response, namely the neuron was "firing" or "not firing", and the model parameters were tuning the *sensitivity* of each neuron. Modern neural networks use asymmetric, semibounded activation functions such as rectified linear units (ReLUs), namely the $\max(\cdot, 0)$ function.

We mentioned that a neural network consists of layers of neurons. Each layer takes as input the output of the previous layer, transforms it in a nonlinear fashion and passes it forward to another layer that can have a different dimensionality. This structure allows neural networks to represent highly nonlinear input-output mappings. Notice that we can compactly represent a neural network's computation with a simple recursive formula:

$$\mathbf{x}^{(l)} = h_l(\mathbf{W}^{(l)}\mathbf{x}^{(l-1)} + \mathbf{b}^{(l)}), \tag{1.17}$$

where $h_l(\cdot)$ denotes the activation function implemented in the l-th layer, $\mathbf{x}^{(0)} = \mathbf{x}$ is an input vector, coming from the *raw* data samples and $\mathbf{x}^{(L)} = f(\mathbf{x};\theta)$ is the network's output vector for a network with L layers. The term *deep* learning describes networks with multiple layers (large L). Intuitively, the more layers a network consists of (the deeper the network), the more complex functions of the inputs it can represent. Each layer of the network gives us a different *representation* of the input, that can be seen as a more high-level description.

1.5.2 Back-Propagation

A convenient property of deep neural networks is that, since we can decompose the network to a sequence of differentiable functions (the layers), we can easily compute the gradient of the loss function with respect to all parameters by simply applying the chain rule of differentiation (Rumelhart et al., 1986).

1.5.3 Stochastic Gradient Descent

In Section 1.1 we mentioned that *gradient descent* is an optimization algorithm that we can use when there is no closed-form solution for the model parameters. Nevertheless, "standard" gradient descent needs the full training set to compute the gradient before doing a single optimization step. This is both expensive and can cause overfitting.

Instead, we can compute the gradient *online* using a single training sample each time to update the model parameters. Namely, we can perform N parameter updates by iterating through the (randomly shuffled) training set of size N and doing a gradient descent step for each sample. The gradient is not precise, but it can be shown that it is a *stochastic* approximation to the full gradient, thus also the name of the algorithm "stochastic gradient descent" or SGD. This version of the algorithm is faster and consumes only $O(1)$ instead of $O(N)$ memory. Further, it can be shown that the stochasticity of the algorithm (the order of the updates) achieves a sort of *regularization* of the model.

In practice, deep learning systems are trained using *minibatches*, namely each gradient descent step is computed using a small set of B samples ($B << N$) that is randomly sampled from the training set without replacement.

1.5.4 Architectures

Feed-Forward Networks

The network described in Section 1.5.1 is one of the first and more straightforward neural network architectures conceived (Rosenblatt, 1961). It is fully connected, namely each neuron in layer l is connected to *all* inputs from layer $l-1$. This architecture is also known as a *feed-forward* network.

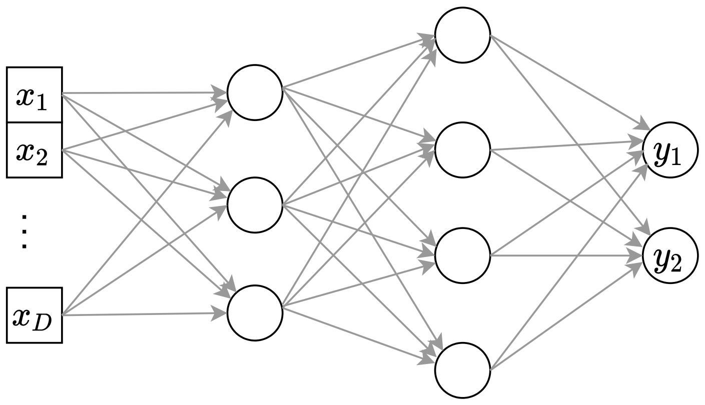


Figure 1.4: A 3-layer feed-forward neural network with layer sizes 3-4-2.

Convolutional Networks

As we mentioned in Section 1.5.1, the latest breakthrough that revamped neural networks as a state-of-the-art machine learning approach was through winning an image recognition challenge. Images are very high-dimensional inputs. Typically an image is represented as a 3-dimensional tensor of size $H \times W \times C$, where H and W are the height and width of the image respectively and C is the number of channels, which is usually 3 for RGB images or 1 for gray-scale images.

Using feed-forward networks for image-type inputs has two major drawbacks. First, the number of trainable parameters in the first layer would explode as we would need $H \cdot W \cdot C + 1$ parameters for each neuron. Second, stacking the image pixels as a vector would completely ignore the structure of images, namely that nearby pixels are more likely to be related than pixels that are far

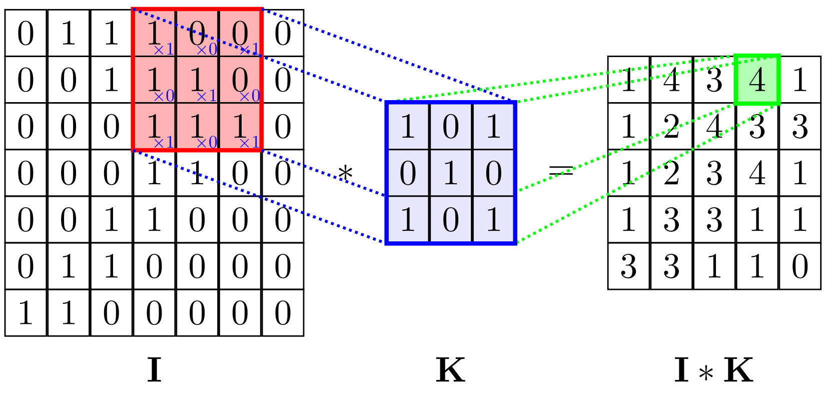


Figure 1.5: Illustration of 2D convolution: A single-channel input image $\mathbf{I}$ of dimensions 7×7 is convolved with a 3×3 convolutional filter $\mathbf{K}$, thereby producing an output image of dimensions 5×5.

apart from each other. This means, we assume that there exist *local* properties in pixel *neighborhoods* that are useful to describe the image content and these can be learned in a more efficient way.

This is where convolutional neural networks shine (Fukushima, 1980; LeCun et al., 1989). Instead of fully connecting each neuron to all input pixels, we learn small *filters* or *kernels* (e.g. of size 3×3) which we *convolve* with the image. Namely, we perform element-wise multiplication of the kernel with a small image region and sum the products. Then, we slide the kernel to the next pixel and convolve with the same kernel. We repeat this process until we cover the whole image. We can apply several different filters to the same image. The output will be another image or *feature map* of dimensions $H' \times W' \times C'$, where the new height H' and the new width W' will depend on the size of the kernel, and C' will be the number of different kernels we apply.

As with feed-forward networks, convolution operations are succeeded by nonlinear activation functions. What we have described until now constitutes a single *convolutional layer*. As with feed-forward networks, we can use the output of a layer as the input of another layer, thereby building deep convolutional networks.

Graph Neural Networks

Over the years, neural networks have been used in many machine learning problems, such as image processing, natural language processing and robotics tasks. The data generated and learned from in these tasks are mostly *regular*, namely the data can be represented as 1D vectors - which can be processed with feed-forward networks - or 2D (or 3D) grids that can be fed to convolutional networks.

Yet, in many applications, the data are generated from non-Euclidean domains that represent data as graphs, with relationships and mutual dependency between objects. This has led to a growing interest in deep learning research that focuses on the structure of graph data. Typical tasks include graph classification (e.g. to classify molecules as harmful or not), node classification (e.g. to detect malicious users in a social network) and link prediction (e.g. to recommend new connections to users of a social network).

Assume we are given a graph $G = (V, E)$ with nodes $V = \{1, 2, \dots, N\}$ and edges $E \subset V \times V$, where an edge $(i, j) \in E$ represents a connection between two nodes i and j. Additionally, we assume that each node i is associated with an feature vector $\mathbf{x}_i \in \mathbb{R}^D$, and each edge with a feature vector $\mathbf{e}_{ij} \in \mathbb{R}^F$. There are many frameworks one can use to describe the computation within a graph neural network (Battaglia et al., 2018). Here, we present a simple instantiation. One layer of a graph neural network will update the node and edge features, resulting in an updated graph with the same structure. First, each edge will get a new feature vector:

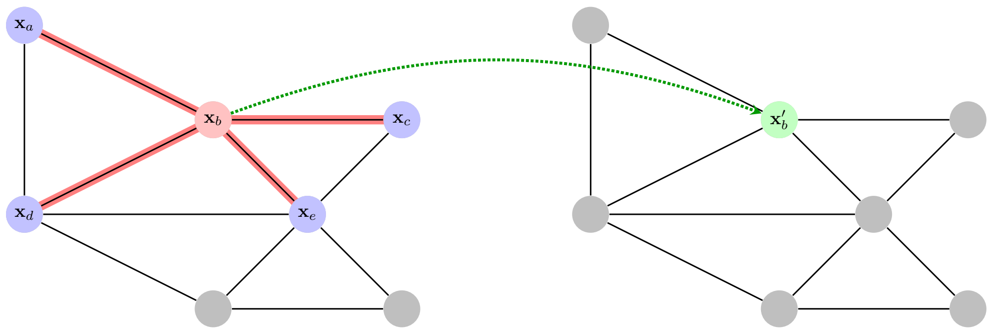


Figure 1.6: Computing a new node feature in a graph neural network: A new feature vector $\mathbf{x}'_b$ (green) is computed for node b. Information is aggregated from b's neighbors (purple) along the edges (red) connecting them.

$$\mathbf{e}'_{ij} = \phi_e(\mathbf{e}_{ij}, \mathbf{x}_i, \mathbf{x}_j), \tag{1.18}$$

where ϕ_e is a differentiable parameterized function, such as a feed-forward neural network. Second, each node will compute an incoming *message* by aggregating information from all edges incident to them:

$$\mathbf{m}_i = \rho(\{\mathbf{e}'_{ij} : j \in N(i)\}), \tag{1.19}$$

where $\rho(\cdot)$ is a permutation invariant function, such as elementwise sum or $\max(\cdot)$, and $N(i)$ denotes the neighbors of node i in G. Finally, each node will get an updated feature vector:

$$\mathbf{x}'_i = \phi_v(\mathbf{m}_i, \mathbf{x}_i), \tag{1.20}$$

where ϕ_v is another differentiable parameterized function, such as another feed-forward neural network.

Chapter 2

Non-Rigid 3D Shape Retrieval

The greatest thing by far is to be a master of metaphor; it is the one thing that cannot be learnt from others; and it is also a sign of genius, since a good metaphor implies an intuitive perception of the similarity in the dissimilar.

– **Aristotle**

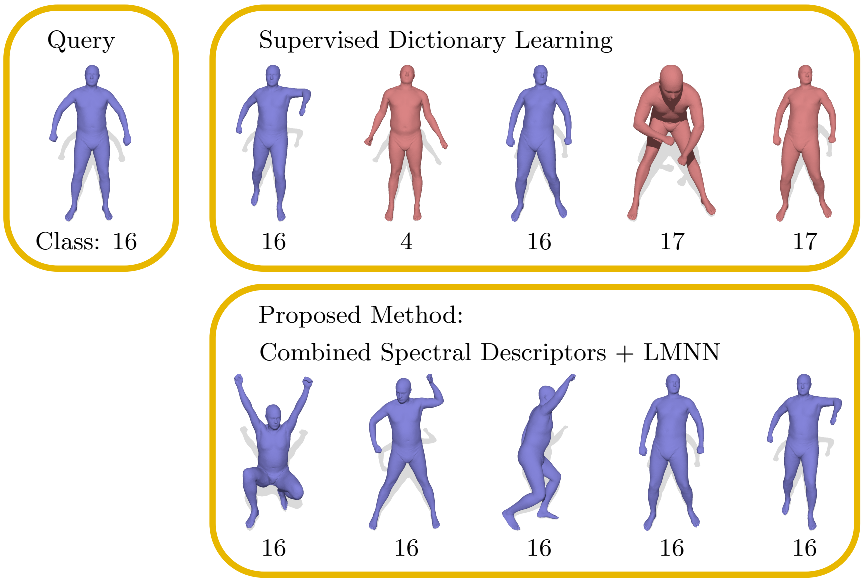

Query
Class: 16
Supervised Dictionary Learning
16
4
16
17
17
Proposed Method:
Combined Spectral Descriptors + LMNN
16
16
16
16
16

Figure 2.1: Example of shape retrieval from SHREC'14 Humans - real (scanned) dataset. The query model (top left) belongs to class 16. The top row shows the best five matches retrieved by the Supervised Dictionary Learning method (Litman et al., 2014). The best five matches retrieved by the proposed method (CSD+LMNN) are shown in the bottom row. The blue color indicates that the retrieved model corresponds to the correct class (i.e. 16) and the red color indicates an incorrect class. The quantitative experiments in Section 2.3 show that the proposed method outperforms the state of the art methods significantly on the SHREC'14 Humans dataset.

2.1 Introduction

This chapter is based on the paper by Chiotellis et al., 2016. The analysis of 3D shapes is becoming more and more important, with increasing amounts of data becoming available through novel 3D scanning technology and 3D modeling software. Among the numerous challenges in 3D shape analysis, in this work we focus on the problems of non-rigid shape similarity and non-rigid 3D shape retrieval: Given a set of 3D shapes and a previously unobserved query shape, we would like to efficiently determine the similarity of the query to all shapes in the database and identify the most similar ones – see Figure 2.1. The computation of shape similarity is a difficult problem, in particular if we wish to allow for non-rigid deformations of the shapes. Under such deformations the appearance of the object may change significantly. For many real-world retrieval applications on large 3D shape databases, it is important that the retrieval of similar shapes can be computed efficiently.

2.1.1 Related work

Much like in image analysis, the analysis of 3D shapes often starts with the extraction of local feature descriptors which are invariant to rigid and robust to non-rigid transformations of the shape. Popular descriptors include the Heat Kernel Signature (Sun et al., 2009), the Wave Kernel Signature (Aubry et al., 2011) and the scale-invariant Heat Kernel Signature (M. M. Bronstein and Kokkinos, 2010). In order to get a *correspondence* between 3D shapes, the shape analysis community has devised a variety of machine learning approaches to learn optimal point descriptors (A. M. Bronstein, 2011; Masci et al., 2015; Rodolà et al., 2014; Windheuser et al., 2014).

Learning approaches have also been used for shape *retrieval*. Litman et al., 2014 define a dictionary of point descriptors and use it to compute sparse representations of the point descriptors of each shape. Then they obtain global shape descriptors by sum pooling. The distances between them are considered to be the dis-similarity between the shapes. The authors go on to use unsupervised and supervised learning methods to optimize the classification results. In the supervised case, the authors try to minimize a triplet loss (Weinberger et al., 2005a) using a subset of the pooled descriptors as a training set. Using stochastic gradient descent (SGD) they propagate the error back to the dictionary of point descriptors. This means their objective is to learn an optimal dictionary of point descriptors for the specific task of shape retrieval and similarity ranking. Although this approach yielded state of the art results (Pickup et al., 2014), the computation time needed to learn the optimal dictionary (3-4 hours) makes it prohibitively expensive to use with larger datasets. In the work of Gasparetto and Torsello, 2015 the main purpose is to learn the invariant representation of each shape of a given dataset. The authors use a statistical framework to address classification tasks. While achieving state of the art performance, a major drawback of this method, from a learning point of view, is that it uses a large subset of the shapes (even 90%) for training.

2.1.2 Contribution

In this work, we propose a 3D shape retrieval method which provides state of the art performance while being substantially faster than previous techniques. We achieve this using a novel combination of stacked shape descriptors and a linear embedding of their distribution by means of a metric learning approach. In contrast to the approach by Litman et al., 2014, we do not employ dictionary learning to obtain shape descriptors from sparse point descriptors, but instead use weighted averaging directly on the point descriptors of a shape. We then learn a metric for the resulting shape descriptors so that samples from the same class are closer to each other than samples from different classes. Letting the learning process operate only on shape descriptors reduces our overall runtime tremendously. One of the main insights of our work is that the stacked shape descriptor alone does not lead to better performance, but in combination with the Large Margin Nearest Neighbor (LMNN) approach for metric learning, classification performance is significantly higher, reaching almost 98% mean average precision on the challenging "SHREC' 14 Humans - scanned" data set. Furthermore, our method is much faster than previous methods, as the individual steps require comparably only a few computations: Rather than several hours, our approach only needs approximately 4 minutes to learn the optimal embedding of shapes using only 40% of the shapes.

2.2 Approach

Our problem dictates to compare non-rigid shapes therefore we aim to obtain representations that capture their intrinsic properties. Our goal is to find representations such that similar shapes have proportionally similar descriptors. This becomes a particularly challenging problem when considering all possible deformations a single shape can have. In the next paragraphs we explain in detail every tool that we use. First we present an overview of our pipeline as illustrated in Figure 2.2.

2.2.1 Overview

A commonly used scheme in shape analysis is to model a shape $\mathcal{S}$ as a two-dimensional manifold $\mathcal{M}$ and representing it as a triangular mesh with a set of n vertices $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$, a set of triangular faces $\mathcal{F} \subset \mathcal{V}^3$ and a set of edges $\mathcal{E} \subset \mathcal{V}^2$ between adjacent vertices.

At first, we compute the Laplace-Beltrami operator (LBO) for each mesh in our dataset. We then compute a point descriptor $d(x)$ - based on the LBO - for each vertex of our mesh. There are different descriptors that can be utilized. We use the Wave Kernel Signature (Aubry et al., 2011) and the scale-invariant Heat Kernel Signature (M. M. Bronstein and Kokkinos, 2010). The reason to choose LBO-based descriptors is their inherent invariance to isometric deformations.

As a mesh can have several thousands vertices and datasets contain a large number of meshes, it becomes intractable to compare all point descriptors. Therefore we compute a weighted average of the point descriptors of each mesh and obtain a q-dimensional descriptor y_f for each shape. The shape descriptors y_f can either be the averaged siHKS, the averaged WKS or a combination of them. Our rational for the particular choice of descriptors is that siHKS captures global, while WKS focuses on local shape features. We argue that a stacked combination of them contains diverse information that can be fully exploited by a metric learning algorithm.

In the end we feed a subset of our shape descriptors y_f along with their labels to a supervised metric learning algorithm (LMNN). The algorithm learns

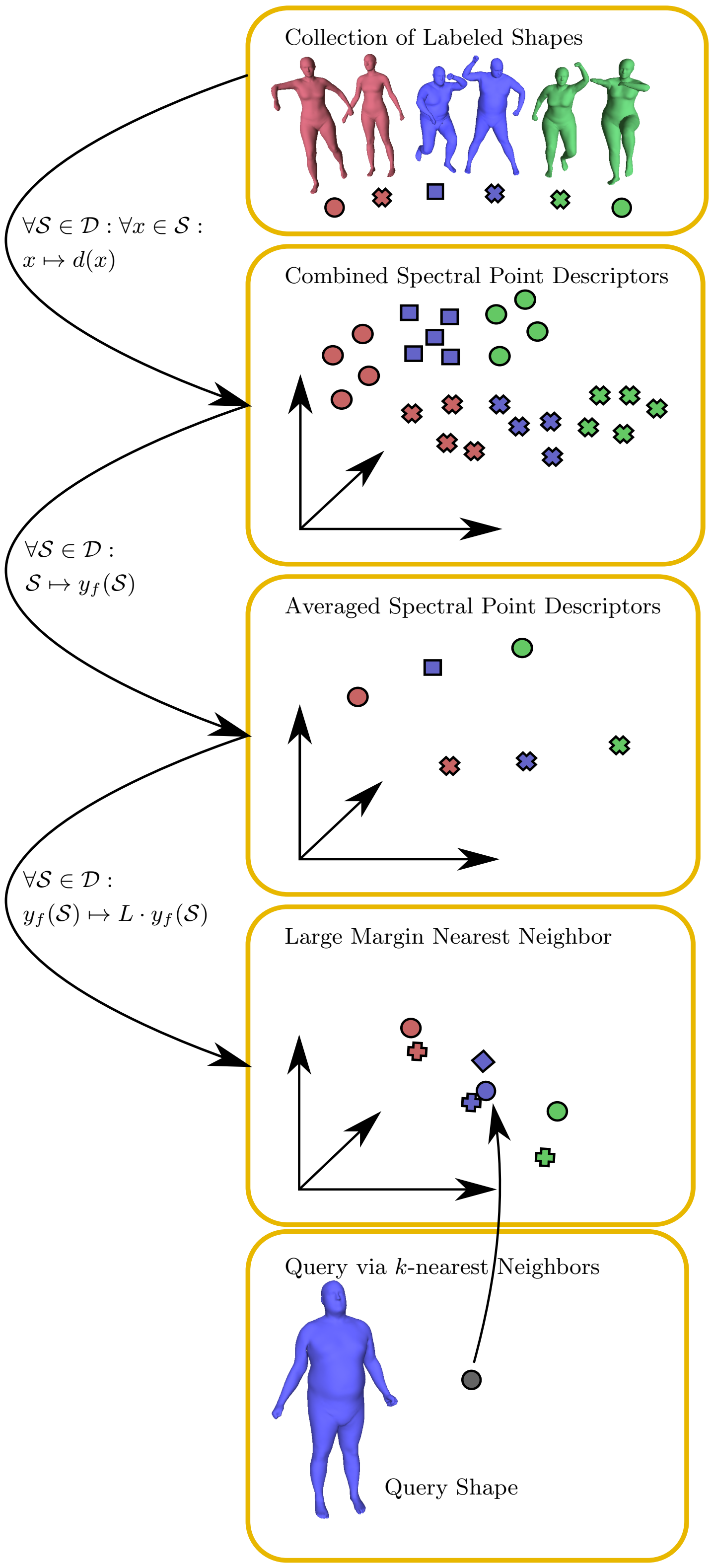

Collection of Labeled Shapes
$\forall \mathcal{S} \in \mathcal{D} : \forall x \in \mathcal{S} :$
$x \mapsto d(x)$
Combined Spectral Point Descriptors
$\forall \mathcal{S} \in \mathcal{D} :$
$\mathcal{S} \mapsto y_f(\mathcal{S})$
Averaged Spectral Point Descriptors
$\forall \mathcal{S} \in \mathcal{D} :$
$y_f(\mathcal{S}) \mapsto L \cdot y_f(\mathcal{S})$
Large Margin Nearest Neighbor
Query via k-nearest Neighbors
Query Shape

Figure 2.2: **Overview:** Schematic illustration of the proposed method.

a linear mapping L of the shape descriptors such that shapes with different labels are easier to distinguish from one another in the new space.

Now when we want to classify a new shape, all we need to do is to compute the same type of shape descriptor y_f as the one we trained our classifier with and transform it into the new space by applying the learned mapping L. The labels of the k closest shapes in the transformed space determine the predicted label for our query shape.

2.2.2 The Laplace-Beltrami operator

The Laplace-Beltrami operator (LBO) is a natural generalization of the Laplace operator for Riemannian manifolds. Like the Laplacian, it is defined as the (negative) divergence of the gradient, and it is a linear operator mapping functions to functions. Therefore the LBO is often also simply referred to as the Laplacian. Formally, given a smooth scalar field $f : \mathcal{M} \to \mathbb{R}$ on the manifold $\mathcal{M}$ associated to shape $\mathcal{S}$, the Laplace-Beltrami operator Δ is defined as

$$\Delta f := -\mathrm{div}(\nabla f). \tag{2.1}$$

One of the most important properties of the Laplacian is that it is invariant under isometric deformations. Particularly useful are the eigenvalues $\lambda_i \in \mathbb{R}$ and the eigenfunctions $\phi_i : \mathcal{M} \to \mathbb{R}$ of the Laplacian, i.e.

$$\Delta \phi_i := \lambda_i \phi_i. \tag{2.2}$$

The eigenvalues λ_i of Equation (2.2) – known as the *Helmholtz equation* – are non-negative and represent a discrete set $(0 = \lambda_0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq +\infty)$. The corresponding eigenfunctions can be chosen to form an orthonormal basis:

$$\langle \phi_i, \phi_j \rangle = \int_{\mathcal{M}} \phi_i(x)\phi_j(x)dx = \begin{cases} 0, & \text{if } i \neq j \\ 1, & \text{if } i = j. \end{cases} \tag{2.3}$$

Discretization

A popular discretization of the LBO is the cotangent scheme (Pinkall and Polthier, 1993; Reuter et al., 2009). It allows to compute the eigenvalues λ_i and eigenvectors ϕ_i as the solutions to the generalized eigenvalue problem

$$A\phi_i = \lambda_i B \phi_i, \tag{2.4}$$

where $A \in \mathbb{R}^{n\times n}$ is the *stiffness matrix* and $B \in \mathbb{R}^{n\times n}$ is the *mass matrix*. Concretely, A is defined as

$$A_{ij} = \begin{cases} \frac{\cot \alpha_{ij} + \cot \alpha_{ji}}{2}, & \text{if } (v_i, v_j) \in \mathcal{E} \\ -\sum_{k\in N(i)} A_{ik}, & \text{if } i = j, \end{cases} \tag{2.5}$$

where α_{ij} and α_{ji} are the two angles opposite of the edge (v_i, v_j) and $N(i)$ is the one-ring neighborhood of vertex v_i. The mass matrix B is defined as

$$B_{ij} = \begin{cases} \frac{a(T_1)+a(T_2)}{12}, & \text{if } (v_i, v_j) \in \mathcal{E} \\ \frac{\sum_{k \in N(i)} a(T_k)}{6}, & \text{if } i = j, \end{cases} \tag{2.6}$$

where T_1, T_2 are the triangles that share the edge (v_i, v_j), and $a(T)$ is the area of triangle T. Often a simplified "lumped" diagonal version of the mass matrix is used:

$$B_{ii} = \frac{\sum_{k \in N(i)} a(T_k)}{3}, \tag{2.7}$$

i.e. B_{ii} is considered as the corresponding area element of vertex v_i. The geometric concepts of these formulas are depicted in Figure 2.3.

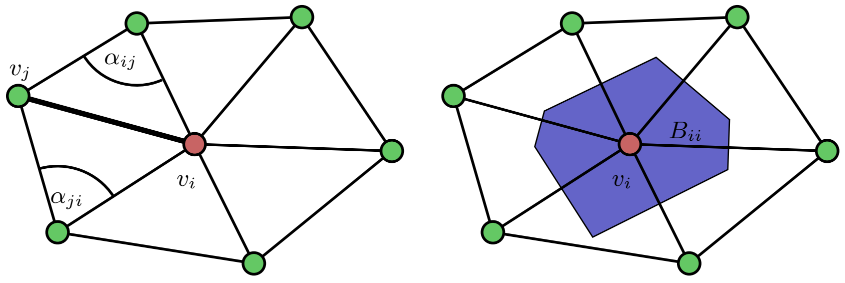


Figure 2.3: **Stiffness Matrix (left):** The entries A_{ij} of the stiffness matrix A contain the average of the cotangents of the angles α_{ij}, α_{ji} opposite to the edge (v_i, v_j). Thus the name *cotangent scheme*. **Mass Matrix (right):** The diagonal entries B_{ii} of the mass matrix B correspond to the Voronoi area around vertex v_i.

2.2.3 Point descriptors

Local feature descriptors have been proven particularly useful in shape analysis tasks such as shape matching (point-to-point correspondence) and shape retrieval. In the following we describe three of the most used ones.

