UiT The Arctic University of Norway

Faculty of Science and Technology

Department of Physics and Technology

Leveraging Kernels for Unsupervised Learning

Sigurd Løkse

A dissertation for the degree of Philosophiae Doctor - September 2020

Abstract

Kernel methods have been a central part of the machine learning arsenal for several decades. Within this framework, unsupervised learning has been a particularly challenging area. This is due to the inherent nature of unsupervised learning tasks, where important information about the structure of the data is unknown to the user, and as such it is difficult to design a kernel or system to solve the problem at hand. This thesis aims to bridge this knowledge gap on a multitude of challenges within the field.

Firstly, we address an important challenge within kernel methods for unsupervised learning, namely that of kernel parameter sensitivity. The process of finding the best parameter for the problem at hand usually depends on information which is unavailable for unsupervised tasks. Inspired by ideas from ensemble learning, we design a kernel for vectorial data with missing elements that automatically adapts to the inherent structures of the data, in order to decouple the parameter choice from the problem at hand. We perform experiments on spectral clustering and unsupervised ranking tasks with promising results.

Next, we develop a kernelized approximation method for unsupervised ranking using the Personalized PageRank (PPR). This method is based on novel insights on the PPR, which through a PPR–specific low–rank representation akin to Kernel PCA naturally leads to an out–of–sample approximation. The method is based on the spectrum of a specific matrix. We provide error bounds for the approximation which is used to order the eigenvectors/eigenvalues to minimize the approximation error. We perform a range of experiments to support our novel insights and show that our method may even outperform the PPR.

The final part of this thesis synergetically combines kernel methods and neural networks in various unsupervised tasks. Firstly, we design a kernelized autoencoder that incorporates similarities between datapoints through a kernel function in order to learn meaningful representations in code space. Secondly, we propose a novel deep learning approach to clustering, utilizing kernel–based information theoretic losses, with promising experimental results when compared to state–of–the art methods on challenging problems. Finally, we incorporate an unsupervised dimensionality reduction method (e.g. Kernel PCA) in–between the reservoir and readout layer of an Echo State Network in order to capture the dynamics of the time series while reducing the dimensionality of the trainable parameters and improving accuracy.

Acknowledgments

Finishing this thesis marks the end of one journey and the beginning of another. In light of that, I have several people I would like to thank.

First and foremost, I would like to express my sincere gratitude to Professor Robert Jenssen for being a great supervisor, with unrelenting positivity and patience. I have no doubt that if it had not been for your excellent lectures, I would have done something completely different right now. When I was studying engineering, my plan was to finish my bachelor and to start working in the industry. I promptly changed my mind when I took the Signal Processing course, which you were teaching at the time. When I got back to reality after my health issues, you managed to secure extra funding for me, which was instrumental for giving me some time to recover. I would never have been able to finish this without your support. Thank you!

To Filippo, thank you for all the discussions, guidance, pizzas and game nights. You're a person one can always rely on, and I am really glad you decided to join us here in Tromsø when the opportunity arose.

To my old office mates, Jonas, Michael, Karl Øyvind and Luigi, thank you for many laughs and fruitful discussions. A special thanks to Michael and Karl Øyvind for proofreading this thesis.

To everyone at the UiT Machine Learning group. When I started my Ph.D., the group consisted of four Ph.D. students crammed into a small office and Robert. Since then, we have outgrown every office space we have been given, and are now too many to mention by name. So instead I would like to thank you collectively for all the discussions, support and laughs along the way.

To my committee members, thank you for taking the time to read my thesis. Due to COVID-19 restrictions, it is unlikely that you will attend the defense in person. If not, I hope I will be able to meet you in person some day.

Last but not least, to my family and friends. Thank you for all the support and patience. Especially in the last months when I haven't been available.

Sigurd Løkse
Tromsø, September 2020

Contents

List of Figures

Abbreviations

CNN Convolutional Neural Network

EM Expectation Maximiztaion

ESN Echo State Network

GMM Gaussian Mixture Model

GRU Gated Recurrent Unit

ITL Information Theoretic Learning

KPCA Kernel Principal Component Analysis

LSTM Long Short Term Memory

MAR Missing At Random

MCAR Missing Completely At Random

MKL Multiple Kernel Learning

MLP Multi–Layer Perceptron

NCut Normalized Cut

NMAR Not Missing At Random

PCA Principal Component Analysis

PCK Probabilistic Cluster Kernel

PDF Probability Density Function

PMF Probability Mass Function

PPR Personalized PageRank

RBF Radial Basis Function

RKHS Reproducing Kernel Hilbert Space

RNN Recurrent Neural Network

SC Spectral Clustering

Chapter 1

Introduction

Kernel methods were introduced in machine learning in the 90's as tools used to transform an inherently linear method to a powerful method capable of dealing with non–linear data [18]. In the nearly three decades since that time, kernel methods have served as one of the fundamental pillars of machine learning research [89], alongside for instance neural networks. Supported by a solid theoretical framework, kernel machines have been leaping forward to a multitude of learning tasks, including classification [18], clustering [36, 47, 71], dimensionality reduction [76, 128] and regression [3, 24, 38], and to different data domains through the choice of kernel function. The ability to deal with both non–vectorial data (e.g. documents/text [56, 97, 151], proteins [13, 49, 93], images [46] and graphs [148]) and vectorial data by selecting an appropriate kernel function, without altering the algorithm itself, is one of the reasons why kernel machines are so powerful and flexible.

