SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Data Engineering and Analytics

Sampling weights of auto-encoder networks

Safa Sadiq

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Data Engineering and Analytics

Sampling weights of auto-encoder networks

Abtastgewichte von Auto-Encoder-Netzwerken

Author: Safa Sadiq
Advisor: Dr. Felix Dietrich
Submission Date: 15th March 2024

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15th March 2024 Safa Sadiq

Acknowledgments

I would like to express my deepest gratitude to my advisor, Dr Felix Dietrich, for his guidance, encouragement, and invaluable insights throughout the research process and for putting up with me canceling meetings last minute. His unwavering support and expertise have been instrumental in shaping this thesis. Without his constructive feedback and critical evaluation, I would never have been able to get this thesis in the shape it is in.

I am grateful to my colleagues and friends for their encouragement and moral support. I would especially like to thank Marcel and Ron for putting up with me even when I was tired and sleep-deprived at work and for their constant reassurance that it would all work out. Thanks to Katharina who taught me how to make the best cup of coffee that would keep me up all night. A huge shoutout to Amna and Ayesha for encouraging me to push harder, for showing up to the library to work with me even on weekends, and for making me feel better about my workload by telling me how theirs is much worse. You were my personal cheerleaders.

A special thanks to my family who are more excited about me finishing this thesis than I am myself. Like the swiss roll, your love and support is truly boundless.

This thesis would not have been possible without the support and contributions of you all, it truly takes a village!

Abstract

Autoencoders are an unsupervised learning technique that leverage representation learning to reconstruct the original data as closely as possible. Traditionally, an autoencoder network is trained to minimize the reconstruction loss between the training data and the data generated as the output of the network, using iterative gradient-based methods. There has been research in the realm of neural networks with sampled weights and biases, instead of iteratively trained ones. The weights and biases are sampled either using a data-agnostic or data-driven approach, which need not be updated during the training process, resulting in much shorter training times. The thesis aims to extend the use of data-driven sampling in neural networks to an autoencoder setting. The "Sampling Where It Matters (SWIM)" [1] algorithm is used to sample the weights and biases of the network that are then fixed and need not be iteratively trained. However, simply sampling points from the input space does not necessarily consider the intrinsic structure and complexities in the data. Hence, kernel representation learning using a contrastive loss function is combined with data-driven sampling to learn an embedding that can be used to reconstruct the data accurately. The approach is tested on benchmark image datasets including MNIST and CIFAR-10 and shown to have promising results. The approach is also shown to have a form similar to kernel canonical correlation analysis.

Kurzfassung

Autoencoder sind eine unbeaufsichtigte Lerntechnik, die das Repräsentationslernen nutzt, um die Originaldaten so genau wie möglich zu rekonstruieren. Traditionell wird ein Autoencoder-Netzwerk mithilfe iterativer, auf Gradienten basierender Methoden trainiert, um den Rekonstruktionsverlust zwischen den Trainingsdaten und den als Ausgabe des Netzwerks generierten Daten zu minimieren. Es gibt Forschungen im Bereich neuronaler Netze mit abgetasteten Gewichten und Verzerrungen anstelle von iterativ trainierten. Die Gewichtungen und Verzerrungen werden entweder mit einem datenunabhängigen oder datengesteuerten Ansatz erfasst, der während des Trainingsprozesses nicht aktualisiert werden muss, was zu deutlich kürzeren Trainingszeiten führt. Ziel der Arbeit ist es, den Einsatz datengesteuerter Abtastung in neuronalen Netzen auf eine Autoencoder-Umgebung auszudehnen. Der Algorithmus „Sampling Where It Matters (SWIM)“ [1], wird verwendet, um die Gewichte und Verzerrungen des Netzwerks abzutasten, die dann festgelegt werden und nicht iterativ trainiert werden müssen. Das einfache Abtasten von Punkten aus dem Eingaberaum berücksichtigt jedoch nicht unbedingt die intrinsische Struktur und Komplexität der Daten. Daher wird das Lernen der Kernel-Repräsentation unter Verwendung einer kontrastiven Verlustfunktion mit datengesteuerter Stichprobenziehung kombiniert, um eine Einbettung zu erlernen, die zur genauen Rekonstruktion der Daten verwendet werden kann. Der Ansatz wird an Benchmark-Bilddatensätzen wie MNIST und CIFAR-10 getestet und zeigt vielversprechende Ergebnisse. Es wird auch gezeigt, dass der Ansatz eine ähnliche Form wie die kanonische Kernel-Korrelationsanalyse hat.

Contents

1 Introduction

An autoencoder is a type of neural network that is commonly used for unsupervised learning tasks, particularly in deep learning. Autoencoders are categorized as generative models and are used to obtain effective data representations, also known as embeddings, by identifying key elements or patterns in the input. The ultimate goal of an autoencoder is to reconstruct the data as closely as possible to the input using these embeddings. The encoder part of the network embeds the data in a low-dimensional space, while the decoder reconstructs it from that space. Traditionally, an autoencoder network is trained to minimize the reconstruction loss between the training data and the data generated as the output of the network, using iterative gradient-based methods. However, recent studies have explored neural networks with sampled weights and biases instead of iteratively trained ones [1, 2, 3, 4, 5]. The weights and biases are sampled either using a data-agnostic or a data-driven approach and do not need to be updated during the training process, resulting in much shorter training times.

The aim of this thesis is to extend the use of sampled neural networks to an autoencoder setting. However, simply sampling points from the input space does not necessarily take into account the intrinsic structure of the data. To overcome this limitation, kernel representation learning is used in combination with data-driven sampling. The input data is first mapped to a higher-dimensional feature space using a sampled linear network. The linear network comprises of a single dense layer with a non-linear activation function. The weights and biases of this layer are sampled via a data-driven scheme. Mapping the data to a higher-dimensional space using a sampled network not only provides us with a good feature representation for the input data, but the use of a non-linear activation function introduces nonlinearity in an otherwise linear regime. The input data is then mapped from this high-dimensional feature space into the kernel space using contrastive kernel learning. The method does not expect the kernel to reveal the underlying structure of the data in the feature space, but rather just finds a linear transformation that preserves the distances between the neighbors and pulls the non-neighbors apart. The kernel space can be reduced to a lower dimension, which allows us to find a low-dimensional embedding of the original data. A modified version of the closed-form solution for a lower-dimensional embedding suggested in [6] is used, along with an explanation

suggesting its semantic similarity to Kernel canonical correlation analysis [7]. The encoder comprises of the linear network followed by the contrastive kernel paradigm mapping the data from the input to a low-dimensional kernelized embedding. The kernelized embedding is then used as the input data to the decoder network with weights and biases sampled again using a data-driven scheme that samples these parameters close to the target function that reconstructs the data points [1].

The proposed approach is validated on benchmark datasets, including MNIST and CIFAR10. The performance is evaluated using metrics tailored to the nature of these datasets, including pixel-wise error, visual inspection of the reconstructed images, and classification accuracy on the reconstructed images.

The remainder of this thesis is structured in the following way: Section 2 discusses the state of the art, focusing mainly on the SWIM method used to sample the parameters of the network in this study. We also discuss other variations of sampled and kernel autoencoders and representation learning frameworks. Section 3 provides an overview of the methodology used in this study along with a theoretical explanation of its link to kernel canonical correlation analysis, and Section 4 presents the experiments conducted and their results. We wrap up with a discussion section where we offer critical insights, address limitations, and suggest avenues for future research.

2 Related Work

2.1 Sampling in feed-forward networks

2.1.1 Data Agnostic Methods

Sampling weights and biases in neural networks using a data agnostic distribution such as a normal distribution, has been studied extensively, with the most common frameworks being the extreme learning machine (ELM) and the random projection networks [2, 8, 9]. The weights between the input and hidden layers as well as the hidden layer's biases are randomly assigned by the network, and these parameters are frozen during training. This approach prioritizes random or uniform sampling methods, enabling the network to effectively handle various data distributions without extensive pre-processing. Non-linearity is introduced in the network via the hidden layer's nonlinear activation functions. The input $x \in \mathbb{R}^{Nxd}$ is mapped onto the hidden layer by linearly combining l nonlinear transformations of the original input, where l is the number of hidden layers. For a network with only one hidden layer, the hidden layer representation is calculated as

$$h_m(x) = g(a_m^T(x) + b_m), \tag{2.1}$$

where g is a non-linear activation function and a and b are the sampled weights and biases respectively.

An illustration of a basic model can be seen in Figure 2.1 with two input points, a single hidden layer, and a single output.

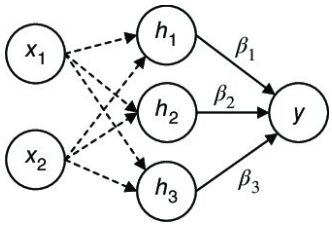


Figure 2.1: An architecture of a sampled network with two inputs, three hidden nodes, and one output [10].

The matrix H is the matrix of hidden layer representations and is denoted as

$$H = \begin{bmatrix} h_1(x_1) & \dots & h_B(x_1) \\ \vdots & \ddots & \vdots \\ h_1(x_N) & \dots & h_B(x_N) \end{bmatrix}.$$

The weights between the output layer and hidden layer are the sole parameters that need to be learned and can be formulated as a standard regularized least-squares problem that in closed form can be written as

$$\beta^* = (H^T H + \lambda I)^{-1} H^T y, \tag{2.2}$$

where I is the identity matrix, y are the labels of the dataset and λ is the regularization parameter. Because such networks learn without iteration, they have much faster convergence than standard algorithms. Furthermore, random hidden nodes claim to be able to approximate any function universally [2]. According to theoretical research, sampled networks are less likely to get stuck at the local optimum and hence more likely than traditional networks to attain the globally optimal solution [10]. By treating each sample equally during the training phase, they can learn robust representations that are not overly tailored to specific datasets or biases present in the training data. As a result, the trained model can perform well on unseen data from various sources, making them highly versatile. Such networks are frequently used in a range of learning problems, including classification, regression, clustering, and feature mapping, because of their excellent generalization capabilities and superior training time. To increase the stability and generalizability for particular applications, several modifications have been proposed [10, 11].

2.1.2 Data Driven Method (SWIM)

The SWIM (Sampling Where It Matters) algorithm [1] is a data-driven sampling strategy for fully connected neural networks. It is founded on the principle of random feature models, but rather than using a data-agnostic distribution, it samples the network parameters using the data points in the input and output training data. The algorithm samples the network's weights and biases using pairs of data points from the input space. Each weight and bias pair in the neural network is entirely characterized by two input space points. The weights are determined by the difference between the two points, while the bias is the inner product of the weight and one of the two points. More specifically the weights and biases of the sampled network are defined as

$$w_{l,i} = s_1 \frac{x^{(2)}_{l-1,i} - x^{(1)}_{l-1,i}}{\|x^{(2)}_{l-1,i} - x^{(1)}_{l-1,i}\|^2}, \quad b_{l,i} = \langle w_{l,i}, x^{(1)}_{l-1,i} \rangle + s_2, \tag{2.3}$$

where $(x^{(1)}_{0,i}, x^{(2)}_{0,i})_{i=1}^{N_l}$ are pairs of points sampled over $X \times X$, $s_1, s_2 \in \mathbb{R}$ are constants related to the activation function, and $x^{(j)}_{l-1,i} = \Phi^{(l-1)}(x^{(j)}_{0,i})$ for $j = 1, 2$, assuming $x^{(1)}_{l-1,i} \neq x^{(2)}_{l-1,i}$ [1].

This data-driven sampling technique guarantees that the network parameters are built based on particular attributes of input data, leading to precise and width-efficient approximations with no need for gradient-based optimizations. The sampled networks formed using the SWIM method have shown to be universal approximators, which means they can estimate any continuous function accurately provided that the layers of the network are composed of a very large number of neurons. Moreover, this method also has several advantages over the data-agnostic sampling methods. They are more accurate because the weights and biases are derived from data points and as such incorporate information about the specific data in the network parameters as shown in Figure 2.2. Knowing which points were sampled for constructing the weights and biases also leads to the model being more interpretable. The SWIM method demonstrates robustness to changing random seeds, ensuring consistent performance across different runs. This robustness is valuable in practical applications where reproducibility and stability are important factors.

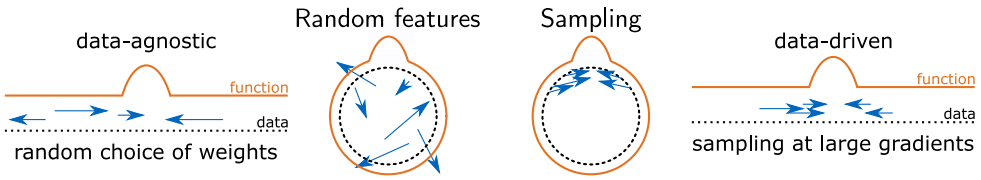


Figure 2.2: Random feature models choose weights in a data-agnostic way, sampling them where it matters: at large gradients. The arrows illustrate where the network weights are placed [1].

The paper by Bolager et al. [1] presents examples and experiments that demonstrate the effectiveness of SWIM. These experiments include approximating Barron functions, constructing deep neural architectures for PDE solution operators, and transfer learning in image classification tasks. The results show that SWIM provides higher accuracy and robustness to changing random seeds, and is much faster than iterative training. These features make it a promising approach for various applications in machine learning and scientific computing. The SWIM method stands out

for its accuracy, interpretability, efficiency, and robustness, making it a promising method for training neural networks in various machine learning tasks.

2.2 Sampling in Autoencoders

When applying sampled weights and biases to networks, it can enhance the network's ability to learn meaningful representations of the input data, which is vital for various applications such as data denoising, dimensionality reduction, and feature learning. Autoencoders are used extensively for these tasks and hence many studies have been conducted in the realm of autoencoders with sampled network parameters.

2.2.1 Sampled Autoencoders

The paper by Kasun et al. [12] draws inspiration from the works of Geoffrey Hinton and Pascal Vincent, who demonstrated the effectiveness of restricted Boltzmann machines and autoencoders for feature engineering. The paper emphasizes the importance of orthogonal random weights and biases in extreme learning machine (ELM) autoencoders to project input data into a different or equal dimension space, as per the Johnson-Lindenstrauss lemma. The representation capability of an extreme learning machine based autoencoder is suggested to offer a promising solution compared to traditional deep networks, showcasing better performance in various applications.

The paper by Sun et al. [4] discusses the concept of integrating the principles of extreme learning machines with an autoencoder like architecture to create a new variant called generalized extreme learning machine autoencoder (GELM-AE). The GELM-AE is designed to extract meaningful features from unlabeled data by incorporating manifold regularization, and it outperforms other unsupervised learning algorithms. Additionally, the paper introduces a new deep neural network called multilayer generalized extreme learning machine autoencoder (ML-GELM), which stacks several GELM-AEs to detect more abstract representations. The experiments conducted on real-world datasets demonstrate the superior performance of GELM-AE and ML-GELM compared to other techniques, with lower time complexity.

There have also been studies conducted in the realm of using sampled autoencoders for learning sparse representations of the input data which can be used for other downstream tasks such as clustering or initialization of deep neural networks. The paper by Ding et al. [13] introduces the notion of using an extreme learning machine as an autoencoder for feature learning. The embedded characteristics are employed in an unsupervised extreme learning machine for clustering. The

paper proves that the features extracted using the extreme learning machine autoencoder recover the principal components that represent original samples and that the method outperforms various traditional clustering algorithms. Such works lay the foundation for using extreme learning machine autoencoders to capture lower dimensional embeddings of high dimensional datasets, which is essential to obtain a good reconstruction.