Heat Kernel Signature

The HKS (Sun et al., 2009) is - as the name indicates - based on the heat diffusion process on a surface S which is governed by the Heat equation:

$$\Delta u(x,t) = -\frac{\partial}{\partial t} u(x,t). \tag{2.8}$$

The solution $k_t(x, x)$ can be interpreted as the amount of heat that remains at point x of surface S after time t when starting with a unit heat source u_0

concentrated at x at $t_0 = 0$. The eigen-decomposition of the Heat Kernel is

$$k_t(x,y) = \sum_{k=0}^{\infty} e^{-\lambda_k t}\phi_k(x)\phi_k(y), \tag{2.9}$$

so the HKS is just

$$k_t(x,x) = \sum_{k=0}^{K-1} e^{-\lambda_k t}\phi_k(x)^2, \tag{2.10}$$

as we truncate the basis to the first K eigenfunctions of the LBO. Concatenating the solutions for different times $\{t_1, t_2, \ldots, t_T\}$ we obtain a descriptor of the form

$$HKS(x) = (k_{t_1}(x,x), k_{t_2}(x,x), \ldots, k_{t_T}(x,x)). \tag{2.11}$$

Scale Invariant Heat Kernel Signature

M. M. Bronstein and Kokkinos, 2010 developed a scale-invariant version of the Heat Kernel Signature (siHKS) using the logarithm, the derivative and the Fourier transform moving from the time domain to the frequencies domain. Assuming a shape is scaled by a factor β, and rewriting time t as α^τ, the heat kernel of the scaled shape would only be shifted in τ by $2\log_\alpha \beta$. The authors first constructed a *scale-covariant heat kernel*:

$$scHKS(x,x) = -\frac{\sum_{k=1}^{K} \lambda_k \alpha^\tau \log \alpha e^{-\lambda_k \alpha^\tau}\phi_k(x)^2}{\sum_{k=1}^{K} e^{-\lambda_k \alpha^\tau}\phi_k(x)^2}. \tag{2.12}$$

In the Fourier domain this shift results in a complex phase $H(\omega)e^{-i\omega 2\log_\alpha \beta}$ where $H(\omega)$ denotes the Fourier transform of *scHKS* w.r.t. τ. Finally the *scale-invariant HKS* is constructed by taking the absolute value of $H(\omega)$ (thus undoing the phase) and then sampling $|H(\omega)|$ at q frequencies $\{\omega_1, \ldots, \omega_q\}$ Litman et al., 2014 :

$$siHKS(x) = (|H(\omega_1)|, \ldots, |H(\omega_q)|)^T. \tag{2.13}$$

Wave Kernel Signature

The Wave Kernel Signature (WKS) (Aubry et al., 2011) - inspired by quantum mechanics - describes the average probability over time to locate a particle with a certain energy distribution f_E at point x. The movement of a quantum particle on a surface is governed by the wave function $\psi(x,t)$ which is a solution of the Schrödinger equation

$$\frac{\partial \psi(x,t)}{\partial t} = i\Delta\psi(x,t). \tag{2.14}$$

The energy distribution of a quantum particle depends on the LBO eigenvalues. Therefore the wave equation for a particle can be written as

$$\psi_E(x,t) = \sum_{k=0}^{\infty} e^{i\lambda_k t}\phi_k(x) f_E(\lambda_k). \tag{2.15}$$

The probability to locate the particle at point x is then $|\psi_E(x,t)|^2$. Therefore the average probability over time is

$$p(x) = \lim_{T\to\infty}\frac{1}{T}\int_0^T |\psi_E(x,t)|^2 = \sum_{k=1}^{\infty}\phi_k(x)^2 f_E(\lambda_k)^2. \tag{2.16}$$

As we described, the LBO and its spectrum capture intrinsic properties of a shape. Therefore different choices of f_E give us shape properties at different scales. Evaluating with energy distributions $\{e_1,\dots,e_q\}$ we get the vector for the Wave Kernel Signature:

$$WKS(E,x) = (p_{e_1}(x),\dots,p_{e_q}(x))^T. \tag{2.17}$$

Note that as with the HKS we must truncate the sum at the first K eigenvalues. Typical values for K are 50 or 100.

2.2.4 Weighted average

Our aim is to use the shape descriptors mentioned above and the learned distance metric to classify shapes. However, for a given shape so far we only have a number of point descriptors, but for classification we would prefer to have one descriptor for the whole shape. To achieve this, we compute a weighted average over all point descriptors $d(x)$ computed from the points x of a given shape $\mathcal{S}$. Thus, our shape descriptor is defined as

$$y_f(\mathcal{S}) = \sum_{x\in\mathcal{S}} w_x d(x) \quad \text{with} \quad w_x = \frac{a_x}{\sum_{y\in\mathcal{S}} a_y}, \tag{2.18}$$

where a_x is the area element associated with vertex $x \in \mathcal{S}$. This weighted averaging is inspired by the pooling step proposed by Litman et al., 2014, however with the difference that we do not use sparse coding.

In the case of WKS we normalize the point descriptors by the L_2-norm. Both averaged shape descriptors are also normalized by the L_2-norm. We compared 3 different shape descriptors, the averaged WKS, the averaged siHKS and a combination of them we refer to as Combined Spectral Descriptor (CSD):

$$y_{CSD}(\mathcal{S}) = \begin{pmatrix} y_{WKS}(\mathcal{S}) \\ y_{siHKS}(\mathcal{S}) \end{pmatrix}. \tag{2.19}$$

Algorithm 1 Shape descriptors

```
procedure GET–AVERAGED –DESCRIPTORS
    for each shape $\mathcal{S} \in \mathcal{D}$ do
        for each point $x \in \mathcal{S}$ do
            $\tilde{d}(x) \leftarrow \text{siHKS}(x)\ |\text{WKS}(x)$
            if WKS then
                $d(x) \leftarrow \frac{\tilde{d}(x)}{||\tilde{d}(x)||_2}$
            else
                $d(x) \leftarrow \tilde{d}(x)$
        end for
        $\tilde{y}_f(\mathcal{S}) \leftarrow \sum_{x \in \mathcal{S}} w_x d(x)$ (see Equation (2.18))
        $y_f(\mathcal{S}) \leftarrow \frac{\tilde{y}_f(\mathcal{S})}{||\tilde{y}_f(\mathcal{S})||_2}$
    end for
```

2.2.5 Large Margin Nearest Neighbor

Large Margin Nearest Neighbor (LMNN) is a machine learning algorithm that was first introduced in 2005 (Weinberger et al., 2005b). The authors keep updating the algorithm and their implementation[1] is very efficient even for applications with very large datasets. As of the latest version that we used, the L-BFGS algorithm is used for optimization by default.

LMNN utilizes both the concept of SVMs of margin maximization and the well known *k-NN* algorithm. It is specifically conceived to learn a Mahalanobis (semi-)metric $\mathcal{D}_M$ that improves the accuracy of k-NN classification. This metric is represented by the positive semi-definite matrix $M \in \mathbb{R}^{n \times n}$, such that

$$\mathcal{D}_M(\vec{x}, \vec{y}) = \langle M(\vec{x} - \vec{y}), (\vec{x} - \vec{y}) \rangle^{\frac{1}{2}}. \tag{2.20}$$

Equivalently $\mathcal{D}_M(\vec{x}, \vec{y})$ can be seen as the Euclidean distance between the points $\vec{x}, \vec{y}$ transformed by the linear transformation $L \in \mathbb{R}^{m \times n}$, i.e.

$$\mathcal{D}_L(\vec{x}, \vec{y}) = \|L\vec{x} - L\vec{y}\|, \tag{2.21}$$

as the positive semi-definiteness of M allows a decomposition $M = L^\top L$.

The main idea of the algorithm is to find a mapping L so that for each input $\vec{x}_i$ there are at least k neighbors that share its label y_i (see Figure 2.4). This is facilitated by choosing *target neighbors* of $\vec{x}_i$, i.e. samples that are desired to be closest to $\vec{x}_i$. The target neighbors for every input are fixed during the whole learning process. Note that target neighbors are not symmetric. For instance if $\vec{x}_j$ is a target neighbor of $\vec{x}_i$ it is not necessary that $\vec{x}_i$ is also a target neighbor of $\vec{x}_j$.

Furthermore, LMNN tries to ensure that differently labeled inputs are farther away from the target neighbors so that they do not get selected by

[1]http://www.cs.cornell.edu/~kilian/code/code.html.

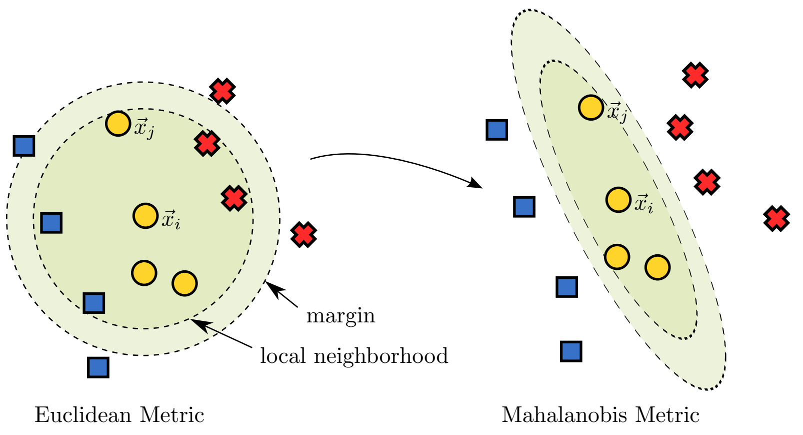


Figure 2.4: **Large Margin Nearest Neighbor (LMNN)** finds the best positive semi-definite matrix M, such that the induced Mahalanobis (semi-)norm $\mathcal{D}_M(\vec{x}_i, \vec{x}_j) = \langle M\vec{x}_i - \vec{x}_j, \vec{x}_i - \vec{x}_j\rangle^{\frac{1}{2}}$ separates the different classes as good as possible.

k-NN. Samples that violate this rule are called *impostors*. Ideally we would like to create a large margin between the perimeter around each input and its target neighbors, and all differently labeled inputs as illustrated in Figure 2.4 on the right. This goal also explains the name of the algorithm.

Loss Function

The loss function consists of two competing terms. The first one pulls target neighbors together:

$$\epsilon_{\text{pull}}(L) = \sum_{i,j \rightsquigarrow i} ||L(\vec{x}_i - \vec{x}_j)||^2. \tag{2.22}$$

The notation in Equation (2.22) implies that $\vec{x}_j$ are target neighbors of $\vec{x}_i$. The pull loss penalizes large distances between inputs and their target neighbors. This is an important difference of LMNN compared to other algorithms where large distances to *all* other similarly labeled samples are penalized. The second term pushes impostors away:

$$\epsilon_{\text{push}}(L) = \sum_{i,j \rightsquigarrow i} \sum_{l} (1 - y_{il})[1 + ||L(\vec{x}_i - \vec{x}_j)||^2 - ||L(\vec{x}_i - \vec{x}_l)||^2]_+, \tag{2.23}$$

where $[x]_+ = \max(x, 0)$ denotes the standard hinge loss and y_{il} is 1 only when $y_i = y_l$ and 0 otherwise. Note that the choice of the unit margin is an arbitrary convention that sets the scale for the linear transformation L. If a different margin $c > 0$ was enforced, the loss function would be minimized by the same linear transformation up to an overall scale factor $\sqrt{c}$. Combining both terms we get the LMNN loss function:

$$\epsilon(L) = \mu \epsilon_{\text{pull}}(L) + (1-\mu)\epsilon_{\text{push}}(L), \tag{2.24}$$

where $\mu \in [0,1]$ is a trade-off parameter between small intra-class and large inter-class distances. Although μ can be estimated with cross validation, in practice setting $\mu = 0.5$ works well. There are several similarities with the SVM's loss function:

- One term penalizes the norm of the *parameter* vector (i.e., $\mathbf{w}$ in SVMs, $\mathbf{L}$ in LMNN)
- The hinge loss is only triggered by samples near the decision boundary
- Both loss functions can be rewritten to utilize the *kernel trick*
- Both problems can be reformulated as convex optimization problems

Convex Optimization

While $\epsilon(L)$ is quadratic in L, Equations (2.20) and (2.21) allow us to restate the loss of ϵ of Equation (2.24) in terms of M. Minimizing this loss becomes a semi-definite program (SDP) which is a convex problem that can be solved globally in polynomial time. For the SDP formulation, Weinberger et al., 2005a introduced slack variables $\{\xi_{ijl}\}$ for all triplets of target neighbors $\vec{x}_i, \vec{x}_j$ and impostors $\vec{x}_l$. The slack variables measure the level of margin violation. Therefore the SDP can be defined as:

$$\begin{array}{lrcll}
\textbf{min.} & \multicolumn{4}{l}{\mu \sum_{i,j \rightsquigarrow i} (\vec{x}_i - \vec{x}_j)^T M (\vec{x}_i - \vec{x}_j) + (1-\mu) \sum_{i,j \rightsquigarrow i,l} (1 - y_{il}) \xi_{ijl}} \\
\textbf{s.t.} & (\vec{x}_i - \vec{x}_l)^T M (\vec{x}_i - \vec{x}_l) - (\vec{x}_i - \vec{x}_j)^T M (\vec{x}_i - \vec{x}_j) & \geq & 1 - \xi_{ijl} & \\
 & \xi_{ijl} & \geq & 0 & \forall i,j,l \\
 & M & \succeq & 0\,. &
\end{array}$$

where the last constraint implies that the matrix M must be positive semi-definite. The authors created their own solver for the SDP in order to take advantage of the sparsity of the slack variables. This leads to much faster solutions.

Optimal training parameters

The LMNN optimization process requires three parameters to be specified beforehand: the dimension m of the lower-dimensional space into which the samples are mapped by L, the number of neighbors k to consider, and the number of iterations r of the L-BFGS optimizer. To find good values for these parameters, a validation set is used, which is a part of the original training data. Then, the LMNN optimization is run on the remaining data with different parameter settings, that are chosen using Bayesian optimization, and evaluated on the validation set. After a given number of iterations, the parameter set that achieved the highest performance on the validation set is used to run LMNN training on the entire training set.

Classification

For classification, we use the k nearest-neighbor classifier in the m-dimensional target space. Thus, for a given test shape we compute its descriptor, map it into $\mathbb{R}^m$ using the mapping L found in the training step, and assign to it the most frequent label of the k closest, by the Euclidean distance, mapped training samples.

2.3 Experiments

2.3.1 Datasets

We evaluated our approaches on 2 datasets from SHREC'14 - Shape Retrieval of Non-Rigid 3D Human Models (Pickup et al., 2014). Of the two main datasets, one consists of synthetic and one of real (scanned) 3D human models. Each class represents a human model and each instance of a class is a different pose of that model. This is a different setting than most classification problems where distinct classes correspond to naturally separate categories (like humans, dogs, cats, etc.). This property along with the fact that some models contain self-intersections makes these datasets particularly challenging.

We used the provided evaluation code from Pickup et al., 2014 that computes several accuracy metrics: nearest neighbor, first tier, second tier, discounted cumulative gain, e-measure, f-measure, precision and recall.

All meshes were down-sampled to 20.000 faces with *Meshlab* (Cignoni et al., 2008).

2.3.2 Evaluation setting

We scaled the shapes as indicated in the available code that accompanies the work by Litman et al., 2014. We truncated the bases of the LBO to the first 100 eigenfunctions. Based on them we computed 50-dimensional siHKS descriptors with the same settings used by Litman et al., 2014 and 100-dimensional WKS descriptors, setting the variance to 6. We used 40% of the shape descriptors to train the LMNN classifier and tested on the rest. We used 25% of the training set as a validation set to find the optimal parameters for LMNN[2].

[2]Our code is available at `https://github.com/tum-vision/csd_lmnn`.

Metric	**CSD**	**CSD+LMNN**
nn	0.5075	0.9792
ft/fm	0.3692	0.9278
st	0.5669	0.9868
em	0.3135	0.2703
dcg	0.6407	0.9760

Table 2.1: Evaluation on the SHREC'14 real dataset.

Metric	**CSD**	**CSD+LMNN**
nn	0.8267	0.9967
ft/fm	0.6789	0.9802
st	0.9147	0.9986
em	0.6358	0.5114
dcg	0.9066	0.9963

Table 2.2: Evaluation on the SHREC'14 synthetic dataset.

Our CSD approach gives remarkable results, when combined with LMNN. Even though the SHREC'14 datasets are considered extremely challenging, our algorithm performed better than the methods that participated in the SHREC'14 contest (see Table 2.3) and the most recent learning approach proposed by Gasparetto and Torsello, 2015. This is a significant result since our approach is comparatively simpler and the computation time very low.

Figure 2.5 shows the result of the LMNN learning step. As one can see, LMNN is able to capture the discriminative features of the classes despite the information loss from the projection onto three dimensions, which is done to facilitate the visualization.

We noticed that using both the siHKS and the WKS performed worse than using each descriptor separately. Nevertheless, when used as input to a metric learning algorithm, the performance of the combined descriptor improved considerably. The CSD with LMNN performs better than either individual descriptor with LMNN (see Table 2.3). In particular, we observe that even if we add a seemingly harmful descriptor, as in the case of the WKS for the real dataset, LMNN is able to select the most useful - in terms of k-NN classification - dimensions of both descriptors, thereby achieving a better accuracy than the siHKS+LMNN approach. This confirms our hypothesis that metric learning can utilize the additional information contained in the combined descriptor. Adding other descriptors to the CSD such as the GPS (Rustamov, 2007) led to no improvement.

Note that in our CSD+LMNN-approach the most time-consuming part is finding the optimal parameters for LMNN. Still the total time needed for the algorithm - excluding the computation of point descriptors - is approximately 2 minutes. In the worst case it never exceeded 4 minutes on a machine with

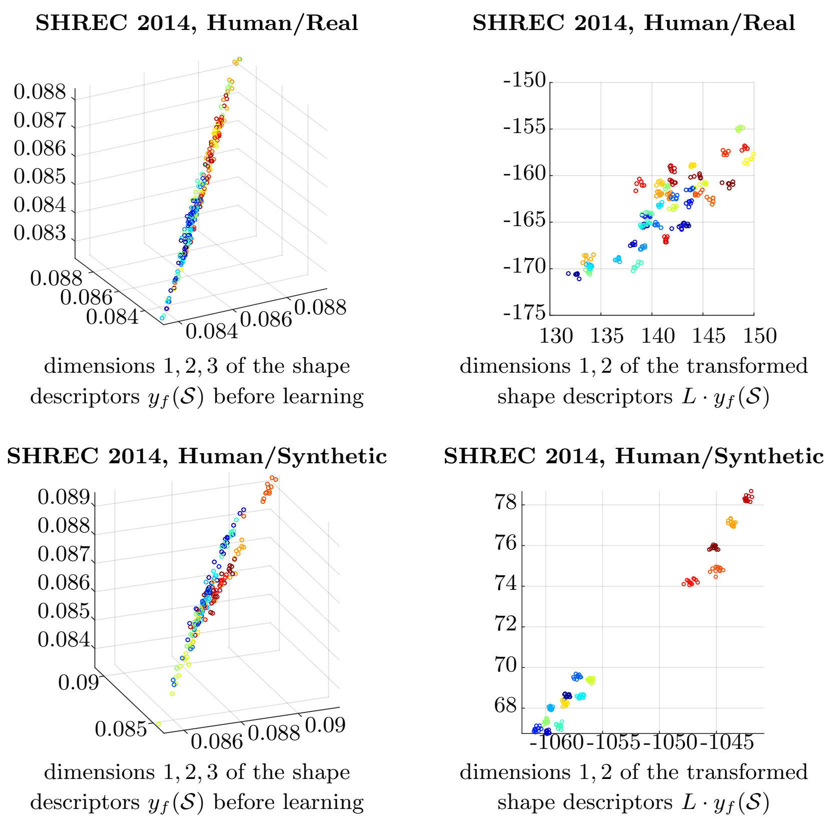


Figure 2.5: **Visualization of the shape descriptors before and after learning:** Each circle corresponds to the descriptor of one shape. The colors correspond to the 40 classes of the SHREC'14 Real dataset (top row) or the 15 classes of the SHREC'14 Synthetic dataset (bottom row). It can be seen by even visualizing only two dimensions that the transformation L, learned by LMNN, results in a much better clustering of the shapes. This is in line with the quantitative evaluation on the datasets.

a 2.0GHz CPU. This is an extremely small amount of time compared to the supervised dictionary learning approach proposed by Litman et al., 2014 which needs nearly 4 hours to converge on a machine with a 3.2GHz CPU.

2.4 Conclusion

In this paper we showed that metric learning can significantly improve the classification accuracy of well known descriptors. Given a large number of features, a learning algorithm such as LMNN can select the most informative ones and weight them appropriately for the problem that we aim to solve, in this case shape retrieval. Our approach is both considerably faster and more

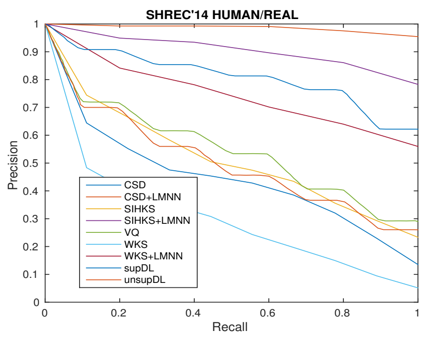


Figure 2.6: Precision-Recall comparison on the SHREC'14 real dataset.

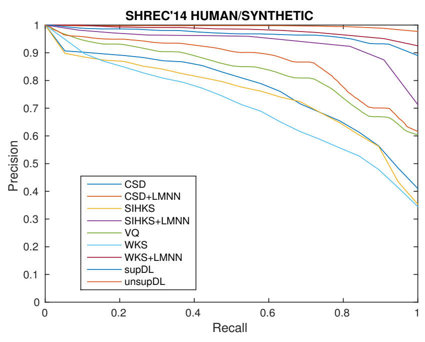


Figure 2.7: Precision-Recall comparison on SHREC'14 synthetic dataset.

Method	**Synthetic**	**Real (Scanned)**
ISPM	90.2	25.8
DBN	84.2	30.4
R-BiHDM	64.2	64.0
HAPT	81.7	63.7
ShapeGoogle(VQ) (A. M. Bronstein et al., 2011)	81.3	51.4
Unsupervised DL (Litman et al., 2014)	84.2	52.3
Supervised DL (Litman et al., 2014)	95.4	79.1
RMVM (Gasparetto and Torsello, 2015)	**96.3**	**79.5**
siHKS	84.33	62.00
siHKS+LMNN	97.11	92.58
WKS	91.33	33.75
WKS+LMNN	98.11	76.92
CSD	82.67	50.75
CSD+LMNN	**99.67**	**97.92**

Table 2.3: Comparison of retrieval methods in terms of mean average precision (mAP, in %) on the SHREC'14 3D Human Models datasets. In the upper part of the table, results of methods that participated in the SHREC'14 contest are documented as in (Pickup et al., 2014) and the most recent learning approach proposed by Gasparetto and Torsello, 2015. In the lower part, we report the results of our approaches, averaged over 5 runs (different training/testing sets splits).

accurate than the state of the art. The comparison in Figure 2.1 demonstrates that our approach is more robust, as it is able to find the true inherent similarities between objects and does not get confused by different classes, even if they are very similar by human standards.

Chapter 3

Learning to Drive from Demonstrations

Setting an example is not the main means of influencing others, it is the only means.

– Albert Einstein

This chapter is based on the paper by Sharifzadeh et al., 2016. In this work we propose an inverse reinforcement learning (IRL) approach using Deep Q-Networks to extract the rewards in problems with large state spaces. We evaluate the performance of this approach in a simulation-based autonomous driving scenario. Our results reflect the intuitive relation between the reward function and readings of distance sensors mounted at different poses on the car. We also show that, after a few learning rounds, our simulated agent generates collision-free motions and performs human-like lane change behavior.

3.1 Introduction

Robots and autonomous systems are becoming more and more a part of everyday life by assisting us in various tasks. An important requirement for these systems is that they behave in a human-acceptable, socially normative way, e.g. by respecting personal spaces or by treating groups of people based on the social relations of the individuals (Kruse et al., 2013; Triebel et al., 2015). This means that humans should not just be regarded as obstacles and that an optimal robot motion should also consider human comfort metrics (Okal and Arras, 2016).

One of the most popular instances of autonomous systems in the current decade are self-driving cars. The ultimate goal of autonomous cars is to drive the passengers from one point to another without any human input, while assuring the comfort of human passengers. Defining "comfort" is not straight forward and this makes it hard to define a suitable objective function for motion planning. One widely used method aiming to fulfill this objective is introduced by Werling et al., 2010. They propose to provide "ease and comfort" by producing jerk-optimal trajectories. Later, lane change experiments conducted by Tehrani et al., 2014 in Japanese highways showed that the human lane change behavior cannot be modeled by a single stage of jerk-optimal trajectories (Tehrani et al., 2015). Instead a two-stage model was proposed. Many other models exist similar to these, however they mostly fail to generate human-like behaviors. Modeling human driving behavior becomes even more complicated when considering scenarios such as driving in large cities with many intersections, traffic lights, pedestrians, etc.. Therefore, applying machine learning methods to extract a model directly from expert demonstrations appears more promising.

In a recent work, Bojarski et al., 2016 proposed an end-to-end supervised learning approach that maps the front facing camera images of a car to steering angles, given expert data. However, this approach requires a large amount of data from different possible driving scenarios in order to get a good approximation of the policy. Still, it might fail when facing scenarios that are very different from the ones in the training data. A more promising formulation for this problem is based on Markov Decision Processes (MDPs).

In this framework, one can apply Inverse Reinforcement Learning (IRL) to extract the unknown reward function of the driving behavior (Ng and Russell, 2000). The hope is that by approximating the reward function rather than directly learning the state-action pairs in a supervised fashion, one can learn policies that generalize better to new scenarios. Finding the reward function of an MDP using IRL was proposed by Ng and Russell, 2000 and further improved by Abbeel and Ng, 2004. Since then, several variations of IRL have been proposed such as Bayesian IRL (Ramachandran and Amir, 2007), maximum entropy based methods (Ziebart et al., 2008) and max margin prediction (Ratliff et al., 2006). Most of the recent methods have been inspired by these works. For example, Wulfmeier et al., 2015 proposed a maximum entropy based approach that handles nonlinear reward functions using deep neural networks. However, most of these approaches are limited to small state spaces that cannot fully describe real-world driving scenarios. One of the main reasons is the difficulty of applying the Reinforcement Learning (RL) step in large state spaces. While RL algorithms using Deep Q-Networks (DQN) (Mnih et al., 2015) have achieved state-of-the-art performance, to the best of our knowledge, DQNs have not been used in IRL methods before.

In this paper, we address the exploding state space problem and build upon the projection-based IRL method by Abbeel and Ng, 2004, using a DQN. We implemented a highway driving simulator and evaluated the performance of our approach by analyzing the extracted rewards. The evaluation is presented in Section 3.3.

3.2 Problem Formulation

A Markov Decision Process (MDP) is defined as a tuple (S, A, T, γ, R), where S and A are the state and action spaces, T is the transition matrix, $\gamma \in [0, 1]$ is the discount factor and $R : S \times A \to \mathbb{R}$ is the reward function. A *policy* $\pi : S \to A$ maps states to actions. It can also be seen as a probability distribution over actions at each state. A *value* or *state-value* function $V^\pi(s_0)$ is defined as the expected discounted future reward if we start from state s_0 and act according to policy π:

$$V^\pi(s_0) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi] \tag{3.1}$$

The discount factor applies the amount of uncertainty that we have about the future rewards. The action-value function Q represents the value that we can gain if we start from state s_0 and take action a and follow policy π after that:

$$Q(s_0, a) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, a] \tag{3.2}$$

The motion planning problem can be formulated as a Markov Decision Process, in which finding the optimal action-value function is the goal. However, similar to many other real-world applications, the transition probabilities and the reward function are unknown. Given expert demonstrations, IRL methods have been shown to effectively find the underlying reward function and consequently, the action-value function. In order to apply such approaches to large state spaces, we propose using DQNs (Mnih et al., 2015) for the Reinforcement Learning step in the inner loop. Previously, using neural networks to approximate the Q-function has been shown to cause instabilities or divergence (Tsitsiklis and Van Roy, 1997). In order to address these problems, Mnih et al., 2015 proposed two key ideas. First, to randomly sample training data from the sequence of past experiences ("experience replay"). Second, to use a second neural network to estimate the *target* Q-values. The weights of this *target network* are only periodically replaced with the weights of the original network. Here, we apply this approach to the projection-based IRL method (Abbeel and Ng, 2004) but other IRL techniques can also benefit from it.

Given expert demonstrations, we want to generate policies π whose values are close to the value of the expert policy π_E:

$$\|V^{\pi}(s_0) - V^{\pi_E}(s_0)\| \leq \epsilon \tag{3.3}$$

Every state s_i is spanned by d-dimensional feature vectors $\phi(s_i)$, with features such as speed, acceleration, sensor readings, etc.. The reward function is defined as a linear combination of these features:

$$R(s_i) = \mathbf{w} \cdot \phi(s_i) \ , \tag{3.4}$$

where $\mathbf{w} \in \mathbb{R}^d$ is the weight vector and $\|\mathbf{w}\|_2 \leq 1$. Plugging (3.4) into (3.1), we get

$$V^{\pi}(s_0) = \mathbf{w} \cdot \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi] \tag{3.5}$$

Furthermore, feature expectations are defined as:

$$\mu(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi] \ . \tag{3.6}$$

Thus, the problem is reduced to generating trajectories whose feature expectations are similar to those of the expert. Abbeel and Ng, 2004 proposed an iterative projection-based method to solve it. In this paper, we propose to use DQN in the RL step of their algorithm. The details of our method are given in Algorithm 2.