Unsupervised learning is one of the main branches in the hierarchy of machine learning methods. In unsupervised learning, the goal is to extract information from data without access to ground truth, with cluster analysis [75, 149, 113, 153] and feature learning [67, 68, 76, 128] being the most prominent applications. In all these applications, the common theme is to learn something about the inherent structure of the data, either by learning relationships between individual datapoints or learning relationships between features. This is important for exploratory data analysis, where unknown insights can be discovered. Unsupervised learning is quickly becoming more important than it has ever been, due to an ever increasing amount of data being available, and manually labeling large amounts of data is a resource intensive task.

Neural networks and deep learning [51, 127] have in recent years revolutionized supervised learning tasks, and achieve great results in a multitude of tasks, including image classification [57, 87], image segmentation [8, 28, 58, 98], speech recognition [9, 60] and time–series analysis [16, 25]. Neural networks are often able to outperform the classical approaches due to their inherent ability to learn optimized feature representations for the task at hand while simultaneously learning to perform the task, instead of relying on hand–crafted features. In supervised learning, there are always ways to measure how well the network is doing, and ground truth labels that tell you if your prediction is right or wrong. Due to unsupervised learning being inherently ambiguous and the sheer amount of trainable parameters in a neural network, the loss surface of an unsupervised task can be highly complex and difficult to optimize. In spite of this, there is an emerging trend within deep learning research to develop methods for unsupervised learning tasks, with for instance clustering [52, 132, 156, 158] and learning meaningful features using autoencoders [86, 122, 126, 147]. These rely on cleverly constructed network architectures, optimization schemes and loss functions in order to train the network for unsupervised tasks.

In the intersection between kernel methods, unsupervised learning and neural networks, there are a multitude of challenges and opportunities which will be addressed by the work presented in this thesis. The next few sections outline these challenges and opportunities and the proposed approaches to address them. These challenges and opportunities all relate in some way or another to kernels and unsupervised learning, and some use kernels in conjunction with neural networks in order to solve unsupervised learning tasks. These methods all rely on – or produce – a *representation* useful for the task at hand e.g. using dimensionality reduction techniques or through transformations using a neural network.

1.1 Challenges and opportunities

The parameter sensitivity problem. When using kernel functions, like with most machine learning methods, there are *hyper–parameters* that need to be specified by the user. The choice of these hyper–parameters are usually *sensitive*, such that changing the parameter slightly could potentially degrade the performance of the algorithm by a large margin. The optimal value of these hyper–parameters are hugely dependent of the data and their inherent structures. For supervised learning, this is not a huge problem since the user has ground truth information at their disposal. This is not the case for unsupervised learning, such that finding an optimal parameter for the task at hand could potentially be borderline impossible. Due to this, a common approach/solution is to use certain rules–of–thumb in order to select the value of parameters. There are several rules–of–thumb available [76, 134, 135, 136], which might all produce different results. The value of the hyper–parameters as calculated by these rules–of–thumb are likely to be either sub–optimal or plain bad. While there are examples of methods dealing with this situation for fully observed data and temporal data [70, 71, 106, 142], the particular situation with vectorial data with missing elements, the literature is sparse.

New insights lead to new methodology There are numerous examples of machine learning methods, where framing the problem from a kernel perspective has been beneficial. For instance, the kernel SVM/SVR [18, 38], Kernel Principal Component Analysis (KPCA) [128] and kernel k-means [47] all enabled previously strictly linear methods to also be able to handle non–linear structures in the data. The latter has been shown to even be capable of minimizing cost functions of several spectral clustering algorithms without needing expensive eigenvector/eigenvalue computations [36], and enabled so–called *constrained clustering* [88]. These examples show that framing a problem from a different perspective can potentially lead to new and useful methodologies. Parts of the work presented in this thesis is based on this idea of looking at an existing problem from a kernel perspective in order to gain new insights and potentially new methodology.

Combining kernel methods with networks As stated earlier, two of the fundamental pillars in machine learning methodology are kernel methods and neural networks. Neural networks are incredibly powerful as they are so–called universal function approximators [34, 65]. In recent years, *deep learning* has emerged as the defacto standard solution for many machine learning problems, as deep neural networks are now viable to be trained successfully. This is mostly due to 1. being able to train networks efficiently on GPUs, 2. clever training methods being developed [85, 139] and 3. the vast amount of data available these days is enough to train networks with a huge number of parameters via transfer learning.

Unsupervised learning using neural networks has until recently been a largely unexplored area with huge potential [63, 122, 137, 146, 147]. Due to great flexibility in network architectures, the network can be designed for the data at hand with e.g. convolutional layers for image data or recurrent units for temporal data. The huge number of trainable parameters in modern neural networks can make unsupervised learning a challenge, due to the fact that an unrestricted network could potentially run wild and learn useless patterns in the data. In traditional kernel methods, dealing with image and time–series data has been a challenge, requiring kernels specifically designed for the data domain which might be computationally expensive with varying results. These challenges can be addressed by combining the power of neural networks with kernel methods, either by directly including kernel based loss functions for unsupervised learning in the neural network training procedure, or as a regularization for the network.