The work done by Zhu et al. [14] discusses the implementation of a novel unsupervised representation model, the hierarchical extreme learning machine network, for learning representations from massive unlabelled data. The proposed network utilizes the weights sampled using a uniform distribution to learn local receptive fields at each layer and uses the outputs from each subsequent layer to form a more comprehensive representation. Additionally, the method reduces rapid information loss in multi-layer unsupervised learning and incorporates local contrast normalization and whitening to enhance performance. The resulting trans-layer representations are processed into block histograms with binary hashing to produce translation and rotation invariant representations, used for high-level tasks such as classification and detection. Experimental results demonstrate the effectiveness of the hierarchical ELM network, achieving 65.97% accuracy on the Caltech 101 task and 99.45% accuracy on the standard MNIST data set.

The main problem associated with all of the approaches discussed in this section is that the sampling for the weights and biases of the network is done using a data-agnostic distribution and does not incorporate any aspect of the training data. As a result of sampling these network parameters independently from the distribution of the input data, the network will require a very large number of samples before it can find a useful embedding space.

2.2.2 Kernel Autoencoders

Traditionally sampled networks may not capture complex data representations effectively, especially in cases where the data is highly non-linear or when a deeper hierarchy of features is necessary. Kernel methods allow sampled networks to model non-linear relationships between input and output variables. By using a kernel function, sampled networks can implicitly map the input data into a higher-dimensional feature space where linear separation might be possible. Furthermore, they can handle a wide range of data types and structures, including non-linear, non-parametric, and even non-stationary data. This makes them suitable for various machine learning tasks where linear methods may not be effective.

In theory, the kernels compute a measure of similarity or distance between pairs of data points in a high-dimensional space, known as the feature space. The choice of

kernel function determines how this similarity is computed. Data points are mapped to a Reproducing Kernel Hilbert Space (RKHS) using a kernel through a process called the kernel trick. Given a dataset $x \in \mathbb{R}^{Nxd}$, the kernel function is applied to pairs of data points to compute a symmetric positive semi-definite kernel matrix K where

$$K_{ij} = K(x_i, x_j). \tag{2.4}$$

The kernel trick exploits the property of Mercer's theorem [15], which states that any positive semi-definite kernel corresponds to an inner product in some (possibly infinite-dimensional) feature space. Instead of explicitly computing the transformed feature vectors $\phi(x_i)$ in the high-dimensional feature space, the kernel function $K(x_i, x_j)$ implicitly captures the inner product of the feature vectors $\phi(x_i)$ and $\phi(x_j)$ in the RKHS. As a result of this property, any algorithm that relies on the inner products can be implicitly mapped to the RKHS simply by evaluating the kernel K [6]. Hence $H^T H$ in Equation 2.2 can be replaced by the kernel matrix Φ such that

$$\Phi = H^T H,$$

where,

$$\Phi_{k,j} = K(x_k, x_j), \qquad k, j = 1 \; to \; N. \tag{2.5}$$

As demonstrated above, kernels can naturally be applied to the randomly sampled network regime. Owing to this, there has been research conducted in the domain of incorporating kernel methods in various sampled networks. The paper by Wong et al. [16] suggests incorporating kernel methods into the extreme learning machine framework to enhance its representational power. Kernel methods allow for nonlinear transformations of the input data into a higher-dimensional space, where linear operations can be performed efficiently. By combining kernel methods with multi-layer structures, the proposed approach aims to learn more expressive and discriminatory feature representations from the data. An added benefit of this approach is that it eliminates the need to manually determine the number of hidden nodes needed for each layer, reducing the number of hyperparameters in the model. As kernel matrices are guaranteed to be invertible, it also allows for the exact inverse to be computed, instead of the pseudo inverse, further improving the performance.

A major drawback of using kernel methods is the storage and computational issues that arise as kernels have a time and space complexity of $O(N^2)$ where N is the size of the data. This issue is addressed in the paper by Vong et al. [17]. The paper proposes a new method called ML-EKM-ELM that encodes every hidden layer in the form of an approximate empirical kernel map with a much smaller size, which is very efficient for learning representations from large-scale data. The Nyström method is adopted in the work to generate the approximate empirical kernel map

due to its efficiency in many large-scale machine learning problems. The paper also compares ML-EKM-ELM with other existing deep neural networks over benchmark datasets and concludes that ML-EKM-ELM is a compact model with a faster solution that maintains satisfactory accuracy.

The paper by Kuss et al. [18] proposes a novel kernel-based framework for building autoencoders based on vector-valued reproducing kernel Hilbert spaces (vv-RKHSs) that learns an optimal kernel embedding. The proposed algorithms rely on the form taken by the minimizers, revealed by a dedicated representer theorem. The paper provides a theoretical analysis of the model, including a generalization bound and a strong connection with kernel principal component analysis (KPCA). The proposed kernel autoencoder's performance is tested on both simulated data and real labeled graphs and shows promising results.

We draw inspiration from these works but incorporate a different flavor of kernel learning in our work, namely contrastive kernel representation learning

2.3 Representation Learning

Representation learning is a type of machine learning technique where the algorithm learns to automatically discover and create meaningful representations of the data. In this context, representation refers to a transformation of the raw data into a different format that captures important features or characteristics of the data in a more useful and informative way. To be able to reconstruct the data as closely as possible to the input, the representations learned by the network must encapsulate the relevant features of the data in the lower dimensional space. Hence, it is imperative to understand the domain of representation learning to be able to construct a network that captures useful representations.

2.3.1 Manifold Learning

Manifold learning is a technique that aims to uncover the underlying structure of high-dimensional data. The theoretical concept of manifold learning is based on the premise that high-dimensional data is frequently found on or near a lower-dimensional manifold within the high-dimensional space [19]. A manifold is essentially a lower-dimensional surface embedded in a higher-dimensional space. Manifold learning algorithms seek to uncover the underlying manifold and represent the data in a more compact and understandable way. By finding and capturing this structure, manifold learning algorithms can minimize the dimensionality of the data while maintaining its significant properties.

Some popular manifold learning algorithms are:

- **Isomap (Isometric Mapping) [20]**: It aims to preserve the geodesic distances between all pairs of points in the dataset. It constructs a neighborhood graph and then approximates the geodesic distances along this graph.
- **Locally Linear Embedding (LLE) [21]**: LLE seeks to represent each data point as a linear combination of its nearest neighbors, aiming to preserve local relationships within the data. It then seeks a low-dimensional embedding where these local relationships are maintained.
- **t-Distributed Stochastic Neighbor Embedding (t-SNE) [22]**: Primarily used for visualization, t-SNE minimizes the divergence between the distributions of pairwise similarities in the high-dimensional space and the low-dimensional embedding.
- **Principal Component Analysis (PCA) [23]**: While not exclusively a manifold learning technique, PCA is widely used for dimensionality reduction. It identifies the directions (principal components) along which the data varies the most and projects the data onto a lower-dimensional subspace while preserving as much variance as possible.
- **Linear Spectral embedding [24]**: Spectral embedding is a dimensionality reduction technique that aims to uncover the underlying structure of data by leveraging the eigenvalues and eigenvectors of a similarity or affinity matrix computed from the data. The basic idea is to represent high-dimensional data points in a lower-dimensional space while preserving pairwise relationships or similarities between the data points.
- **Diffusion Maps [25]**: This method is based on the concept of diffusion processes, where similarity between data points is measured by their probability of transitioning from one point to another over a series of steps. By constructing a diffusion matrix from pairwise similarities and then computing its eigenvectors, diffusion maps generate a low-dimensional embedding of the data that preserves the underlying geometric and topological relationships.

The swiss roll dataset is a popular dataset used to test various manifold learning techniques. It is a two-dimensional manifold that is embedded in three-dimensional space and resembles a rolled-up sheet of paper or a spiral-like structure. The figure 2.3 shows the swiss roll in three dimensions and the figure 2.4 shows a comparison of each of these manifold learning techniques on the swiss roll dataset. Each subplot illustrates a distinct technique in its attempt to unfold the inherently nonlinear structure of the swiss roll into a lower-dimensional representation. Each

method reduces the swiss roll from three dimensions to two dimensions and, where needed, the number of neighboring points is specified as twelve. Differences in the unwrapped shapes highlight the varying capabilities of each method in capturing and preserving the underlying geometric properties of the dataset.

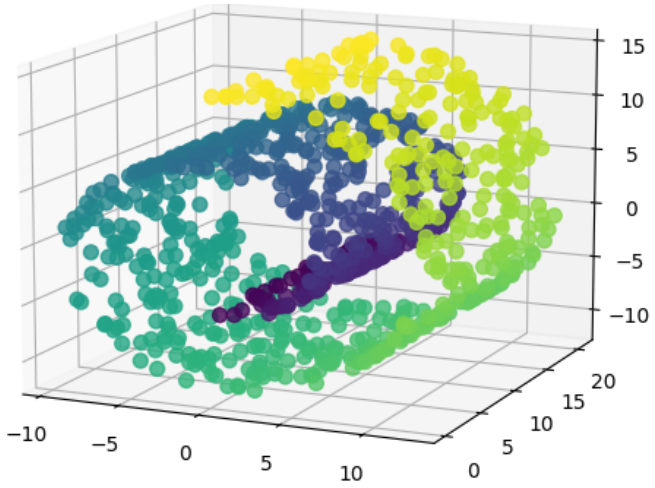
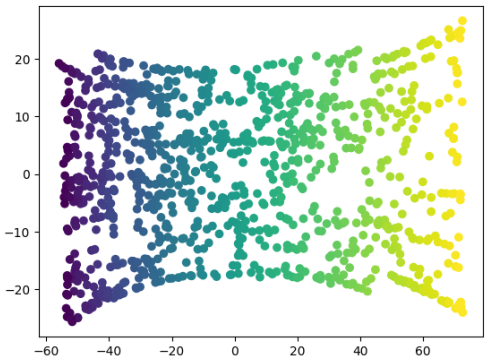


Figure 2.3: The Swiss Roll manifold in 3 dimensions

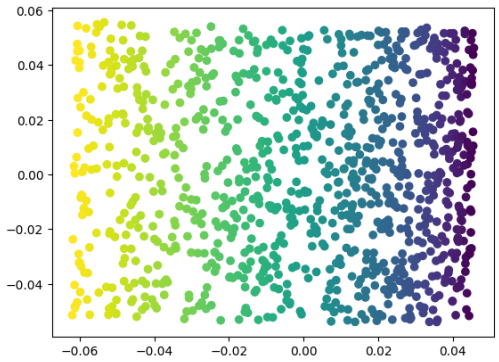
2.3.2 Kernel Representation Learning

Kernel representation learning is aimed at learning feature representations of data using the concept of kernels. The key idea behind kernel representation learning is to find a suitable kernel function, given by Equation 2.4 that maps the input data into a higher-dimensional space where the data becomes linearly separable or more amenable to analysis. This can be especially useful in scenarios where the original data lies on a complex, nonlinear manifold, making traditional linear methods less effective. Some common kernel functions include:

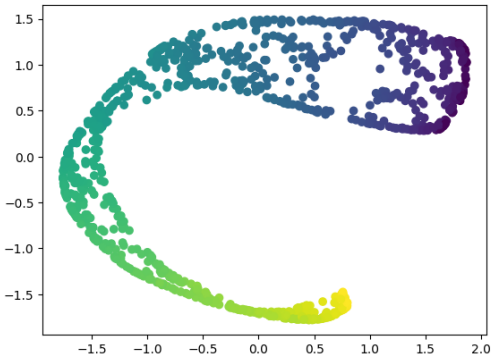
- **Linear Kernel**: It computes the inner product between two data points in the original feature space, effectively measuring their similarity without any nonlinearity. It is given by the form $k(x, y) = x^T y$.
- **Polynomial Kernel**: It computes the similarity between two data points as the polynomial of their inner product, often with an additional user-defined degree parameter. It is defined as $k(x, y) = (x^T y + c)^d$.
- **Gaussian Radial Basis Function (RBF) Kernel**: It measures the similarity between two data points based on the Euclidean distance between them in



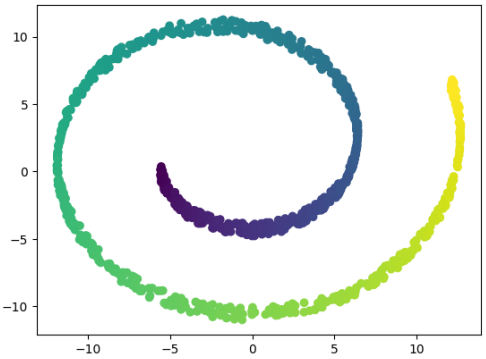
(a) Isomap Embedding



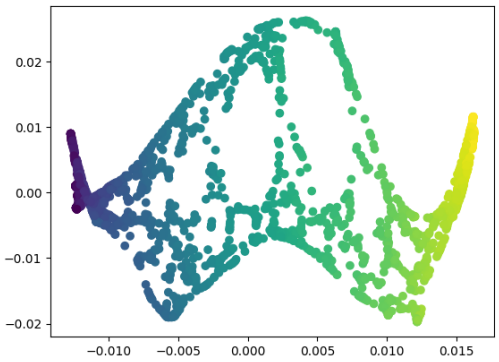
(b) LLE Embedding



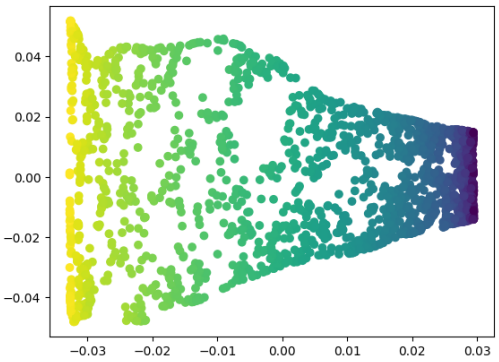
(c) TSNE Embedding



(d) PCA Embedding



(e) Spectral Embedding



(f) Diffusion Maps

Figure 2.4: Two-dimensional Embeddings of the Swiss Roll Dataset

the original feature space. It is defined as $k(x, y) = \exp(-\gamma(x - y)^2))$. The similarity decreases exponentially with distance, resulting in a smooth and continuous kernel.

- **Sigmoid Kernel**: It computes the similarity between two data points using the hyperbolic tangent function, given by $k(x, y) = \tanh(\alpha xy + \beta)$.

The success of kernel representation learning methods can be attributed to several factors. They capture complex, nonlinear relationships between data points by implicitly mapping them to a higher-dimensional space. They can easily deal with very high dimensional data and even an infinite dimensional feature space. Kernel methods, particularly those based on regularization techniques are often less prone to overfitting compared to other complex models [26]. Many kernel methods come with strong theoretical guarantees, such as convergence properties and performance bounds, which provide insights into their behavior and hence make them more interpretable.

A common principle of these methods is to construct nonlinear variants of linear algorithms by substituting the linear inner product with kernel functions. Examples include kernel principle component analysis (kPCA) and kernel linear discriminant analysis (kLDA) which are non-linear extensions of these linear dimensionality reduction algorithms. Significant for our work is the kernel version of canonical correlation analysis. Linear canonical correlation analysis (CCA) is used to explore the relationship between two sets of variables by identifying linear combinations of variables (canonical variates) that have maximum correlation across the two sets [26]. The goal of CCA is to find linear combinations of the canonical variates from each set that maximize their correlation. This is achieved by finding weights for each variable such that the correlation between the resulting canonical variates is maximized.