Algorithm 2 Projection-based IRL using DQN

```
 1: Randomly initialize the parameters of the DQN and some policy $\pi^{(0)}$.
 2: Compute or approximate feature expectations $\mu^{(0)}$.
 3: Initialize $\mathbf{w}^{(1)} = \mu_E - \mu^{(0)}$, $\bar{\mu}^{(0)} = \mu^{(0)}$ and $i = 2$.
 4: while $\|\mathbf{w}\|_2 > \epsilon$ do
 5:     Let $\Delta\mu := \mu^{(i-1)} - \bar{\mu}^{(i-2)}$
 6:     Set $\bar{\mu}^{(i-1)} = \frac{\Delta\mu^T(\mu_E - \bar{\mu}^{(i-2)})}{\Delta\mu^T \Delta\mu} \Delta\mu$                ▷ Projection
 7:     Set $\mathbf{w}^{(i)} = \mu_E - \bar{\mu}^{(i-1)}$
 8:     Train the DQN using $R = \mathbf{w}^{(i)} \cdot \phi$ and obtain policy $\pi^{(i)}$      ▷ RL step
 9:     Compute or estimate features expectations $\mu^{(i)}$ for $\pi^{(i)}$
10:     Set $i \leftarrow i + 1$
```

3.3 Evaluation

In this section, we present the evaluation results of the proposed approach. We considered the driving scenario in a highway and implemented a simulator for collecting expert trajectories and testing. The simulating environment was programmed in Python. The user interface is shown in Figure 3.2 on the right side. The red car is the agent being trained to drive. The dynamics of this car are implemented based on the single track model (LaValle, 2006) with three degrees of freedom. The Deep Q-network architecture used in our approach is shown in Figure 3.1. It consists of an input layer of features, 2 fully connected hidden layers with 160 units each and rectified linear units (ReLUs), followed by a fully connected output layer to the action values.

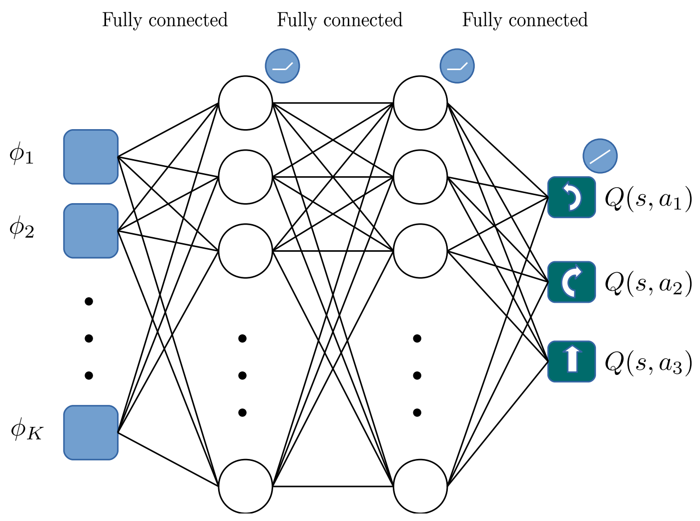


Figure 3.1: The architecture of the proposed Deep Q-Network. The input is the set of features and the output layer consists of three possible actions to steer left, steer right or not steer.

In these experiments we used 13 sensors. The sensors had a maximum sensing radius equal to 64% of the environment length, discretized to 16 bins of equal size. Each feature ϕ_k, indicated whether or not there was an obstacle in the interval of each bin. If the sensor was not sensing any obstacles, the maximum possible reading distance was assigned to it. Therefore, we had a total of 208 binary features which gave rise to 2^{208} possible states.

In the driving experiments by Abbeel and Ng, 2004, the car could only have discrete transitions to the lane on its left or right. However, in our experiments, we allowed steering with three different angles $(0, \frac{\pi}{12}, -\frac{\pi}{12})$ giving rise to more realistic, continuous transitions. For simplification in our experiments we set the acceleration to zero. The highway in our experiments had 3 lanes and at most two other cars could appear in front of the agent. For training, we collected 90 expert demonstrations from this setup.

The algorithm achieved satisfactory results after only 6 IRL iterations with 3000 inner loop iterations each. Since we did not have access to the true reward function of driving, we evaluated our proposed algorithm in the following ways:

a. **Analyzing the extracted weights:** The extracted weights for features from 7 sensors are plotted in Figure 3.2. The color-code guide of each sensor is shown in the upper right corner. As shown in this figure, there is a nonlinear relationship between readings of the same sensor and their extracted weights. This means that if the sensor readings had not been discretized into binary features, the algorithm would not have been able to capture the weights correctly. We have represented the maximum weight of features for each sensor

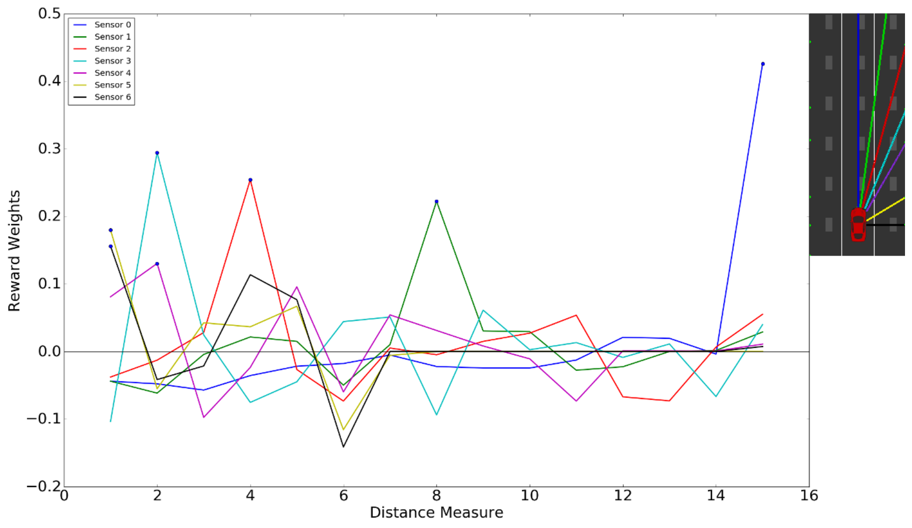


Figure 3.2: Weights extracted using the IRL algorithm plotted for features of 7 sensors. Each sensor has been assigned a color which is shown in the upper right corner.

		Sensor						
		0	1	2	3	4	5	6
$\lvert\hat{\mu}_E - \hat{\mu}_A\rvert$	1	0.000	0.000	0.000	0.004	0.158	0.005	0.011
	2	0.000	0.001	0.005	0.136	0.209	0.110	0.167
	3	0.001	0.004	0.006	0.173	0.079	0.126	0.085
	4	0.003	0.004	0.016	0.116	0.121	0.158	0.026
	5	0.002	0.016	0.094	0.096	0.004	0.039	0.030
	6	0.001	0.041	0.095	0.000	0.176	0.045	0.038
	7	0.001	0.039	0.057	0.046	0.017	0.000	0.000
	8	0.006	0.075	0.002	0.156	0.016	0.000	0.000

Table 3.1: The absolute differences between the trained and expert mean feature expectations. Each column refers to one of the first 7 sensors and each row refers to one of the first 8 distance bins.

with blue dots. As one can see, for sensor 0 (blue) that is pointing straight in front of the car, the higher the reading is, the larger the weight is. This means, the agent learns to keep maximum distance from obstacles (cars) in front of them. In the optimal state, where the car stays as far away from all obstacles and walls, the sensor readings form an ellipse. Therefore, as the angle of the sensor from the vertical axis increases, the reward weights peak at a smaller distance.

Another notable observation is that sensor 6 (side sensor), gets the highest weight when reading the smallest distance from the obstacles. The explanation is that at this distance, the car is placed next to a highway wall. This has been the expert's preference in the demonstrations. This reward weight immediately drops when the reading is increased, which perfectly explains that staying in the lanes is preferable to driving between the lanes. The same can be observed for most of the other sensors. Note that some of the distances were never read by the sensors during the experiments and the extracted weights for these cases is 0. For example, sensors 5 and 6 never sense obstacles in distances larger than the width of the highway.

b. Comparing feature expectation values of the expert to the trained agent: We trained the DQN using the rewards computed by the final weights. Then, we let the agent drive for several scenarios, acting according to the policy of the trained network. The mean difference over feature expectations in these scenarios was computed. Part of these values are presented in Table 3.1.

c. Evaluating using classical motion planning objectives: Classical motion planning methods are evaluated based on their performance in obstacle avoidance, jerk optimality, driving in the lanes, etc.. In our experiments,

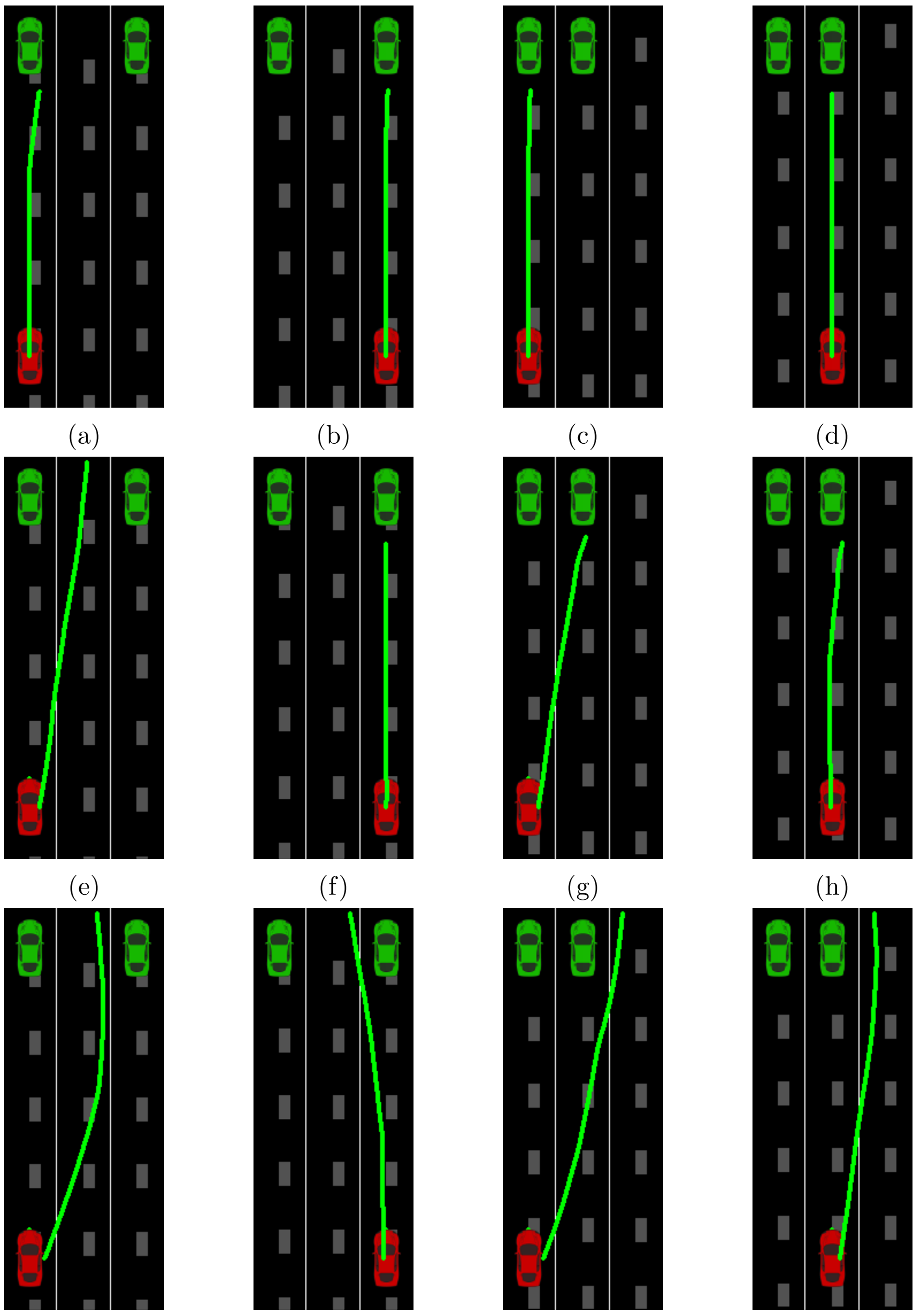


Figure 3.3: The agent's motion when facing four particular scenarios during training. The top row depicts the planned motion after the first IRL iteration (with 3000 DQN inner iterations), the middle row after four and the bottom row after six IRL iterations.

similar to the expert, the agent avoided the walls and obstacles in 100% of the scenarios, while maintaining their position in the lane except during obstacle avoidance. Jerk values were also found to be close to the ones of the expert. Figure 3.3 demonstrates the agent's planned motion based on the rewards extracted at different stages of the training phase.

3.4 Conclusion

In this paper we proposed using Deep Q-Networks as the refinement step in Inverse Reinforcement Learning approaches. This enabled us to extract the rewards in scenarios with large state spaces such as driving, given expert demonstrations. The aim of this work was to extend the general approach to IRL. Exploring more advanced methods like Maximum Entropy IRL and the support for nonlinear reward functions is an interesting future direction.

Chapter 4

Incremental Semi-Supervised Learning from Streams

Anybody can become angry — that is easy, but to be angry with the right person and to the right degree and at the right time and for the right purpose, and in the right way — that is not within everybody's power and is not easy.

– Aristotle

This chapter is based on the paper by Chiotellis* et al., 2018. The Label Propagation (LP) algorithm, first introduced by Zhu and Ghahramani, 2002 is a semi-supervised method used in transductive learning scenarios, where all data are available already in the beginning. In this work, we present a novel extension of the LP algorithm for applications where data samples are observed sequentially – as is the case in autonomous driving. Specifically, our “Incremental Label Propagation” algorithm efficiently approximates the so called harmonic solution on a nearest-neighbor graph that is regularly updated by new labeled and unlabeled nodes. We achieve this by reformulating the original algorithm based on an active set of nodes and by introducing a threshold to decide whether the label of a given node should be updated or not. Our method can also deal with graphs that are not fully connected, and we give a formal convergence proof for this general case. In experiments on the challenging KITTI benchmark data stream, we show superior performance in terms of both test accuracy and number of required training labels compared to state-of-the-art online learning methods.

4.1 Introduction

Most standard learning approaches used for classification tasks in robotics have the drawback that they either require a large amount of hand-labeled training data, or they are hardly adaptive to newly observed data samples. In particular, when considering an object classification problem from a given stream of training data, there is often just not enough ground truth information available to train a complex model such as a deep neural network. And even if there were sufficient training data, it is very difficult to efficiently update the classifier on newly arriving data from the stream, be it labeled or unlabeled.

Therefore, in this paper, we propose a learning algorithm for classification that is both semi-supervised and incremental, i.e. it can perform very fast model updates for new data samples. Furthermore, and in contrast to other semi-supervised approaches like transductive SVMs (V. N. Vapnik, 1995) or more recent deep learning approaches (Häusser et al., 2017; Kingma et al., 2014; Sajjadi et al., 2016), we can more flexibly and more directly trigger the learning process from a single sample instead of using batches of data with a fixed size. Our approach is based on the idea that newly observed data samples – both labeled and unlabeled ones – only have a *local* influence on the class predictions of the given unlabeled graph nodes. Thus, it is actually not necessary to recompute LP from scratch every time a sample arrives. In our formulation, this notion of locality is guided by a threshold ϑ, by which the trade-off between efficiency and accuracy is managed. In practice, however, it turns out that a relatively small area of influence (i.e. a large value of ϑ) already results in a very good performance while reducing the number of propagation iterations significantly.

To summarize, our key contributions are:

- A novel incremental semi-supervised learning method, we call "Incremental Label Propagation", where the area of influence of the algorithm can be easily tuned with the one hyper-parameter ϑ.
- A proof of convergence of the Label Propagation algorithm for partially connected graphs.
- An empirical evaluation of our algorithm on a challenging benchmark data stream (See Fig. 4.1), showing the effectiveness of exploiting unlabeled data for stream-based classification.

4.2 Related Work

The Label Propagation (LP) algorithm (Zhu and Ghahramani, 2002) has been used in several works for semi-supervised learning tasks. Chapelle et al., 2006 show that the LP algorithm is equivalent to the well known Jacobi algorithm for solving sparse linear systems. In fact, LP solves the problem of inverting the matrix Δ_{UU}, i.e. the submatrix of the graph Laplacian corresponding to the unlabeled nodes. As introduced by Zhu et al., 2003, the resulting solution is the so called harmonic solution on the graph. Zhu and Ghahramani, 2002 prove convergence of their algorithm for fully connected graphs, but we also give a convergence proof on partially connected graphs, where each connected component contains at least one labeled node.

In the literature, one can find several ideas for efficiently approximating the harmonic solution for a large non-growing graph. The algorithm of Ganu and Kveton, 2013 computes the harmonic solution on a subgraph of nodes for which the label can be predicted with high certainty. However, for the algorithm to be fast, the subgraph must be small, which means that only the labels of a small subset of nodes will be predicted. Delalleau et al., 2005 suggest to compute the harmonic solution on a subset of unlabeled nodes $\mathcal{S}$, where the label vectors of the nodes outside of $\mathcal{S}$ are set equal to the weighted average of their labeled neighbors and the neighbors in $\mathcal{S}$. While this is more efficient, no theoretical analysis is given on how much the solution differs from the exact harmonic solution on the full graph. Moreover, it is not clear how to get from the harmonic function approximation on a fixed graph to the approximation on an enlarged graph without computing everything from scratch.

Delalleau et al., 2005 derive an inductive formula to assign an optimal label to a new point given the harmonic solution on the old points. However, optimality only holds under the assumption that the label vectors of old points cannot change. If many unlabeled or a few labeled points arrive, the new information is not utilized to update the labels of previously classified points. In the algorithm of Valko et al., 2010, new incoming points are first assigned to one of k clusters using the doubling algorithm for incremental k-center clustering (Charikar et al., 2004). Then, each new point is replaced by its cluster center and the harmonic solution is computed on the cluster centers. Therefore, only a $k \times k$ instead of a $n \times n$ matrix has to be inverted to get the exact harmonic solution. Moreover, only the cluster centers and the cluster sizes have to be stored instead of all the data. However, the number of clusters k, that shall represent the data points, must be fixed beforehand. Also, once a point is assigned to a cluster, its original feature vector is discarded, so its position inside the cluster is not taken into account any more.

Zhu, 2005 also uses the idea of clustering to approximate the harmonic solution on large graphs. He computes the harmonic solution on a backbone graph consisting of mixture components, the so called harmonic mixture solution. An advantage of this approach is that a generative mixture model can naturally handle unseen points. Although a prediction for the label of a new point is easy to obtain, new points do not influence the model, except if the whole model is retrained. This is problematic if there are only a few points available in the beginning. Moreover, the problem of determining the number of clusters remains. The authors show that for a good approximation of the harmonic solution, the number of mixture components must be above a certain threshold that is data-dependent. Thus, the determination of a good number of clusters that keeps the effort of finding the harmonic mixture solution low, but still yields an acceptable approximation of the harmonic solution remains an open question.

Therefore, in this work, we do not reduce the data to a backbone graph but present the “Incremental Label Propagation” algorithm that allows us to approximate the harmonic solution on the full graph consisting of all nodes efficiently, in the context of a permanently growing graph.

4.3 Incremental Label Propagation

4.3.1 Reviewing Offline Label Propagation

We denote a partially labeled data set $\mathcal{D}$ of size n as the union of two subsets: the set $\mathcal{L}$ with l labeled data points $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)$ where $\mathbf{x}_i \in \mathbb{R}^d$ are feature vectors and $y_i \in \{1, \ldots, C\}$ are class labels, and the set $\mathcal{U}$ with u unlabeled points $\mathbf{x}_{l+1}, \ldots, \mathbf{x}_{l+u}$, i.e. $n = l + u$. Our aim is to infer class labels for unlabeled data points from the labeled ones. To do this, we compute edge weights $w_{i,j}$ based on a given Mahalanobis matrix M and build a k-nn graph $\mathcal{G}$ with edge weights

$$w_{i,j} = \begin{cases} \exp(-(\mathbf{x}_i - \mathbf{x}_j)^T M (\mathbf{x}_i - \mathbf{x}_j)) & \text{if } j \in \mathcal{N}(i) \\ 0 & \text{otherwise,} \end{cases} \tag{4.1}$$

where by $j \in \mathcal{N}(i)$ we denote that j is a neighbor of i. In particular, we find the k_l labeled and k_u unlabeled nearest neighbors of i. We define the *weight*

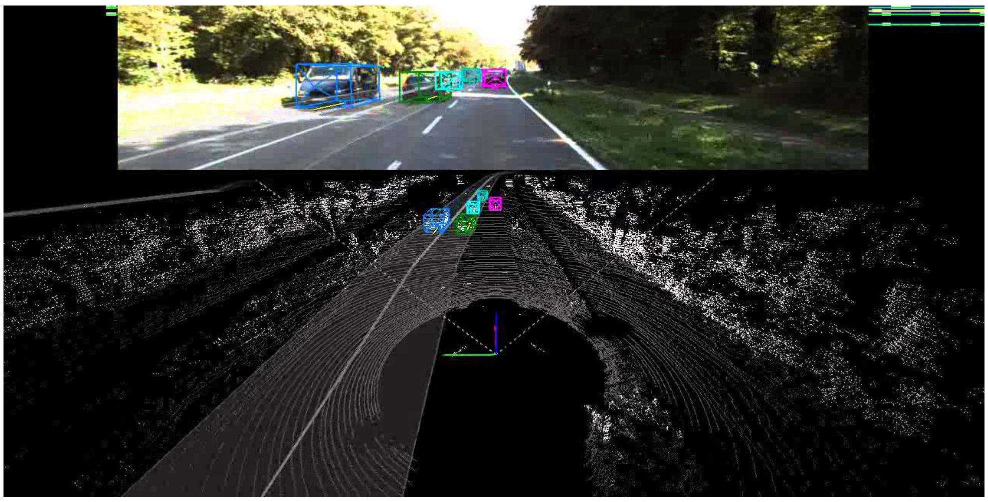


Figure 4.1: An example image (top) and the corresponding reconstructed 3D environment (bottom) with found objects from the KITTI Vision Benchmark Suite (Geiger et al., 2012).

matrix of all $w_{i,j}$ with $W \in \mathbb{R}^{n \times n}$, the diagonal matrix $D = \text{diag}(d_1, \dots, d_n)$ where $d_i = \sum_{j=1}^{n} w_{i,j}$, and the *transition matrix* $P = D^{-1}W$. Labels are represented in a *label matrix* $F \in \mathbb{R}^{n \times C}$, where ideally entry $f_{k,c} = 1$ if $\mathbf{x}_k$ has label $y_k = c$ and $f_{k,c} = 0$ otherwise. From the convention that all labeled points appear first and all unlabeled ones afterwards, it follows that P consists of four blocks:

$$P = \begin{pmatrix} P_{LL} & P_{LU} \\ P_{UL} & P_{UU} \end{pmatrix}, \tag{4.2}$$

where the subscripts of the blocks indicate transitions between labeled, unlabeled and mixed point pairs. Similarly, F consists of a labeled block F_L - the one-hot encoding of the true labels - and an unlabeled block F_U which initially contains zero vectors:

$$F = \begin{pmatrix} F_L \\ F_U \end{pmatrix}. \tag{4.3}$$

With this notation, the LP algorithm, introduced by Zhu and Ghahramani, 2002, can be formalized in two steps that are repeated until convergence. In the first step, a new label matrix F^{new} is computed by propagating the given labels according to the transitions, i.e. $F^{new} \leftarrow PF^{old}$. In the second step, the labels of the labeled samples are reset to those from the ground truth, i.e. $F_L^{new} \leftarrow F_L$. A possible convergence criterion is whether the norm $||F_U^{new} - F_U^{old}||_\infty$ drops under a given threshold.

4.3.2 Convergence on Partially Connected Graphs

Zhu and Ghahramani, 2002 prove the convergence of the LP algorithm but their proof (and the equivalent proof of Zhu, 2005) only applies if there is a $\gamma < 1$ s.th. $\forall i \in \{1, ..., u\}: \sum_{j=1}^{u} (P_{UU})_{i,j} \leq \gamma$, which means that each unlabeled node has to be connected to a labeled node by an edge of weight greater than 0. This assumption is of course true for fully connected graphs with positive weights. We will show here convergence on not fully connected graphs. First we assume that each connected component of nodes contains at least one labeled node. The LP algorithm can be written as

$$\begin{aligned} F_U^{t+1} &= P_{UU} F_U^t + P_{UL} F_L^t \,, \\ F_L^{t+1} &= F_L^0 \qquad \forall t \,. \end{aligned} \tag{4.4}$$

Obviously F_L stays constant. To show the convergence of F_U, we show the convergence of the columns of F_U. Let f_U be the j-th column of F_U and f_L the j-th column of F_L, $j \in \{1, \ldots, C\}$. We use the Banach fixed-point theorem to show that the sequence defined by $f_U^{t+1} = P_{UU} f_U^t + P_{UL} f_L$ converges independently of the starting point f_U^0 to a fixed point. Let us define $T : [0,1]^u \to \mathbb{R}^u$ as $T(x) = P_{UU} x + P_{UL} f_L$. It is easy to show that $T([0,1]^u) \subseteq [0,1]^u$ since P and F are both row stochastic. Thus, we need to prove the following

Theorem 1. *The mapping $T : x \mapsto P_{UU} x + P_{UL} f_L$ is a contraction.*

Proof. As a first step, we show that $\rho(P_{UU}) = \max_i |\lambda_i| < 1$ where λ_i are the eigenvalues of P_{UU}. Let λ be an eigenvalue of P_{UU} and $v \in \mathbb{R}^u$ a corresponding eigenvector. W.l.o.g. we can assume that

$$\|v\|_\infty = 1 \;\Rightarrow\; |v_j| \leq 1 \quad \forall j. \tag{4.5}$$

Using the definition of P, one can show that $\|P_{UU} v\|_\infty \leq 1$ and therefore it must hold that $|\lambda| \leq 1$ because for $|\lambda| > 1$ we would get the contradiction $\|P_{UU} v\|_\infty = |\lambda| \|v\|_\infty > \|v\|_\infty = 1$. It remains to show that $|\lambda| \neq 1$: Assume that $|\lambda| = 1$. We define the set $\mathcal{K} = \{k \in \{1, ..., u\} : |v_k| = 1\}$ that is not empty since $\|v\|_\infty = 1$. With a short proof one can show that

$$\forall\, i \in \mathcal{K},\; j \in \{1, ..., u\} \setminus \mathcal{K} :\; w_{i+l, j+l} = 0. \tag{4.6}$$

Let us define the set $\tilde{\mathcal{K}} = \{j \in \{l+1, \ldots, l+u\} \mid j - l \in \mathcal{K}\}$. $\tilde{\mathcal{K}}$ and $\mathcal{U} \setminus \tilde{\mathcal{K}}$ are separate connected components of unlabeled nodes and each of them contains at least one labeled node by assumption. It follows that

$$\exists r \in \tilde{\mathcal{K}}, s \in \mathcal{L}, \text{ s.th. } w_{r,s} > 0 \Rightarrow \frac{\sum_{j=1}^{u} (W_{UU})_{r-l,j}}{\sum_{k=1}^{n} w_{r,k}} < 1 \tag{4.7}$$

With that we get

$$\begin{aligned}1 =&|v_{r-l}| = |\lambda||v_{r-l}| = |(\lambda v)_{r-l}| = |\left(P_{UU}v\right)_{r-l}| = |\left(\left(D^{-1}W\right)_{UU}v\right)_{r-l}| \\ =&|\sum_{j=1}^{u}\frac{w_{r,j+l}}{\sum_{k=1}^{n}w_{r,k}}v_j| \leq \frac{1}{\sum_{k=1}^{n}w_{r,k}}\sum_{j=1}^{u}w_{r,j+l}|v_j| \leq \frac{\sum_{j=1}^{u}w_{r,j+l}}{\sum_{k=1}^{n}w_{r,k}} < 1\,.\end{aligned} \tag{4.8}$$

Because of this contradiction, the assumption $|\lambda| = 1$ must have been wrong and therefore we get $\rho\left(P_{UU}\right) < 1$. It immediately follows that there is an $\epsilon > 0$ s.th. $\rho\left(P_{UU}\right) + \epsilon < 1$. Now, we define a special vector norm $||.||_\omega$ and induced matrix norm $|||.|||_\omega$ s.th. $|||P_{UU}|||_\omega \leq \rho\left(P_{UU}\right) + \epsilon$. With that it follows that T is a contraction (with Lipschitz constant $\rho\left(P_{UU}\right) + \epsilon$) in the normed space $\left(\mathbb{R}^u, \|.\|_\omega\right)$. For details, see A.