1.2 Objectives

The main interrelated objectives of this thesis are summarized as follows:

1. Provide a solution for key challenges in kernel methods for unsupervised learning, in particular getting rid of sensitive user–specified hyper–parameters when dealing with missing data.
2. Leverage kernels in order to (i) provide new insights to existing methodology, (ii) develop new methodology based on these insights and (iii)

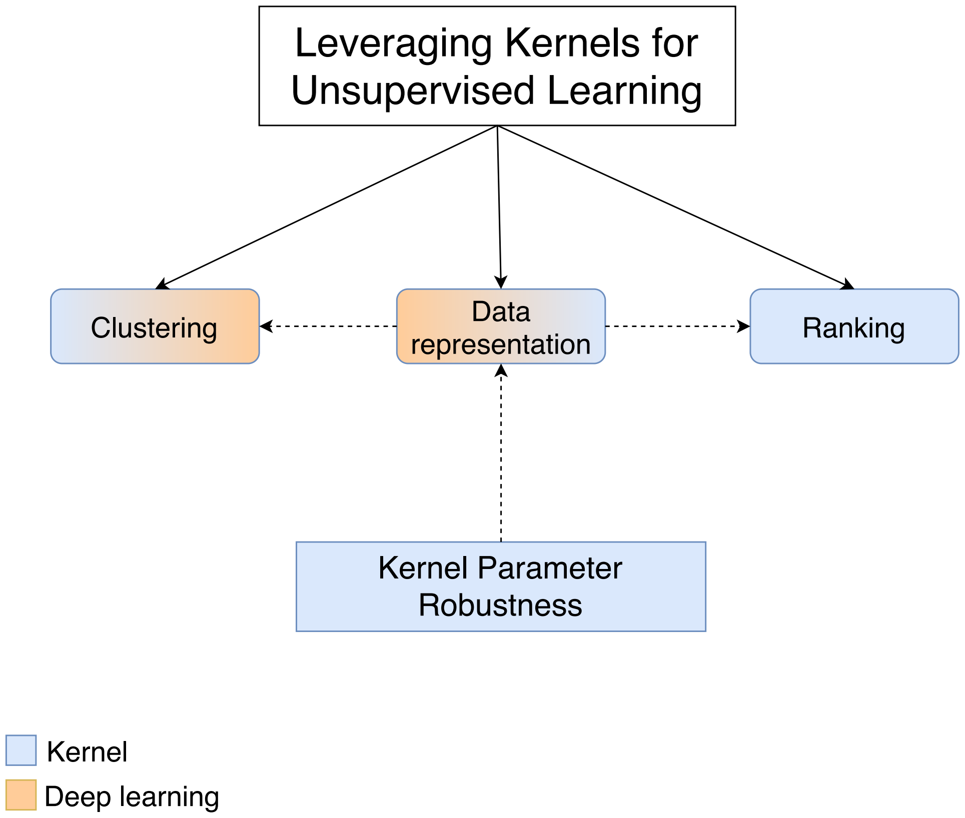

Leveraging Kernels for Unsupervised Learning
Clustering
Data representation
Ranking
Kernel Parameter Robustness
Kernel
Deep learning

Figure 1.1: Thesis overview.

improve previously non–kernelized methodologies.

3. Leverage kernels in unsupervised learning for neural networks in order to exploit the best from both worlds.

1.3 Proposed approaches

An overview of the topics discussed in this thesis is shown in Fig. 1.1.

In order to address the first challenge, we design a novel kernel function based on an *ensemble* approach, in which missing elements and parameter sensitivity are handled simultaneously. The kernel function is adaptive, in the sense that it is constructed by an ensemble of statistical models that we fit to the data. Each of these statistical models is inherently capable of dealing with missing elements in the data.

In order to address challenge 2, we propose the *Kernel Personalized PageRank* in Paper II. Here, we interpret the Personalized PageRank in terms of mathematical operations in a specific kernel space and leverage this in order to propose new methodology. The method is based on computing an embedding in the empirical kernel space using a special kernel and computing scores by projecting the elements in this space on a weighted mean vector defined by the query. The method naturally extends to out–of–sample data.

We address challenge 3 in Paper III, Paper IV and Paper V. In Paper III, we regularize the loss function of an autoencoder, in order to ensure that the feature representation in the code layer is meaningful. In particular, we guide the *inner products* in the code layer to approximate values from a precomputed kernel. In Paper IV, we propose a novel clustering network, in which information theoretic/kernel based losses are incorporated in order to train the network. This network can be trained end–to–end and does not require pretraining procedures. In Paper V, we incorporate an unsupervised dimensionality reduction layer in an echo state network [74] as a regularization before the readout layer. This dimensionality reduction procedure tends to capture the underlying dynamics of the input time series.

1.4 Potential synergies

Robust kernels are important for all kernel methods, and have potential utility for the other works. For instance, the kernel could be used in conjunction with the work in Paper II in order to rank data with missing values. It could also potentially be combined with the work in Paper III in order to enable the autoencoder to handle missing data. There is also a potential synergy between Paper II and Paper III. For instance, one could leverage the new insights in Paper II in order to train an autoencoder in such a way that the code layer possesses properties beneficial for ranking.

1.5 Brief summary of included papers

This section briefly summarizes the papers included in this thesis. The following papers are included in this thesis:

I. Sigurd Løkse, Filippo M. Bianchi, Arnt-Børre Salberg and Robert Jenssen. **”Unsupervised learning using PCKID – A Probabilistic Cluster Kernel for Incomplete Data”**, In submission.

II. Sigurd Løkse and Robert Jenssen. **”Kernel Personalized PageRank”**, In submission.

III. Michael Kampffmeyer, Sigurd Løkse, Filippo M. Bianchi, Robert Jenssen and Lorenzo Livi. **”The deep kernelized autoencoder”**, Applied Soft Computing, 2018.

IV. Michael Kampffmeyer, Sigurd Løkse, Filippo M. Bianchi, Lorenzo Livi, Anrt-Børre Salberg and Robert Jenssen. **”Deep divergence-based approach to clustering”**, Neural Networks, 2019.