The objective is to find two vectors $v_j \in L(X')$ and $w_j \in L(Y')$ where $L(X')$ is the row space of the input data X such that the variates $a_j = Xv_j$ and $b_j = Yv_j$ are maximally correlated. This objective is given by

$$corr(a_j, b_j) = \frac{\langle a_j, b_j \rangle}{\|a_j\| \|b_j\|}. \tag{2.6}$$

The objective can be formulated as the optimization problem of the form

$$\begin{gathered} argmax_{v_j \in L(X'),\ w_j \in L(Y')}\ v_j' X' Y w_j \\ subject\ to\ v_j' X' X v_j = w_j' Y' Y w_j = 1. \end{gathered} \tag{2.7}$$

Once the canonical variates are obtained, CCA computes the canonical correlations, which measure the strength of the relationship between the sets of canonical variates. The canonical correlation coefficients indicate how well the two sets of variables are related to each other. The kernel version of canonical correlation analysis (KCCA) differs from the linear version in how it operates on the data. While linear canonical correlation analysis works directly with the original data in the feature space, KCCA operates in a higher-dimensional space implicitly defined by a kernel function.

2.3.3 Contrastive Representation Learning

The dimensionality reduction approaches described earlier achieve the goal of obtaining low-dimensional features from high-dimensional data. However, those methods have significant limitations, which drove the development of contrastive learning. Linear spectral embedding and PCA are linear techniques and cannot handle non-linear data or manifolds, Isomap, LLE and Diffusion maps have a high computational cost and are sensitive to their parameters, while TSNE does not generalize to new data. Contrastive learning overcomes the flaws mentioned above by learning characteristics from data using neural networks. The sole assumption made about the data structure is that similar inputs are close to each other in latent space. Contrastive learning, like manifold learning, aims to map similar points to close together while mapping dissimilar points far apart. Unlike multidimensional scaling or kernel approximation, it is not required to specify a similarity measure (or kernel) between every pair of points in the data. Instead, all that remains to be done is to determine pairs of similar inputs (positive samples) and dissimilar inputs (negative samples). The desired representations are then learned via the minimization of a loss function, called contrastive loss, which depends on a notion of similarity such as the Euclidean distance between two points. It encourages this similarity to be as high as possible for positive samples and as low as possible for negative samples [27].

Two of the most popular contrastive learning losses are the Barlow Twins loss and the Triplet loss. For Barlow Twins, the input data X is used to construct another matrix Y where Y is an augmentation of each data point from the input X. Augmenting the input creates points that are semantically similar to the input points and hence should be mapped identically in the feature space. In other words, the representations or features learned from the augmented points should be identical to the ones learned from their corresponding input point. The objective function measures the cross-correlation matrix between the embeddings of two identical networks fed one fed with X and the other fed with Y and tries to make this matrix close to the identity [28]. The algorithm is detailed in Figure 2.5.

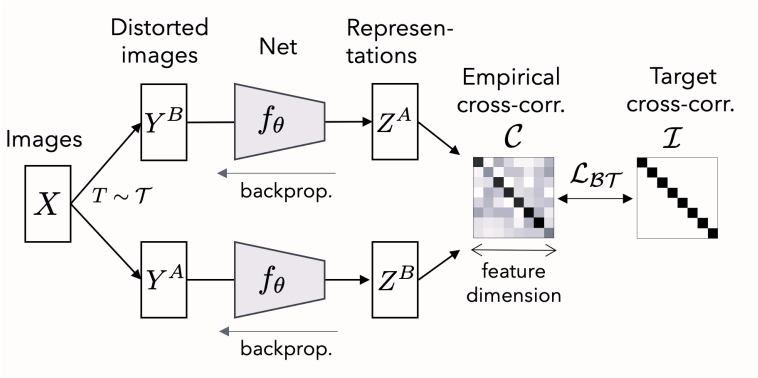


Figure 2.5: The Barlow Twins algorithm uses distorted versions Y^A and Y^B of the original Data X, constructed via augmentations and fed into identical neural networks. The objective function then tries to map the cross-correlation matrices of the embeddings of Y^A and Y^B to an identity matrix [28].

The Triplet loss is another popular contrastive loss. Triplet loss requires three inputs: anchor, positive, and negative. The positive point here refers to a data point that is semantically similar to the anchor. This could be based on class labels or some distance measure. We can also generate positive points from the data using augmentations. The negative, in contrast, refers to a data point that is dissimilar to the data. The objective of triplet loss is to reduce the distance between the anchor and the positive example while increasing the gap between the anchor and the negative example. The work by Esser et al. in the paper [6] combines kernel methods and the triplet loss for non-parametric representation learning models, focusing on contrastive self-supervised learning models. It presents kernel variants of a single hidden layer network that minimize the simple and spectral contrastive loss functions, which are both variants of the triplet loss. The formulation of the two losses is shown in Figure 2.6. Here x denotes the original data points, x^+ denotes positive pairs of the data, and x^- denotes negative pairs of the data.

The paper also presents a closed-form solution for an optimal embedding using contrastive kernel learning. This is very important for our work as a closed-form solution enables us to compute an optimal embedding without the need for iterative gradient descent. It emphasizes the suitability of kernel methods for small data problems and their interpretability. The authors also extend the existing representer theorem under orthogonal constraints and derive generalization error bounds for the proposed kernel models, showing that model predictions improve with an increased number of unlabelled data. Furthermore, the paper compares the proposed kernel

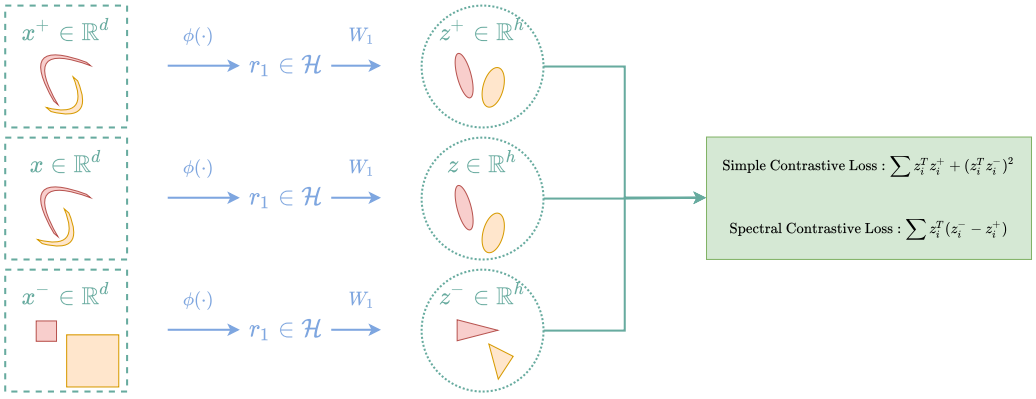


Figure 2.6: Kernel contrastive loss using a Triplet setting [6]. The simple and spectral contrastive losses are optimized by decreasing the distance between the embedding of the anchor z and the positive z^+ and increasing the distance between embedding of anchor z and the negative z^-.

contrastive method to neural network models for classification and de-noising tasks, demonstrating that the approach suggested performs on par with or outperforms the neural network counterparts.

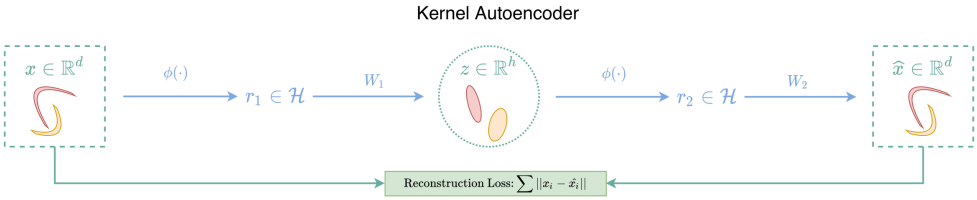
2.3.4 Autoencoders for Representation Learning

Much like these manifold learning algorithms, autoencoders also aim to learn a lower dimensional representation of the input data. The embedding space in an autoencoder is usually lower dimensional compared to the input dimension and aims to encapsulate the most meaningful features or representations of the input data. Hence, it forms a compressed and informative representation of the input data. This is why the embedding space can be thought of as a lower dimensional representation of the data allowing autoencoders to be used for dimensionality reduction and representation learning.

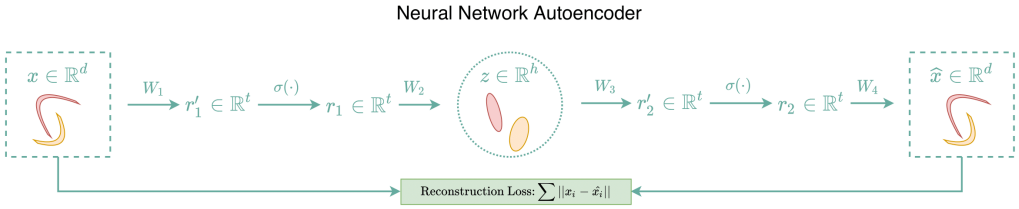
In recent times, the concept of feature learning for dimensionality reduction has gained significant attention and has been applied to various real-world tasks. As sampled networks gain more attention and praise, the focus has also been on using sampled autoencoders to study the representations they create, especially in a low-dimensional setting, and to try to ascertain if such networks can be used for dimensionality reduction. Researchers have been inspired by the hierarchical architecture of deep neural networks and have focused on constructing multilayer ELM feature extractors, the most prominent of which were discussed in Section 2.2.1 and the kernel variants which were discussed in Section 2.2.2.

The paper by Esser et al. [6] has also touched upon the use of kernel methods with the autoencoder network. The paper presents the concept of a kernel autoencoder (kAE), in which both the encoder and decoder represent kernel machines. It defines the kernel autoencoder as a model that requires transferring the input to a lower-dimensional latent space (encoding) and then returning to the reconstruction of the original data (decoding). The work provides a formal specification of the kernel autoencoder and defends its design, which includes norm regularization on both the encoder and decoder. The data from the input space is first mapped to a feature space by computing a kernel function over the input data points. This feature space is then used as the input to an encoder that maps it to an embedding space by multiplying it with a weight matrix. A kernel is computed over the embedding space before it is fed to a decoder network which multiplies it by its weight matrix to reconstruct the data. The approach is shown in Figure 2.7, compared against the traditional autoencoder network. and the kernel PCA regime.

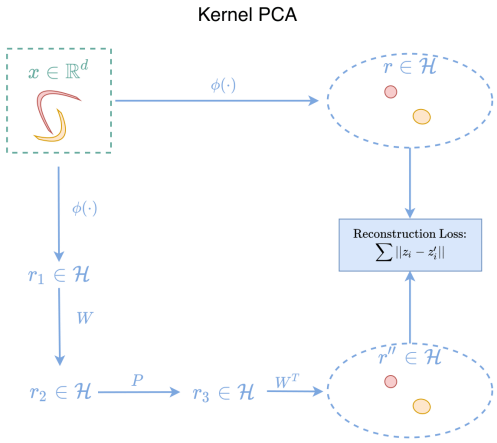
In this approach, both weight matrices for the encoder and decoder are learned using iterative gradient descent. Furthermore, the paper examines the relationship between kernel AE and kernel PCA, emphasizing the contrasts and similarities between the two methodologies. Although different from our purpose of learning reconstructions using sampled weights and biases, this method does provide a good baseline for our work.



(a) The Kernel Autoencoder where the data is first mapped to a kernel space before it is mapped to a lower dimensional space. The lower dimensional data is also first mapped to a kernel space before it is reconstructed.



(b) A traditional neural network where the data is mapped to a lower dimensional space using one or more layers and reconstructed from that space.



(c) Kernel PCA that performs PCA in the RKHS and therefore computes distances in the feature space

Figure 2.7: Kernel Autoencoder compared against the neural network autoencoder and kernel PCA [6].

3 Data Driven Sampling in Autoencoders

3.1 Auto Encoder Network

A traditional autoencoder consists of an encoder and a decoder, which work together to learn a compact representation of the input data. It comprises of three parts.

1. Encoder: The encoder takes the input data and maps it to a lower-dimensional representation. The encoder can consist of one or more layers of neurons, each performing transformations on the input data to gradually reduce its dimensionality.

2. Embedding Space: This is the compressed representation of the input data learned by the encoder. It has a lower dimensionality compared to the input data and captures the most salient features. The dimensionality of the embedding space is a hyperparameter that needs to be defined before training the autoencoder.

3. Decoder: The decoder takes the compressed representation generated by the encoder and attempts to reconstruct the original input data from it. Like the encoder, the decoder can consist of one or more layers of neurons, which gradually expand the dimensionality of the embedding space back to the dimensionality of the original input data.

The structure of our autoencoder also consists of the encoder, the embedding space, and the decoder. It differs from a traditional autoencoder in two respects. Firstly, the weights for both the encoder and decoder are not trained using gradient descent but are sampled using the data-driven sampling from [1]. Secondly, the output of the encoder is not inherently the embedding space. Rather, the output of the encoder is a high-dimensional feature map that is then mapped to a lower-dimensional embedding space using a modified version of the kernel representation learning scheme from [6], which uses the feature map from the encoder as well as a similarity map which specifies which points are positive pairs (have identical labels) of each other. Each of the parts is discussed in detail in the following subsections. A visual representation is shown in Figure 3.1

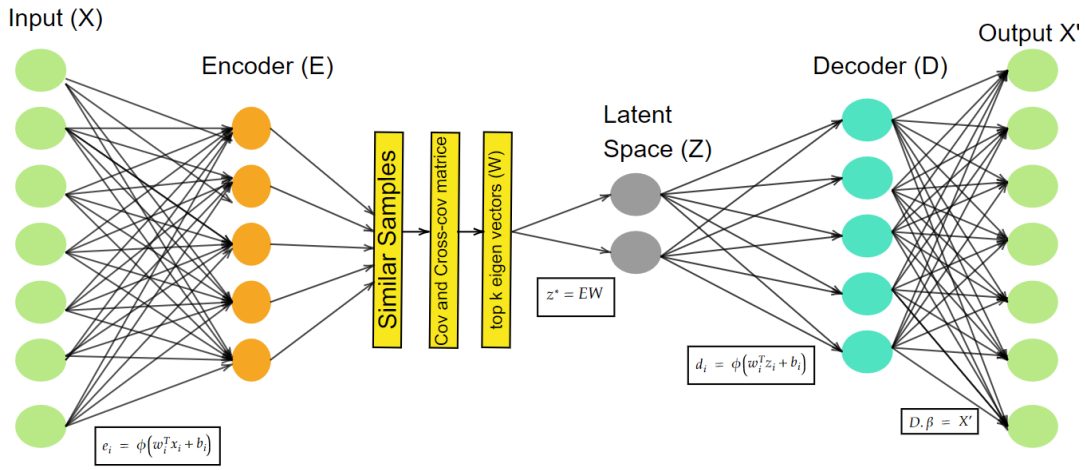


Figure 3.1: The autoencoder network with the encoder that maps the data into a high dimensional feature space, the embedding space which is constructed from the encoded feature space and a similarity graph, and the decoder which maps the points from the embedding space to the reconstruction.

3.1.1 Computing the Similarity Map

The similarity map is a matrix denoting which points from the input space are related to each other. Given the input $X \in \mathbb{R}^{Nxd}$ of the form

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

where each x_m is a d dimensional data point, the matrix for the similarity map is demoted by $G \in \{0,1\}^{NxN}$ where each row of the matrix corresponds to one data point and each point in the row denotes its similarity to each of the other points. Hence the matrix G represents a pairwise positive relation between the samples in the data where $G_{i,j} = 1 \Leftrightarrow x_i$ and x_j are semantically related and $i \neq j$. We can also restrict the matrix G to identify a limited number of positive samples for each point. We do this by randomly sampling any z points from each $pairs(i)$ where $pairs(i)$ are positive samples for each data sample x_i, and then setting $G_{i,j} = 1 \Leftrightarrow G_{i,j} \in pairs(i)$ for each i.

There are several ways to determine whether two points are semantically related to each other. The most commonly used method for image datasets is to compute

augmentations of each data sample. Some common methods of augmentations include rotation, scaling, translation, contrast and brightness adjustments among others. These augmentations can be applied individually or in combination, and they are often used with parameters randomly sampled from predefined ranges to increase variability. Another method is to use any notion of similarity to compute closeness. For example, the Euclidean distance or cosine distance between two points can be a measure of similarity with points having a smaller distance being more similar to each other. These are both examples of unsupervised approaches.