$\square$

So far we assumed that each connected component contains at least one labeled node. We can extend the proof also for graphs that do not satisfy this assumption. The nodes in connected components that do not contain a labeled node (isolated nodes) keep their label vector equal to the zero vector throughout the algorithm. The steps performed then are equivalent to removing the isolated nodes from the graph, performing LP on the remaining nodes and assigning all isolated nodes a zero label vector. And for graphs without isolated nodes, we have proven convergence already.

4.3.3 Incremental Label Propagation

Our main idea is that when a new sample arrives, many computation steps can be saved by propagating labels only locally and stopping the propagation process if no significant change of labels is achieved, similar to a diffusion process. The latter is formulated by introducing a *threshold* ϑ and a decision function that returns only indices of data samples, for which the label change is larger than ϑ according to some norm (we use the ℓ_1-norm). Another difference to standard LP is that we formulate the algorithm based on a set $\mathcal{Q}$ of candidate nodes to propagate labels to and a set $\mathcal{A}$ of nodes which actually get updated. This leads to a simple formulation of the stopping criterion, since label propagation is stopped when $\mathcal{Q}$ becomes empty[1].

The pseudocode for Incremental Label Propagation is given in Alg. 3. We denote by $F_{U(i)}$ the i-th row of matrix F_U, namely the estimated label for point i. Here, we give details for the individual steps. First, we compute the edge weights $\mathbf{w}_{n+1} = (w_{n+1,1}, \ldots, w_{n+1,n})$ between the new data sample $\mathbf{x}_{n+1}$

[1]Note that the use of the sets $\mathcal{Q}$ and $\mathcal{A}$ alone does not turn LP into an incremental algorithm. In fact, for $\vartheta = 0$, ILP with sets is equivalent to offline LP for the observed samples at the time. The insight is, that for any point, only its reverse neighbors can change their label in every LP iteration, i.e. the graph can be equally processed using a region-growing strategy.

Algorithm 3 Incremental Label Propagation (ILP)

```
Require:
    data set $\mathcal{D}$ of size $n$; new data sample $\mathbf{x}_{n+1}$;
    Mahalanobis matrix $M$; graph $\mathcal{G}$; labels $F$; transitions $P$;
    number of neighbors $k_l$, $k_u$
    thresholds $\vartheta$, $T_{max}$
Ensure: updated label matrix $F_U$;

 1: $\mathbf{w}_{n+1} \leftarrow$ ComputeWeights($\mathcal{D}, \mathbf{x}_{n+1}, M, k_l, k_u$)    ▷ Eq. (4.1)
 2: $P \leftarrow$ UpdateTransition($W, \mathbf{w}_{n+1}$ )
 3: $\mathbf{f}_{n+1} \leftarrow P_{n+1,1:n}F$                                  ▷ Estimation
 4: $\mathcal{Q} \leftarrow \{k \mid k \in \mathcal{N}^T(n+1) \cap \mathcal{U}\}$       ▷ Initialize candidates
 5: $\tilde{F}_U \leftarrow F_U$
 6: for $k \in \mathcal{Q}$ do
 7:     $\tilde{F}_{U(k)} \leftarrow P_{UL(k)}F_L + P_{UU(k)}F_U$
 8: $t_i \leftarrow 0$
 9: while $\mathcal{Q} \neq \emptyset$ and $t_i < T_{max}$ do
10:     $\mathcal{A}, F_U \leftarrow$ FilterAndUpdate($\mathcal{Q}, \tilde{F}_U, F_U, \vartheta$)
11:     $\mathcal{Q}, \tilde{F}_U \leftarrow$ GetNextCandidates($\mathcal{A}, \tilde{F}_U, P_{UU}$)
12:     $t_i \leftarrow t_i + 1$

 1: function FilterAndUpdate($\mathcal{Q}, \tilde{F}_U, F_U, \vartheta$)
 2:     $\mathcal{A} \leftarrow \emptyset$
 3:     for $i \in \mathcal{Q}$ do
 4:         $\Delta\mathbf{f}_i \leftarrow \tilde{F}_{U(i)} - F_{U(i)}$
 5:         if $|\Delta\mathbf{f}_i| > \vartheta$ then                           ▷ Filter
 6:             $F_{U(i)} \leftarrow \tilde{F}_{U(i)}$                           ▷ Update
 7:             $\mathcal{A} \leftarrow \mathcal{A} \cup (i, \Delta\mathbf{f}_i)$
 8:     return $\mathcal{A}, F_U$

 1: function GetNextCandidates($\mathcal{A}, \tilde{F}_U, P_{UU}$)
 2:     $\mathcal{Q} \leftarrow \emptyset$
 3:     for $(j, \Delta\mathbf{f}_j) \in \mathcal{A}$ do
 4:         for $k \in \{i \mid i \in \mathcal{N}^T(j) \cap \mathcal{U}\}$ do
 5:             $\tilde{F}_{U(k)} \leftarrow \tilde{F}_{U(k)} + P_{UU(k,j)}\Delta\mathbf{f}_j$
 6:             $\mathcal{Q} \leftarrow \mathcal{Q} \cup k$
 7:     return $\mathcal{Q}, \tilde{F}_U$
```

and the observed samples $\mathbf{x}_1, \dots, \mathbf{x}_n$ according to (4.1). This we use to update the matrices W, D, and P, which then have $n+1$ rows and columns. With P, we estimate a label vector $\mathbf{f}_{n+1}$ for the new node as a weighted average of the labels of its labeled and unlabeled neighbors (line 3). After that, we initialize the candidates set $\mathcal{Q}$, which at first contains only the nodes from the unlabeled set $\mathcal{U}$, that have the new node as a nearest neighbor. We denote these reverse nearest neighbors of sample $\mathbf{x}_{n+1}$ as $\mathcal{N}^T(n+1)$. We also initialize the *tentative* label updates matrix $\tilde{F}_U$ (line 5). In practice we only need to store the rows of $\tilde{F}_U$ that get updated during the algorithm. In lines 6 to 7, the label propagation starts by computing the tentative labels of the reverse nearest neighbors of the new node. This is the only step where information from the transition submatrix P_{UL} is used.

In lines 9 to 12, the main idea of the ILP algorithm is presented. From the set $\mathcal{Q}$ of candidates that received a tentative label update, we identify the ones whose update is considered *significant*. In particular, the ℓ_1-distance between their previous and tentative label has to be larger than ϑ. Note that ϑ implicitly defines the area of influence of the algorithm: for $\vartheta = 0$ all changes are considered significant and we have offline LP with sets starting from the new node. Any other value for ϑ constrains the range of the region growing process, leading to a more local effect. We apply the significant tentative updates by storing the new labels in the original matrix F_U (line 6 of function *FilterAndUpdate*). In set $\mathcal{A}$ we store the indices of the updated nodes along with their label difference $\Delta\mathbf{f}$, so we can continue the propagation with their reverse neighbors in the next iteration.
The ILP algorithm presented in Alg. 3 for new unlabeled points is almost identical for new labeled points with the only difference that $\mathbf{f}_{n+1}$ does not need to be estimated.

4.4 Runtime Analysis

We consider the insertion of a single node. We denote by k_l and k_u the number of labeled and unlabeled neighbors of each node in the graph respectively. The main burden of the computation of the node's edge weights lies in the distance computation between the new node and the existing nodes in the graph which in turn depends on the choice of nearest neighbor algorithm. The trivial computation takes $O(n_t d)$ where d is the data dimensionality and $n_t = l_t + u_t$ is the number of nodes at iteration t. This can be reduced to logarithmic time using an efficient data structure for online node insertion and nearest neighbor search, such as Ball Trees as described in the online insertion algorithm of Omohundro, 1989. We denote this runtime as O_{knn}.

A major advantage of using a k-nn graph as opposed to an ε-nn graph in an incremental setting is that updating the old entries of W and P is independent of the number of nodes n_t. Using a fixed capacity heap for storing the nearest neighbors of each node, a node's labeled neighbors can be updated in $\log(k_l)$ and equivalently unlabeled neighbors in $\log(k_u)$ time, as only one neighbor might get pushed out of the heap to be replaced by the new node. Therefore the weight matrix can be updated in $O((k_l+k_u)\log(\max(k_l,k_u)))$. The update of P is done in $O((k_l+k_u)^2)$ since only the rows of nodes connected to the new point must be updated.

The estimation of the new label takes $O((k_l + k_u)C)$ time. Accessing the neighbors and reverse neighbors of points can be done in constant time using a hash map at the cost of $O(n)$ auxiliary space. Each inner iteration in the functions *GetNextCandidates* and *FilterAndUpdate* takes $O(C)$ time, therefore the runtime depends mainly on how the sizes of $\mathcal{Q}$ and $\mathcal{A}$ evolve. The size of $\mathcal{A}$ is always bounded by the size of $\mathcal{Q}$ but the size of $\mathcal{Q}$ depends on the number of reverse neighbors, which is not deterministic. In practice though this is close to k_u[2]. With this assumption and with $\vartheta = 0$, $\mathcal{Q}$ and $\mathcal{A}$ grow as $k_u, k_u^2, k_u^3, \ldots$, etc., as we consider all label updates significant. Therefore, the number of iterations until all unlabeled points have been reached by label propagation is bounded by $\log_{k_u}(u_t)$. The total number of operations in the main loop then is

$$C(k_u + k_u^2 + \ldots + k_u^{\log_{k_u} u_t}) = C\frac{u_t - 1}{k_u - 1} = O(u_t C \frac{1}{k_u - 1}). \tag{4.9}$$

And thus the total runtime of incremental label propagation for a node arriving at time t is

$$O_{knn} + O((k_l + k_u)^2 + (k_l + k_u)C + u_t C \frac{1}{k_u - 1}). \tag{4.10}$$

4.5 Experiments

Our implementation and code for all experiments is publicly available[3].

4.5.1 Evaluation and Setup

In this work, we use the standard Euclidean metric ($M = I$) to compute the edge weights (see Equation (4.1)). For evaluation we use several metrics: the ℓ_1 error, the 0-1 classification error and the cross-entropy between ground truth labels and the predictions F_U. We also compute the entropy of our predictions F_U, the number of label propagation iterations per new point and the wall-clock computation time per new point. We set $T_{max} = \log_{k_u} u_t$ if $k_u > 1$ and $T_{max} = 30$ if $k_u = 1$.

We first examine the influence of different hyperparameters and the performance of ILP against its fully supervised counterpart, using the MNIST dataset of handwritten digits (LeCun et al., 1998) that consists of 60000 training and 10000 test images of dimensions 28 by 28. We do not compute any features but use the raw pixel values normalized to lie in $[0, 1]$.

Due to lack of space, we present results with $k_u = k_l = 3$. However, we found the algorithm to be very robust with respect to values of k_u and k_l up to 19, except for the case of $k_l = 1$, where the final accuracy dropped by 5%.

[2]One can enforce the number of reverse neighbors to be equal to k_u by using a *mutual* instead of a regular k-nn graph.

[3]Code available at `https://github.com/johny-c/incremental-label-propagation`.

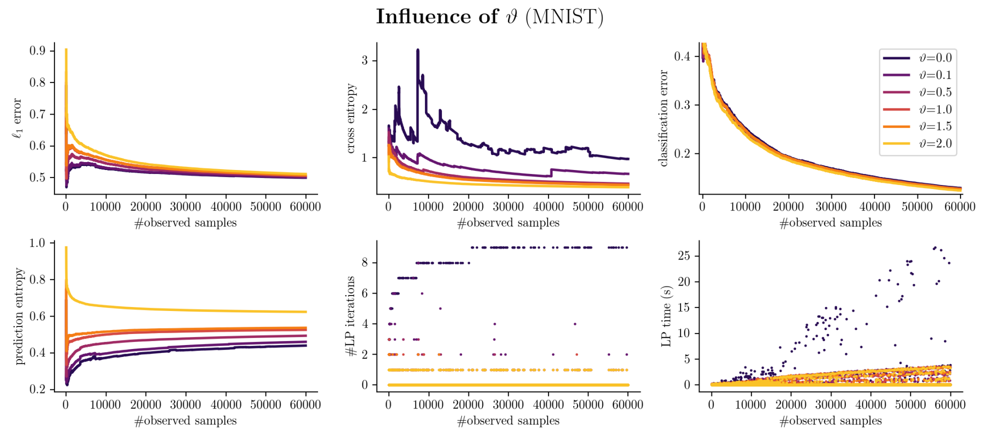


Figure 4.2: Illustration of how ϑ influences different metrics. **Top row**: ℓ_1 error, cross entropy and classification error (based on arg max) w.r.t. the predictions F_U. **Bottom row**: Entropy of the predictions, number of label propagation iterations and actual computation time after each new node.

4.5.2 Influence of ϑ

The ratio of observed labels is fixed to 5% and we set $k_l = 3$ and $k_u = 3$. In Fig. 4.2 we show how different metrics are affected by ϑ. As expected, the ℓ_1-error decreases as we decrease ϑ, namely as we move closer to offline LP. On the other hand, the number of iterations per observation and therefore the runtime decreases when we increase ϑ, as the algorithm is constrained to act more locally. For reference, with $\vartheta = 0$, the total runtime was 4162.40s, while with $\vartheta = 1.0$ only 3143.46s. We observe several interesting facts:

- The estimation error is robust to changes in ϑ, as the maximum likelihood of the correct predictions is mostly preserved.

- The entropy of the predicted label distributions behaves very similar to the ℓ_1-error making it a good candidate metric for self-evaluation or introspection during learning.

- From the cross-entropy evaluation, it seems that choosing a large enough ϑ has a regularizing effect: When ϑ is small, the cross-entropy is strongly oscillating, as every label update is trusted equally. When ϑ is larger, the "propagation region" is narrower and therefore cross-entropy is smoother. In fact for $\vartheta = 2$, where we only have label estimation and no propagation, the cross-entropy and often also the estimation error is minimized. However, when testing on a holdout set, we find that intermediate values of ϑ achieve the best accuracy. This suggests that a too small ϑ causes overfitting, and a ϑ that is too large causes underfitting.

4.5.3 Influence of number of observed labels

In this experiment we evaluate how the accuracy of the algorithm is affected by the number of observed labels (Fig. 4.3). We compare the test error when using just the labeled samples against the test error when also using the labels estimated by ILP for the unlabeled training samples. We use $k_l = 3, k_u = 3$ and $\vartheta = 0.3$. Predicting the labels of the test set amounts to computing $F_T = P_{TL}F_L + P_{TU}F_U$. Taking into account only the first summand $P_{TL}F_L$ is weighted k-nn querying with respect to the given labeled samples. In Table 4.1 we show that ILP is consistently better than weighted k-nn, demonstrating the utilization of the unlabeled samples.

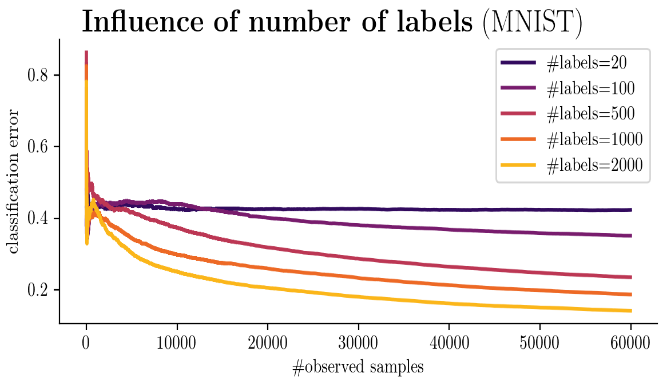


Figure 4.3: Estimation error on the observed unlabeled samples of MNIST for different number of observed labels in total.

#Labels	Est. error (%)	$k$nn error (%)	ILP error (%)
20	42.35	41.17	40.11
100	35.17	27.01	26.61
500	23.54	14.54	14.09
1000	18.73	11.27	10.68
2000	14.17	8.56	8.2

Table 4.1: Final estimation error on the unlabeled training set, $k$nn error and ILP error on the test set of MNIST.

4.5.4 Confusion Analysis

In continuation of the previous experiment, we trained incrementally on MNIST observing only 5% of the labels. In Fig. 4.4 we show the final confusion matrices of k-nn, where only labeled neighbors vote for the label of test nodes, and ILP, where the estimated labels of the unlabeled training nodes are also

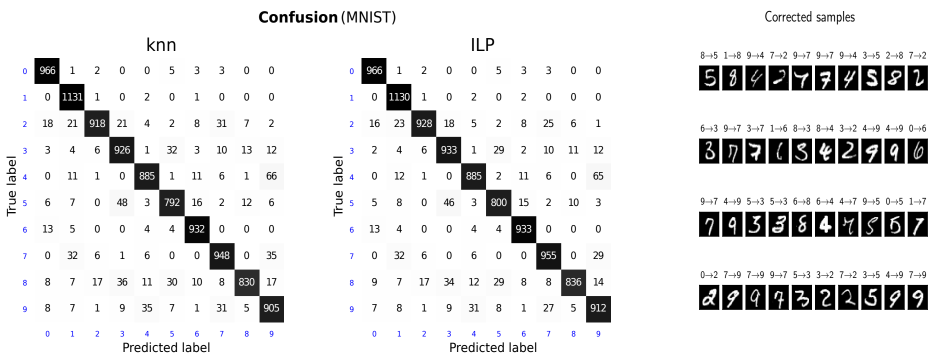


Figure 4.4: Confusion matrices for the test set of MNIST. **Left:** Predicting only with 3 labeled neighbors (test error=7.67%), **Middle:** Predicting with ILP, taking into account also the estimated labels of 3 unlabeled neighbors (test error=7.22%). **Right:** Some samples for which the label predictions were corrected by ILP.

considered. We present examples of images misclassified by k-nn that were correctly classified by ILP.

4.5.5 KITTI Benchmark

We further evaluate Incremental Label Propagation on the more challenging setting of a *stream* of data. For this experiment we use data from the KITTI benchmark[4]. We conduct the same experiment as Narr et al., 2016[5]. We concatenate 18 streams of segmented 3D point clouds from urban traffic environments (Geiger et al., 2012) to form one long stream. Each of the 25090 segments corresponds to a 3D bounding box containing points that represent a given object candidate. For each such candidate, a 60-dimensional feature vector was computed as proposed by Himmelsbach et al., 2009. These features consist of global characteristics such as box volume and mean intensity, as well as of distributions of local features such as scatterness or flatness. These features can be computed in real time. For the test data, each of the 18 subsets was split at a 2:1 ratio to obtain a stream of 16000 training samples and a set of 9,090 test samples. We used the 100 first training samples (50 labeled and 50 unlabeled) as a “burn-in” set to initialize the algorithm. In Fig. 4.6, we show the order of appearance of labeled and unlabeled samples from each class in the described training stream, as well as the behavior of ILP when only 10% of the labels are observed.

[4]http://www.cvlibs.net/datasets/kitti.

[5]Data were provided by the authors upon our request.

The methods mentioned by Narr et al., 2016 were evaluated on the test set after every 1000 observations and their final test error is presented in Table 4.2.

Method	Final Test error(%)
OMCGB (Saffari et al., 2010	13.00
ORF (Saffari et al., 2009)	13.70
MF (Lakshminarayanan et al., 2014)	10.00
ILP (5% labels)	10.75
ILP (10% labels)	10.18
ILP (20% labels)	**9.53**

Table 4.2: Final test error of different online learning methods for the described experiment on the KITTI benchmark.

We note that we do not compare against offline methods, as they are not relevant in a stream-based learning scenario. We fixed ϑ to 1.0 and ran the experiment for different ratios of given labels (see Figure 4.5). With only 5% of the labels, ILP already outperforms Online Random Forests (Saffari et al., 2009) and online multi-class Gradient Boost (Saffari et al., 2010) and achieves comparable accuracy to Mondrian Forests (Lakshminarayanan et al., 2014). Note that all other methods use all training labels. With 20% of the labels, we achieved a test error of 9.53%, which outperforms even the Mondrian Forest method.

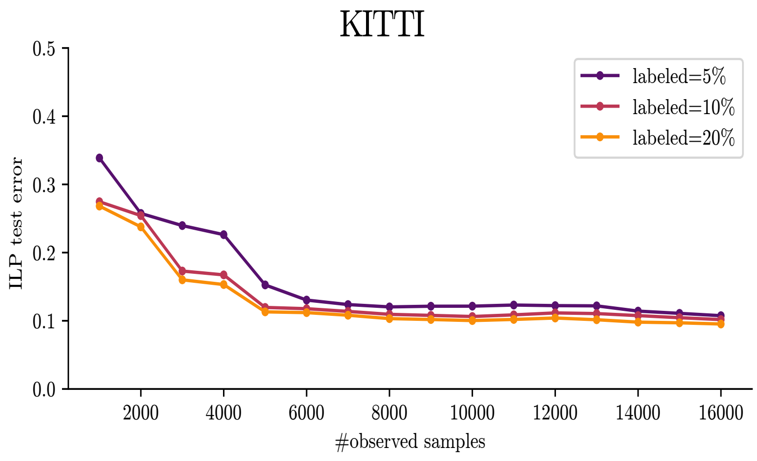


Figure 4.5: Test error of ILP over time for different ratios of labeled data in the stream.

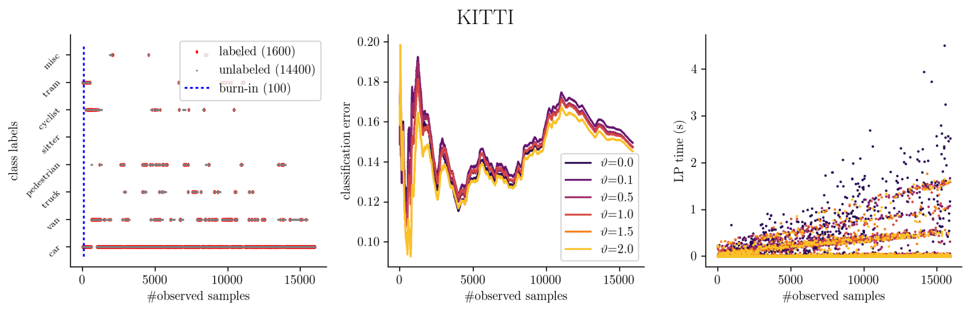


Figure 4.6: Results on the constructed stream from the KITTI dataset given 10% of labels (see text). **Left:** The order of appearance of labels in the stream, **Middle:** ILP error on the unlabeled training set, **Right:** Computation time for different values of ϑ.

4.6 Conclusion

In this paper we presented “Incremental Label Propagation”, an efficient incremental variant of the Label Propagation algorithm introduced by Zhu and Ghahramani, 2002 that is useful for object classification from sparsely labeled streams of data. We provided a proof of convergence of the original algorithm for partially connected graphs and gave an analysis for the runtime of our algorithm. With various experiments we investigated the influence of hyperparameters on the behavior of ILP, showed the utilization of the unlabeled samples over its purely supervised counterpart and demonstrated the performance improvement, even over fully supervised online learning methods on a challenging benchmark dataset.

Chapter 5

Effective Version Space Reduction for ConvNets

He who is different from me does not impoverish me - he enriches me. Our unity is constituted in something higher than ourselves - in Man... For no man seeks to hear his own echo, or to find his reflection in the glass.

– **Antoine de Saint-Exupéry**

This chapter is based on the paper by Liu et al., 2020. In active learning, sampling bias could pose a serious inconsistency problem and hinder the algorithm from finding the optimal hypothesis. However, many methods for neural networks are hypothesis space agnostic and do not address this problem. We examine active learning with convolutional neural networks through the principled lens of version space reduction. We identify the connection between two approaches – prior mass reduction and diameter reduction – and propose a new diameter-based querying method – the minimum Gibbs-vote disagreement. By estimating version space diameter and bias, we illustrate how the version space of neural networks evolves and examine the realizability assumption. With experiments on MNIST, Fashion-MNIST, SVHN and STL-10 datasets, we demonstrate that diameter reduction methods reduce the version space more effectively and perform better than prior mass reduction and other baselines, and that the Gibbs vote disagreement is on par with the best query method.

5.1 Introduction

Active learning is a supervised learning framework in which the learner is given access to a pool or stream of unlabeled samples and is allowed to selectively query labels from an oracle (e.g. , a human annotator). In each query round, the learner queries the labels of some unlabeled samples and trains on the augmented labeled set to obtain new classifiers. The goal is to learn a good classifier or *hypothesis* using as few labels as possible. This setting is relevant in many real-world problems, where labeled data are scarce or expensive to obtain, but unlabeled data are cheap and abundant.

Many active learning methods for neural networks rely on measures of the "informativeness" of a query, in the form of classifier uncertainty, margin (Ducoffe and Precioso, 2018; Joshi et al., 2009) or information gain (Gal et al., 2017; Houlsby et al., 2011; Kirsch et al., 2019). Other methods capture the informativeness by representativeness of the query set using geometry-based (Sener and Savarese, 2018) or discriminative methods (Gissin and Shalev-Shwartz, 2019). However, most of these methods ignore the notion of the hypothesis space and do not address the problem of sampling bias (Dasgupta, 2009), which plague many active learning methods. Without carefully handling this problem, an active learning algorithm is not guaranteed to be *consistent*, i.e. capable of finding the optimal classifier in the hypothesis space.

We consider the hypothesis space of convolutional neural networks (ConvNets) and study version space reduction methods. Version space reduction works by removing hypotheses that are inconsistent with the observed labels from a predefined hypothesis space and maintaining the consistent sub-space, the *version space*. A key condition called the *realizability assumption* is that the hypothesis space contains the classifier that provides the ground truth – if not, there are no guarantees that the best hypothesis will not be removed,

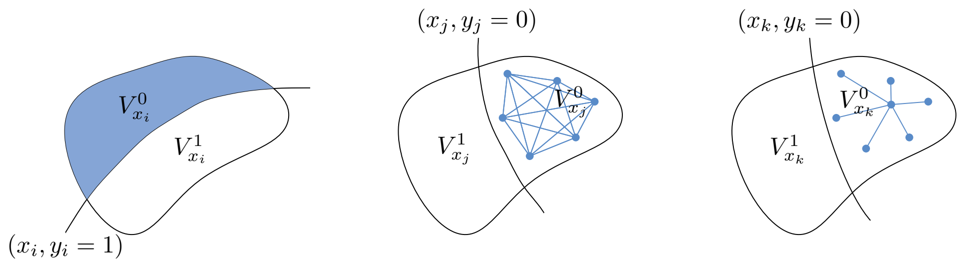


Figure 5.1: Version space reduction for binary classification. Upon observing the label of x, the current version space V is split into subspaces V_x^0 and V_x^1, one of which will be removed and the other remains. **Left**: Prior mass reduction methods remove approximately half of the mass. **Middle**: Diameter reduction methods, like pairwise disagreement, query a sample that lead to sub-spaces of small diameter. **Right**: Proposed method, the *Gibbs-vote disagreement*, measures diameter by the expected distance between random hypotheses and their majority vote.

because a hypothesis might make mistakes on the queried samples but perform well on the data distribution.

For neural networks, the realizability assumption may not hold for all cases. For instance, no neural networks can achieve arbitrarily small test error on some classification datasets. A workaround is to consider the effective labelings on a set of i.i.d. pool samples. To avoid the problem of an unreasonably large effective hypothesis space, as implied by the result of C. Zhang et al., 2017, we only consider the labelings achievable by training on unaltered samples and correct labels. We examine experimentally whether the realizability holds with this restriction and analyze its implications on version space reduction methods.

Prior mass reduction (Dasgupta, 2004; Golovin and Krause, 2010; Nguyen et al., 2013) and diameter reduction (Dasgupta, 2005; Tosh and Dasgupta, 2017) are two widely used version space reduction approaches. See Fig. 5.1 for illustration. However, prior mass reduction is not an appropriate objective for active learning (Tosh and Dasgupta, 2017) since any intermediate version spaces containing more than one hypothesis may still have a large diameter, i.e. large error rate in the worst-case scenario, despite having substantially reduced mass. We derive connections between prior mass and diameter reduction and introduce a new interpretation of diameter reduction as prior mass "reducibility reduction".