V. Sigurd Løkse, Filippo M. Bianchi and Robert Jenssen. **”Training Echo State Networks with Regularization Through Dimensionality Reduction”**, Cognitive Computation, 2017.

Paper I In this paper, we develop a novel kernel function designed for vectorial data with missing elements. The kernel function is founded on ensemble methods, leading to an adaptive kernel function which is not sensitive to hyper–parameter choices, and thus is especially suited for unsupervised learning where no ground truth data is available for parameter cross–validation. We perform experiments on spectral clustering and unsupervised ranking tasks. These are compared to various methods and kernels, with promising results.

Paper II In this paper, we provide new insights on the Personalized PageRank and develop new methodology based on these insights. In particular, we show that the score vector of the Personalized PageRank can be computed using simple projections in a particular kernel space. These insights naturally lead to a low–rank embedding space in which the scores can easily be approximated both for in–sample data and previously unseen out–of–sample data. We provide error approximation bounds and use these to order eigenvectors and eigenvalues such that these error bounds are minimized. These insights are supported by experimental results, which also show that our method may even outperform the PPR.

Paper III and IV In these papers, we introduce kernels in deep learning methodology in order to leverage key properties of kernels and existing kernel–based methods for unsupervised learning. In particular, **Paper III** introduces the Deep Kernelized Autoencoder in which an auto–encoder architecture is developed that learns data representations using a kernel–alignment based regularization term. This combination of kernels and deep learning allows for learning meaningful features in the code layer, with promising results. In **Paper IV**, we develop a deep architecture for clustering, in which we adapt and incorporate kernel based/information theoretic clustering losses in the training procedure, which enforces compactness within clusters and separation between clusters. Equivalently, the loss can be interpreted as maximizing the *angle* between cluster mean vectors in kernel space. This method achieved state–of–the–art results on challenging problems at the time.

Paper V In this paper, we introduce unsupervised *dimensionality reduction* (including Kernel PCA) as a regularization to *Echo State Networks*. The dimensionality reduction layer creates a low–rank representation which extracs the important information from the large and sparse reservoir. This improves both accuracy and efficiency of the network.

1.6 Other papers

6. Jonas N. Myhre, Karl Øyvind Mikalsen, Sigurd Løkse and Robert Jenssen. **"Consensus clustering using kNN mode seeking"** In 2015 Scandinavian Conference on Image Analysis (SCIA), Springer,

2015.

7. Michael Kampffmeyer, Sigurd Løkse, Filippo M. Bianchi, Robert Jenssen and Lorenzo Livi. **"Deep kernelized autoencoders"** In 2017 Scandinavian Conference on Image Analysis (SCIA), Springer, 2017.

8. Sigurd Løkse, Filippo M. Bianchi, Arnt–Børre Salberg and Robert Jenssen. **"Spectral Clustering Using PCKID – A Probabilistic Cluster Kernel for Incomplete Data"** In 2017 Scandinavian Conference on Image Analysis (SCIA). Springer, 2017.

9. Filippo M. Bianchi, Simone Scardapane, Sigurd Løkse and Robert Jenssen. **"Bidirectional deep-readout echo state networks"**, 26th European Symposium on Artificial Neural Networks, Computational, 2018.

10. Jonas N. Myhre, Karl Øyvind Mikalsen, Sigurd Løkse and Robert Jenssen. **"Robust clustering using a kNN mode seeking ensemble"**, Pattern Recognition 76, 491-505, 2018.

11. Sigurd Løkse and Robert Jenssen. **"Ranking Using Transition Probabilities Learned From Multi–Attribute Data"** In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2851-2855. IEEE, 2018.

12. Kristoffer Wickstrøm, Sigurd Løkse, Michael Kampffmeyer, Shujian Yu, Jose Príncipe and Robert Jenssen. **"Analysis of Deep Neural Networks using Tensor Kernels and Matrix–Based Rényi's Entropy."** In Workshop on Information Theory and Machine Learning, 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), volume 60, pages 8-11, 2019.

13. Filippo M. Bianchi, Simone Scardapane, Sigurd Løkse and Robert Jenssen. **"Reservoir computing approaches for representation and classification of multivariate time series"**, In IEEE Transactions on Neural Networks and Learning Systems, 2020.

14. Van Nhan Nguyen, Sigurd Løkse, Kristoffer Wickstrøm, Michael Kampffmeyer, Davide Roverso and Robert Jenssen. **"SEN: A Novel Dissimilarity Measure for Prototypical Few–Shot Learning Net-**

works", In Workshop on Visual Learning with Limited Labels, IEEE/-CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.

15. Van Nhan Nguyen, Sigurd Løkse, Kristoffer Wickstrøm, Michael Kampffmeyer, Davide Roverso and Robert Jenssen. **"SEN: A Novel Dissimilarity Measure for Prototypical Few–Shot Learning Networks"**, In 16th European Conference on Computer Vision (ECCV), 2020.

1.7 Reading guide

The remainder of this thesis is organized into four parts, *methodology, summary of research, included papers* and an *appendix.*

The *methodology* part is organized into three chapters. *Chapter 2* provides a brief overview of the necessary theory behind kernel methods and kernels, with examples of both. *Chapter 3* introduces unsupervised learning, and presents unsupervised learning methods, which are relevant for the works presented in this thesis. The chapter is split into three sections; *dimensionality reduction/feature extraction* (relevant for Paper I, Paper II and Paper V), *clustering* (relevant for Paper I and Paper IV) and *ranking with the Personalized PageRank* (relevant for Paper II). *Chapter 4* provides a brief introduction to neural networks, various network architectures and unsupervised learning with neural networks. This is relevant for Paper III, Paper IV and Paper V.