However, for our work, we used a supervised approach for computing the similarity map. As the class labels were available for all the datasets used, the similarity map was defined based on these labels, with $G_{i,j} = 1$ if x_i and x_j had the same label. The use of class labels to determine the positive pairs simplifies the process of positive selection while avoiding potential false negatives [29]. The approach also results in a better and more varied selection of positive pairs that contain semantically relevant features and generalize better on unseen data. An illustration of how using a supervised approach harnessing class labels for generating a similarity map can be helpful to more closely align images that have similar semantic properties can be seen in Figure 3.2

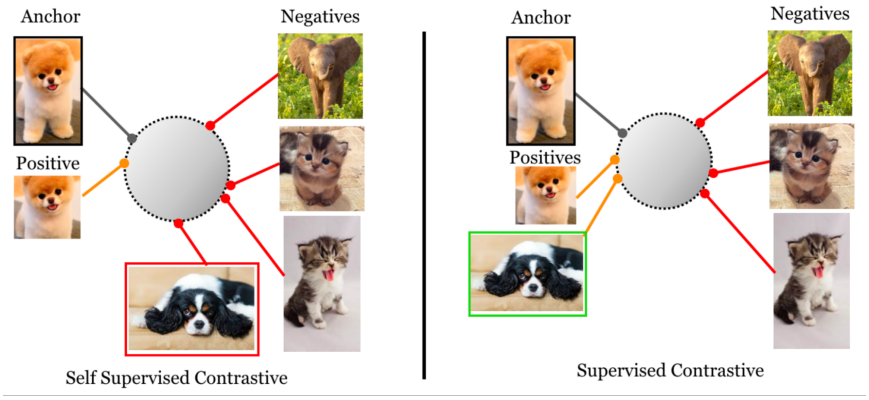


Figure 3.2: Embeddings found using data augmentations labels to generate positive pairs vs using class labels. As demonstrated by the photo of the black and white puppy, taking class label information into account results in an embedding space where elements of the same class are more closely aligned than in the self-supervised case [29].

3.1.2 Sampling the Encoder

The encoder part of the network comprises of a single dense layer that maps the input space to a feature map. Given the input $X \in \mathbb{R}^{Nxd}$, the encoder maps the input to a feature space $E \in \mathbb{R}^{Nxm}$ where m is the width of the dense layer. The layer width m can be smaller or larger compared to the dimensionality of the data space d and is chosen as a hyperparameter. The mapping has the form

$$e_i(x) = \phi(w_i^T(x) + b_i), \tag{3.1}$$

where w and b are sampled from the data X using the form given in 2.3 and ϕ is a non linear activation function. The activation functions that can be used with this data-driven sampling scheme are rectified linear unit (ReLU) and Tanh. The scalars in the equation 2.3 are used to fix what values the points x_1 and x_2 take on when the activation function is applied. For ReLU, s_1 is set to 1 and s_2 is set to 0, to let x_1 be mapped to zero and x_2 to 1. For tanh, $s_1 = 2s_2$ and $s_2 = ln(3)/2$, which implies that x_1 and x_2 are mapped to $+1/2$ and $-1/2$ respectively, and the midpoint between the two points is assigned to zero [1].

The output of the encoder is a m dimensional, non-linear map of the input space. It is important to note that either $m < d$ or $m >= d$, but in either case, the mapping $E(x)$ does not represent the embedding, which is the case for a traditional autoencoder, where the embedding is given by the output of the encoder. In our case, the output of the encoder is a feature map from which the embedding will be calculated in the next step. The feature map is constructed only to obtain a non-linear, high-dimensional representation of the input, similar to constructing a kernel over the data space.

3.1.3 Optimal Embedding for the Latent Space

For this section, we take inspiration from the Barlow Twins loss [28] which is based on the covariance and cross-covariance matrices. This loss aims to encourage the learned representations to be invariant to certain differences in data points the having same class label while still preserving useful information. It achieves this by penalizing the cross-covariance matrix of the representations obtained from the positive pairs of the data samples.

To extend into the non-linear regime, instead of using the original data samples, we use the feature map $E(x)$, which is the output of the encoder. Once the feature map is constructed, the optimal embedding is computed. We compute the embedding by finding the optimal weights $W \in \mathbb{R}^{mxk}$ where k is the dimension of the embedding space such that the embedding can be obtained using

$$z^* = E(x)W. \tag{3.2}$$

To compute the optimal weights, first, we construct two matrices X_a and X_b as follows

$$X_a = [e_1, e_1, ...e_1, ...e_N, e_N, ...e_N]^T, \tag{3.3}$$

where each point e_i is the feature map of the input point x_i obtained from the encoder and is repeated z number of times where z is the number of positive pairs chosen for each point.

$$X_b = [\hat{e}_{1,1}, \hat{e}_{1,2}, ...\hat{e}_{1,z}, ...\hat{e}_{N,1}, \hat{e}_{N,2}, ...\hat{e}_{N,z}]^T, \tag{3.4}$$

where $\hat{e}_{i,\hat{z}}$ is the feature map of the positive pair for x_i based on the similarity graph G. We then compute the covariance and cross-covariance matrices as follows

$$\begin{aligned} C_{aa} &= X_a^T X_a, \\ C_{bb} &= X_b^T X_b, \\ C_{ab} &= X_a^T X_b, \\ C_{ba} &= X_b^T X_a, \end{aligned} \tag{3.5}$$

and then the matrix C is calculated as

$$C = {C_{aa}}^{-1} C_{ab} {C_{bb}}^{-1} C_{ba}. \tag{3.6}$$

The optimal weights W are then given by the top k eigenvectors of C and using these the optimal embedding can be calculated using Equation 3.2. Hence we have a closed-form solution for our latent space.

As computing the eigen values and eigen vectors of a high dimensional matrix can be very computationally expensive, we solve them using the `lobpcg` function from the library `scipy.sparse.linalg`. It is a solver for large-scale eigenvalue problems for symmetric positive definite generalized eigenvalue problems. It requires a block preconditioner, which is an approximation to the inverse of the matrix being solved. The block preconditioner is used to accelerate the convergence of the iterative solver by reducing the condition number of the problem. As the block preconditioner, we use the variance matrix C_{aa}. It iteratively refines an initial guess for the eigenvectors and eigenvalues of the problem. At each iteration, it constructs a subspace spanned by the current approximation to the eigenvectors and solves a small generalized eigenvalue problem within this subspace. `lobpcg` is designed to handle large-scale problems efficiently by working with sparse matrix representations, which reduces memory requirements and computational cost for operations like matrix-vector product. This greatly helps us to speed up the computation of the eigenvalue problem.

3.1.4 Sampling the Decoder

From the embedding constructed using Equation 3.2 in the previous subsection, the original image can now be reconstructed using the decoder. The decoder of our network consists of two layers. The first dense layer maps the embedding to a high-dimensional non-linear feature map. This is done based on the equation

$$d_i(z) = \phi(w_i^T(z_i) + b_i), \tag{3.7}$$

where $z \in \mathbb{R}^{Nxk}$ is the embedding and ϕ is a non linear activation function. The weights w and biases b are again sampled using the data-driven sampling method from [1], but this time, the samples are derived from the points in the embedding space and not the original dataset. Although this has changed our sample space, the construction of the sampling scheme, based on the two activation functions ReLU and Tanh, is invariant to the sample space, so that the sampling is also valid when the sample space changes from the original data space.

The next part of the decoder is the linear layer where the output weights are calculated analytically. The network solves a simple linear regression problem to determine the output weights. This involves solving the following equation

$$D(z)\beta = Y, \tag{3.8}$$

where $D(z)$ is the feature space obtained from the dense layer of the decoder, β are the weights to be solved and Y is the target matrix which in the case of an autoencoder is simply the input matrix X. In this way, the last linear layer learns a matrix representation that exactly maps the feature map of the decoder $D(z)$ to the target X. The equation is solved using linear least squares, where if $D(z)$ has full rank then we get the exact solution, otherwise β minimizes the Euclidean norm $\|D(z)\beta - Y\|$.

Moreover, to prevent the weights β from overfitting on the train set and hence exhibit poor performance on the test set, we can introduce regularization to the Equation 3.8 to penalize large weights and encourage smoother solutions, helping to prevent overfitting. This is especially useful when the model's capacity is large relative to the size of the training dataset or when dealing with noisy data.

The full algorithm for the autoencoder is given in Algorithm 2.

3.1.5 Hyperparameters for the network.

In an autoencoder, hyperparameters are the parameters that are set before the training process begins and are not learned from the data. These parameters control various aspects of the autoencoder's architecture, optimization, and regularization.

Algorithm 1 The SWIM algorithm, for activation function ϕ, and norms on input, output of the hidden layers, and output space, $\|\cdot\|_{\mathcal{X}_0}$, $\|\cdot\|_{\mathcal{X}_1}$, $\|\cdot\|_{\mathcal{Y}}$ respectively, taken from [1], and amended to sample only for one layer.

Constants: $\epsilon \in \mathbb{R}_{>0}$, $\varsigma \in \mathbb{N}_{>0}$, $N \in \mathbb{N}_{>0}$, and $s_1, s_2 \in \mathbb{R}$

Data: $X = \{x_i \colon x_i \in \mathbb{R}^D, i = 1, 2, \dots, M\}$, $Y = \{y_i \colon f(x_i) = y_i \in \mathbb{R}^N, i = 1, 2, \dots, M\}$
$\Phi(x) = x$
$\tilde{M} \leftarrow \varsigma \cdot \lceil \frac{N_l}{M} \rceil \cdot M$
$P \in \mathbb{R}^{\tilde{M}}$; $P_i \leftarrow 0 \quad \forall i$
$\tilde{X} = \{(x_i^{(1)}, x_i^{(2)}) \colon \Phi(x_i^{(1)}) \neq \Phi(x_i^{(2)})\}_{i=1}^{\tilde{M}} \sim \text{Uniform}(X \times X)$
for $i = 1, 2, \dots, \tilde{M}$ **do**

$\tilde{x}_i^{(1)}, \tilde{x}_i^{(2)} \leftarrow \Phi(x_i^{(1)}), \Phi(x_i^{(2)})$
$\tilde{y}_i^{(1)}, \tilde{y}_i^{(2)} = f(x_i^{(1)}), f(x_i^{(2)})$
$P_i \leftarrow \frac{\|\tilde{y}_i^{(2)} - \tilde{y}_i^{(1)}\|_{\mathcal{Y}}}{\max\{\|\tilde{x}_i^{(2)} - \tilde{x}_i^{(1)}\|_{\mathcal{X}}, \epsilon\}}$

end for
$W \in \mathbb{R}^{N,D}, b \in \mathbb{R}$
for $k = 1, 2, \dots, N$ **do**
Sample $(x^{(1)}, x^{(2)})$ from $\tilde{X}$, with replacement and with probability proportional to P
$\tilde{x}^{(1)}, \tilde{x}^{(2)} \leftarrow \Phi(x^{(1)}), \Phi(x^{(2)})$
$W^{(k,:)} \leftarrow s_1 \frac{\tilde{x}^{(2)} - \tilde{x}^{(1)}}{\|\tilde{x}^{(2)} - \tilde{x}^{(1)}\|^2}$; $b_l^{(k)} \leftarrow \langle W^{(k,:)}, \tilde{x}^{(1)} \rangle + s_2$
end for
$\Phi(\cdot) \leftarrow \phi(WX - b)$
return $\{W, b, \Phi(\cdot)\}$

Algorithm 2 The full algorithm for the data-driven sampled autoencoder. $\mathcal{L}$ is a loss function, which in our case is always L^2 loss, and $\arg\min \mathcal{L}(\cdot,\cdot)$ becomes a linear optimization problem.

Constants: `num_positives` $\in \mathbb{N}_{>0}$, $k \in \mathbb{N}_{>0}$, $\phi \in \{ReLU, Tanh\}$

Data: $X = \{x_i \colon x_i \in \mathbb{R}^d, i = 1, 2, \ldots, N\}$, $Y = \{y_i \colon f(x_i) = y_i \in \mathbb{Z}^+, i = 1, 2, \ldots, N\}$

for $i = 1, 2, \cdots, N$ **do**
  `pairs(i)` $= \{x_j \quad | \quad \forall j$ where $y_j = y_i$ and $i \neq j\}$
  `pairs(i)` = sample `num_positives` points from `pairs(i)`
end for

find $E = \phi(\cdot)$ using Algorithm 1 with Inputs X and X

find X_a using Equation 3.3
find X_b using Equation 3.4

$C_{aa} = X_a^T X_a$
$C_{bb} = X_b^T X_b$
$C_{ab} = X_a^T X_b$
$C_{ba} = X_b^T X_a$

$cov = C_{aa}^{-1} C_{ab} C_{bb}^{-1} C_{ba}$

$W =$ top k eigen vectors of cov in descending order

$z* = E \cdot W$

find $D = \phi(\cdot)$ using Algorithm 1 with Inputs Z and X

$W, b = \arg\min \mathcal{L}(D, X)$;

$\hat{X} = WX - b$
return $\hat{X}, W, b$

Hyperparameters significantly influence the performance, generalization, and convergence speed of the autoencoder. Proper tuning of these hyperparameters is crucial to achieve the desired performance and prevent issues like overfitting or slow convergence. By design, our network does not require some of the hyperparameters required by traditional autoencoders such as the learning rate, dropout rate, optimizer amongst others. The hyperparameters used in the work are given in Table 3.1. All these hyperparameters are tuned manually to find optimal values for best results.

Table 3.1: Hyperparameters required for the network.