We propose a new diameter measure called the *Gibbs-vote disagreement*, which equals the expected distance between the random hypotheses and their majority vote classifier. We show its relation to a common diameter measure, the *pairwise disagreement*, and discuss under which situations the former may be advantageous. We show experimentally on four image classification datasets that diameter reduction methods perform better than all baselines and that prior mass reduction (Dasgupta, 2004; Golovin and Krause, 2010; Nguyen et al., 2013) and other baselines (Ducoffe and Precioso, 2018; Gal et al., 2017; Houlsby et al., 2011; Sener and Savarese, 2018) do not perform consistently better than random query and sometimes fail completely.

5.2 Related Work

A lot of research has been conducted to study the label complexity for active learning and optimality guarantees for greedy version space reduction. Hanneke, 2007 and Balcan et al., 2006 prove upper-bounds on the label complexity in the realizable and non-realizable cases, using a parameter called the disagreement coefficient. Tosh and Dasgupta, 2017 propose a diameter-based active learning algorithm and characterize its sample complexity using a parameter called the splitting index. Dasgupta, 2004 shows that a greedy strategy maximizing the worst-case prior mass reduction is approximately as good as the optimal strategy. Golovin and Krause, 2010 show that the prior mass reduction utility function is adaptive submodular and a greedy algorithm is guaranteed to obtain near-optimal solutions in the average-case scenario. Cuong et al., 2014 prove a worst-case optimality guarantee for pointwise submodular functions.

A variety of methods relying on the informativeness of a query have been proposed for neural networks. Gal et al., 2017 use the Monte Carlo dropout to approximate the mutual information between predictions and model posterior (Houlsby et al., 2011) in a Bayesian setting. Kirsch et al., 2019 extend the work of Gal et al., 2017 and Houlsby et al., 2011 to a batch query method. Ducoffe and Precioso, 2018 use adversarial attacks to generate samples close to the decision boundaries. Sener and Savarese, 2018 adopt a core-set approach to select representative samples for query. Gissin and Shalev-Shwartz, 2019 use a discriminative method to select samples such that the labeled and the unlabeled set are indistinguishable. Pinsler et al., 2019 formulate batch query as a sparse approximation to the expected complete data posterior of model parameters in a Bayesian setting. Beluch et al., 2018 show that ensemble methods consistently outperform the geometry-based method of Sener and Savarese, 2018 and the Monte Carlo dropout method of Gal and Ghahramani, 2016 and Gal et al., 2017.

5.3 Preliminaries

Let $\mathcal{X}$ be the input feature space and $\mathcal{Y}$ the label space. Let $\mathcal{H}$ be a hypothesis space of functions $h : \mathcal{X} \to \mathcal{Y}$ and assume a prior π over $\mathcal{H}$. A hypothesis randomly drawn from the prior is called a *Gibbs classifier*. Denote $S =$

$\{(x_i, y_i)\}_{i=1}^n$ a pool of i.i.d. samples from the data distribution P_{XY} and $Q \subseteq S$ the set of queried labeled samples. Define the version space V corresponding to Q as

$$V := \{h \in \mathcal{H} : h(x) = y,\ \forall (x, y) \in Q\}\,. \tag{5.1}$$

Denote the subset of V that is consistent with x being labeled as y as

$$V_x^y := \{h \in \mathcal{H} : h(x) = y,\ h \in V\}\,, \tag{5.2}$$

and the pseudo-metric induced by the marginal distribution P_X as

$$d(h, h') := \Pr_x\big(h(x) \neq h'(x)\big)\,. \tag{5.3}$$

The disagreement and agreement region are defined as

$$\mathrm{DIS}(V) := \big\{x \in \mathcal{X} : \exists h, h' \in V, h(x) \neq h'(x)\big\}\,, \tag{5.4}$$

$$\mathrm{AGR}(V) := \mathcal{X} \setminus \mathrm{DIS}(V). \tag{5.5}$$

5.4 Prior Mass Reduction

5.4.1 Gibbs Error

The Gibbs error (Nguyen et al., 2013) of an unlabeled sample x is the average-case relative prior mass reduction:

$$\mathrm{GE}(x|V) := \ \mathbb{E}_y\big[1 - \Pr_{h\sim\pi|_V}(h(x) = y)\big] \ = \ \mathbb{E}_y[1 - \pi|_V(V_x^y)]\,, \tag{5.6}$$

where $\pi|_V(h) = \pi(h)/\pi(V)$ is the conditional distribution of $\mathcal{H}$ restricted to V. Gibbs error measures the proportion of inconsistent hypotheses taking expectation over all possible labelings of x, achievable by hypotheses in the version space. A greedy strategy that considers maximizing the average-case absolute prior mass reduction in each query can equivalently select the unlabeled sample that maximizes the Gibbs error

$$\arg\max_x \mathrm{GE}(x|V). \tag{5.7}$$

Define the prior mass reduction utility function as

$$f(Q) := \ 1 - \Pr\left(\{h \in \mathcal{H} : h(x) = y,\ \forall (x, y) \in Q\}\right) = \ 1 - \pi(V). \tag{5.8}$$

The optimization problem in (5.7) can be written, up to a scaling factor, as

$$\arg\max_x \pi(V)\mathrm{GE}(x|V) = \arg\max_x \mathbb{E}_y[f(Q \cup \{(x, y)\}) - f(Q)] \tag{5.9}$$

$$= \arg\max_x \Delta_{\mathrm{avg}}(x|Q), \tag{5.10}$$

where the notation $\Delta_{\text{avg}}(x|Q)$ denotes the expected marginal gain of x in terms of prior mass reduction given the labeled samples in Q.

A closely related objective for active learning is the label entropy given x. It can be shown that the Gibbs error lower bounds the entropy. However, a greedy strategy that maximizes the entropy is not guaranteed to be near-optimal in the adaptive case (Cuong et al., 2014). Furthermore, empirically this criterion performs similarly or worse than the maximum Gibbs error. For the sake of simplicity, we do not consider this method in this paper.

5.4.2 Variation Ratio

The variation ratio of an unlabeled sample x is the worst-case relative prior mass reduction upon the reveal of its label:

$$\mathrm{VR}(x|V) := \min_y \left[1 - \mathrm{Pr}_{h\sim\pi|_V}(h(x)=y)\right] = \min_y \left[1 - \pi|_V(V_x^y)\right]. \tag{5.11}$$

It measures the proportion of inconsistent hypotheses considering the worst-case labeling of x and is a lower bound on the Gibbs error. A greedy strategy that considers maximizing the worst-case absolute prior mass reduction in each query selects the unlabeled sample that maximizes the variation ratio

$$\arg\max_x \mathrm{VR}(x|V), \tag{5.12}$$

which can be expressed in terms of the prior mass reduction utility function, up to a scaling factor, as

$$\begin{aligned}\arg\max_x \pi(V)\mathrm{VR}(x|V) &= \arg\max_x \min_y \left[f(Q\cup\{(x,y)\}) - f(Q)\right] &(5.13)\\ &= \arg\max_x \Delta_{\text{wc}}\left(x|Q\right), &(5.14)\end{aligned}$$

where the notation $\Delta_{\text{wc}}\left(x|Q\right)$ denotes the worst-case marginal gain of x in terms of prior mass reduction given the labeled samples in Q.

5.5 Diameter Reduction

5.5.1 Worst-Case Pairwise Disagreement

The size of the version space can be measured by the expected pairwise disagreement between hypotheses drawn from the conditional distribution:

$$\mathrm{PWD}(V) := \mathbb{E}_{h,h'\sim\pi|_V}\left[d(h,h')\right]. \tag{5.15}$$

It is the *average diameter* of the version space. A greedy strategy selects the unlabeled sample that minimizes the worst-case pairwise disagreement

$$\arg\min_x \max_y \mathrm{PWD}(V_x^y) = \arg\min_x \max_y \mathbb{E}_{h,h'\sim\pi|_{V_x^y}}\left[d(h,h')\right]. \tag{5.16}$$

Other measures of diameter based on the supremum distance (Dasgupta, 2005; Kääriäinen, 2005) are not amenable to implementation because evaluation of such diameters involves optimization. The pairwise disagreement can be estimated from a finite set of sample hypotheses from the version space.

5.5.2 Worst-Case Gibbs-Vote Disagreement

We propose a new diameter measure called the *Gibbs-vote disagreement*. It is the expected disagreement between random hypotheses and their majority vote:

$$\mathrm{GVD}(V) \coloneqq \mathbb{E}_{h\sim\pi|_V}[d(h, h_{\text{vote}}|_V)]\,, \tag{5.17}$$

where $h_{\text{vote}}|_V$ is the majority vote classifier of hypotheses from V. For each x, it induces a prediction

$$h_{\text{vote}}|_V(x) = \arg\max_y \mathbb{E}_{h\sim\pi|_V}[p(y|x;h)]\,, \tag{5.18}$$

where $p(y|x;h)$ is the predicted probability of x belonging to class y given by a hypothesis h. The majority vote classifier is the deterministic classifier that has the smallest expected distance to the Gibbs classifier (Devroye et al., 2013; Kääriäinen, 2005):

$$\mathbb{E}_{h'}\left[d(h', h_{\text{vote}})\right] = \min_h \mathbb{E}_{h'}\left[d(h', h)\right]. \tag{5.19}$$

Hence the Gibbs-vote disagreement measures the size of the version space by the expected distance of the random hypotheses to their "center". Further, the following relation holds

$$\frac{1}{2}\mathrm{PWD}(V) \leq \mathrm{GVD}(V) \leq \mathrm{PWD}(V) \tag{5.20}$$

We defer the proof to the appendix[1]. Essentially, Equation (5.20) reveals that the Gibbs-vote disagreement is sandwiched between the average radius and diameter.

A greedy strategy selects the unlabeled sample that minimizes the worst-case Gibbs-vote disagreement

$$\arg\min_x \max_y \mathrm{GVD}(V_x^y) = \arg\min_x \max_y \mathbb{E}_{h\sim\pi|_{V_x^y}}\left[d(h, h_{\text{vote}}|_{V_x^y})\right], \tag{5.21}$$

where $h_{\text{vote}}|_{V_x^y}$ is the majority vote of hypotheses from the subspace V_x^y.

[1]The paper with appendix is available at https://arxiv.org/abs/2006.12456.

5.5.3 Diameter Reduction as Reducibility Reduction

Pairwise disagreement shares a simple relation with Gibbs error – it is the expected Gibbs error:

$$\begin{aligned}
\mathrm{PWD}(V) &= \mathbb{E}_{h,h'\sim\pi|_V}\big[\,\mathbb{E}_x\big[\mathbb{1}(h(x)\neq h'(x))\big]\,\big] && (5.22)\\
&= \mathbb{E}_x\big[\,\mathbb{E}_{h\sim\pi|_V}\big[\,\mathrm{Pr}_{h'\sim\pi|_V}(h(x)\neq h'(x))\big]\,\big] && (5.23)\\
&= \mathbb{E}_x[\,\mathrm{GE}(x|V)]\,. && (5.24)
\end{aligned}$$

A similar relation holds between Gibbs-vote disagreement and the variation ratio:

$$\begin{aligned}
\mathrm{GVD}(V) &= \mathbb{E}_{h\sim\pi|_V}[\,\mathbb{E}_x[\mathbb{1}(h(x)\neq h_{\mathrm{vote}}|_V(x))]\,] && (5.25)\\
&= \mathbb{E}_x\big[\,\mathbb{E}_{h\sim\pi|_V}[\mathbb{1}(h(x)\neq h_{\mathrm{vote}}|_V(x))]\,\big] && (5.26)\\
&= \mathbb{E}_x[\,\mathrm{VR}(x|V)]\,, && (5.27)
\end{aligned}$$

where the last equality holds because the predictions of the majority vote classifier are always the worst-case labels for prior mass reduction. Diameter reduction selects samples such that, upon revealing their labels, the induced subspaces have minimum possibility to be further reduced by a potential random query. Thus, it can be thought of as reducing the expected prior mass "reducibility".

Prior mass reduction finds splits in directions that evenly partition the version space, but could result in version spaces that have irregular shapes, in the sense that the space can be whittled down finely in some directions while being under-split in others. The worst-case error rate of the resulted version space could still be large. Diameter reduction correctly resolve this issue. Fig. 5.1 illustrates the differences between prior mass and diameter reduction.

5.5.4 Weighted Diameter Reduction

Tosh and Dasgupta, 2017 show that in general average diameter cannot be decreased at steady rate and propose to query the unlabeled samples that minimize the diameter weighted by the *squared* prior mass in the worst-case scenario

$$\begin{aligned}
&\arg\min_x \max_y \mathbb{E}_{h,h'\sim\pi}\big[\mathbb{1}(h,h'\in V_x^y)\,d(h,h')\big] && (5.28)\\
&\quad= \arg\min_x \max_y \pi(V_x^y)^2\,\mathbb{E}_{h,h'\sim\pi|_{V_x^y}}\big[d(h,h')\big]\,. && (5.29)
\end{aligned}$$

The potential to be minimized is a surrogate for the "amount" of edges between hypotheses and is closely related to the splittablity of the version space (Dasgupta, 2005; Tosh and Dasgupta, 2017).

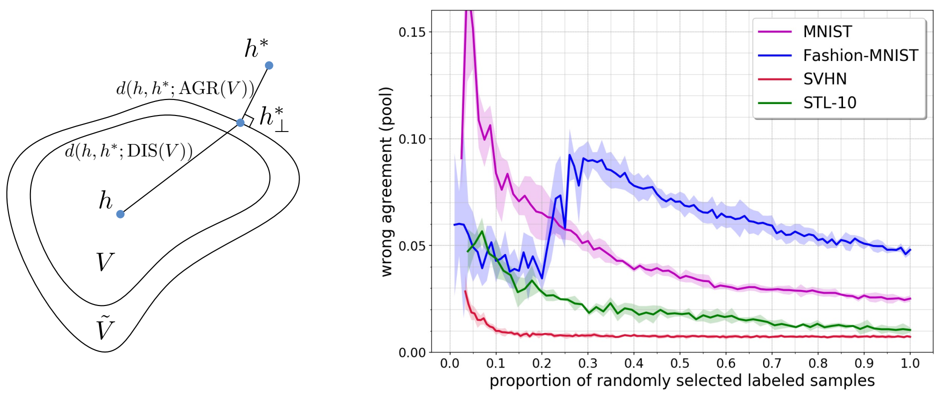


Figure 5.2: **Left**: Projection of h^* to the samplable version space. **Right**: Wrong agreement of version spaces trained on random samples. Total numbers of samples are 400, 1000, 3000 and 2580 for MNIST, Fashion-MNIST, SVHN and STL-10 respectively.

5.6 Realizability Assumption

Even though neural networks are capable of fitting an arbitrary pool set, we show experimentally that the version space obtained by training on a subset of the pool set with stochastic gradient descent – the "samplable" version space – is biased and not likely to contain the correct labeling of the pool set. Indeed, the distance from the *Bayes classifier*, which provides the ground truth labeling, to the "boundary" of the version space is non-negligible.

Let $h^*_\perp$ be the projection of the Bayes classifier h^* to the set of hypotheses $\tilde{V}$ that agree with V on $\mathrm{AGR}(V)$ (see the left plot of Fig. 5.2), i.e. ,

$$h^*_\perp := \arg\min_{h \in \tilde{V}} d(h, h^*), \tag{5.30}$$

$$\tilde{V} := \left\{h : h(\mathrm{AGR(V)}) = h'(\mathrm{AGR}(V)), h' \in V\right\}. \tag{5.31}$$

It is easy to see that $h^*_\perp$ provides the ground truth on $\mathrm{DIS}(V)$ and predicts the same labels on $\mathrm{AGR}(V)$ as hypotheses in V do, hence

$$d(h^*_\perp, h^*) = d(h, h^*; \mathrm{AGR(V)}) = \mathbb{E}_x[\mathbb{1}(x \in \mathrm{AGV}(V))\mathbb{1}(h(x) \neq h^*(x))]\,, \, \forall h \in V. \tag{5.32}$$

where $d(h, h^*; \mathrm{AGR(V)})$ is the disagreement probability restricted to AGR(V), or equivalently the *wrong agreement* of hypotheses in V.

We show the evolution of wrong agreement in the right plot of Fig. 5.2. As more random samples are queried, the wrong agreement decreases for all datasets, but for some much slower than the others. In Fig. 5.3, we show for MNIST a 2-D embedding of version spaces using Multi-Dimensional Scaling (MDS) (Kruskal, 1978), which finds a low-dimensional representation

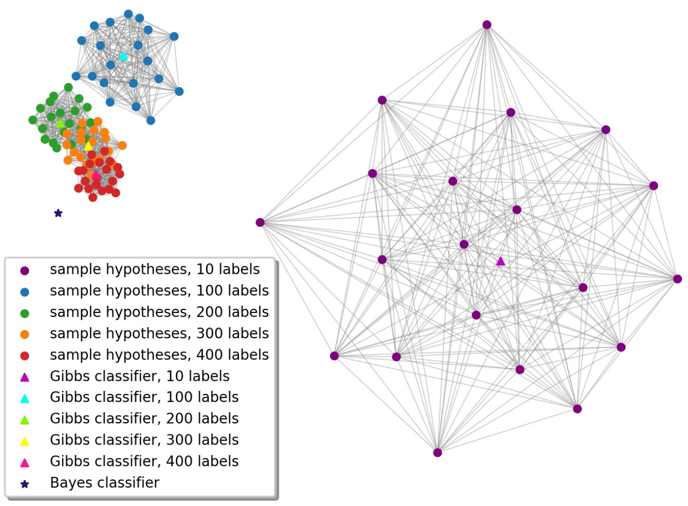


Figure 5.3: Embedding of version spaces on MNIST using MDS. As more random samples are used for training, the samplable version spaces move closer to the Bayes classifier but hardly cover it.

of potentially high-dimensional data by preserving pairwise distances between the data points. The Bayes classifier is not contained in any of the samplable version spaces although the distances between them decrease steadily.

In general neural networks trained with a random subset do not automatically predict all labels in the pool set correctly, unless a relatively large proportion of samples are used for training. However, this fact does not render version space reduction inconsistent, because the samplable version space is not fixed, but it shifts towards the correct labeling and finally covers it when the whole pool set has been used.

We conjecture that the dynamics of active learning with neural networks have two major components: (1) shrinkage of the samplable version space, which is explicitly optimized by the learning algorithm and (2) reduction of bias, which is not directly controllable. Empirical evidence is provided in the next section.

5.7 Evaluation

Datasets and Architectures We conduct active learning experiments[2] on four image classification datasets: MNIST, Fashion-MNIST, SVHN and STL-10. Neural network architectures are chosen to be competent for each dataset but as simple as possible in the hope of controlling the model complexity and

[2]Source code is available at https://github.com/jiayu-liu/effective-version-space-reduction-for-convnets.

mitigating the effect of overfitting. See Table 5.1 for the complete experiment settings.

Active Learning Methods We compare nine querying methods: Random, variation ratio (VR), Gibbs error (GE), Bayesian Active Learning by Disagreement with Monte Carlo dropout (BALD-MCD) (Gal et al., 2017; Houlsby et al., 2011), Core-Set (Sener and Savarese, 2018), Deep-Fool Active Learning (DFAL) (Ducoffe and Precioso, 2018), pairwise disagreement (PWD), Gibbs-vote disagreement (GVD), and double-weighted pairwise disagreement (M^2-PWD) (Tosh and Dasgupta, 2017). For each method on each dataset, at least three runs of active learning with different random balanced initial training set are performed.

Ensemble Size We train networks multiple times from scratch to obtain sample hypotheses and use them for prior mass and diameter estimation. Since diameters are estimated by considering partitioned version spaces, the ensemble size should be at least in the order of number of classes. We set the size to 20. Larger ensemble improves estimation but at the cost of longer training time. In preliminary experiments, we tried larger ensembles (40) and did not observe significant differences. Hence we do not include experiments on changing this hyper-parameter in the paper.

Query Size We set a small query budget for each round to reduce the correlation between queries. Larger budget may alleviate the pressure of frequent retraining, but the effect of each query can not be estimated and examined reliably. We observed in preliminary experiments that using larger budget (one or two orders larger) hides the differences between methods.

5.7.1 Diameter Reduction is More Effective Than Prior Mass Reduction

Fig. 5.4 and Table 5.2 show that direct diameter reduction methods PWD and GVD are consistently better than Random and achieve higher accuracy than other baselines while weighted diameter reduction M2-PWD is on par with Random. Diameter reduction methods usually exhibit less variances because training on samples queried by PWD, GVD and M2-PWD yields version

Dataset	Pool/Val/Test	Model	Ensemble Size	Init/Query/Total	Runs
MNIST	45000/5000/10000	2-conv-layer ConvNet	20	10/5/400	4
Fashion-MNIST	55000/5000/10000	3-conv-layer ConvNet	20	10/10/1000	4
SVHN	40000/5000/15000	6-conv-layer ConvNet	20	100/20/3000	4
STL-10	4000/1000/8000	ResNet18	20	100/40/2580	3

Table 5.1: Settings for each dataset used in the active learning experiments.

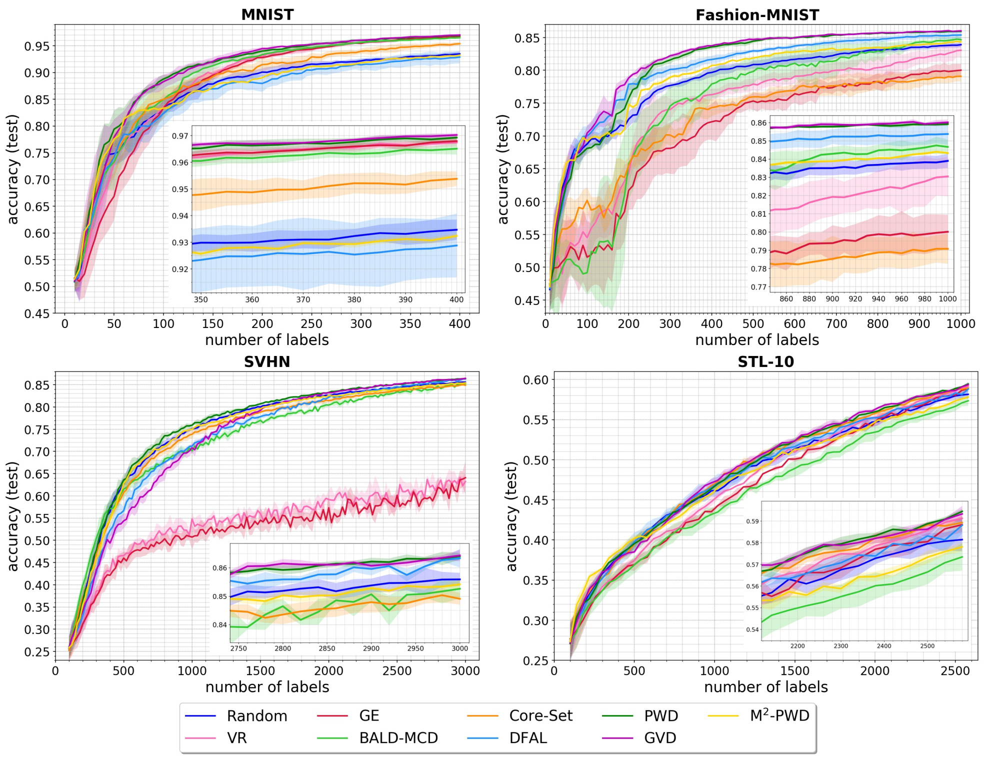


Figure 5.4: Accuracy over number of queried labels on the test set. Direct diameter reduction methods PWD and GVD are consistently better than Random and are among the best methods. Weighted diameter reduction M^2-PWD is on par with Random. Other baselines are effective on some datasets but inferior to Random on the others. Note that PWD, GVD and M^2-PWD exhibit smaller variances than the others.

#labels	**MNIST** 400	**Fashion-MNIST** 1000	**SVHN** 3000	**STL-10** 2580
Random	93.47 ± 0.38	83.90 ± 0.38	85.60 ± 0.23	58.15 ± 0.54
VR	96.74 ± 0.15	83.05 ± 1.09	63.23 ± 1.99	59.13 ± 0.21
GE	96.79 ± 0.10	80.01 ± 0.94	64.08 ± 3.77	58.84 ± 0.34
BALD-MCD	96.51 ± 0.22	84.67 ± 0.41	85.26 ± 0.34	57.35 ± 0.64
Core-Set	95.38 ± 0.28	79.08 ± 0.82	84.91 ± 0.20	58.93 ± 0.33
DFAL	92.88 ± 1.19	85.38 ± 0.60	86.34 ± 0.33	58.81 ± 0.37
PWD	96.92 ± 0.12	85.92 ± 0.10	86.41 ± 0.12	**59.45 ± 0.11**
GVD	**97.02 ± 0.06**	**86.01 ± 0.15**	**86.44 ± 0.20**	59.33 ± 0.37
M^2-PWD	93.24 ± 0.09	84.33 ± 0.03	85.42 ± 0.16	57.81 ± 0.20

Table 5.2: Accuracy on the test set in percentage.

spaces with smaller diameters and less diverse sample hypotheses. Prior mass reduction is not always effective and even fails on SVHN. This failure is an example of prior mass reduction being incapable of reducing the diameter, and provides empirical evidence that it may not be an appropriate objective for active learning.

5.7.2 Comparison to Other Baselines

BALD-MCD, Core-Set and DFAL are not consistently better than Random although each of them achieves comparative test accuracy on certain dataset. Their inferiority to Random in terms of test accuracy usually correlates with higher diameter (See description in Fig. 5.6 and Table 5.3). BALD-MCD and DFAL are highly related to prior mass reduction methods in that BALD (Houlsby et al., 2011) seeks samples for which the model parameters under the posterior disagree the most about the prediction (Houlsby et al., 2011), and that DFAL, inspired by margin-based active learning (Balcan et al., 2007), tries to locate the decision boundary with fewer labels which is essentially removing inconsistent hypotheses in the realizable case. However, none of them explicitly minimize the diameter, neither does Core-Set.

Note that for a fair comparison, we do not augment the training set by also adding the adversarial samples as the original DFAL paper by Ducoffe and Precioso, 2018 does. Samples with minimum adversarial perturbation are then verified reliably to be less effective than those lead to minimum diameter. The original Core-Set paper (Sener and Savarese, 2018) uses a large query batch size (in the order of 1000). However, many baselines rely on greedy selection and do not perform any batch optimization. To reduce query correlation, we adopt as small batch size as possible. This allows reliable evaluation of the effectiveness of queried samples as in the online setting. We are therefore able to identify one major cause of inferiority to Random as failing to effectively reduce the version space diameter.

	MNIST	**Fashion-MNIST**	**SVHN**	**STL-10**
#labels	400	1000	3000	2580
Random	2.86 ± 0.18	7.55 ± 0.26	13.13 ± 0.29	32.88 ± 0.43
VR	2.27 ± 0.18	10.64 ± 0.72	46.88 ± 2.76	34.21 ± 0.08
GE	2.30 ± 0.04	11.38 ± 1.52	44.87 ± 4.25	34.25 ± 0.09
BALD-MCD	2.39 ± 0.15	8.11 ± 0.51	16.58 ± 0.42	33.55 ± 0.44
Core-Set	2.91 ± 0.18	10.79 ± 1.34	14.66 ± 0.47	33.13 ± 0.64
DFAL	3.79 ± 0.60	7.06 ± 0.60	13.98 ± 0.31	32.41 ± 0.27
PWD	**1.93** ± 0.04	**6.91** ± 0.16	**12.80** ± 0.08	**32.25** ± 0.26
GVD	1.98 ± 0.05	6.98 ± 0.26	12.88 ± 0.25	32.96 ± 0.48
M^2-PWD	3.37 ± 0.13	7.22 ± 0.08	13.31 ± 0.13	33.23 ± 0.18

Table 5.3: Diameter (pairwise disagreement) on the test set in percentage.

5.7.3 Evolution of Samplable Version Space and its Implications

As shown in Fig. 5.6 and 5.3, the samplable version space shifts closer to the correct labeling while reducing its diameter as more labels are queried. These two processes together result in smaller test error.

No Direct Control Over Reduction of Version Space Bias Interestingly, the Core-Set method, which queries representative samples from the pool set by solving a k-center problem in the feature space learned by neural networks, is incapable of achieving negligible wrong agreement on the learned version spaces. Indeed, it suffers larger version space bias than the direct diameter reduction methods. After all, random queries which are i.i.d. by assumption fail to achieve this goal as concluded in Section 5.6 and other attempts without augmenting the training data seem doomed.