The *summary of research* part provides a brief summary of the included papers and the author's contributions to the works as well as concluding remarks and limitations of the works. The research papers are included in the *included papers* part. The final part of this thesis is an appendix containing a brief overview of *missing data*, which is relevant for Paper I.

Part I

Methodology and context

Chapter 2

Kernel methods

This chapter presents kernel theory, which is mostly relevant for Paper I and Paper II, but also relevant for Paper III, Paper IV and Paper V.

Up until the non–linear Support Vector Machine (SVM) was invented in the early 90's [18], most machine learning methods were either not capable of dealing with non–linear data, or difficult to train with the computational power available at the time. By incorporating the mathematical framework associated with kernels, the SVM was transformed into a non–linear method leading to a *convex* optimization problem, which could be solved using existing optimization methods.

The *Kernel SVM* is an important example of a so–called *kernel method*, where kernels are utilized in order to *implicitly* map the data via a non–linear transformation to a high dimensional space, in which a linear SVM is applied. That is, instead of learning a classical linear classifier on the form

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle, \tag{2.1}$$

where $\langle \cdot, \cdot \rangle$ denotes an inner product, the Kernel SVM utilizes the so–called *kernel trick* [2] both during training and evaluation in order to implicitly map the data to a high dimensional space and learn a function on the form

$$f(\mathbf{x}) = \sum_i \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}). \tag{2.2}$$

In this case, $\kappa(\cdot, \cdot)$ is a so–called *kernel function* which often can be interpreted as some kind of similarity between the data points in its arguments. The kernel is a function with special properties which will be explained in detail shortly.

2.1 Kernel theory

Although the application of kernels in machine learning is fairly recent in historical context, the mathematical foundation of these methods were invented in the early 1900's [103]. In this section, we will briefly introduce the theory behind kernel methods.

2.1.1 Kernels

The term *kernel* has numerous definitions in mathematics. In the context of kernel methods in machine learning, a kernel is simply a function computing an inner product in a Reproducing Kernel Hilbert Space (RKHS). In particular, the function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if there exists a RKHS $\mathcal{H}$ and a function $\phi : \mathcal{X} \to \mathcal{H}$ such that $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}}$. That is, given $\mathbf{x}_i$, $\mathbf{x}_j \in \mathcal{X}$, the kernel function *implicitly* maps the data via ϕ to the possibly infinitely dimensional RKHS $\mathcal{H}$ and computes the inner product of the mapped data in $\mathcal{H}$. Computing inner products in this manner allows for transforming inner product based linear methods (e.g. SVM) to powerful non–linear methods via the so–called *kernel trick*. This is illustrated in Fig. 2.1, where a non–linear problem in input space can be solved as a linear problem in kernel space. The benefit of using a kernel is best illustrated through an example.

Example 2.1 (Polynomial kernel.)**.** *The polynomial kernel [163] is defined as*

$$\kappa(\mathbf{x}, \mathbf{y}) = (1 + \mathbf{x}^T \mathbf{y})^d. \tag{2.3}$$

Considering a low dimensional feature space with $\mathbf{x}, \mathbf{y} \in \mathbb{R}^2$ *and a quadratic*

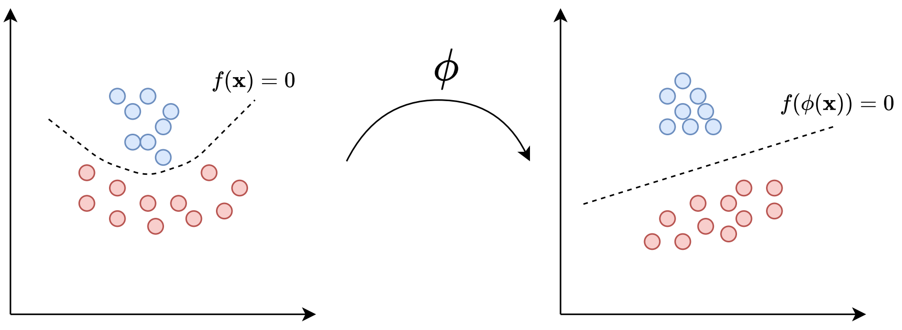


Figure 2.1: Illustration showing a non–linear problem in input space that can be solved as a linear problem in kernel space.

kernel with $d = 2$, the kernel function can be expressed as

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{y}) &= (1 + \mathbf{x}^T\mathbf{y})^2 \\ &= 1^2 + 2\mathbf{x}^T\mathbf{y} + \left(\mathbf{x}^T\mathbf{y}\right)^2 \\ &= 1 + 2\sum_{i=1}^{2} x_i y_i + \sum_{i=1}^{2} x_i y_i \sum_{j=1}^{2} x_j y_j \\ &= 1 + \sqrt{2}x_1\sqrt{2}y_1 + \sqrt{2}x_2\sqrt{2}y_2 + x_1^2y_1^2 + x_2^2y_2^2 + \sqrt{2}x_1x_2\sqrt{2}y_1y_2 \\ &= \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle_{\mathcal{H}},\end{aligned}$$

where

$$\phi(\mathbf{x}) = \begin{pmatrix} 1 & \sqrt{2}x_1 & \sqrt{2}x_2 & \sqrt{2}x_1x_2 & x_1^2 & x_2^2 \end{pmatrix}^T.$$

In this example, it was possible to find an explicit mapping ϕ such that $\kappa(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$. Although it is indeed possible in this example to first compute this mapping and then compute the inner product in kernel space, this is ill–advised as both the computational complexity and the memory complexity are increased in comparison to simply evaluating the kernel function. Furthermore, if d or the dimensionality of the data is increased, the dimensionality of the mapped data increases drastically. If only the inner product in kernel space is needed, it is therefore unnecessary to map the data first and then compute the inner product when the inner product itself is easy to compute via the kernel function.