Hyperparameter	**Description**
Positive pairs	The number of samples taken as the positive pairs for each input data sample
Encoder width	Number of neurons in the encoder. This also specifies the dimensionality for the feature space of the encoder
Decoder width	Number of neurons in the decoder. This also specifies the dimensionality for the feature space of the decoder
Activation function	The function to introduce non-linearity into the network, enabling it to learn complex patterns in the data and make the network capable of approximating any arbitrary function. The choice is between ReLU and Tanh
Embedding dimension	The dimension of the embedding space which the input is reduced to
Regularization constant	The constant that controls the degree of regularization for the linear layer

3.2 Association to Kernel Canonical Correlation Analysis

Canonical correlation analysis (CCA) is a statistical technique used to analyze the relationship between two sets of variables. CCA is formulated as an optimization problem, where the goal is to find the optimal linear transformation of the input variables, usually represented by a matrix of weights denoted as W, such that the correlation between the transformed variables is maximized. In simpler terms, CCA seeks to find linear combinations of the original variables from each set in such a way that the correlation between these combinations is maximized. This is achieved by adjusting the weights W in the transformation matrices. The optimization process involves searching for the set of weights that yields the highest correlation between the transformed variables, and captures the most meaningful relationship between the two sets of variables. There are numerous approaches to framing the CCA problem. Essentially, the basic objective is to iteratively discover pairs of filters that generate features with the highest correlation possible as in

$$\max_{w_a, w_b} \frac{\langle X_a W_a,\ X_b W_b \rangle}{\|X_a W_a\|_2 \|X_b W_b\|_2}. \tag{3.9}$$

Inspired by the work in [30], we now show that this has a natural translation to Equation 3.6 which we use to find the optimal weights for the embedding. Equation 3.9 can be reformulated as a constrained optimization problem of the form

$$\max_{w_a, w_b} \langle X_a W_a,\ X_b W_b \rangle \quad s.t. \quad \|X_a W_a\|_2 = 1 \quad and \quad \|X_b W_b\|_2 = 1. \tag{3.10}$$

Because adjusting the scale of the weights doesn't affect the learned filters, they can be converted into a Lagrangian function, thus

$$\mathcal{L} = \langle X_a W_a,\ X_b W_b \rangle - \lambda_1 (\quad \|X_a W_a\|_2 - 1) - \lambda_2 (\|X_b W_b\|_2 - 1). \tag{3.11}$$

Computing the partial derivatives gives us

$$\begin{aligned} \nabla_{w_a} \mathcal{L} &= X_a X_b w_b - 2\lambda_1 X_a X_a w_a \\ \nabla_{w_b} \mathcal{L} &= X_b X_a w_a - 2\lambda_2 X_b X_b w_b \\ \frac{\delta \mathcal{L}}{\delta \lambda_1} &= w_a^T X_a X_a w_a \\ \frac{\delta \mathcal{L}}{\delta \lambda_2} &= w_b^T X_b X_b w_b \end{aligned} \tag{3.12}$$

Replacing $X_a X_a = \Sigma_{aa}$, $X_a X_b = \Sigma_{ab}$, $X_b X_a = \Sigma_{ba}$, $X_b X_b = \Sigma_{bb}$, we get

$$\begin{aligned} \nabla_{w_a} \mathcal{L} &= \Sigma_{ab} w_b - 2\lambda_1 \Sigma_{aa} w_a \\ \nabla_{w_b} \mathcal{L} &= \Sigma_{ba} w_a - 2\lambda_2 \Sigma_{bb} w_b \\ \frac{\delta \mathcal{L}}{\delta \lambda_1} &= w_a^T \Sigma_{aa} w_a \\ \frac{\delta \mathcal{L}}{\delta \lambda_2} &= w_b^T \Sigma_{bb} w_b \end{aligned} \tag{3.13}$$

Setting the first two equations to 0 will give us

$$\begin{aligned} \Sigma_{ab} w_b - 2\lambda_1 \Sigma_{aa} w_a = 0 \Rightarrow w_a^T \Sigma_{ab} w_b - 2\lambda_1 w_a^T \Sigma_{aa} w_a = 0 \\ \lambda_1 = \frac{1}{2} w_a^T \Sigma_{ab} w_b \\ \Sigma_{ba} w_a - 2\lambda_2 \Sigma_{bb} w_b = 0 \Rightarrow w_b^T \Sigma_{ba} w_a - 2\lambda_1 w_b^T \Sigma_{bb} w_b = 0 \\ \lambda_1 = \frac{1}{2} w_b^T \Sigma_{ba} w_a \end{aligned} \tag{3.14}$$

From this equation, it is clear that $\lambda_1 = \lambda_2$. Hence these both can be replaced by a single variable λ and the equations can then be reformulated as

$$\begin{aligned} \Sigma_{ab} w_b - 2\lambda \Sigma_{aa} w_a = 0 \\ w_a = \frac{1}{2\lambda} \Sigma_{aa}^{-1} \Sigma_{ab} w_b \\ \Sigma_{ba} w_a - 2\lambda_2 \Sigma_{bb} w_b = 0 \\ (\frac{1}{2} \Sigma_{bb}^{-1} \Sigma_{ba} \Sigma_{aa}^{-1} \Sigma_{ab} - \lambda^2 \mathcal{I}) w_b = 0 \end{aligned} \tag{3.15}$$

From the last equation, we know that we can obtain w_b by solving the generalized eigenvalue problem. Note that this also has the same form as the Equation 3.6 whose eigenvalues give the optimal weights for embedding.

While the proof of equivalence to CCA has been constructed using the input data X, we can also extend this proof to the kernel version where the input data X is replaced by some kernel version $\phi(x)$. Since in our work, we have not constructed the embedding from the input space but rather from the feature space obtained from the encoder, we can equate the feature space equal to $\phi(x)$. Hence, our work would more closely align with kernel canonical correlation analysis.

3.3 Details of datasets used experiments conducted by varying hyperparameters

We conduct a number of experiments using the two benchmark image datasets: MNIST and CIFAR-10. For each experiment, we change one of the three hyperparamaters, length of the encoder and decoder, the embedding dimension or the number of positive pairs while keeping all others constant. We use the same subset of training and test samples, for each experiment, to prevent any discrepancies that could arise based on a different selection of training and test data. We monitor the train and test reconstruction error which is denoted by the mean squared error between the original image and the reconstructed image, averaged over the training and test set respectively. We also record the training time for each experiment and the classification error on the original datasets vs the reconstructed datasets.

3.3.1 Datasets

MNIST

The MNIST dataset is a large collection of handwritten digits that is used to evaluate and compare different machine learning algorithms. It consists of a total of 70,000 images, which are divided into training and testing sets of 60,000 and 10,000, respectively. Each image is a grayscale square that measures 28x28 pixels, resulting in a total of 784 pixels per image. Additionally, each image is associated with a label that indicates the digit it represents, with the digits ranging from 0 through 9. While the MNIST dataset is relatively simple by modern standards, it provides a useful benchmark for assessing machine learning algorithms. However, achieving high accuracy on MNIST is not considered a significant achievement in itself, but rather serves as a baseline for more difficult tasks.

CIFAR-10

The CIFAR-10 dataset is a widely used benchmark dataset in the fields of machine learning and computer vision research. This dataset comprises a total of 60,000 images, with 50,000 images in the training set and 10,000 images in the test set. Each image in the CIFAR-10 dataset is a 32x32 pixels color image. These images are RGB (Red, Green, Blue) images, meaning they have three color channels. The dataset includes 10 classes or categories, where each category represents a distinct object or classification. The categories include airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The CIFAR-10 dataset is more challenging than MNIST as it

includes higher resolution color images, more diverse objects and backgrounds, and a broader range of object categories.

3.3.2 Preprocessing the data

Preprocessing data is a critical step in machine learning. Scaling or standardizing features (for example, subtracting the mean and dividing by the standard deviation) can improve optimization and algorithm convergence and prevent certain features from dominating others. It can increase the robustness of machine learning models by making them less susceptible to fluctuations in input data.

For our experiments, we do not use the full datasets with 50,000 train images and 10,000 test images. Instead, we randomly choose 20,000 train images and 2,500 test images to prevent running out of memory on the machine. The data is normalized by dividing the pixel values by 255 and centered by subtracting the column-wise mean of the training images from both the training and test sets.

3.3.3 Network structure and Hyperparameters used for the experiments

A range of experiments were conducted with both datasets with varying values for the number of positive pairs, the encoder and decoder layer width, and the embedding dimension to uncover insights into their performance, robustness, and generalization. The structure of the network is shown in Figure 3.1. It consists of the encoder, which includes a single dense layer to map the data to a high dimensional feature space, the embedding space constructed using the linear weights found using the eigenvalue decomposition of the kernel correlation matrix, and a single dense layer followed by a linear layer as the decoder. The details of the experiments conducted with the varying hyperparameters are shown in Table 3.2.

3.4 Results

To assess the performance of the model, we have used different measures to evaluate the reconstruction. Firstly, the plots of the reconstructed images were constructed to visualize the reconstruction quality. The second measure used was the mean squared error given by the Equation 3.16. Lastly, a very basic classifier was used that was trained on the original training set and then was used to predict the labels for the reconstructed training set, the original test set and the reconstructed test set. The classification error on the original data was compared to the classification error on the reconstructed data.

Table 3.2: Experiments conducted on MNIST and CIFAR-10 and the configurations for their hyperparameters.

	MNIST	**CIFAR10**
Train Samples	20000	20000
Test Samples	2500	2500
Activation Function	Tanh	Tanh
Encoder Width	[500, 1000, 1500, 2000, 300, 3500, 4000, 4500]	[2000, 2500, 3000, 3500, 4000]
Embedding Dimension	[50, 100, 150, 200, 250, 300]	[400, 450, 500, 550, 600]
Decoder Width	[500, 1000, 1500, 2000, 300, 3500, 4000, 4500]	[2000, 2500, 3000, 3500, 4000]
Regularization parameter	$1e^{-10}$	$1e^{-10}$
Positive Pairs	[2, 6, 10, 14, 18, 22]	[2, 6, 10, 14, 18, 22]

We evaluate the difference in the mean squared error, classification accuracy, and the training time across the different combinations of hyperparameters used. The hyperparameters we varied to test the network were: the number of positive pairs, the layer width of the encoder and decoder, and the embedding dimension. Overall, what is observed is that the mean squared error and classification accuracy do not change with the number of positive pairs. The mean squared error increases and the classification accuracy decreases on the test set as the layer width for the encoder and decoder is increased. The reconstruction error decreases and the classification accuracy increases with an increase in the embedding dimension. The training time correlates most strongly with the width of the encoder and decoder, increasing as the width is increased.

3.4.1 Reconstruction

The first step to assess the quality of the reconstructed images is to plot them to visualize the reconstruction. This is a form of qualitative evaluation that involves visually inspecting the reconstructed images to assess the quality of reconstructions, including sharpness, clarity, and preservation of details.

Layer width for encoder and decoder

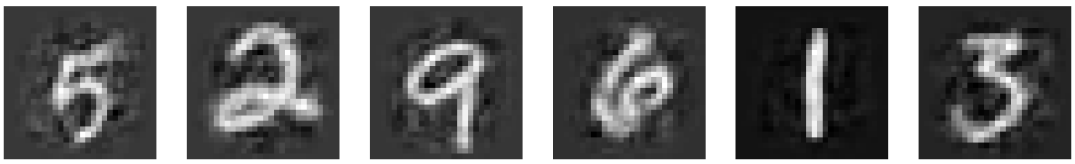
Figure 3.3 shows a sample of images from the test set of MNIST together with their reconstructed counterparts using different layer widths for the encoder and decoder. Figure 3.4 shows the same for CIFAR-10. While, visually there is not much difference for MNIST, the difference between the reconstructions in CIFAR-10 is clear. Surprisingly, increasing the layer width does not improve the quality of reconstruction, rather it makes it worse. This is due to the fact that as the layer width is increased, the network tends to overfit on the training set and has worse generalization for samples on the test set. If the layer width is too small, the sampled parameters are too few and cannot accurately capture all the intrinsic information in the data, but a layer width that is too large tends to lead to poor generalization. Hence, it is best to have a moderately large layer width.

Embedding Dimension

Not surprisingly, our experiments show that as the encoding dimension is increased, the quality of reconstruction also increases. A larger embedding dimension allows the autoencoder to capture more complex patterns and relationships in the input data. With a higher-dimensional embedding space, the autoencoder has more capacity to represent diverse variations and nuances present in the data. The model

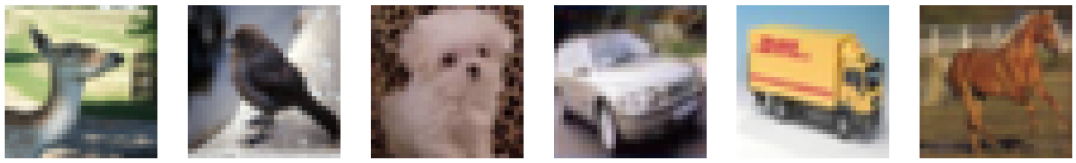
(a) Images from Test Set

(b) Reconstructed Images with layer width = 1000

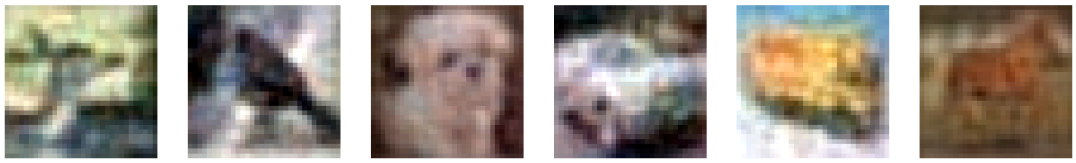


(c) Reconstructed Images with layer width = 3000

Figure 3.3: Reconstructed MNIST Test Set Images with the widths of the encoder and decoder = [1000, 3000]. The number of positive pairs for this experiment was set to 2 and the encoding dimension is set to 50.



(a) Images from Test Set



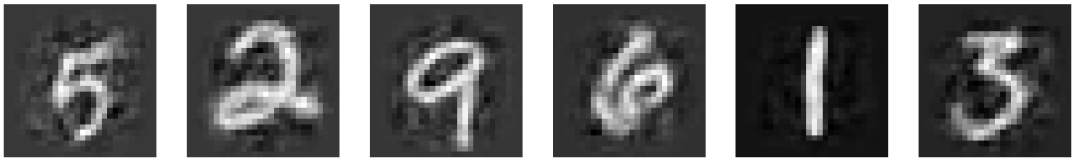
(b) Reconstructed Images with layer width = 2000

(c) Reconstructed Images with layer width = 4000

Figure 3.4: Reconstructed CIFAR10 Test Set Images with the widths of the encoder and decoder = [2000, 4000]. The number of positive pairs for this experiment was set to 2 and the encoding dimension is set to 400.

can better preserve important features and details from the input when mapping it to the latent space and decoding it back to the original space. It also facilitates better generalization to unseen data by learning more robust and informative representations. With a larger embedding dimension, the autoencoder can compress the input data into a lower-dimensional space with less information loss and enables smoother interpolation between different data points in the latent space. The results for varying the embedding dimension for MNIST and CIFAR-10 are shown in Figure 3.5 and 3.6 respectively. What is very promising is that even when the dimension of the data is reduced to as low as approximately one-fifth of the original data dimension, the reconstruction is visually very similar to the original data.

(a) Images from Test Set



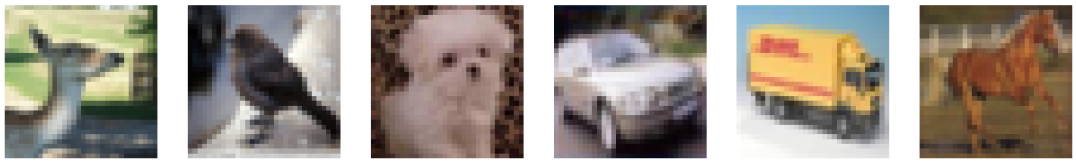
(b) Reconstructed Images with encoding dimension = 50

(c) Reconstructed Images with encoding dimension = 300

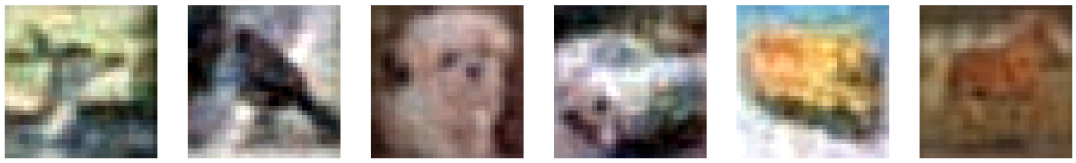
Figure 3.5: Reconstructed MNIST Test Set Images with the encoding dimension = [50, 300]. The number of positive pairs for this experiment was set to 2 and the layer width is set to 3000.

Number of positive pairs

The number of positive pairs is the number of samples chosen as the similar points for a single data sample, the embeddings of which are used to calculate the covariance matrix. While it may seem that increasing the number of positive pairs, would lead



(a) Images from Test Set



(b) Reconstructed Images with encoding dimension = 400

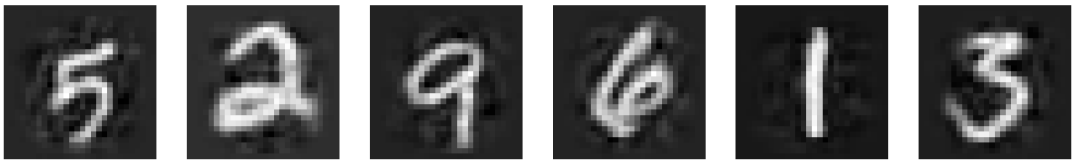
(c) Reconstructed Images with encoding dimension = 800

Figure 3.6: Reconstructed CIFAR-10 Test Set Images with the encoding dimension = [400, 800]. The number of positive pairs for this experiment was set to 2 and the layer width is set to 2000.

to better reconstruction and better generalization, that does not seem to be the case as can be seen by the results of the experiments in Figures 3.7 and 3.8. Even when the number of positive pairs is changed from 2 to 15, there is visually very little difference in the reconstruction. It can be argued that this small improvement does not justify the additional cost associated with using a larger number of positive pairs.