Prior Mass Induced by Stochastic Gradient Descent May Not Be a Reliable Surrogate Measure The continued decline in wrong agreement indicates that the distribution over labelings changes over time. This fact of shifting density over samplable labelings renders the notion of prior mass problematic, hence all notions relying on prior mass may not be well-defined. A direct consequence is that an estimate of the worst-case version space reduction would be more reliable than the average-case one. For example, VR provides a more reliable estimate of version space reduction than GE does.

Inferiority of Weighted Diameter Reduction Method The estimation of weighted diameter involves estimating the prior mass. Hence, the inferiority of M^2-PWD to PWD and GVD can be attributed to the intrinsic difficulty of obtaining unbiased samplable version spaces and the resulted density shift. A supportive evidence can be seen by noting that on MNIST and Fashion-MNIST, where the wrong agreement is large (hence large density shift), the weighted variant performs worse, while on SVHN and STL-10, where the wrong agreement is small (hence small shift), the gap is less significant.

5.7.4 Gibbs-Vote Disagreement

The Gibbs-vote disagreement is among the best methods on all datasets, except for the early learning stage on SVHN. Its effectiveness can be ascribed to an interesting phenomenon – majority voting reduces mistakes. Although it need not necessarily be the case, this phenomenon occurs in many situations and the boost to accuracy depends on the variance of errors of Gibbs classifiers (Lacasse et al., 2006). We show empirically that the majority vote classifier indeed has smaller error rate than random hypotheses in the version space in Fig. 5.5. Hence, optimizing the Gibbs-vote disagreement not only reduces

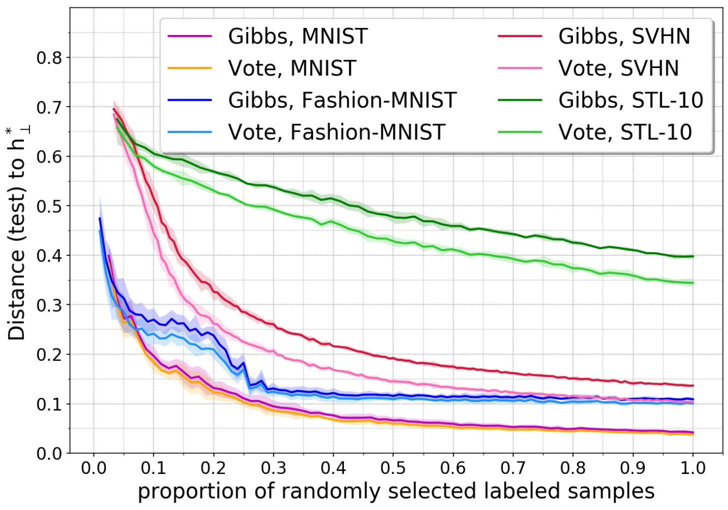


Figure 5.5: Distance from the Gibbs and the majority vote classifier to the projection of h^*. On four datasets, the majority vote classifier has a smaller distance, hence smaller error rate. See description of Fig. 5.2 for total numbers of random samples.

the diameter but also implicitly moves the consistent hypotheses closer to the correct labeling, which is useful when the samplable version spaces are biased and do not contain the Bayes classifier.

5.8 Conclusion

In this work, we studied version space reduction for convolutional neural networks. We revealed the differences and connections between prior mass and diameter reduction methods and proposed the Gibbs-vote disagreement as a new effective diameter-reduction method. With experiments on four datasets, we shed light into how version space reduction works in the deep active learning setting and demonstrated the superiority of diameter reduction over prior mass reduction methods and other baselines.

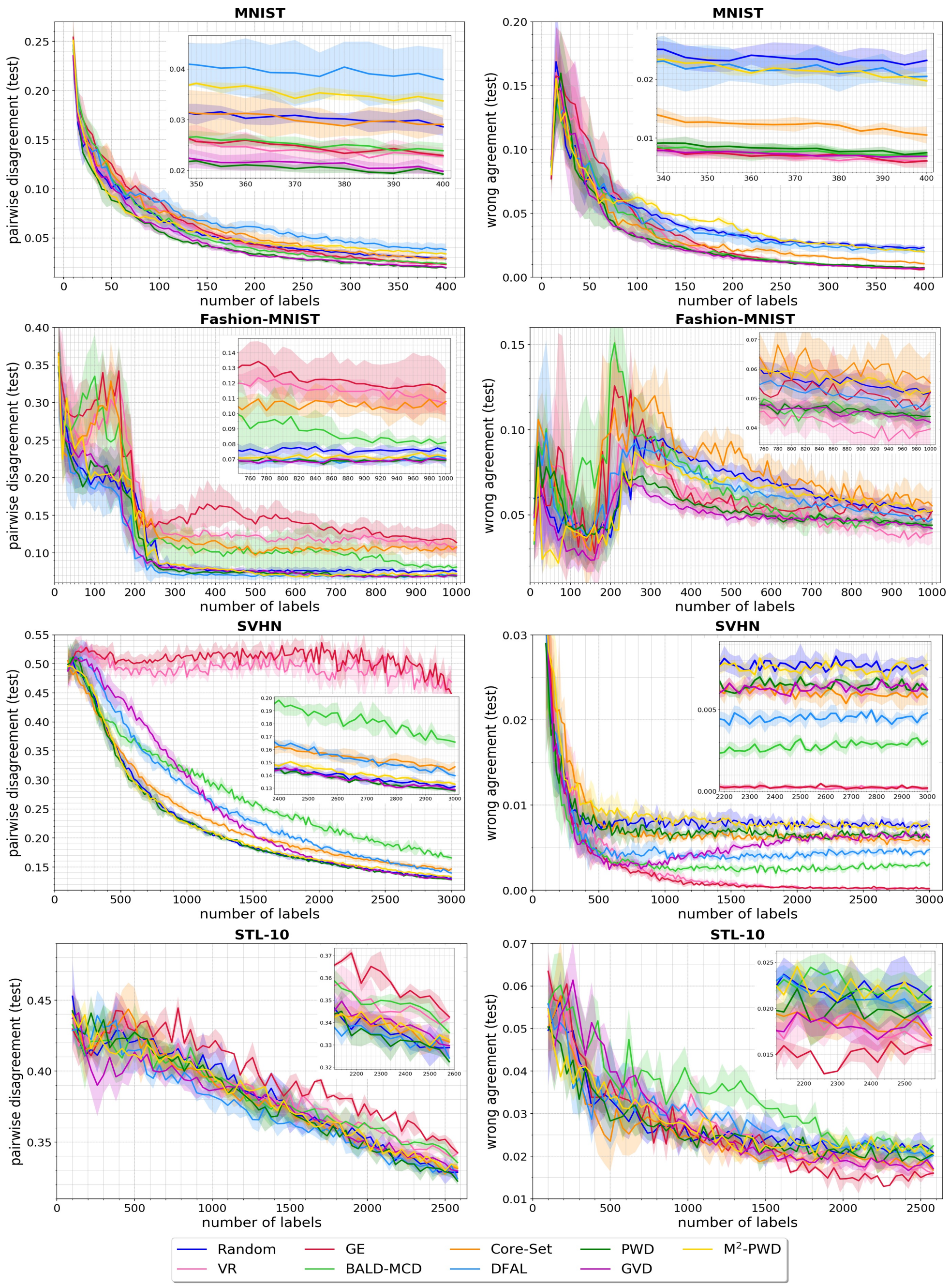


Figure 5.6: Pairwise disagreement and wrong agreement over number of queried labels on the test set. Except direct diameter reduction methods PWD and GVD, other baselines are not consistently better than or on par with Random at reducing version space diameter. Performing worse than Random: GE, VR and BALD-MCD on datasets except MNIST, Core-Set on Fashion-MNIST and SVHN, and DFAL on MNIST and SVHN, and M^2-PWD on MNIST.

Chapter 6

Learning to Explore

The journey, not the arrival matters.

– T.S. Eliot

This chapter is based on the preprint by Chiotellis and Cremers, 2020. Can we *learn* how to explore unknown spaces efficiently? To answer this question, we study the problem of Online Graph Exploration, the online version of the Traveling Salesperson Problem. We reformulate graph exploration as a reinforcement learning problem and apply Direct Future Prediction (Dosovitskiy and Koltun, 2017) to solve it. As the graph is discovered online, the corresponding Markov Decision Process entails a dynamic state space, namely the observable graph and a dynamic action space, namely the nodes forming the graph's frontier. To the best of our knowledge, this is the first attempt to solve online graph exploration in a data-driven way. We conduct experiments on six data sets of procedurally generated graphs and three real city road networks. We demonstrate that our agent can learn strategies superior to many well known graph traversal algorithms, confirming that exploration can be learned.

6.1 Introduction

In online graph exploration, an agent is immersed in a completely unknown environment. Located at a node of an unknown graph, they can only see the node's immediate neighbors. The agent moves and discovers the graph as they go. Whenever they visit a new node, all incident edges are revealed, along with their weights and their end nodes. To visit a new node, the agent has to traverse a path of known edges in the discovered graph. For each of these edges, the agent pays their weight as a cost. The goal of the agent is to visit all nodes in the graph, while paying the minimum cost.

By removing any particular geometric constraints, a large number of problems can be reduced to online graph exploration, as it basically is *search* with partial information in a discrete space. We revisit online graph exploration for undirected unweighted connected graphs. This task can be directly associated with many major problems in robotics such as planning, navigation, tracking and mapping (Yamauchi, 1997). All these subfields have been thoroughly investigated and a large number of algorithms have been devised, both classical and recently also learning-based. While path planning and navigation algorithms consider the question *"How can I get from A to B the fastest?"*, exploration algorithms consider a more abstract question: *"Where should I go beginning from A in order to discover the world the fastest?"*. In other words, while path planning studies *how* to reach a given destination, exploration is concerned with *which* destination should be reached next, a problem more akin to dynamic planning. We argue that exploration is a fundamental sequential decision making problem and it is therefore worth investigating if algorithms can *learn* which destinations are worth reaching and when.

The best known exploration strategy remains a simple greedy method - the nearest neighbor algorithm (NN). However, as NN selects the nearest (in terms of shortest path distance) unexplored node, its decisions are optimal only when considering a horizon of a single decision step. Therefore, a reasonable question is whether there are algorithms that can consider a longer horizon and thus minimize the *cumulative* path length, which is the true objective of exploration.

We present a learning algorithm that does exactly this. Our contributions can be summarized as follows:

- reformulate online graph exploration as a reinforcement learning problem,
- propose a neural network that can handle the associated dynamic state and action space,
- show experimentally that the proposed approach solves graph exploration as fast or faster than many classical graph exploration algorithms.

6.2 Related Work

The problem of graph exploration has been studied by the graph theory community for decades. A large number of works has been conducted, studying the problem for specific classes of graphs (Higashikawa et al., 2014; Miyazaki et al., 2009), with multiple collaborative agents (Dereniowski et al., 2013) or for variations of the problem with additional information (Dobrev et al., 2012) or energy constraints (Das et al., 2015; Duncan et al., 2006). Successful exploration algorithms are often sophisticated variations of depth first search (DFS), where the algorithm has to decide when to diverge from DFS (Kalyanasundaram and Pruhs, 1994; Megow et al., 2012). A similar problem has been studied by (Deng and Papadimitriou, 1999) for unweighted *directed* graphs, where the agent has to traverse all edges instead of visiting all nodes. In this setting, the offline equivalent problem is known as the Chinese Postman Problem (CPP) (Guan, 1962) which is solvable in polynomial time. In contrast, the offline equivalent of Online Graph Exploration is the Traveling Salesperson Problem, an NP-hard problem.

Besides the large volume of research, for general graphs, the best known exploration algorithm remains a simple greedy method - the nearest neighbor algorithm (NN). The trajectories followed by NN are provably at most $O(\log n)$[1] longer than the optimal ones (Rosenkrantz et al., 1977).

However, to the best of our knowledge, it has not been attempted to solve graph exploration in a data-driven way. In this work, we investigate whether, given a training set of graphs, a learning algorithm is able to find good exploration strategies that are competitive or even superior to traditional methods. Recently, there has been growing interest in applying learning to combinatorial optimization problems. In the work of Vinyals et al., 2015,

[1]where n is the number of nodes

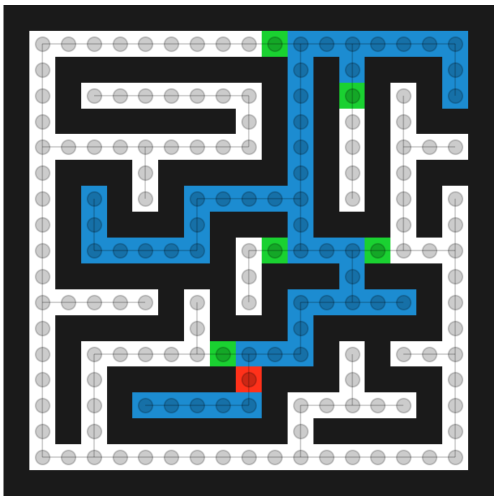
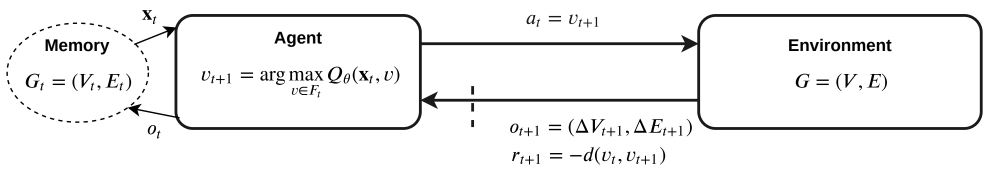


Figure 6.1: A graph exploration agent (**red**) keeps track of the nodes already visited (**blue**) and the *frontier* nodes that could be visited next (**green**). The frontier nodes form the boundary between known and unknown (**white**) space. The goal is to discover and visit all nodes as fast as possible.

three problems were studied and solved by learning to "point" to elements of a set with a neural network. Among these problems was the Traveling Salesperson Problem, which can be thought of as the offline equivalent to graph exploration. Vinyals et al., 2015 proposed a recurrent neural network (RNN) architecture, called Pointer Network, based on the attention mechanism introduced by Bahdanau et al., 2015. However, using an RNN might introduce a bias due to the ordering of the elements in the input sequence. To alleviate this bias, several works have studied neural architectures that can preserve the permutation-invariance property of sets (Edwards and Storkey, 2017; Lee et al., 2019; Zaheer et al., 2017). Moreover, in recent years, Graph Neural Networks (GNNs) (Battaglia et al., 2018) have emerged. These neural networks consider not just sets of nodes but also their pairwise connections or relationships as inputs and can learn how to solve problems such as node classification (Kipf and Welling, 2017) and link prediction (M. Zhang and Chen, 2018). Further, methods such as DeepWalk (Perozzi et al., 2014) and node2vec (Grover and Leskovec, 2016) have been used to learn node embeddings in an unsupervised way, borrowing ideas from natural language processing (Mikolov et al., 2013). These methods make an implicit assumption that nodes that co-appear in a random walk on the graph are more similar than nodes that don't. Nevertheless, it remains a challenge to learn node embeddings for dynamic graphs that are changing while an algorithm makes decisions about the graph's structure.

Close to our work is the work of Dai et al., 2019 which studies exploration on environments with graph-structured state-spaces, such as software testing. In contrast to Dai et al., 2019, we study the original problem as defined by the



$\mathbf{x}_t$
Memory
$G_t = (V_t, E_t)$
o_t
Agent
$v_{t+1} = \arg\max_{v \in F_t} Q_\theta(\mathbf{x}_t, v)$
$a_t = v_{t+1}$
Environment
$G = (V, E)$
$o_{t+1} = (\Delta V_{t+1}, \Delta E_{t+1})$
$r_{t+1} = -d(v_t, v_{t+1})$

Figure 6.2: The task of a graph exploring agent is to select the frontier node $v_{t+1} \in F_t$ to visit next. Once the node is visited, the environment reveals a new set of nodes ΔV_{t+1} and edges ΔE_{t+1}, expanding the agent's knowledge to the graph $G_{t+1} = (V_{t+1}, E_{t+1})$. The distance traveled by the agent from v_t to v_{t+1} is paid as a negative reward r_{t+1}. The agent acts upon an integrated observation $\mathbf{x}_{t+1}$ retrieved from its memory. The subset of nodes that have been discovered but are not yet visited are labeled as the frontier F_{t+1}.

graph theory community which prohibits the revisiting of past nodes. This allows us to study the problem in isolation, without the interference of other tasks such as visual feature learning and the perceptual aliasing problem.

In Section 6.3 we formally describe the problem of online graph exploration and the corresponding Markov Decision Process (MDP). In Section 6.4 we describe our method, the proposed network architecture, the used input features and the training algorithm. Lastly, in Section 6.5 we report experimental results on procedurally generated graphs and real road networks.

6.3 Formulation

6.3.1 Graph Exploration Overview

An agent explores a connected unweighted graph $G = (V, E)$. At time step $t = 0$, they start at an arbitrary node $v_0 \in V$ and they can only observe an initial *map* $G_0 = (V_0, E_0)$ comprised of the neighbors and incident edges of v_0. We assume the agent has a memory where it can store and integrate observations. Therefore, at any time step t, the agent observes a subgraph $G_t = (V_t, E_t)$ with a subset of visited nodes C_t and a subset of frontier nodes F_t to be explored. Being at node $v_t \in C_t$, the agent has to choose a node $v_{t+1} \in F_t$ to visit next. Once a decision is made, the agent follows a path from v_t to v_{t+1} of length $l_{t+1} = d_{G_t}(v_t, v_{t+1})$. Note that this is a shortest path in G_t but not necessarily in G[2]. The new node v_{t+1} gets removed from the frontier and becomes part of the set of visited nodes: $C_{t+1} \leftarrow C_t \cup \{v_{t+1}\}$. Finally, the agent observes the neighbors $N(v_{t+1})$ of v_{t+1} and the frontier gets expanded

[2]In the rest of the paper, we omit G_t and use the shorter notation $d(v_t, v_{t+1})$ wherever possible.

by the subset of neighbors that have not been observed in the past:

$$F_{t+1} \leftarrow F_t \cup N(v_{t+1}) \setminus C_{t+1}. \tag{6.1}$$

The goal is to visit the nodes in such an *order* that the total path length is minimized. Notice that we use a different timescale than commonly used e.g. in navigation problems. In a single time step, the exploration agent can traverse a path of arbitrary length in the known graph G_t. The differences between exploration algorithms lie in the way they choose the node $v \in F_t$ to visit next. For instance, DFS considers the order of entry in the frontier and chooses the most recently entered node. The nearest neighbor algorithm (NN) chooses the node closest to the current node $v_t \in C_t$:

$$v_{t+1}^{NN} = \arg\min_{v \in F_t} d(v_t, v) \tag{6.2}$$

However, since the NN selection rule is greedy, it might be suboptimal. Namely, an algorithm A could exist that takes into account the *expected* future path lengths and could therefore make better decisions:

$$v_{t+1}^{A} = \arg\min_{v \in F_t} \mathbb{E}[d(v_t, v) + \sum_{i=t+1}^{\infty} d(v_i, v_{i+1})]. \tag{6.3}$$

This formulation is reminiscent of reinforcement learning (RL). In RL an agent in a state s_t and following a policy π, chooses action $a_t = \pi(s_t)$ and receives an immediate reward r_{t+1}. The true objective of the agent is to maximize the cumulative reward:

$$a_t = \arg\max_{a \in \mathcal{A}} \mathbb{E}[r_{t+1} + \sum_{i=t+1}^{\infty} \gamma^{i-t} r_{i+1} | \pi, a], \tag{6.4}$$

where $\gamma \in [0, 1]$ is a discount factor that weighs distant future rewards less than imminent rewards. The expectation of cumulative rewards is also known as the action value $Q^\pi(s_t, a_t)$. Notice that the NN algorithm can be exactly recovered for $\gamma = 0$.

6.3.2 Markov Decision Process

RL problems are formally described as Markov Decision Processes (MDPs). An MDP is defined as a 5-tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, namely a state space $\mathcal{S}$, an action space $\mathcal{A}$, a state transition probability function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, a reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ and a discount factor $\gamma \in [0, 1]$. A partially observable Markov decision process (POMDP) is a generalization of a MDP, where the agent cannot directly observe the state $s_t \in \mathcal{S}$ but has partial information through observations $o_t \in \mathcal{O}$. An agent with a memory component can integrate partial observations to cumulative observations $\mathbf{x}_t \in \mathcal{X}$. Notice that the observation space $\mathcal{X}$ is a subset of the state space $\mathcal{S}$. Thus, we refer to the setting of Online Graph Exploration as a memory-augmented POMDP. In Figure 6.2, we illustrate this setting. In the following, we describe the components of this MDP.

State Space Let $\mathcal{G}$ be the set of all conceivable graphs, and let $\mathcal{P}_G$ denote the set of all conceivable visit orderings P_G for a graph $G \in \mathcal{G}$. Then the state space $\mathcal{S}$ is defined as the set of all pairs (G, P_G) of graphs $G \in \mathcal{G}$ and associated visit orderings $P_G \in \mathcal{P}_G$:

Observation Space At each time step, the environment reveals the neighborhood of the visited node. Therefore, the observation space is exactly the subset of graphs that are star graphs.

Action Space It is common in RL problems with discrete action spaces, for the agent to have access to a fixed set of actions $\mathcal{A}$ as in Eq. (6.4). Instead, in graph exploration, a new unique action set $\mathcal{A}_t$ is induced from the state at each time step. This action set corresponds to the nodes that have been observed but not visited yet, namely the nodes in the frontier:

$$\mathcal{A}_t = F_t. \tag{6.5}$$

The general action space can be described by the power set of all nodes: $\mathcal{A} = 2^{\mathcal{V}}$. Note that the frontier can be derived from the known graph $G_t = (V_t, E_t)$ and the path P_t as $F_t = V_t \setminus C_t$, where C_t denotes the *set* of visited nodes found in the sequence P_t.

Reward Function As defined in section 6.3.1, the rewards correspond to negative geodesic distances. Therefore, assuming unweighted graphs, all rewards are strictly negative:

$$r(s_t, a_t) = -d(v_t, v_{t+1}) < 0. \tag{6.6}$$

State Transition Function Let v_{t+1} be the node to visit next. Then, if $s_t = (G, P_t)$ is the state described by the graph G and the path P_t, the new state is described by the same graph G and the extended path $P_{t+1} \leftarrow P_t \mid\mid v_{t+1}$[3].

Memory Update Upon observing $\Delta V_{t+1} = N(v_{t+1})$, namely the neighbors of v_{t+1}, and $\Delta E_{t+1} = E(v_{t+1})$, the edges from v_{t+1} to $N(v_{t+1})$, the agent's

[3]By $\mid\mid$ we denote concatenation of a sequence with a new element.

memory is updated as:

$$
\begin{align}
V_{t+1} &\leftarrow V_t \cup \Delta V_{t+1} \tag{6.7}\\
E_{t+1} &\leftarrow E_t \cup \Delta E_{t+1} \tag{6.8}\\
P_{t+1} &\leftarrow P_t \mid\mid v_{t+1} \tag{6.9}\\
C_{t+1} &\leftarrow C_t \cup \{v_{t+1}\} \tag{6.10}\\
F_{t+1} &\leftarrow F_t \cup \Delta V_{t+1} \setminus C_{t+1}. \tag{6.11}
\end{align}
$$

6.4 Methodology

6.4.1 Predicting the Future Path Lengths

Our premise is that a learning agent can perform better than traditional exploration algorithms, as long as they can predict the future distances to be traveled. Inspired by the framework introduced by Dosovitskiy and Koltun, 2017, we use Direct Future Prediction (DFP) to learn a predictor of future path lengths. Confirming the authors' observations, we found that reducing policy learning to a supervised regression problem makes training faster and more stable[4]. In particular, at time t, we aim to predict the vector

$$
\mathbf{y}_t = (\mathbf{m}_{t+\tau_1} - \mathbf{m}_t, \mathbf{m}_{t+\tau_2} - \mathbf{m}_t, \dots, \mathbf{m}_{t+\tau_M} - \mathbf{m}_t), \tag{6.12}
$$

where $\mathbf{m}_t$ is a low-dimensional measurement vector augmenting the agent's high-dimensional observation $\mathbf{x}_t$, and $\{\tau_j\}_{j=1}^{M}$ are temporal offsets. Following Dosovitskiy and Koltun, 2017, we choose exponential offsets $\tau_j = 2^{j-1}$. We could directly use a scalar measurement $L_t = \sum_{i=0}^{t} l_i$, namely the cumulative path length up to time t. However, there are several disadvantages with this choice. For unweighted graphs, we know that any *one-step* path length l_t lies in the range $[1, N_{max} - 1]$, where N_{max} is the maximum number of nodes we are considering. Thus, directly predicting path lengths would limit our ability to generalize to graphs larger than our training graphs. Second, the distribution of path lengths naturally grows over time together with the observable graph's diameter. To avoid these problems, instead of minimizing path lengths, we maximize the agent's exploration rate $u_t = \frac{|C_t|}{L_t} = \frac{t}{L_t}$ which always lies in the $[0, 1]$ interval and thus the entries of $\mathbf{y}$ always lie in $[-1, 1]$. At test time, we choose the node to visit next by simply taking the arg max:

$$
v_{t+1} = \operatorname*{arg\,max}_{v \in F_t} \mathbf{g}^\top \mathbf{f}_\theta(\mathbf{x}_t, \mathbf{m}_t, v), \tag{6.13}
$$

where $\mathbf{f}_\theta$ is our parameterized predictor network, $\mathbf{x}_t = (V_t, E_t, X_t)$ is the observable graph with node features X_t and $\mathbf{g}$ is a goal vector expressing how much we care about different future horizons. Another advantage of using DFP

[4]See also supplementary material for a related experiment.

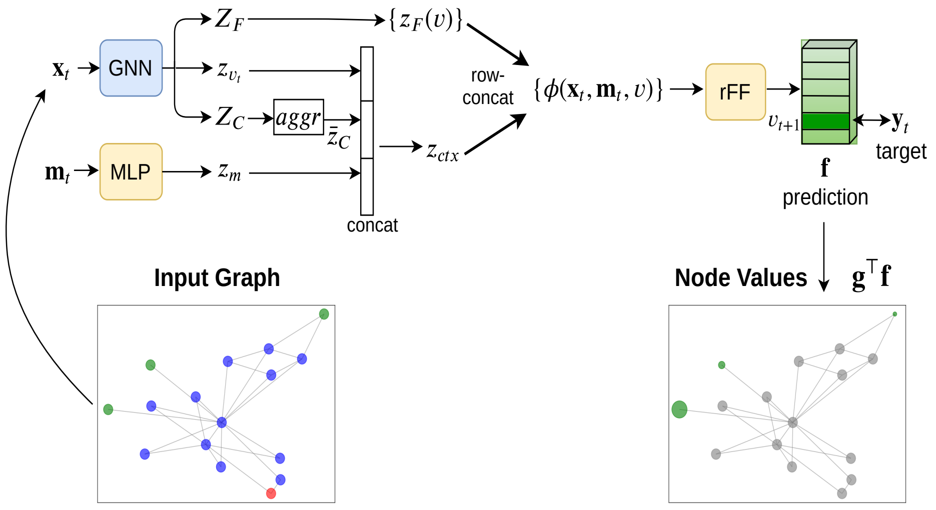


Figure 6.3: The proposed neural network architecture for online graph exploration.

instead of RL is that, given information about the remaining time available, we can directly incorporate it in the goal vector, both at training and at test time without the need of retraining the network. In contrast to Dosovitskiy and Koltun, 2017, we don't use $\mathbf{g}$ as an input to the network[5], but only as a weighting of the predictions to obtain a policy.

6.4.2 Network Architecture

In Figure 6.3 we show our network architecture. We first obtain node embeddings Z_F, Z_C and z_{v_t} by passing the observable graph $\mathbf{x}_t = (V_t, E_t, X_t)$ through a graph neural network (GNN). The embeddings correspond to the node subsets F_t, C_t and the current node v_t. We use a standard graph convolutional network (GCN) (Kipf and Welling, 2017). The node embeddings Z_C of the visited set are aggregated to obtain a subgraph embedding $\bar{z}_C$. Even though more sophisticated pooling methods (e.g. attention (Velickovic et al., 2018)) may be used, we use simple mean pooling.

We pass $\mathbf{m}_t$ through a multi-layer Perceptron (MLP) to obtain a measurement embedding z_m. The vectors $\bar{z}_C$, z_{v_t} and z_m are concatenated to form a *context* vector z_{ctx}. Each frontier node embedding $z_F(v) \in Z_F$ is concatenated with z_{ctx}, resulting in a set of state-action encodings $\phi(\mathbf{x}_t, \mathbf{m}_t, v)$, one for each node $v \in F_t$. Finally we pass these encodings through a row-wise feed-forward network (another MLP) to obtain a prediction vector $\mathbf{f}(\mathbf{x}_t, \mathbf{m}_t, v)$ for each node

[5]We found that adding a goal module does not improve and some times even hurts performance.