In addition to the simplicity of computing inner products implicitly using the kernel function, the explicit mapping ϕ is in general unknown. For instance with the Gaussian kernel, the kernel feature space is *infinitely dimensional* (a function space) [133], such that an explicit vector mapping is not possible.

In some situations, it is not necessarily the nonlinear property of the kernel methods which is the most appealing. As seen in for instance [133], it is possible to construct kernel matrices for non–vectorial data like documents and DNA. This allows for kernel methods to be used on types of data for which vector based methods are not compatible.

Properties of Kernels

Definition 2.1. *Reproducing kernel. Let $\mathcal{H}$ be a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$ and let $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel with $\kappa \in \mathcal{H}$. Then κ is a reproducing kernel if*

$$\langle f, \kappa(\mathbf{x}, \cdot)\rangle_{\mathcal{H}} = f(\mathbf{x}).$$

Def. 2.1 is the so–called *reproducing property* of a kernel. The term reproducing kernel stems from the fact that the kernel is the evaluation functional of the Hilbert space. In particular, since the kernel $\kappa(\mathbf{x}, \cdot) \in \mathcal{H}$, we have $\kappa(\mathbf{x}, \mathbf{y}) = \langle \kappa(\mathbf{y}, \cdot), \kappa(\mathbf{x}, \cdot)\rangle_{\mathcal{H}}$, hence the term *reproducing* kernel. If we define the transformation function $\phi(\mathbf{x}) = \kappa(\mathbf{x}, \cdot)$, this yields $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \kappa(\mathbf{x}_i, \cdot), \kappa(\mathbf{x}_j, \cdot)\rangle_{\mathcal{H}} = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle_{\mathcal{H}}$. That is, $\kappa(\cdot, \cdot)$ computes an *inner product* in $\mathcal{H}$. The term *Reproducing Kernel Hilbert Space* (RKHS) simply refers to a Hilbert space endowed with a reproducing kernel.

At first glance, the idea of a function computing inner products in some RKHS seems somewhat abstract and arbitrary. However, it turns out that these kernel functions inhibit some particular properties that makes it possible to evaluate if a candidate function is in fact a kernel.

Definition 2.2. *Positive Semi–definiteness. Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathcal{X}$ and let $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. The function $\kappa(\cdot, \cdot)$ is a kernel iff $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \kappa(\mathbf{x}_j, \mathbf{x}_i)$ and $\forall\, \mathbf{z} \in \mathbb{R}^N \backslash \{\mathbf{0}\}$*

$$\mathbf{z}^T\mathbf{K}\mathbf{z} = \sum_{i,j=1}^{N} z_i z_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \geq 0.$$

Here $\mathbf{K} \in \mathbb{R}^{N \times N}$ is the *kernel matrix*[1], which contains all pairwise evaluations of the kernel function, i.e. $(\mathbf{K})_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$. Note that $\mathbf{K}$ is symmetric since $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \kappa(\mathbf{x}_j, \mathbf{x}_i)$.

Def. 2.2 states that any symmetric positive semi–definite function is a valid kernel, and can thus be used in kernel based machine learning methods. Showing that any kernel is positive semi–definite is trivial, since we have

$$\begin{aligned}\sum_{i,j=1}^{N} z_i z_j \kappa(\mathbf{x}_i, \mathbf{x}_j) &= \sum_{i,j=1}^{N} z_i z_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}} \\ &= \langle \sum_{i=1}^{N} z_i \phi(\mathbf{x}_i), \sum_{j=1}^{N} z_j \phi(\mathbf{x}_j) \rangle_{\mathcal{H}} \\ &= \| \sum_{i=1}^{N} z_i \phi(\mathbf{x}_i) \|_{\mathcal{H}}^2 \geq 0.\end{aligned}$$

The opposite statement (any positive semi–definite function is a kernel) is a consequence of *Moore-Aronzajn's theorem* [6], in which they show that given a positive semi–definite function, it is always possible to construct a valid RKHS endowed with that function as a reproducing kernel.

Definition 2.3. *Composite kernel. Given kernel functions $\kappa_\ell(\cdot, \cdot)$ and coefficients $\alpha_\ell \geq 0$, $\ell = 1, 2, \dots, L$, the composite kernel is defined as*

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \sum_{\ell=1}^{L} \alpha_\ell \kappa_\ell(\mathbf{x}_i, \mathbf{x}_j) \tag{2.4}$$

To show that the composite kernel is indeed a valid kernel, the only thing that needs to be checked is if it is positive semi–definite. This can be done using Eq. 2.4. In particular,

$$\sum_{i,j=1}^{N} z_i z_j \kappa(\mathbf{x}_i, \mathbf{x}_j) = \sum_{i,j=1}^{N} z_i z_j \sum_{\ell=1}^{L} \alpha_\ell \kappa_\ell(\mathbf{x}_i, \mathbf{x}_j) = \sum_{\ell=1}^{L} \alpha_\ell \sum_{i,j=1}^{N} z_i z_j \kappa_\ell(\mathbf{x}_i, \mathbf{x}_j) \geq 0,$$

since each kernel function is positive semi–definite and the coefficients are non–negative. These composite kernels are the foundation for Multiple Kernel Learning (MKL) [50], and are necessary for the work in Paper I.