(a) Images from Test Set

(b) Reconstructed Images with number of positive pairs = 2



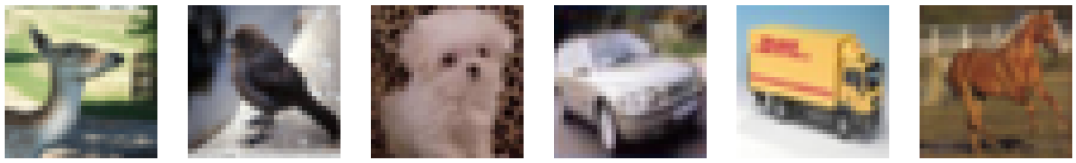
(c) Reconstructed Images with number of positive pairs = 15

Figure 3.7: Reconstructed MNIST Test Set Images with number of positive pairs = [2, 15]. The encoding dimension for this experiment was set to 100 and the layer width is set to 1000.

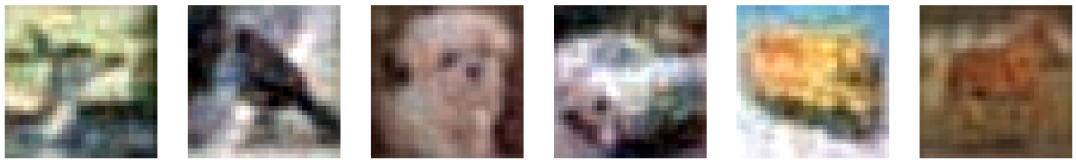
3.4.2 Reconstruction Error

The primary metric used to assess the performance of the autoencoder is the reconstruction error, quantifying the discrepancy between the input images and their reconstructions. Lower reconstruction error values indicate better reconstruction fidelity. We have used the mean squared error to measure the reconstruction error. The form of the error is given by

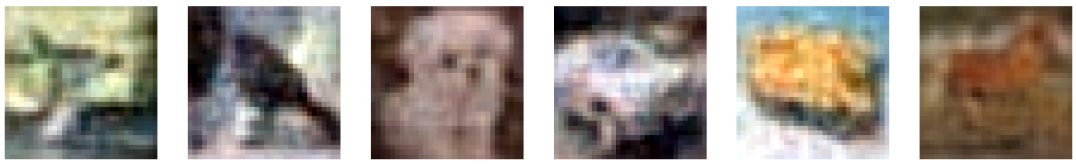
$$MSE = \frac{1}{N} = \sum_{i=1}^{N} (X_i - \hat{X}_i)^2. \tag{3.16}$$



(a) Images from Test Set



(b) Reconstructed Images with number of positive pairs = 2

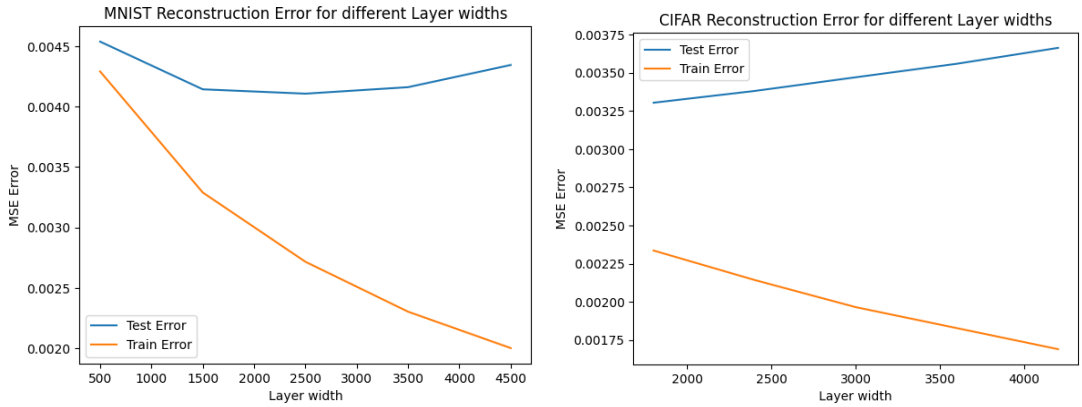


(c) Reconstructed Images with number of positive pairs = 15

Figure 3.8: Reconstructed MNIST Test Set Images with number of positive pairs = [2, 15]. The encoding dimension for this experiment was set to 400 and the layer width is set to 2000.

Layer width for encoder and decoder

Having a larger layer width means that there are more neurons in the encoder and the decoder. For our experiments, the layer width was set to be the same for the encoder and the decoder. More neurons allow the model to capture more complex patterns and relationships in the data. This can help discover underlying structures and features crucial for constructing a good latent space representation. With more neurons, the model can better approximate intricate decision boundaries between classes, leading to more accurate dimensionality reduction. This would lead us to believe that increasing the layer width should reduce the reconstruction error. However, as seen in Figure 3.9, increasing the layer width does decrease the reconstruction error on the training set, but a considerablly large layer width leads to a higher reconstruction error on the test set. This is reflective of the model overfitting on the training set due to a large layer width. On MNIST, the optimal layer width was 1500 with a reconstruction error of 0.0042 on the standardized images. On CIFAR-10, as the layer width was increased beyond 2000, the test error constantly increased and a layer width of less than 2000 is too small to be used with the eigen solver and a larger embedding dimension like 800. Hence the most optimal layer width to be used is 2000 with a test error of 0.0033.



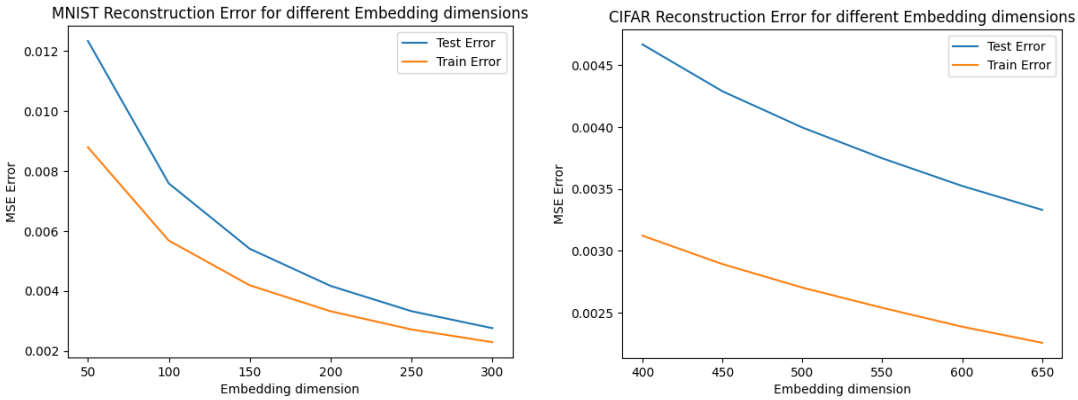
(a) MNIST
Embedding Dimension = 200
Number of positive pairs = 2

(b) CIFAR
Embedding Dimension = 650
Number of positive pairs = 2

Figure 3.9: Train and Test Reconstruction Error for MNIST and CIFAR-10 for different layer widths.

Embedding Dimension

The results of this section are consistent with expectations. A larger embedding dimension would lead to more data from the original images being preserved, which would ultimately lead to better reconstruction and hence, a lower reconstruction error. This is seen in Figure 3.10 where the train and test reconstruction error decreases as the encoding dimension is increased. The rate of decrease of the error, however, is not linear but also decreases. When the embedding dimension is set to almost a quarter of the original dimension, the reconstruction errors are very low, reaching 0.003 and 0.0035 for MNIST and CIFAR-10 respectively. This goes to show that even when the input samples are reduced to a quarter of their original size, enough information is preserved by the network to reconstruct them accurately enough.



(a) MNIST
Layer Width = 1000
Number of positive pairs = 2

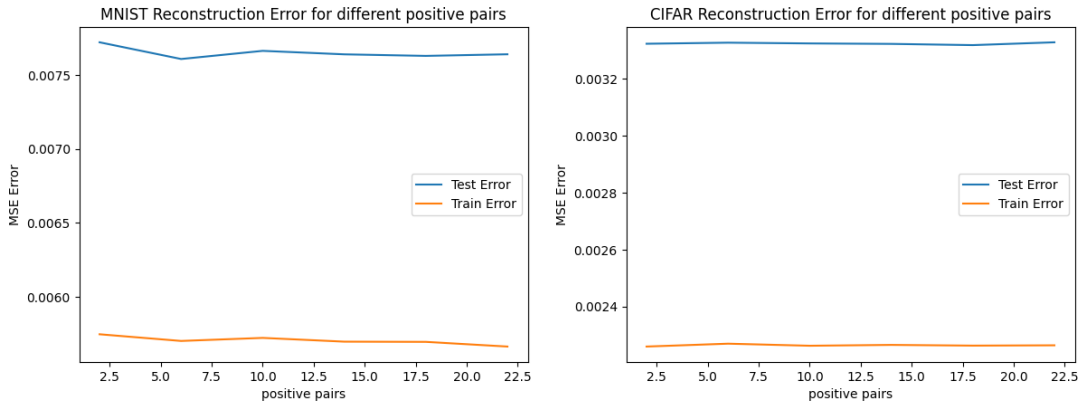
(b) CIFAR
Layer Width = 2000
Number of positive pairs = 2

Figure 3.10: Train and Test Reconstruction Error for MNIST and CIFAR for different embedding dimensions.

Number of positive pairs

Experiments conducted by keeping constant the layer width and embedding dimensions and varying the number of samples used as positive pairs for each data sample revealed surprising results. While the hypothesis was that increasing the number of positive pairs would improve the network performance, the results revealed otherwise. Having more similar samples was believed to lead to clearer interpretations of the correlations and canonical variates. This is because the patterns

of association between the two sets of variables become more pronounced when more similar samples are associated to each other. This was especially believed to be true for the CIFAR-10 dataset, where samples that belong to the same class can still be very different from each other, so using a larger number of positive pairs would uncover specific patterns of association that may be obscured in more diverse samples. However, as shown in figure 3.11, the reconstruction error for both the training and test set on both datasets remained relatively constant when using a different number of positive pairs.



(a) MNIST
Layer Width = 1000
Embedding Dimension = 200

(b) CIFAR
Layer Width = 2000
Embedding Dimension = 650

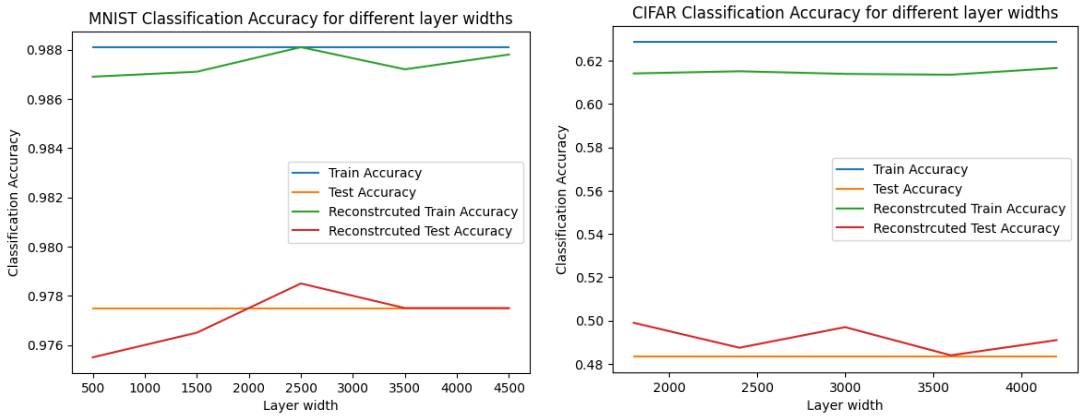
Figure 3.11: Train and Test Reconstruction Error for MNIST and CIFAR for different numbers of positive pairs.

3.4.3 Accuracy of classification on Reconstructed Images

We initialized a classification model again using the SWIM algorithm to compare the classification of the reconstructed images against the original images. This is used as a metric to judge the reconstruction because if a classifier assigns the same labels to both the original and reconstructed images, the reconstruction can be considered a good one. The model is comprised of a single dense layer which is a fully connected layer followed by a linear linear that analytically computes the weights and is trained on the training set. Figure 3.12, Figure 3.13 and Figure 3.14 show the classification accuracy on the training, test, reconstructed training, and reconstructed test sets based on varying the layer width for the encoder and decoder, the embedding dimension and the number of positive pairs respectively.

Two important deductions can be made from these figures. Firstly, for all three hyperparameters: the layer width, embedding dimension, and the number of positive pairs, the values used have a very small impact on the classification accuracy. In the previous subsection, we saw that changing the layer width and embedding dimension did impact the reconstruction error, so the reconstruction is indeed affected by these parameters. However, it is not affected to the extent that it would be classified differently. Even when using suboptimal values of these hyperparameters, the reconstruction is good enough that a decent classifier can correctly identify the class that the sample belongs to. What these two hyperparameters do is control the amount of detail preserved in the reconstruction.

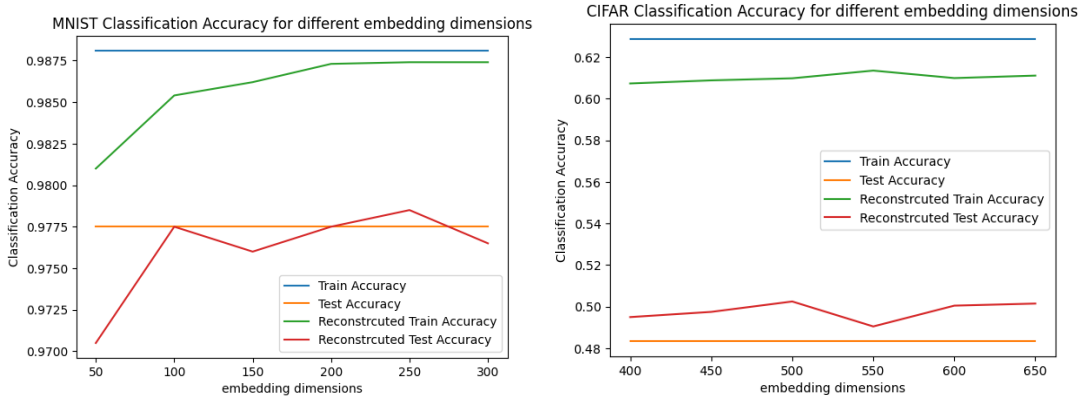
The second important point concerns the accuracy values of the classification. For the MNIST dataset, the classification accuracy on the test set is in the range of 97%, which for a simple classifier that we have experimented with is good enough. However for CIFAR10, although the the classification accuracy on the training sets is approximately 60%, the accuracy on the test sets falls within the range of 48% to 50%. However, it is not the absolute accuracy values that are important here but the difference between the accuracy in the training and reconstructed training set and the test and reconstructed test set. We can see from the figures that the accuracy on the reconstructed sets is very close to the accuracy on the original training and test sets, suggesting that the reconstructed images were very similar to the original ones.