$v \in F_t$. The state-action value $Q(\mathbf{x}_t, \mathbf{m}_t, v, \mathbf{g})$ of each node in the frontier can be obtained by multiplying their prediction vector $\mathbf{f}(\mathbf{x}_t, \mathbf{m}_t, v)$ with the goal vector $\mathbf{g}$.

6.4.3 Input Features

Solving online graph exploration as a learning problem depends critically on what the learning algorithm "sees" as input. The state s_t consists of the known graph G_t and the path P_t. Therefore, we have to decide on the nodes and perhaps also edge input features of the graph G_t. Second, we have to consider a representation of the path P_t.

We used categorical node features indicating if a node belongs to the visited set C_t, the frontier set F_t or if it is the current node v_t:

$$x(v_i) = [\mathbf{1}(v_i \in C_t), \mathbf{1}(v_i \in F_t), \mathbf{1}(v_i = v_t)]. \tag{6.14}$$

Note that a categorical node feature space can incorporate many classical graph traversal algorithms (e.g. DFS and NN) as special cases by simply adding a binary channel that indicates the node that would be selected by the respective algorithm. Furthermore, this representation allows the learning algorithm to potentially learn a *hyper-policy* (Precup et al., 1998) by combining greedy algorithms in novel ways.

In preliminary experiments, we investigated ways to utilize the order of visit of the nodes, P_t, by using positional encodings (Vaswani et al., 2017) as continuous node features. We found that these features degraded the agent's performance both when used on their own and also when combined with the categorical features.

6.4.4 Training

In Algorithm 4 we describe our training procedure. In each episode, we randomly sample a graph from the training set and then randomly set one of its nodes as source. This virtually increases the training set size from the number of training graphs $|\mathcal{G}_{train}|$ to the total number of nodes in all training graphs $\sum_{G \in \mathcal{G}_{train}} |V(G)|$.

Every few exploration steps, a minibatch of graphs $\{G_t\}$ and associated exploration sequences $\{P_t\}$ are sampled from a replay buffer, along with the respective future measurement vectors $\{\mathbf{y}_t\}$. The network is trained to regress the future measurements by minimizing the mean squared error using the Adam optimizer (Kingma and Ba, 2015).

Algorithm 4 Training NOGE

```
1: Input: network $\mathbf{f}_\theta$, training set of graphs $\mathcal{G}_{train}$, time limit $T_{max}$, goal
   vector $\mathbf{g}$, minibatch size B.
2: Output: trained network $\mathbf{f}_\theta$.
3: Initialize $\theta$ randomly.
4: Initialize an experience replay buffer $\mathcal{R}$.
5: while training do
6:     Sample a graph $G = (V, E) \sim \mathcal{G}_{train}$.
7:     Sample a source node $v_0 \sim V$.
8:     Explore $G$ using $\mathbf{f}_\theta$, $\mathbf{g}$ and $\epsilon$-greedy policy for up to $T_{max}$ episode steps.
9:     Store $G_T$ and the followed path $P_T$ in $\mathcal{R}$.
10:    Sample a minibatch $\{G_t, P_t, \mathbf{y}_t\}$ from $\mathcal{R}$.
11:    Reconstruct tuples $\{\mathbf{x}_t, \mathbf{m}_t, v_{t+1}, \mathbf{y}_t\}$.
12:    Train $\mathbf{f}_\theta$ using the minibatch loss:
13:    $L(\theta) = \frac{1}{B}\sum_{i=1}^{B} ||\mathbf{y}_t^{(i)} - \mathbf{f}_\theta(\mathbf{x}_t^{(i)}, \mathbf{m}_t^{(i)}, v_{t+1}^{(i)})||^2$.
```

6.5 Experiments

The hyperparameters used are reported in the Appendix B. Our source code to generate the data sets and reproduce the experiments is publicly available[6].

6.5.1 Evaluation Protocol

We evaluate our algorithm - NOGE (Neural Online Graph Exploration) on data sets of generated and real networks. In addition to the basic version of our algorithm, we evaluate NOGE with an extra node feature, indicating the nearest neighbor, as described in section 6.4.3. We call this variant NOGE-NN. We use three well known graph exploration algorithms as baselines: Breadth First Search (BFS), Depth First Search (DFS) and Nearest Neighbor (NN). We note that these heuristics do not need any training. For completeness, we also report a random exploration baseline (RANDOM). We compare the algorithms in terms of the *exploration rate* u_T, namely the number of visited nodes over the total path length at the end of episodes:

$$u_T = \frac{|C_T|}{|P_T|} = \frac{T}{\sum_{i=0}^{T} l_i}. \tag{6.15}$$

For the test sets we fix a set of source nodes per graph, to compare all methods given the same initial conditions. The metrics reported are computed on the test sets after either all nodes have been explored or a fixed number of $T_{max} = 500$ exploration steps has been reached. For all experiments we report mean and standard deviation over 5 random seeds.

[6] https://github.com/johny-c/noge

6.5.2 Procedurally Generated Graphs

We first examine six classes of procedurally generated graphs (Figure 6.4). We used the *networkx* library[7] to generate a diverse (in terms of size and connectivity) set of graphs for each class. In Table 6.1 we report basic statistics of the data sets, namely the number of graphs, the minimum and maximum number of nodes and the minimum and maximum number of edges.

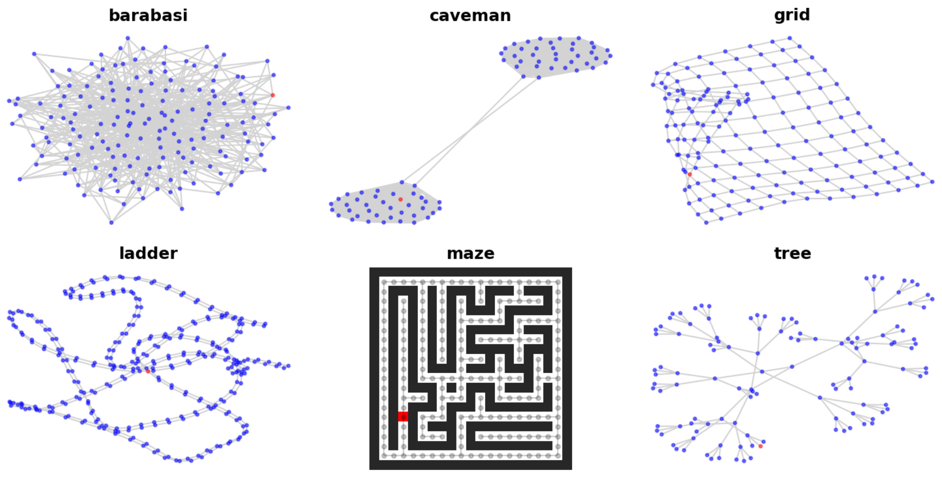


Figure 6.4: Samples from the procedurally generated data sets.

Dataset	Size	$\|V\|_{min,max}$	$\|E\|_{min,max}$
barabasi(tr)	400	(100, 199)	(384, 780)
barabasi(te)	100	(100, 199)	(384, 780)
ladder(tr)	80	(200, 398)	(298, 595)
ladder(te)	20	(220, 386)	(328, 577)
tree(tr)	4	(121, 1365)	(120, 1364)
tree(te)	2	(364, 1093)	(363, 1092)
grid(tr)	80	(64, 289)	(112, 544)
grid(te)	20	(72, 240)	(127, 449)
caveman(tr)	120	(60, 316)	(870, 12324)
caveman(te)	30	(70, 304)	(1190, 11400)
maze(tr)	400	(97, 251)	(96, 262)
maze(te)	100	(97, 255)	(96, 276)

Table 6.1: Basic statistics of the procedurally generated data sets. In the parentheses, "tr" signifies training set and "te" signifies test set.

[7]https://github.com/networkx/networkx

We split each data set in a training (80%) and test set (20%) of graphs. For some of these data sets, an optimal strategy is known. For instance, DFS explores trees optimally by traversing each edge two times - once to explore and once to backtrack. Thus its exploration rate is approximately 0.5. It is worthwhile examining if NOGE can find this optimal strategy.

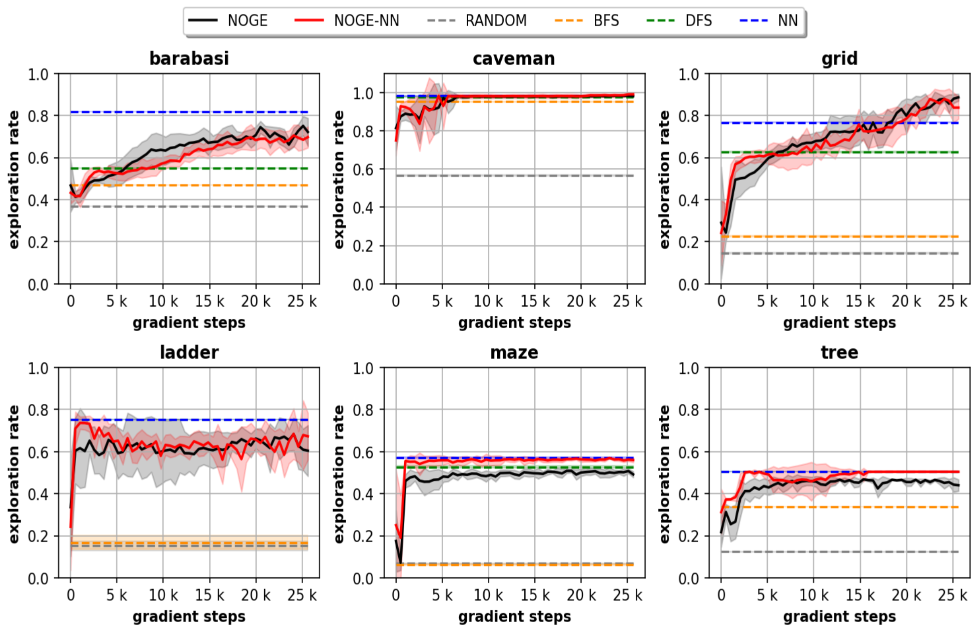


Figure 6.5: Exploration rate over gradient steps for the six procedurally generated data sets.

Algorithm	barabasi	caveman	grid
RANDOM	0.3695 (0.0006)	0.5664 (0.0050)	0.1461 (0.0037)
BFS	0.4695 (0.0013)	0.9526 (0.0025)	0.2264 (0.0039)
DFS	0.5494 (0.0009)	0.9778 (0.0006)	0.6272 (0.0041)
NN	**0.8179**(**0.0014**)	0.9827 (0.0015)	0.7670 (0.0028)
NOGE	0.7214 (0.0663)	0.9817 (0.0029)	**0.8861**(**0.0162**)
NOGE-NN	0.6970 (0.0497)	**0.9907**(**0.0031**)	**0.8373**(**0.0536**)
Algorithm	ladder	maze	tree
RANDOM	0.1531 (0.0226)	0.0688 (0.0027)	0.1242 (0.0011)
BFS	0.1691 (0.0341)	0.0626 (0.0025)	0.3397 (0.0002)
DFS	**0.7519**(**0.0010**)	0.5266 (0.0050)	**0.5044**(**0.0002**)
NN	**0.7530**(**0.0009**)	**0.5723**(**0.0033**)	**0.5044**(**0.0003**)
NOGE	0.6046 (0.1208)	0.4921 (0.0140)	0.4403 (0.0272)
NOGE-NN	**0.6729**(**0.1114**)	0.5601 (0.0106)	**0.5043**(**0.0004**)

Table 6.2: Final exploration rate: Mean and standard deviation on the generated data sets.

In Figure 6.5 we show the test performance of NOGE over 25600 training steps. In Table 6.2 we report the final performance of the algorithms compared to the baselines. NOGE is able to outperform other methods on the "grid" and the "caveman" data set and finds the optimal strategy on trees. Somewhat surprisingly the NN feature seems to only help on the "maze" and "tree" data sets. Note that in "ladder" and "tree", the exploration rate line of DFS is hidden as its performance matches that of NN.

6.5.3 City Road Networks

To examine its capabilities, we also evaluate NOGE, on three real road networks. We use openly available drivable road networks from OpenStreetMap (Haklay and Weber, 2008). We explore three cities with diverse networks: Munich (MUC), Oxford (OXF) and San Francisco (SFO), shown in Figure 6.6.

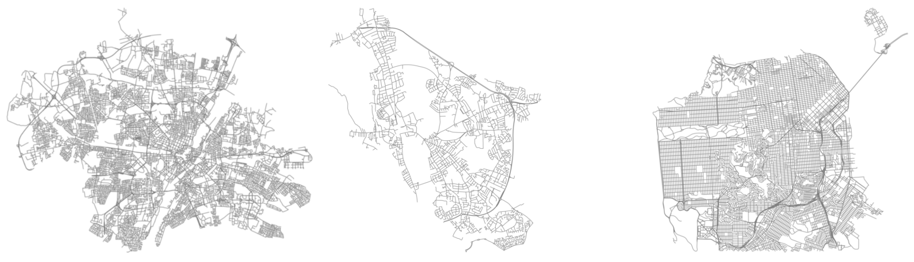

(a) Munich (b) Oxford (c) San Francisco

Figure 6.6: The road city networks used for evaluating NOGE.

In Table 6.3, we show basic statistics for these data sets. We constructed a training set and test set for each city by cutting each graph in two components and removing any edges connecting them. The cut was defined by the diagonal line over each city's 2D bounding box. The larger component - consisting of approximately 60% of the nodes - was used for training and the smaller one for testing.

Dataset	Size	$\|V\|_{min,max}$	$\|E\|_{min,max}$
MUC(tr)	1	(8559, 8559)	(12821, 12821)
MUC(te)	1	(5441, 5441)	(7772, 7772)
OXF(tr)	1	(2197, 2197)	(2561, 2561)
OXF(te)	1	(1185, 1185)	(1430, 1430)
SFO(tr)	1	(5691, 5691)	(9002, 9002)
SFO(te)	1	(3885, 3885)	(6579, 6579)

Table 6.3: Basic statistics of the city road network data sets. In the parentheses, "tr" signifies training set and "te" signifies test set.

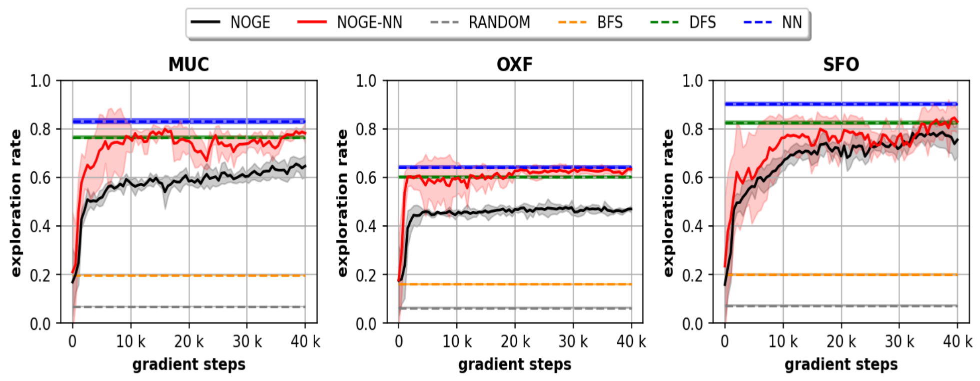


Figure 6.7: Exploration rate over gradient steps for the city road networks data sets.

Algorithm	MUC	OXF	SFO
RANDOM	0.0674 (0.0003)	0.0624 (0.0009)	0.0726 (0.0015)
BFS	0.1961 (0.0007)	0.1608 (0.0019)	0.2007 (0.0033)
DFS	0.7644 (0.0053)	0.6012 (0.0048)	0.8252 (0.0073)
NN	**0.8314**(**0.0091**)	**0.6422**(**0.0037**)	**0.9017**(**0.0064**)
NOGE	0.6458 (0.0441)	0.4695 (0.0136)	0.7541 (0.0679)
NOGE-NN	0.7814 (0.0386)	**0.6328**(**0.0141**)	0.8289 (0.0456)

Table 6.4: Final exploration rate: Mean and standard deviation on the city road networks data sets.

In Figure 6.7 we show the test performance of NOGE over 40000 training steps and in Table 6.4 we report the final performance. Nearest Neighbor performs clearly better in San Francisco and Munich. NOGE-NN is able to match and surpass DFS, as the second best method. In Oxford, NOGE-NN is within a standard deviation from the best performance. In these graphs the NN feature clearly improves performance.

6.6 Conclusion

In this work, we presented NOGE, a learning-based algorithm for exploring graphs online. First, we formulated an appropriate memory-augmented Markov Decision Process. Second, we proposed a neural architecture that can handle the growing graph as input and the dynamic frontier as output. Third, we devised a node feature space that can represent greedy methods as *options* (Precup et al., 1998). Finally, we showed experimentally that NOGE is competitive to well known classical graph exploration algorithms in terms of the exploration rate of unseen graphs.

Chapter 7

Conclusion

In this thesis, we described machine learning algorithms that reduce the need for human supervision in problems from a variety of domains. To achieve that we extended and applied techniques from Metric Learning, Semi-Supervised Learning, Active Learning and Reinforcement Learning.

In Chapter 2, we showed how a metric learning algorithm, applied on top of standard spectral-based point descriptors, improved the state of the art in non-rigid shape retrieval by a large margin.

In Chapter 3, we described an inverse reinforcement learning system that learned how to avoid collisions and how to maximize human comfort in a simulated driving environment. We showed that, given only a few demonstrations from a human driver, the agent can learn how the human approaches the task and can optimize their driving behavior with respect to human criteria that are hard to formulate explicitly.

In Chapter 4, we presented a semi-supervised algorithm that can learn to classify objects in an incoming stream of data. By combining an incremental graph update mechanism and a novel stopping criterion for the label propagation process (Zhu and Ghahramani, 2002), we showed that our algorithm can surpass fully supervised online algorithms in terms of classification accuracy, while using only a small fraction of the labels.

In Chapter 5, we investigated active learning with convolutional neural networks through the lens of version space reduction. We showed why prior mass reduction are not principled and proposed a novel and efficient query strategy based on diameter reduction and model disagreement. We demonstrated that the proposed criterion in many cases improves the state-of-the-art in image classification.

Finally, in Chapter 6, we studied Online Graph Exploration, an important combinatorial optimization problem that we viewed as a reinforcement learning problem. We found that a learning agent can find exploration strategies competitive to and in some cases faster than classical graph traversal algorithms.

While deep learning has "unlocked" new possibilities in machine learning and artificial intelligence, it is important that the autonomous systems of this new era can work without requiring enormous amounts of labeled data. We believe, in this thesis, we have demonstrated that this can be achieved. We are excited to see where the new challenges in computer vision and robotics will lie.

Appendix A

Appendix for Chapter 4

A.1 Details for proof of Theorem 1

To get a suitable norm on $\mathbb{R}^u$ to show that T is a contraction we use lemma 5.6.10 of Roger A. Horn, 2013. It states that for any matrix $A \in \mathbb{R}^{n\times n}$ and $\epsilon > 0$ there is a matrix norm $|||.|||_\omega$, s.th. $|||A|||_\omega \leq \rho(A) + \epsilon$. This matrix norm is constructed as follows: Because of theorem 2.3.1 in Roger A. Horn, 2013 there is an unitary matrix $U \in \mathbb{R}^{n\times n}$ and an upper triangular matrix $\Delta \in \mathbb{R}^{n\times n}$ s.th. $A = U\Delta U^*$. Let $D_t := \operatorname{diag}(t, t^2, t^3, ..., t^n)$ and $Q := D_t U^*$. Then the matrix norm we are interested in is defined by

$$\begin{aligned} |||A|||_\omega &:= |||D_t U^* A U D_t^{-1}|||_1 = |||\left(D_t U^*\right) A \left(D_t U^*\right)^{-1}|||_1 \\ \text{where } |||A|||_1 &:= \max_{1\leq j\leq n} \sum_{i=1}^{n} |a_{i,j}|\,. \end{aligned} \tag{A.1}$$

Example 5.6.4 in Roger A. Horn, 2013 shows that the norm $|||.|||_1$ is induced by the ℓ_1-vector-norm $||x||_1 = \sum_{i=1}^n |x_i|$ on $\mathbb{R}^n$. Following theorem 5.6.7 in Roger A. Horn, 2013 the matrix norm $|||.|||_\omega$ is then induced by the vector-norm $||x||_\omega = ||D_t U^* x||_1$. As shown in the proof of lemma 5.6.10 in Roger A. Horn, 2013, for t large enough the matrix norm $|||.|||_\omega$ fulfills $|||A|||_\omega \leq \rho(A) + \epsilon\,$.

Appendix B

Appendix for Chapter 6

B.1 Implementation Details

B.1.1 Hyperparameters

In Table B.1 we show the hyperparameters used for the experiments on the procedurally generated data sets. The only differences in the experiments for the city road networks are the number of training steps which was set to 40000 and the hidden layer width of the neural network (see next subsection).

Parameter	Value
test set ratio	0.2
max. episode steps (T_{max})	500
node history	2
feature range	[-0.5, 0.5]
target normalization	True
training steps	25600
evaluation episodes	50
env. steps per tr. step	32
tr. steps per evaluation	512
replay buffer size ($\lvert\mathcal{R}\rvert$)	20000
ε_{max}	1
ε_{min}	0.15
temporal coefficients ($\mathbf{g}$)	$[0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, 1]$
minibatch size (B)	32
learning rate	0.0001

Table B.1: Hyperparameters used in experiments evaluating NOGE.

We elaborate on hyperparameters, the usage of which may not be clear:

Node History It is common in deep reinforcement learning to replace the input observation $\mathbf{x}_t$ with a stack of the last k observations $[\mathbf{x}_{t-k+1}, \mathbf{x}_{t-k+2}, \dots, \mathbf{x}_t]$, particularly when the observations are images. This gives the agent a sense of the environment dynamics. We found that using a stack of the last 2 feature vectors for each node also improves performance in graph exploration, as it gives a sense of direction.

Feature Range As a preprocessing step, shifting input features to the $[-0.5, 0.5]$ range speeds up learning.

Target Normalization As a postprocessing step, we used target normalization. We scaled targets $\mathbf{y}$ by the standard deviation of measurements collected during random exploration, as described by Dosovitskiy and Koltun, 2017.

Evaluation Episodes For evaluation, we sampled 50 graphs from the test set and fixed one source node per graph. If the test set contained less than 50 graphs, we sampled 50 source nodes uniformly from all test graphs.

ε-greedy Policy As described in our training algorithm, we used an ε-greedy policy to collect experiences, namely a random frontier node was selected to be visited with probability ε and a node was selected by the network's policy with probability 1 - ε. We linearly interpolated ε from 1 to 0.15 over the course of training. During testing, the greedy policy ($\varepsilon = 0$) was used.

B.1.2 Network Architecture

The architecture of our network, used for the procedurally generated graphs, is shown in Table B.2. The same architecture was used for the city road networks, except that all hidden layers are wider by a factor of two. The input dimension for the graph neural network (GNN) was 3 for NOGE and 4 for NOGE-NN. In all networks we use the ReLU nonlinearity after all layers except for the output layer of the row-wise feed-forward (rFF) network.

module	input dimension	output dimension
GNN	3 or 4	32
	32	64
MLP	1	64
	64	64
rFF	256	128
	128	8

Table B.2: Network architecture.

B.1.3 Replay Buffer for Graphs

To use the replay buffer for training, we need to be able to sample graph observations G_t from any time step in an episode. To do that, for each episode we store the discovered graph $G_T = (V_T, E_T)$ at the end of the episode and two arrays: an array of node counts and an array of edge counts, indicating the size of the graph at each time step. To be able to reconstruct the frontier at an arbitrary time step t, we need to store two integers per node v: the time of discovery $t_{dis}(v)$ and the time of visit $t_{dis}(v)$. Then the frontier F_t at any time step t is:

$$F_t = \{v \in V_t : t \geq t_{dis}(v) \wedge t < t_{vis}(v)\} \qquad \text{(B.1)}$$

B.2 Comparison to DQN

We originally tried using standard RL algorithms but found them to be unstable compared to DFP. In Figure B.1, we additionally show the test exploration rate of a DQN (Mnih et al., 2015) during training on the procedurally generated data sets. The curves show mean and standard deviation over 5 random seeds. Except for the original path length reward function (PL), described in the paper, we also tried using the exploration rate difference between two time steps (ER). In both cases the DQN had access to the NN feature. We can see that in 3 out of 6 data sets, the DQN struggles and degrades to solutions much worse than those found by NOGE.

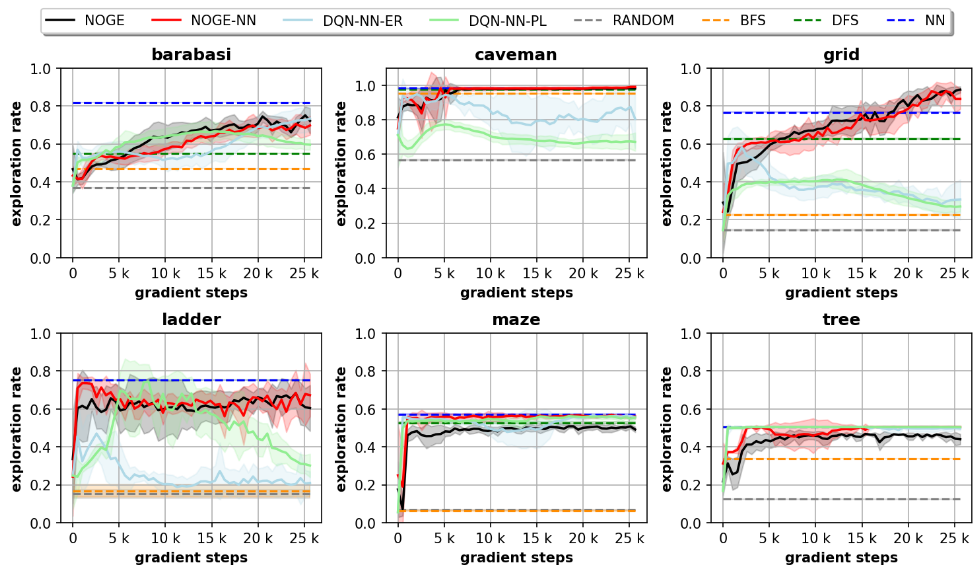


Figure B.1: Exploration rate over gradient steps for the six procedurally generated data sets.

Own Publications

Chiotellis, I. and D. Cremers (2020). "Neural Online Graph Exploration". In: arXiv: 2012.03345 [cs.LG] (cit. on p. 74).

Chiotellis, I., R. Triebel, T. Windheuser, and D. Cremers (2016). "Non-Rigid 3D Shape Retrieval via Large Margin Nearest Neighbor Embedding". In: *European Conference on Computer Vision (ECCV)*. Springer, pp. 327–342 (cit. on p. 16).

Chiotellis*, I., F. Zimmermann*, D. Cremers, and R. Triebel (2018). "Incremental Semi-Supervised Learning from Streams for Object Classification". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 5743–5749 (cit. on p. 42).

Liu, J., I. Chiotellis, R. Triebel, and D. Cremers (2020). "Effective Version Space Reduction for Convolutional Neural Networks". In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2020, Ghent, Belgium, September 14-18, 2020, Proceedings, Part II*. Ed. by F. Hutter, K. Kersting, J. Lijffijt, and I. Valera. Vol. 12458. Lecture Notes in Computer Science. Springer, pp. 85–100. DOI: 10.1007/978-3-030-67661-2_6. URL: https://doi.org/10.1007/978-3-030-67661-2%5C_6 (cit. on p. 58).

Sharifzadeh, S., I. Chiotellis, R. Triebel, and D. Cremers (2016). "Learning to Drive Using Inverse Reinforcement Learning and Deep Q-Networks". In: *Workshop on "Deep Learning for Action and Interaction", Conference on Neural Information Processing Systems (NIPS)*. (Cit. on p. 32).

Bibliography

Abbeel, P. and A. Y. Ng (2004). “Apprenticeship learning via inverse reinforcement learning”. In: *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*. Ed. by C. E. Brodley. Vol. 69. ACM International Conference Proceeding Series. ACM. DOI: 10.1145/1015330.1015430. URL: https://doi.org/10.1145/1015330.1015430 (cit. on pp. 33, 34, 36).