[1] Also called the Gram matrix

2.1.2 Examples of kernel methods

There are many examples of kernel–based algorithms in the field of machine learning, both for supervised learning [18, 38, 120], unsupervised learning [36, 47, 76, 128] and semi–supervised learning [26, 27, 161, 166]. Many of these methods share the same end–goal: to learn a linear function as shown in (2.1). In this section, a few important methods will be described briefly.

Kernel Principal Component Analysis Kernel Principal Component Analysis (KPCA) [128] is a non–linear feature extraction method based on Principal Component Analysis (PCA). The idea is to find a vectorial representation of the data, corresponding to projections of the data on the principal components in kernel space. Since Kernel PCA is an integral part of the work presented in this thesis, details on the mathematics behind the method will be presented in Sec. 3.1.2. However, the main result is that the projection of $\phi(\mathbf{x})$ on the k'th principal component in kernel space is given by

$$\text{Proj}_{\mathbf{v}_k}(\phi(\mathbf{x})) = \frac{1}{\sqrt{\lambda_k}} \sum_i (\mathbf{a}_k)_i \kappa(\mathbf{x}_i, \mathbf{x}),$$

where λ_k is the k'th largest eigenvalue of the kernel matrix and $(\mathbf{a}_k)_i$ is element i of the corresponding eigenvector. Note that this is on the same form as Eq. 2.2, with $\alpha_i = \frac{(\mathbf{a}_k)_i}{\sqrt{\lambda_k}}$. This is not incidental, as expressions on this form shows up in many kernel methods due to the *representer theorem* [129], which in in essence states that any minimizer of an empirical risk function admits a representation on the form of Eq. 2.2.

Kernel SVM The SVM is a classification method in which the end–goal is to find the *best* decision hyperplane that optimally separates the classes. That is, the optimal decision hyperplane maximizes the *margin* between the hyperplane and each of the classes as shown in Fig. 2.2. The hyperplane can be expressed as a linear function on the form

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b,$$

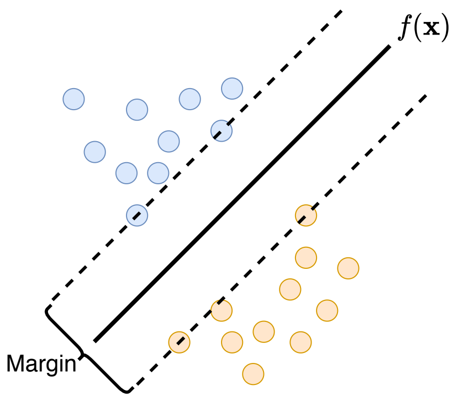


Figure 2.2: Illustration of a Support Vector Machine.

where $\mathbf{w}$ is a weight vector and b is a bias term. The most basic form of a SVM has a constraint that all data points must lie outside of the margin. This can be optimized by Lagrangian optimization, where the Lagrangian can be shown to be

$$\mathcal{L}(\mathbf{X}, \mathbf{y}, \mathbf{w}, \boldsymbol{\lambda}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i \lambda_i (y_i[\langle \mathbf{w}, \mathbf{x}_i \rangle + b] - 1),$$

where y_i is the label for data point i and λ_i is a Lagrange multiplier. The optimal weight vector for this problem can by differentiation be shown to be on the form $\mathbf{w} = \sum_i \lambda_i y_i \mathbf{x}_i$, such that

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = \langle \sum_i \lambda_i y_i \mathbf{x}_i, \mathbf{x} \rangle + b = \sum_i \lambda_i y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b.$$

The important part of this expression is that it only depends on the input data through an inner product. This means that one can utilize the kernel trick in order to kernelize this method, i.e. substitute $\langle \mathbf{x}_i, \mathbf{x} \rangle$ with $\kappa(\mathbf{x}_i, \mathbf{x}) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \rangle_{\mathcal{H}}$ [18]. This yields

$$f(\mathbf{x}) = \sum_i \lambda_i y_i \kappa(\mathbf{x}_i, \mathbf{x}) + b,$$

which is on the same form as (2.2). This is in general a non–linear function in input space. In a similar manner, the dual problem of the Lagrangian can be expressed with dependence on the input data solely through an inner product, such that the kernel trick can be used during training as well.

Other noteworthy methods There are numerous other examples of kernel methods for various applications. In the following we mention a few noteworthy methods. For supervised learning, we have for instance Fisher discriminant analysis [105], Kernel Ridge Regression [3], R-SVM [38] and Gaussian Processes [123, 80]. Unsupervised methods include kernel k–means [47, 36], various multiple kernel k–means methods [39, 95, 96], kernel self–organizing maps [69, 101] and Kernel Entropy Component Analysis [76].

2.2 Examples of kernel functions

Choosing which kernel function to use is always important, as which kernel function works best is largely dependent on both the input data and the task at hand. Undoubtedy, the most popular kernel function is the Radial Basis Function (RBF) kernel[2], which is on the form

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{1}{2\sigma^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2}, \tag{2.5}$$

where σ is a tunable hyper–parameter representing the kernel width (i.e. the scale of the data). This kernel function belongs to the family of *shift invariant* kernels, which have special properties and are notably used to connect kernel methods to both Information Theoretic Learning (ITL) and Gaussian Processes [77, 80]. Other shift invariant kernels include the Laplacian kernel and the Cauchy kernel. The aforementioned kernels are commonly used for vectorial data. If the data is non–vectorial in nature, one has to choose a kernel specifically designed for the data. There are for instance kernels designed for time–series [106], images [46], text [56, 97, 151], graphs [148] and even protein data [13, 49, 93].