(a) MNIST
Embedding Dimension = 200
Number of positive pairs = 2

(b) CIFAR
Embedding Dimension = 650
Number of positive pairs = 2

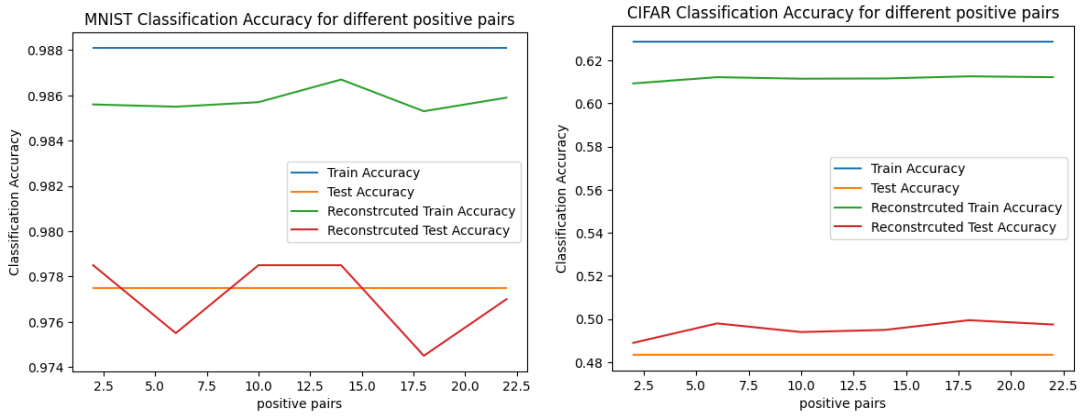
Figure 3.12: Classification Accuracy on the original training and test sets and the reconstructed sets for MNIST and CIFAR-10 for different layer widths.



(a) MNIST
Layer Width = 1000
Number of positive pairs = 2

(b) CIFAR
Layer Width = 2000
Number of positive pairs = 2

Figure 3.13: Classification Accuracy on the original training and test sets and the reconstructed sets for MNIST and CIFAR-10 for different embedding dimensions.



(a) MNIST
Layer Width = 1000
Embedding Dimension = 200

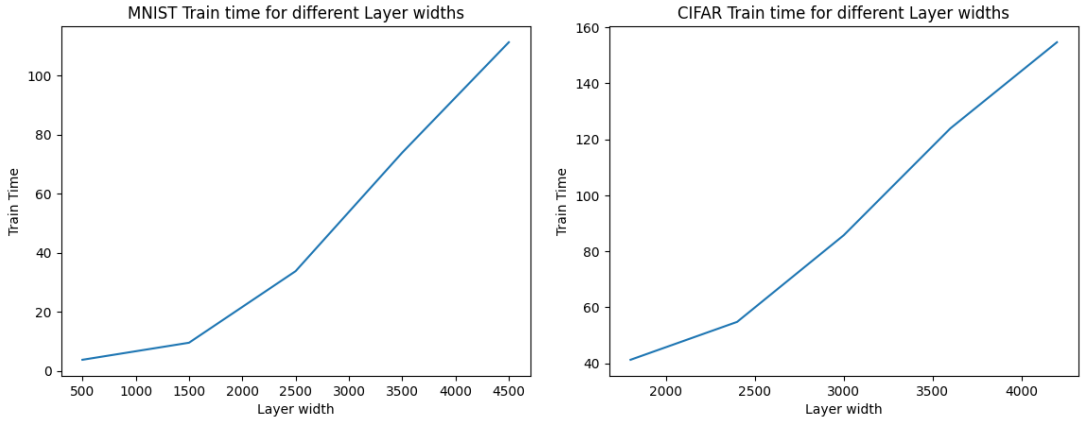
(b) CIFAR
Layer Width = 2000
Embedding Dimension = 650

Figure 3.14: Classification Accuracy on the original training and test sets and the reconstructed sets for MNIST and CIFAR-10 for different numbers of positive pairs.

3.4.4 Timing

Layer width for encoder and decoder

The training time increase almost linearly as the layer width of the encoder and decoder is increased. Increasing the layer width typically increases the computational complexity of the network. This is because the number of parameters (weights and biases) to be sampled grows linearly with the number of neurons in the layer. Consequently, training the model may require more computational resources and time. Morover, solving the weights of the last linear layer involves matrix multiplication and inversion, so as the layer width increases, the size of the weight matrices also increases, leading to larger matrix operations. These operations can be computationally intensive and contribute to longer training times. Figure 3.15 confirms this hypothesis.



(a) MNIST
Embedding Dimension = 200
Number of positive pairs = 2

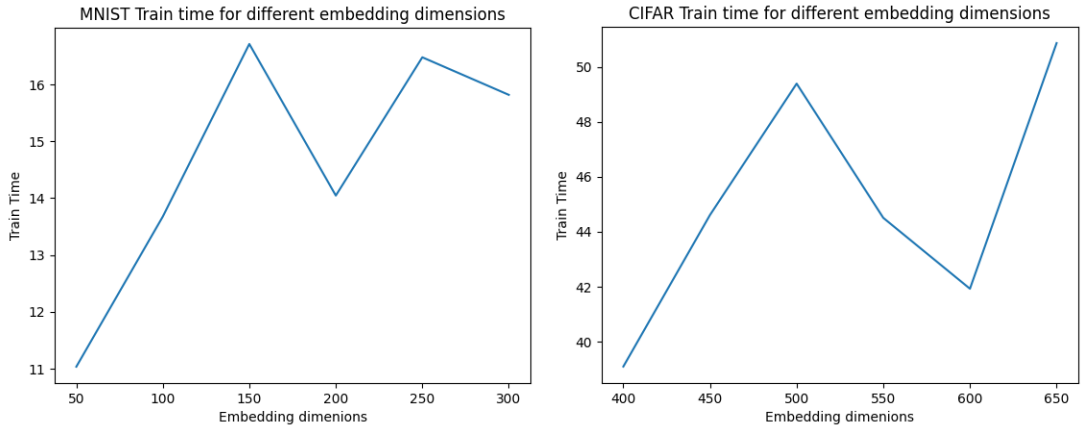
(b) CIFAR
Embedding Dimension = 650
Number of positive pairs = 2

Figure 3.15: Training time (in seconds) for MNIST and CIFAR-10 for different layer widths.

Embedding Dimension

The size of the embedding space shows no significant relationship to the training time as shown in figure 3.16. While a larger embedding dimension means that there are more neurons in the second hidden layer of the network which, in turn, would imply that sampling more weights and biases would take more time, the results

are not consistent with this assumption. A possible explanation for this could be that for a larger embedding dimension, the `lobpcg` method takes fewer iterations to converge to a solution, which is why for some embedding dimensions, the training time dips. Nevertheless, the difference in training times is just a few seconds so we can conclude that the embedding dimension does not influence the training time.



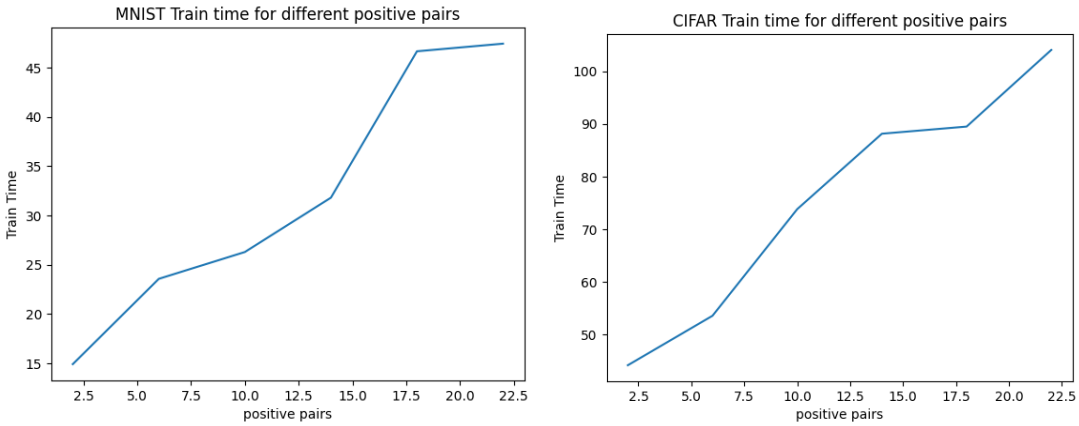
(a) MNIST
Layer Width = 1000
Number of positive pairs = 2

(b) CIFAR
Layer Width = 2000
Number of positive pairs = 2

Figure 3.16: Training time (in seconds) for MNIST and CIFAR-10 for different embedding dimensions.

Number of positive pairs

The number of positive pairs used also impacts the training time. The two are directly proportional; an increase in the number of positive pairs leads to an increase in training time. The reason for this could be attributed to the fact that as the number of positive pairs increases, the size of the covariance matrix from which the weights for the embedding are calculated also increases. The size of the covariance matrix also affects the performance and convergence of the `lobpcg` method. While `lobpcg` is designed for large-scale problems, its computational cost can become significant for very large sets of eigenvalues. The algorithm involves iterative matrix-vector multiplications and preconditioning steps, which can become time-consuming as the problem size grows. For smaller sets, it may converge faster and more reliably, as the iterations are performed on a smaller-dimensional subspace.



(a) MNIST
Layer Width = 1000
Embedding Dimension = 200

(b) CIFAR
Layer Width = 2000
Embedding Dimension = 650

Figure 3.17: Training Time (in seconds) for MNIST and CIFAR-10 for different numbers of positive pairs.

3.5 Comparison to other approaches

To determine how our approach compares to other approaches in the domain, we carried out experiments comparing our network with the following:

- A network with the same architecture as ours, shown in Figure 3.1, but with weights and biases sampled using a normal and uniform distribution respectively, as opposed to our approach in which the network parameters are sampled using the SWIM algorithm.
- A fully connected network, with an encoding layer, a latent space layer and a decoding layer, trained iteratively using gradient descent. The optimizer used was the Adam optimizer. The loss function used was the mean squared error loss and we trained the network for 100 epochs with an early stopping criteria set on the loss with a patience of 3.

The train and test set of 20,000 and 2,000 samples is used for all experiments. The embedding dimension for MNIST is set to 100, while for CIFAR-10 it is set to 650. The results for the test reconstruction error, classification accuracies, and training times are shown in Table 3.3 for MNIST and Table 3.4 for CIFAR10. Figure 3.18 shows a few of the test samples of MNIST and Figure 3.19 shows their equivalent reconstructed samples using our approach. Figure 3.20 and Figure 3.21 show the

same for CIFAR-10. The following subsections show the reconstruction of the same test set using other approaches, along with a brief discussion.

Table 3.3: Hyperparameters and Results for comparison experiment on MNIST.

	Our approach	**Random sampling with Tanh Activation**	**Random sampling with ReLU Activation**	**Fully connected network trained iteratively**
Activation Function	Tanh	Tanh	RelU	Tanh
Encoder Width	1000	1000	1000	1000
Decoder Width	1000	1000	1000	784
Positive Pairs	2	2	2	-
Learning Rate	-	-	-	0.01
Test Reconstruction Error	0.0045	0.0105	0.0098	0.0029
Train Reconstruction Classification Accuracy	0.9845	0.9299	0.922	0.9847
Test Reconstruction Classification Accuracy	0.9785	0.90	0.918	0.98
Train Time	7.12	5.48	5.84	65.6

3.5.1 Data-agnostic sampling

A visual inspection of the reconstructed images in Figures 3.22 and 3.23 shows that the approach is not viable as the quality of the images is very poor and in the case of the CIFAR-10 data, it is not visually possible to identify the classes of the image samples. Using the ReLU activation function shows slightly better performance but the reconstruction is still considerably worse than our approach. The values in table 3.3 and 3.4 show that the reconstruction error is higher and the train accuracy

Figure 3.18: Subset of images from the MNIST Test set.

Figure 3.19: Reconstructed MNIST images using the SWIM Algorithm for parameter sampling.

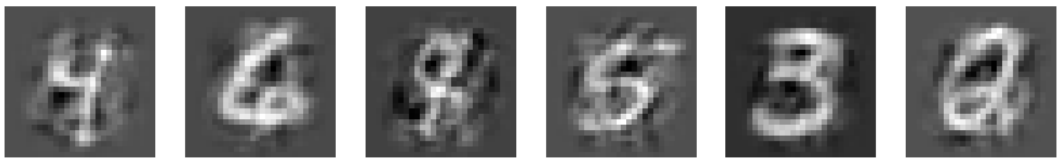
Figure 3.20: Subset of images from the CIFAR-10 Test set.

Figure 3.21: Reconstructed CIFAR-10 images using the SWIM Algorithm for parameter sampling.

Table 3.4: Hyperparameters and Results for comparison experiment on CIFAR-10.

	Our approach	**Random sampling with Tanh Activation**	**Random sampling with ReLU Activation**	**Fully connected network trained iteratively**
Activation Function	Tanh	Tanh	ReLU	Tanh
Encoder Width	2000	2000	2000	2000
Decoder Width	2000	2000	2000	3072
Positive Pairs	2	2	2	-
Learning Rate	-	-	-	0.01
Test Reconstruction Error	0.0031	0.01345	0.0195	0.0035
Train Reconstruction Classification Accuracy	0.6149	0.5553	0.5874	0.6083
Test Reconstruction Classification Accuracy	0.507	0.474	0.4935	0.5025
Train Time	45.35	47.25	38.02	103.16

is lower for the data agnostic sampling approach compared our our data-driven approach. The training time for our approach is slightly longer since data-agnostic sampling is simpler than data-driven sampling. However, a few extra seconds of training time is an acceptable trade-off for better reconstruction quality.



(a) Reconstructed MNIST images using Tanh as the activation function.

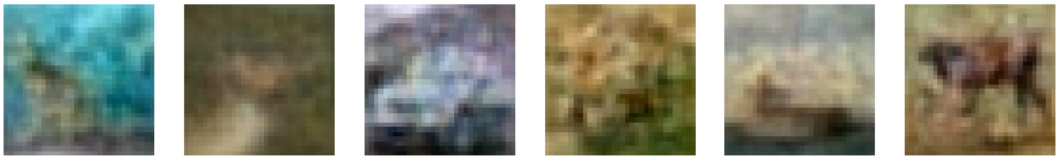
(b) Reconstructed images using ReLU as the activation function.

Figure 3.22: Reconstructed MNIST images using the data agnostic sampling method, where the random and uniform distributions are used for sampling weights and biases respectively.

3.5.2 Iteratively trained fully connected networks

The reconstructions of the subset of images is shown in Figure 3.24 for MNIST and figure 3.25 for CIFAR-10. Visually, the reconstructions look better and clearer than the ones we get using our approach and more detail seems to be preserved. This is further confirmed for the MNIST dataset in Table 3.3 where it can be seen that the reconstruction error is lower and the classification accuracy on the training and test set is higher for the iteratively trained network as compared to our approach. For CIFAR-10, Table 3.4 shows that the iteratively trained network has slightly higher reconstruction error and slightly lower training and test classification accuracy compared our our approach, but the difference is minimal. However, the iteratively trained network has a train time almost thrice the train time of the sampled network and requires tuning more hyperparameters such as the learning rate, the optimizer, the number of epochs, etc. Overall, our network performs on par, has fewer hyperparameters, and saves considerable train time.

(a) Reconstructed images using Tanh as the activation function.



(b) Reconstructed images using ReLU as the activation function.

Figure 3.23: Reconstructed CIFAR-10 images using the data agnostic sampling method, where the random and uniform distributions are used for sampling weights and biases respectively.

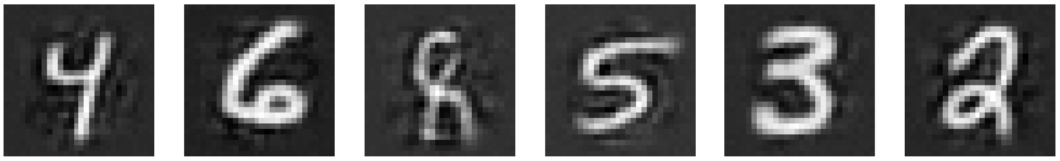


Figure 3.24: Reconstructed MNIST images with parameters trained via gradient descent, and Tanh used as the activation function.