Aubry, M., U. Schlickewei, and D. Cremers (2011). “The wave kernel signature: A quantum mechanical approach to shape analysis”. In: *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE, pp. 1626–1633 (cit. on pp. 17, 18, 22).

Bahdanau, D., K. Cho, and Y. Bengio (2015). “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. URL: http://arxiv.org/abs/1409.0473 (cit. on p. 76).

Balcan, M.-F., A. Beygelzimer, and J. Langford (2006). “Agnostic active learning”. In: *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*. Ed. by W. W. Cohen and A. W. Moore. Vol. 148. ACM International Conference Proceeding Series. ACM, pp. 65–72. DOI: 10.1145/1143844.1143853. URL: https://doi.org/10.1145/1143844.1143853 (cit. on p. 60).

Balcan, M.-F., A. Z. Broder, and T. Zhang (2007). “Margin Based Active Learning”. In: *Learning Theory, 20th Annual Conference on Learning Theory, COLT 2007, San Diego, CA, USA, June 13-15, 2007, Proceedings*. Ed. by N. H. Bshouty and C. Gentile. Vol. 4539. Lecture Notes in Computer Science. Springer, pp. 35–50. DOI: 10.1007/978-3-540-72927-3_5. URL: https://doi.org/10.1007/978-3-540-72927-3%5C_5 (cit. on p. 69).

Battaglia, P. W., J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. (2018). “Relational inductive biases, deep learning, and graph networks”. In: *arXiv preprint arXiv:1806.01261* (cit. on pp. 12, 76).

Beluch, W. H., T. Genewein, A. Nürnberger, and J. M. Köhler (2018). "The Power of Ensembles for Active Learning in Image Classification". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, pp. 9368–9377. DOI: 10.1109/CVPR.2018.00976. URL: http://openaccess.thecvf.com/content%5C_cvpr%5C_2018/html/Beluch%5C_The%5C_Power%5C_of%5C_CVPR%5C_2018%5C_paper.html (cit. on p. 60).

Bojarski, M., D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. (2016). "End to End Learning for Self-Driving Cars". In: *arXiv preprint arXiv:1604.07316* (cit. on p. 32).

Bronstein, A. M. (2011). "Spectral descriptors for deformable shapes". In: *arXiv preprint arXiv:1110.5015* (cit. on p. 17).

Bronstein, A. M., M. M. Bronstein, L. J. Guibas, and M. Ovsjanikov (2011). "Shape google: Geometric words and expressions for invariant shape retrieval". In: *ACM Transactions on Graphics (TOG)* 30.1, p. 1 (cit. on p. 30).

Bronstein, M. M. and I. Kokkinos (2010). "Scale-invariant heat kernel signatures for non-rigid shape recognition". In: *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*. IEEE Computer Society, pp. 1704–1711. DOI: 10.1109/CVPR.2010.5539838. URL: https://doi.org/10.1109/CVPR.2010.5539838 (cit. on pp. 17, 18, 22).

Chapelle, O., B. Schölkopf, and A. Zien (2006). *Semi-Supervised Learning*. The MIT Press (cit. on p. 43).

Charikar, M., C. Chekuri, T. Feder, and R. Motwani (2004). "Incremental clustering and dynamic information retrieval". In: *SIAM Journal on Computing* 33.6, pp. 1417–1440 (cit. on p. 44).

Cignoni, P., M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia (2008). "MeshLab: an Open-Source Mesh Processing Tool". In: *Eurographics Italian Chapter Conference*. Ed. by V. Scarano, R. D. Chiara, and U. Erra. The Eurographics Association. ISBN: 978-3-905673-68-5. DOI: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136 (cit. on p. 27).

Cuong, N. V., W. S. Lee, and N. Ye (2014). "Near-optimal Adaptive Pool-based Active Learning with General Loss". In: *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, UAI 2014, Quebec City, Quebec, Canada, July 23-27, 2014*. Ed. by N. L. Zhang and J. Tian. AUAI Press, pp. 122–131. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1%5C&smnu=2%5C&article%5C_id=2447%5C&proceeding%5C_id=30 (cit. on pp. 60, 62).

Dai, H., Y. Li, C. Wang, R. Singh, P.-S. Huang, and P. Kohli (2019). "Learning Transferable Graph Exploration". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, pp. 2514–2525. URL: https://proceedings.neurips.cc/paper/2019/hash/afe434653a898da20044041262b3ac74-Abstract.html (cit. on p. 76).

Das, S., D. Dereniowski, and C. Karousatou (2015). "Collaborative exploration by energy-constrained mobile robots". In: *International Colloquium on Structural Information and Communication Complexity*. Springer, pp. 357–369 (cit. on p. 75).

Dasgupta, S. (2004). "Analysis of a greedy active learning strategy". In: *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*, pp. 337–344. URL: https://proceedings.neurips.cc/paper/2004/hash/c61fbef63df5ff317aecdc3670094472-Abstract.html (cit. on pp. 59, 60).

— (2005). "Coarse sample complexity bounds for active learning". In: *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada]*, pp. 235–242. URL: https://proceedings.neurips.cc/paper/2005/hash/6e82873a32b95af115de1c414a1849cb-Abstract.html (cit. on pp. 59, 63, 64).

— (2009). "The Two Faces of Active Learning". In: *Proceedings of the 12th International Conference on Discovery Science*, pp. 35–35 (cit. on p. 58).

Delalleau, O., Y. Bengio, and N. Le Roux (2005). "Efficient Non-Parametric Function Induction in Semi-Supervised Learning." In: *AISTATS*. Vol. 27, p. 100 (cit. on p. 43).

Deng, X. and C. H. Papadimitriou (1999). "Exploring an unknown graph". In: *Journal of Graph Theory* 32.3, pp. 265–297 (cit. on p. 75).

Dereniowski, D., Y. Disser, A. Kosowski, D. Pajkak, and P. Uznański (2013). "Fast collaborative graph exploration". In: *International Colloquium on Automata, Languages, and Programming*. Springer, pp. 520–532 (cit. on p. 75).

Devroye, L., L. Györfi, and G. Lugosi (2013). *A probabilistic theory of pattern recognition*. Vol. 31. Springer Science & Business Media (cit. on p. 63).

Dobrev, S., R. Královič, and E. Markou (2012). "Online graph exploration with advice". In: *International Colloquium on Structural Information and Communication Complexity*. Springer, pp. 267–278 (cit. on p. 75).

Dosovitskiy, A. and V. Koltun (2017). "Learning to Act by Predicting the Future". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=rJLS7qKel (cit. on pp. 74, 80, 81, 94).

Ducoffe, M. and F. Precioso (2018). "Adversarial active learning for deep networks: a margin based approach". In: *arXiv preprint arXiv:1802.09841* (cit. on pp. 58, 60, 67, 69).

Duncan, C. A., S. G. Kobourov, and V. A. Kumar (2006). "Optimal constrained graph exploration". In: *ACM Transactions on Algorithms (TALG)* 2.3, pp. 380–402 (cit. on p. 75).

Edwards, H. and A. J. Storkey (2017). "Towards a Neural Statistician". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=HJDBUF5le (cit. on p. 76).

Fukushima, K. (1980). "A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biol. Cybern.* 36, pp. 193–202 (cit. on p. 12).

Gal, Y. and Z. Ghahramani (2016). "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. Ed. by M.-F. Balcan and K. Q. Weinberger. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 1050–1059. URL: http://proceedings.mlr.press/v48/gal16.html (cit. on p. 60).

Gal, Y., R. Islam, and Z. Ghahramani (2017). "Deep Bayesian Active Learning with Image Data". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by D. Precup and Y. W. Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 1183–1192. URL: http://proceedings.mlr.press/v70/gal17a.html (cit. on pp. 58, 60, 67).

Ganu, G. and B. Kveton (2013). *Nearly Optimal Semi-Supervised Learning on Subgraphs* (cit. on p. 43).

Gasparetto, A. and A. Torsello (2015). "A statistical model of Riemannian metric variation for deformable shape analysis". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, pp. 1219–1228. DOI: 10.1109/CVPR.2015.7298726. URL: https://doi.org/10.1109/CVPR.2015.7298726 (cit. on pp. 17, 28, 30).

Geiger, A., P. Lenz, and R. Urtasun (2012). "Are we ready for autonomous driving? The KITTI vision benchmark suite". In: *2012 IEEE Conference* *on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*. IEEE Computer Society, pp. 3354–3361. DOI: 10.1109/CVPR.2012.6248074. URL: https://doi.org/10.1109/CVPR.2012.6248074 (cit. on pp. 45, 53).

Gissin, D. and S. Shalev-Shwartz (2019). “Discriminative active learning”. In: *arXiv preprint arXiv:1907.06347* (cit. on pp. 58, 60).

Golovin, D. and A. Krause (2010). “Adaptive Submodularity: A New Approach to Active Learning and Stochastic Optimization”. In: *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010*. Ed. by A. T. Kalai and M. Mohri. Omnipress, pp. 333–345. URL: http://colt2010.haifa.il.ibm.com/papers/COLT2010proceedings.pdf%5C#page=341 (cit. on pp. 59, 60).

Grover, A. and J. Leskovec (2016). “node2vec: Scalable Feature Learning for Networks”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. Ed. by B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi. ACM, pp. 855–864. DOI: 10.1145/2939672.2939754. URL: https://doi.org/10.1145/2939672.2939754 (cit. on p. 76).

Guan, M. G. (1962). “Graphic programming using odd or even points Chinese Mathematics”. In: *V ol* 1.2, pp. 73–2 (cit. on p. 75).

Haklay, M. and P. Weber (2008). “Openstreetmap: User-generated street maps”. In: *IEEE Pervasive Computing* 7.4, pp. 12–18 (cit. on p. 86).

Hanneke, S. (2007). “A bound on the label complexity of agnostic active learning”. In: *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20-24, 2007*. Ed. by Z. Ghahramani. Vol. 227. ACM International Conference Proceeding Series. ACM, pp. 353–360. DOI: 10.1145/1273496.1273541. URL: https://doi.org/10.1145/1273496.1273541 (cit. on p. 60).

Häusser, P., A. Mordvintsev, and D. Cremers (2017). “Learning by Association - A Versatile Semi-Supervised Training Method for Neural Networks”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, pp. 626–635. DOI: 10.1109/CVPR.2017.74. URL: https://doi.org/10.1109/CVPR.2017.74 (cit. on p. 42).

Higashikawa, Y., N. Katoh, S. Langerman, and S.-i. Tanigawa (2014). “Online graph exploration algorithms for cycles and trees by multiple searchers”. In: *Journal of Combinatorial Optimization* 28.2, pp. 480–495 (cit. on p. 75).

Himmelsbach, M., T. Luettel, and H.-J. Wuensche (2009). “Real-time object classification in 3D point clouds using point feature histograms”. In: *IEEE/RSJ IROS*, pp. 994–1000 (cit. on p. 53).

Houlsby, N., F. Huszár, Z. Ghahramani, and M. Lengyel (2011). “Bayesian active learning for classification and preference learning”. In: *arXiv preprint arXiv:1112.5745* (cit. on pp. 58, 60, 67, 69).

Joshi, A. J., F. Porikli, and N. Papanikolopoulos (2009). “Multi-class active learning for image classification”. In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, pp. 2372–2379. DOI: 10.1109/CVPR.2009.5206627. URL: https://doi.org/10.1109/CVPR.2009.5206627 (cit. on p. 58).

Kääriäinen, M. (2005). “Generalization Error Bounds Using Unlabeled Data”. In: *Learning Theory, 18th Annual Conference on Learning Theory, COLT 2005, Bertinoro, Italy, June 27-30, 2005, Proceedings*. Ed. by P. Auer and R. Meir. Vol. 3559. Lecture Notes in Computer Science. Springer, pp. 127–142. DOI: 10.1007/11503415_9. URL: https://doi.org/10.1007/11503415%5C_9 (cit. on p. 63).

Kalyanasundaram, B. and K. R. Pruhs (1994). “Constructing competitive tours from local information”. In: *Theoretical Computer Science* 130.1, pp. 125–138 (cit. on p. 75).

Kingma, D. P. and J. Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. URL: http://arxiv.org/abs/1412.6980 (cit. on p. 82).

Kingma, D. P., S. Mohamed, D. J. Rezende, and M. Welling (2014). “Semi-supervised Learning with Deep Generative Models”. In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, pp. 3581–3589. URL: https://proceedings.neurips.cc/paper/2014/hash/d523773c6b194f37b938d340d5d02232-Abstract.html (cit. on p. 42).

Kipf, T. N. and M. Welling (2017). “Semi-Supervised Classification with Graph Convolutional Networks”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=SJU4ayYgl (cit. on pp. 76, 81).

Kirsch, A., J. van Amersfoort, and Y. Gal (2019). “BatchBALD: Efficient and Diverse Batch Acquisition for Deep Bayesian Active Learning”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by H. M. Wallach, H. Larochelle,

A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, pp. 7024–7035. URL: https://proceedings.neurips.cc/paper/2019/hash/95323660ed2124450caaac2c46b5ed90-Abstract.html (cit. on pp. 58, 60).

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. Ed. by P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, pp. 1106–1114. URL: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html (cit. on p. 8).

Kruse, T., A. K. Pandey, R. Alami, and A. Kirsch (2013). "Human-aware robot navigation: A survey". In: *Robotics and Autonomous Systems* 61.12, pp. 1726–1743 (cit. on p. 32).

Kruskal, J. B. (1978). *Multidimensional scaling*. 11. Sage (cit. on p. 65).

Lacasse, A., F. Laviolette, M. Marchand, P. Germain, and N. Usunier (2006). "PAC-Bayes Bounds for the Risk of the Majority Vote and the Variance of the Gibbs Classifier". In: *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*. Ed. by B. Schölkopf, J. C. Platt, and T. Hofmann. MIT Press, pp. 769–776. URL: https://proceedings.neurips.cc/paper/2006/hash/779efbd24d5a7e37ce8dc93e7c04d572-Abstract.html (cit. on p. 70).

Lakshminarayanan, B., D. M. Roy, and Y. W. Teh (2014). "Mondrian Forests: Efficient Online Random Forests". In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, pp. 3140–3148. URL: https://proceedings.neurips.cc/paper/2014/hash/d1dc3a8270a6f9394f88847d7f0050cf-Abstract.html (cit. on p. 54).

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press (cit. on p. 35).

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4, pp. 541–551 (cit. on p. 12).

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324 (cit. on p. 50).

Lee, J., Y. Lee, J. Kim, A. R. Kosiorek, S. Choi, and Y. W. Teh (2019). "Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Ed. by K. Chaudhuri and R. Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 3744–3753. URL: http://proceedings.mlr.press/v97/lee19d.html (cit. on p. 76).

Litman, R., A. Bronstein, M. Bronstein, and U. Castellani (2014). "Supervised learning of bag-of-features shape descriptors using sparse coding". In: *Computer Graphics Forum*. Vol. 33. Wiley Online Library, pp. 127–136 (cit. on pp. 16, 17, 22, 23, 27, 29, 30).

Masci, J., D. Boscaini, M. Bronstein, and P. Vandergheynst (2015). "Geodesic Convolutional Neural Networks on Riemannian Manifolds". In: *ICCV Workshops* (cit. on p. 17).

McCulloch, W. S. and W. Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133 (cit. on p. 8).

Megow, N., K. Mehlhorn, and P. Schweitzer (2012). "Online graph exploration: New results on old and new algorithms". In: *Theoretical Computer Science* 463, pp. 62–72 (cit. on p. 75).

Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean (2013). "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. Ed. by C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, pp. 3111–3119. URL: https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html (cit. on p. 76).

Miyazaki, S., N. Morimoto, and Y. Okabe (2009). "The online graph exploration problem on restricted graphs". In: *IEICE transactions on information and systems* 92.9, pp. 1620–1627 (cit. on p. 75).

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, pp. 529–533 (cit. on pp. 33, 34, 95).

Narr, A., R. Triebel, and D. Cremers (2016). "Stream-based active learning for efficient and adaptive classification of 3d objects". In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 227–233 (cit. on pp. 53, 54).

Ng, A. Y. and S. J. Russell (2000). "Algorithms for Inverse Reinforcement Learning". In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*. Ed. by P. Langley. Morgan Kaufmann, pp. 663–670 (cit. on p. 33).

Nguyen, V. C., W. S. Lee, N. Ye, K. M. A. Chai, and H. L. Chieu (2013). "Active Learning for Probabilistic Hypotheses Using the Maximum Gibbs Error Criterion". In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. Ed. by C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, pp. 1457–1465. URL: https://proceedings.neurips.cc/paper/2013/hash/fb89705ae6d743bf1e848c206e16a1d7-Abstract.html (cit. on pp. 59–61).

Okal, B. and K. O. Arras (2016). "Learning socially normative robot navigation behaviors with bayesian inverse reinforcement learning". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2889–2895 (cit. on p. 32).

Omohundro, S. M. (1989). *Five balltree construction algorithms*. International Computer Science Institute Berkeley (cit. on p. 49).

Perozzi, B., R. Al-Rfou, and S. Skiena (2014). "DeepWalk: online learning of social representations". In: *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*. Ed. by S. A. Macskassy, C. Perlich, J. Leskovec, W. Wang, and R. Ghani. ACM, pp. 701–710. DOI: 10.1145/2623330.2623732. URL: https://doi.org/10.1145/2623330.2623732 (cit. on p. 76).

Pickup, D., X. Sun, P. L. Rosin, R. R. Martin, Z. Cheng, Z. Lian, M. Aono, A. Ben Hamza, A. Bronstein, M. Bronstein, S. Bu, U. Castellani, S. Cheng, V. Garro, A. Giachetti, A. Godil, J. Han, H. Johan, L. Lai, B. Li, C. Li, H. Li, R. Litman, X. Liu, Z. Liu, Y. Lu, A. Tatsuma, and J. Ye (2014). "SHREC'14 track: Shape Retrieval of Non-Rigid 3D Human Models". In: *Proceedings of the 7th Eurographics workshop on 3D Object Retrieval*. EG 3DOR'14. Eurographics Association. URL: https://www.cs.cf.ac.uk/shaperetrieval/shrec14/ (cit. on pp. 17, 27, 30).

Pinkall, U. and K. Polthier (1993). "Computing discrete minimal surfaces and their conjugates". In: *Experimental mathematics* 2.1, pp. 15–36 (cit. on p. 20).

Pinsler, R., J. Gordon, E. T. Nalisnick, and J. M. Hernández-Lobato (2019). "Bayesian Batch Active Learning as Sparse Subset Approximation". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December*

8-14, 2019, Vancouver, BC, Canada. Ed. by H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, pp. 6356–6367. URL: https://proceedings.neurips.cc/paper/2019/hash/84c2d4860a0fc27bcf854c444fb8b400-Abstract.html (cit. on p. 60).

Precup, D., R. S. Sutton, and S. Singh (1998). "Theoretical results on reinforcement learning with temporally abstract options". In: *European conference on machine learning*. Springer, pp. 382–393 (cit. on pp. 82, 87).

Ramachandran, D. and E. Amir (2007). "Bayesian inverse reinforcement learning". In: *Urbana* 51.61801, pp. 1–4 (cit. on p. 33).

Ratliff, N. D., J. A. Bagnell, and M. Zinkevich (2006). "Maximum margin planning". In: *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*. Ed. by W. W. Cohen and A. W. Moore. Vol. 148. ACM International Conference Proceeding Series. ACM, pp. 729–736. DOI: 10.1145/1143844.1143936. URL: https://doi.org/10.1145/1143844.1143936 (cit. on p. 33).

Reuter, M., S. Biasotti, D. Giorgi, G. Patanè, and M. Spagnuolo (2009). "Discrete Laplace-Beltrami operators for shape analysis and segmentation". In: *IEEE International Conference on Shape Modelling and Applications* (cit. on p. 20).

Rodolà, E., S. R. Bulò, T. Windheuser, M. Vestner, and D. Cremers (2014). "Dense Non-rigid Shape Correspondence Using Random Forests". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, pp. 4177–4184. DOI: 10.1109/CVPR.2014.532. URL: https://doi.org/10.1109/CVPR.2014.532 (cit. on p. 17).

Roger A. Horn, C. R. J. (2013). *Matrix analysis*. New York: Cambridge University Press (cit. on p. 91).

Rosenblatt, F. (1961). *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. Cornell Aeronautical Lab Inc Buffalo NY (cit. on p. 10).

Rosenkrantz, D. J., R. E. Stearns, and P. M. Lewis II (1977). "An analysis of several heuristics for the traveling salesman problem". In: *SIAM journal on computing* 6.3, pp. 563–581 (cit. on p. 75).

Rumelhart, D. E., G. Hinton, and R. Williams (1986). "Learning internal representations by error propagation". In: *Foundations. MIT Press, Cambridge* (cit. on p. 10).

Rustamov, R. M. (2007). "Laplace–Beltrami eigenfunctions for deformation invariant shape representation". In: *Proceedings of the fifth Eurographics symposium on Geometry processing*. Eurographics Association, pp. 225–233 (cit. on p. 28).

Saffari, A., M. Godec, T. Pock, C. Leistner, and H. Bischof (2010). “Online multi-class LPBoost”. In: *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*. IEEE Computer Society, pp. 3570–3577. DOI: 10.1109/CVPR.2010.5539937. URL: https://doi.org/10.1109/CVPR.2010.5539937 (cit. on p. 54).

Saffari, A., C. Leistner, J. Santner, M. Godec, and H. Bischof (2009). “On-line random forests”. In: *Computer Vision Workshops at International Conference on Computer Vision (ICCV)*, pp. 1393–1400 (cit. on p. 54).

Sajjadi, M., M. Javanmardi, and T. Tasdizen (2016). “Regularization With Stochastic Transformations and Perturbations for Deep Semi-Supervised Learning”. In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, pp. 1163–1171. URL: https://proceedings.neurips.cc/paper/2016/hash/30ef30b64204a3088a26bc2e6ecf7602-Abstract.html (cit. on p. 42).

Sener, O. and S. Savarese (2018). “Active Learning for Convolutional Neural Networks: A Core-Set Approach”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=H1aIuk-RW (cit. on pp. 58, 60, 67, 69).

Sun, J., M. Ovsjanikov, and L. Guibas (2009). “A Concise and Provably Informative Multi-Scale Signature Based on Heat Diffusion”. In: *Computer graphics forum*. Vol. 28. Wiley Online Library, pp. 1383–1392 (cit. on pp. 17, 21).

Tehrani, H., Q. H. Do, M. Egawa, K. Muto, K. Yoneda, and S. Mita (2015). “General behavior and motion model for automated lane change”. In: *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, pp. 1154–1159 (cit. on p. 32).

Tehrani, H., K. Muto, K. Yoneda, and S. Mita (2014). “Evaluating human & computer for expressway lane changing”. In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, pp. 382–387 (cit. on p. 32).

Tosh, C. and S. Dasgupta (2017). “Diameter-Based Active Learning”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. Ed. by D. Precup and Y. W. Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 3444–3452. URL: http://proceedings.mlr.press/v70/tosh17a.html (cit. on pp. 59, 60, 64, 67).

Triebel, R., K. Arras, R. Alami, L. Beyer, S. Breuers, R. Chatila, M. Chetouani, D. Cremers, V. Evers, M. Fiore, H. Hung, O. A. I. Ramírez, M. Joosse, H. Khambhaita, T. Kucner, B. Leibe, A. J. Lilienthal, T. Linder, M. Lohse, M. Magnusson, B. Okal, L. Palmieri, U. Rafi, M. van Rooij, and L. Zhang (2015). "SPENCER: A Socially Aware Service Robot for Passenger Guidance and Help in Busy Airports". In: *Proc. Field and Service Robotics (FSR)* (cit. on p. 32).

Tsitsiklis, J. N. and B. Van Roy (1997). "An analysis of temporal-difference learning with function approximation". In: *IEEE transactions on automatic control* 42.5, pp. 674–690 (cit. on p. 34).

Valko, M., B. Kveton, L. Huang, and D. Ting (2010). "Online Semi-Supervised Learning on Quantized Graphs". In: *UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, July 8-11, 2010.* Ed. by P. Grünwald and P. Spirtes. AUAI Press, pp. 606–614. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1%5C&smnu=2%5C&article%5C_id=2077%5C&proceeding%5C_id=26 (cit. on p. 43).

Vapnik, V. (1998). "Statistical learning theory new york". In: *NY: Wiley* (cit. on p. 6).

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory.* New York, NY, USA: Springer-Verlag New York, Inc. ISBN: 0-387-94559-8 (cit. on p. 42).

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA.* Ed. by I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (cit. on p. 82).

Velickovic, P., G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio (2018). "Graph Attention Networks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.* OpenReview.net. URL: https://openreview.net/forum?id=rJXMpikCZ (cit. on p. 81).

Vinyals, O., M. Fortunato, and N. Jaitly (2015). "Pointer Networks". In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada.* Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, pp. 2692–2700. URL: https://proceedings.

neurips.cc/paper/2015/hash/29921001f2f04bd3baee84a12e98098f-Abstract.html (cit. on pp. 75, 76).

Weinberger, K. Q., J. Blitzer, and L. K. Saul (2005a). “Distance Metric Learning for Large Margin Nearest Neighbor Classification”. In: *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada]*, pp. 1473–1480. URL: https://proceedings.neurips.cc/paper/2005/hash/a7f592cef8b130a6967a90617db5681b-Abstract.html (cit. on pp. 17, 26).

— (2005b). “Distance Metric Learning for Large Margin Nearest Neighbor Classification”. In: *Advances in Neural Information Processing Systems 18 [Neural Information Processing Systems, NIPS 2005, December 5-8, 2005, Vancouver, British Columbia, Canada]*, pp. 1473–1480. URL: https://proceedings.neurips.cc/paper/2005/hash/a7f592cef8b130a6967a90617db5681b-Abstract.html (cit. on p. 24).

Werling, M., J. Ziegler, S. Kammel, and S. Thrun (2010). “Optimal trajectory generation for dynamic street scenarios in a frenet frame”. In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, pp. 987–993 (cit. on p. 32).

Windheuser, T., M. Vestner, E. Rodolà, R. Triebel, and D. Cremers (2014). “Optimal Intrinsic Descriptors for Non-Rigid Shape Analysis”. In: *British Machine Vision Conference, BMVC 2014, Nottingham, UK, September 1-5, 2014*. Ed. by M. F. Valstar, A. P. French, and T. P. Pridmore. BMVA Press. URL: http://www.bmva.org/bmvc/2014/papers/paper015/index.html (cit. on p. 17).

Wulfmeier, M., P. Ondruska, and I. Posner (2015). “Deep Inverse Reinforcement Learning”. In: *arXiv preprint arXiv:1507.04888* (cit. on p. 33).

Yamauchi, B. (1997). “A frontier-based approach for autonomous exploration”. In: *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA’97.’Towards New Computational Principles for Robotics and Automation’*. IEEE, pp. 146–151 (cit. on p. 74).

Zaheer, M., S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola (2017). “Deep Sets”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, pp. 3391–3401. URL: https://proceedings.neurips.cc/paper/2017/hash/f22e4747da1aa27e363d86d40ff442fe-Abstract.html (cit. on p. 76).

Zhang, C., S. Bengio, M. Hardt, B. Recht, and O. Vinyals (2017). “Understanding deep learning requires rethinking generalization”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=Sy8gdB9xx (cit. on p. 59).

Zhang, M. and Y. Chen (2018). “Link Prediction Based on Graph Neural Networks”. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, pp. 5171–5181. URL: https://proceedings.neurips.cc/paper/2018/hash/53f0d7c537d99b3824f0f99d62ea2428-Abstract.html (cit. on p. 76).

Zhu, X. (2005). “Semi-Supervised Learning with Graphs”. PhD thesis. Language Technologies Institute, School of Computer Science, Carnegie Mellon University (cit. on pp. 44, 46).

Zhu, X. and Z. Ghahramani (2002). *Learning from Labeled and Unlabeled Data with Label Propagation*. Tech. rep. CMU-CALD-02-107. Pittsburgh: Carnegie-Mellon Univ. (cit. on pp. 42, 43, 45, 46, 55, 89).

Zhu, X., Z. Ghahramani, and J. D. Lafferty (2003). “Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions”. In: *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*. Ed. by T. Fawcett and N. Mishra. AAAI Press, pp. 912–919. URL: http://www.aaai.org/Library/ICML/2003/icml03-118.php (cit. on p. 43).

Ziebart, B. D., A. L. Maas, J. A. Bagnell, and A. K. Dey (2008). “Maximum Entropy Inverse Reinforcement Learning.” In: *AAAI*, pp. 1433–1438 (cit. on p. 33).