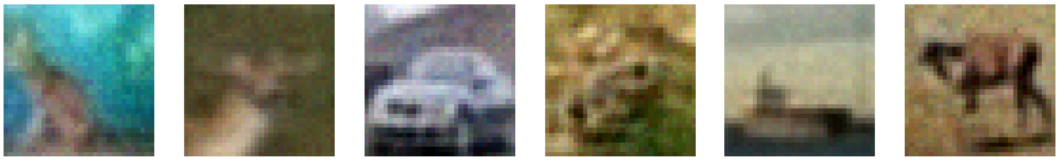


Figure 3.25: Reconstructed CIFAR-10 images with parameters trained via gradient descent, and Tanh used as the activation function.

3.6 Limitations

Although our approach demonstrated good results, it does have its limitations. These include the memory requirements when dealing with large, high-dimensional data, the computational complexity of calculating the eigenvalues and eigenvectors of a large data matrix and its inability to be used with low-dimensional manifolds like the swiss roll.

3.6.1 Scalability

Scalability issues primarily arise when dealing with large-scale datasets or models with a high number of parameters. The model requires storing the entire training dataset in memory to compute the hidden layer activations and solve the output weights analytically. Hence, for datasets having a very high dimension or a very large number of samples, this can become impractical due to memory limitations. Storing and processing large matrices representing the input data and hidden layer activations exhausts available memory resources, leading to memory overflow or slowdowns. Moreover, the computational cost of computing the output weights analytically scales quadratically with the number of hidden neurons, which can become prohibitive for large neural networks. To overcome this, one approach is to store the data on disk, but reading the data from the disk each time also results in longer training time. We did encounter this issue in our experiments as well and had to sub-sample the training and test sets to 20,000 and 2,500 images respectively. In resource-constrained environments, such as embedded systems or IoT devices, the memory and computational overhead of this model may limit their practical utility.

3.6.2 Computational complexity of computing Eigenvectors

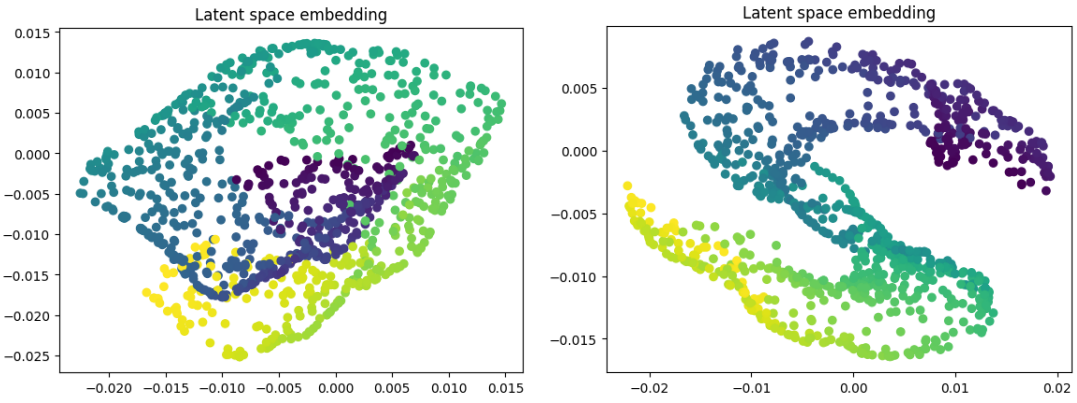
Our approach requires us to compute the eigenvectors of the covariance matrix of the feature space to determine the optimal weights for the embedding. However, computing the eigenvectors can be computationally expensive, especially for large matrices. We have used the `lobpcg` function from `scipy` to compute the eigenvectors. Although it is designed to be efficient for large-scale eigenvalue problems, it may still exhibit computational challenges for extremely large matrices. It has a computational complexity of $O(n^2)$ for dense matrices and $O(n_{nz})$ for sparse matrices, where n is the size of the matrix and n_{nz} is the number of non-zero entries. For very large matrices, this computational cost may become prohibitive. Another problem with this approach is that it requires the number of eigenvectors to be much larger than the dimensionality of the data on which they are computed. This, in our case, can

be achieved by having a large with for the encoder but that drastically increases the training time.

3.6.3 Poor reconstruction for globally non-linear, low dimensional manifolds

Whilst our approach shows good results with high dimensional data, it fails where data is low dimensional and globally non-linear. An example of this is the swiss roll manifold which can be described as can be described as a two-dimensional surface embedded in three-dimensional space shown in Figure 2.4. The approach relies on finding linear combinations of the data in a high-dimensional feature space. While the kernel trick allows us to implicitly map the data into a higher-dimensional space, it still assumes that the underlying relationships between the variables are linear. For datasets with globally nonlinear manifolds, linear transformations may not adequately capture the complex nonlinear relationships. Moreover, the low dimensionality of the data does not allow for a good translation into a high-dimensional feature space via the sampling algorithm.

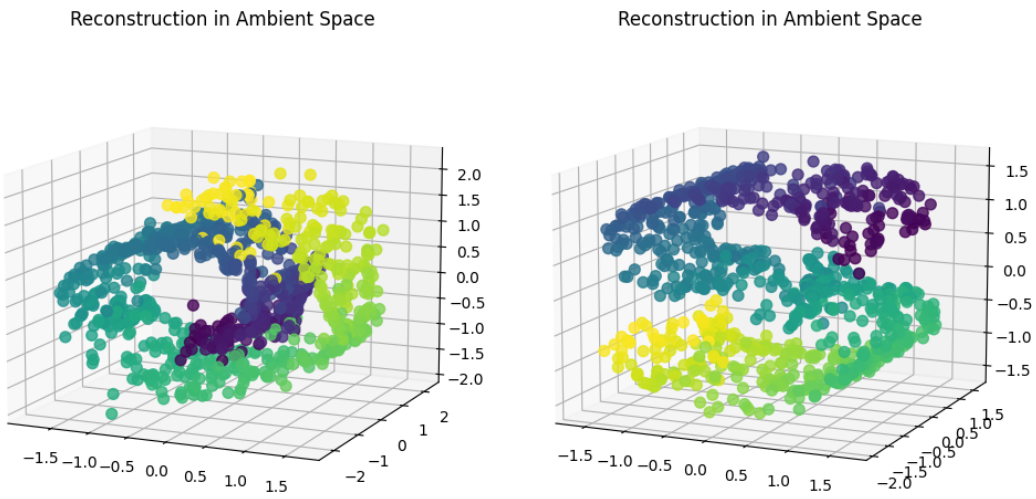
Figure 3.26 shows the two-dimensional embedding of the Swiss Roll and S Curve dataset. It can be seen that the embeddings fail to unwarp both datasets and capture the intrinsic structure of the manifolds, the way that some manifold learning algorithms, such as Isomap, do. Owing to the poor embeddings that are captured, the reconstruction, shown in Figure 3.27 is also far from perfect.



(a) Swiss Roll

(b) S Curve

Figure 3.26: Two dimensional embeddings for the swiss roll and s curve datasets using our method. The figure shows how our method fails to "unwrap" the manifolds.



(a) Swiss Roll

(b) S Curve

Figure 3.27: Reconstruction of the swiss roll and s curve data using our method.

4 Discussion

4.1 Conclusion

In this thesis, we presented a novel strategy for extending the use of sampled neural networks to the autoencoder setting. We addressed the challenge of capturing the intrinsic structure of data by combining kernel representation learning with data-driven sampling from the SWIM algorithm. We were able to obtain a low-dimensional embedding of the original data by first mapping it to a higher-dimensional feature space with a sampled linear network and then translating it into a lower-dimensional space using contrastive kernel learning. This lower dimensional space was representative enough to reconstruct the original data accurately, again by using a linear sampled network.

Through our experimental results, we demonstrated the effectiveness of our approach in capturing meaningful data representations and reconstructing input data accurately. To validate our proposed approach, we conducted experiments on benchmark image datasets, including MNIST and CIFAR-10. We evaluated the performance using various metrics tailored to the nature of the datasets, such as pixel-wise error, visual inspection of reconstructed images, and classification error on reconstructed images. The approach also reduces the number of hyperparameters that require manual tuning.

The results for the approach are very promising, showing a very low reconstruction error and visually appearing to have accurately reconstructed the data. Classification of the original images vs the reconstructed images also shows that the classification error is similar in both sets of images. The reconstruction works best when the encoder and decoder have a sufficiently large width and the activation function used is Tanh. Not surprisingly, the quality of the reconstructed data improves as the dimension of the encoding space is increased. However, a sufficiently good reconstruction can be obtained with the encoding space having a size of a third of the original data. We also prove how our approach has ties to kernel canonical correlation analysis, with the two having the same functional form.

In conclusion, our study contributes to the advancement of autoencoder models by introducing a novel framework that combines sampled neural networks with kernel representation learning. We believe that our findings provide valuable insights

and pave the way for further research in the field of data-driven sampling and representation learning.

4.2 Future Work

The domain of sampled networks is a relatively unexplored one and there remains a lot of potential for research and advancements. One area for future research could be to develop a novel approach for using sampled networks for nonlinear dimensionality reduction and manifold learning. These networks could leverage their simplicity and efficiency while incorporating nonlinear transformations to capture complex data structures more effectively. In this regard, future work could also be directed towards different forms of sampling techniques that allow the sampling scheme to accurately capture globally non-linear manifolds.

We mentioned the issues of sampled networks concerning scalability to very large datasets. To address scalability issues, future research could explore areas such as mini-batch training, distributed computing, model compression, and approximation methods. These approaches aim to reduce memory usage, improve computational efficiency, and enable the training of sampled networks on large-scale datasets while mitigating scalability challenges.

Another interesting area of research could be the shift from semi-supervised or supervised dimensionality reduction approaches (as we have used in our work) to unsupervised approaches for reconstruction in sampled autoencoders. The current approaches that work with sampled autoencoders usually involve using either the data labels or some information about the data, such as semantic similarities between data samples, thus belonging to the supervised or semi-supervised class of algorithms respectively. A big leap will be to shift from such a supervised or semi-supervised setting into a completely unsupervised setting so that the autoencoders can be used for further downstream tasks such as anomaly detection, denoising or domain adaptations.

Bibliography

[1] E. L. Bolager, I. Burak, C. Datar, Q. Sun, and F. Dietrich. "Sampling weights of deep neural networks". In: *Advances in Neural Information Processing Systems* 36 (2024).

[2] C. Gallicchio and S. Scardapane. "Deep randomized neural networks". In: *Recent Trends in Learning From Data: Tutorials from the INNS Big Data and Deep Learning Conference (INNSBDDL2019)*. Springer. 2020, pp. 43–68.

[3] G. Dudek. "Autoencoder based Randomized Learning of Feedforward Neural Networks for Regression". In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, pp. 1–8.

[4] K. Sun, J. Zhang, C. Zhang, and J. Hu. "Generalized extreme learning machine autoencoder and a new deep neural network". In: *Neurocomputing* 230 (2017), pp. 374–381.

[5] S. Zhou, C. Deng, W. Wang, G.-B. Huang, and B. Zhao. "GenELM: Generative extreme learning machine feature representation". In: *Neurocomputing* 362 (2019), pp. 41–50.

[6] P. Esser, M. Fleissner, and D. Ghoshdastidar. "Non-Parametric Representation Learning with Kernels". In: *arXiv preprint arXiv:2309.02028* (2023).

[7] S.-Y. Huang, M.-H. Lee, and C. K. Hsiao. "Kernel canonical correlation analysis and its applications to nonlinear measures of association and test of independence". In: *Institute of Statistical Science: Academia Sinica, Taiwan* (2006).

[8] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. "Extreme learning machine: theory and applications". In: *Neurocomputing* 70.1-3 (2006), pp. 489–501.

[9] P. I. Wójcik. "Random projection in deep neural networks". In: *arXiv preprint arXiv:1812.09489* (2018).

[10] S. Scardapane and D. Wang. "Randomness in neural networks: an overview". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7.2 (2017), e1200.

[11] J. Wang, S. Lu, S.-H. Wang, and Y.-D. Zhang. "A review on extreme learning machine". In: *Multimedia Tools and Applications* 81.29 (2022), pp. 41611–41660.

[12] L. L. C. Kasun, H. Zhou, G.-B. Huang, and C. M. Vong. "Representational learning with extreme learning machine for big data". In: (2013).

[13] S. Ding, N. Zhang, J. Zhang, X. Xu, and Z. Shi. "Unsupervised extreme learning machine with representational features". In: *International Journal of Machine Learning and Cybernetics* 8.2 (2017), pp. 587–595.

[14] W. Zhu, J. Miao, L. Qing, and G.-B. Huang. "Hierarchical extreme learning machine for unsupervised representation learning". In: *2015 international joint conference on neural networks (ijcnn)*. IEEE. 2015, pp. 1–8.

[15] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley. "Reproducing Kernel Hilbert Space, Mercer's Theorem, Eigenfunctions, Nystr\" om Method, and Use of Kernels in Machine Learning: Tutorial and Survey". In: *arXiv preprint arXiv:2106.08443* (2021).

[16] C. M. Wong, C. M. Vong, P. K. Wong, and J. Cao. "Kernel-based multilayer extreme learning machines for representation learning". In: *IEEE transactions on neural networks and learning systems* 29.3 (2016), pp. 757–762.

[17] C.-M. Vong, C. Chen, and P.-K. Wong. "Empirical kernel map-based multilayer extreme learning machines for representation learning". In: *Neurocomputing* 310 (2018), pp. 265–276.

[18] P. Laforgue, S. Clémençon, and F. d'Alché-Buc. "Autoencoding any data through kernel autoencoders". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR. 2019, pp. 1061–1069.

[19] M. Meilă and H. Zhang. "Manifold learning: what, how, and why". In: *Annual Review of Statistics and Its Application* 11 (2023).

[20] M. Balasubramanian and E. L. Schwartz. "The isomap algorithm and topological stability". In: *Science* 295.5552 (2002), pp. 7–7.

[21] L. K. Saul and S. T. Roweis. "An introduction to locally linear embedding". In: *unpublished. Available at: http://www. cs. toronto. edu/~ roweis/lle/publications. html* (2000).

[22] M. C. Cieslak, A. M. Castelfranco, V. Roncalli, P. H. Lenz, and D. K. Hartline. "t-Distributed Stochastic Neighbor Embedding (t-SNE): A tool for eco-physiological transcriptomic analysis". In: *Marine genomics* 51 (2020), p. 100723.

[23] A. Maćkiewicz and W. Ratajczak. "Principal components analysis (PCA)". In: *Computers & Geosciences* 19.3 (1993), pp. 303–342.

[24] Q. Wang, Z. Qin, F. Nie, and X. Li. "Spectral embedded adaptive neighbors clustering". In: *IEEE transactions on neural networks and learning systems* 30.4 (2018), pp. 1265–1271.

[25] J. De la Porte, B. Herbst, W. Hereman, and S. Van Der Walt. "An introduction to diffusion maps". In: *Proceedings of the 19th symposium of the pattern recognition association of South Africa (PRASA 2008), Cape Town, South Africa*. 2008, pp. 15–25.

[26] M. Kuss and T. Graepel. "The geometry of kernel canonical correlation analysis". In: (2003).

[27] K. C. Tsiolis. "Contrastive Learning as Kernel Approximation". In: *arXiv preprint arXiv:2309.02651* (2023).

[28] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny. "Barlow twins: Self-supervised learning via redundancy reduction". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 12310–12320.

[29] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan. "Supervised contrastive learning". In: *Advances in neural information processing systems* 33 (2020), pp. 18661–18673.

[30] R. Balestriero and Y. LeCun. "Contrastive and non-contrastive self-supervised learning recover global and local spectral embedding methods". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 26671–26685.