A common theme for most kernel functions is that they require the user to choose a hyper–parameter, whose optimal value is highly dependent on the data and the task. This might not be a big problem if the task is classification or regression. For unsupervised tasks this could be a big problem, since except from controlled lab experiments, one cannot expect ground truth information to be available in real applications. Thus, the performance of

[2]Often referred to as the Gaussian kernel

the kernel machine cannot be evaluated easily for unsupervised tasks and choosing a good parameter value either comes down to knowledge of the data or heuristic rules–of–thumb.

Chapter 3

Unsupervised Learning

In modern times, the amount of data available is massive due to data storage being cheap and data acquisition being cheap and easy. Manual labelling of the data however is labour intensive, time consuming, and in some cases impossible due to the amount of data available. In unsupervised learning, the goal is to extract information from data without, or at least with minimal, prior knowledge. Unlike classical supervised learning tasks (classification, regression, etc.), unsupervised learning can be applied to data where there is no ground truth. Unsupervised learning has many real world applications, including medical data analysis [4, 11, 109, 155], market research [82, 90], identifying fake news [66, 160], image segmentation [8, 134, 162] and social media segmentation [54, 107].

There are many types of unsupervised learning algorithms. Examples include, but are not limited to dimensionality reduction, clustering, and ranking. Dimensionality reduction techniques are often used as a pre–processing step of the data in order to extract/generate relevant information for the task at hand (e.g. clustering). Most dimensionality reduction techniques can be considered unsupervised. Clustering is the unsupervised analogue to classification, where the goal is to find natural groups of similar objects within the data without relying on labels. Ranking techniques attempt to generate an ordered list of objects in a meaningful way, such that the most important objects are located early in the list. The most successful application of such ranking techniques is arguably with search engines on the internet [20], but it has also been successfully applied to other types of data/objects like proteins [43], genes [110], images [79] and many more [48].

This chapter contains relevant background theory for the research presented in this thesis, namely dimensionality reduction, clustering and unsupervised ranking.

3.1 Dimensionality reduction/feature extraction

This section describes methods for unsupervised dimensionality reduction and feature extraction, which are relevant for Paper I, Paper II and Paper V. For a more comprehensive guide on dimensionality reduction methods, the interested reader is directed towards the numerous surveys on this topic [23, 33, 143, 150].

In machine learning, dimensionality reduction and feature extraction is a commonly utilized pre–processing step which is applied to the data in order to generate a more useful representation than the raw data. What is a *useful* representation depends on the application (clustering, classification, ranking etc.) and the data.

3.1.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) [67] is possibly the most used dimensionality reduction method in existence. The idea behind PCA is to transform data with correlated components such that the components in the transformed data are uncorrelated. This is achieved using a linear transformation. These uncorrelated components are called the *principal components*, and are ordered such that a given principal component has larger variance than the succeeding ones.

Let $\mathbf{x} \in \mathbb{R}^d$ be a random vector with expectation $\boldsymbol{\mu}_{\mathbf{x}} = \mathbb{E}[\mathbf{x}]$ and covariance matrix $\boldsymbol{\Sigma}_{\mathbf{x}} = \mathbb{E}\left[(\mathbf{x} - \boldsymbol{\mu}_{\mathbf{x}})(\mathbf{x} - \boldsymbol{\mu}_{\mathbf{x}})^T\right]$ and define the linear transformation

$$\mathbf{y} = \mathbf{A}^T \mathbf{x}, \tag{3.1}$$

where $\mathbf{A}$ is some nonrandom transformation matrix. Given Eq. 3.1, the expectation of $\mathbf{y}$ is given by $\boldsymbol{\mu}_\mathbf{y} = \mathbb{E}\left[\mathbf{A}^T\mathbf{x}\right] = \mathbf{A}^T\boldsymbol{\mu}_\mathbf{X}$ and the covariance matrix is given by

$$\boldsymbol{\Sigma}_\mathbf{y} = \mathbb{E}\left[\left(\mathbf{A}^T\mathbf{x} - \mathbf{A}^T\boldsymbol{\mu}_\mathbf{x}\right)\left(\mathbf{A}^T\mathbf{x} - \mathbf{A}^T\boldsymbol{\mu}_\mathbf{x}\right)^T\right] = \mathbf{A}^T\boldsymbol{\Sigma}_\mathbf{x}\mathbf{A}. \tag{3.2}$$

To ensure that the components in $\mathbf{y}$ are uncorrelated, we need to find a transformation matrix $\mathbf{A}$ such that the covariance matrix $\boldsymbol{\Sigma}_\mathbf{y}$ is *diagonal*. The covariance matrix $\boldsymbol{\Sigma}_\mathbf{x}$ is symmetric and can be diagonalized by $\boldsymbol{\Sigma}_\mathbf{x} = \mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^T$, where $\mathbf{E}$ is the eigenvector matrix of $\boldsymbol{\Sigma}_\mathbf{x}$ and $\boldsymbol{\Lambda}$ is diagonal eigenvalue matrix of $\boldsymbol{\Sigma}_\mathbf{x}$. Substituting this into Eq. 3.2 yields $\boldsymbol{\Sigma}_\mathbf{y} = \mathbf{A}^T\mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^T\mathbf{A}$, from which we see that letting the transformation matrix $\mathbf{A} = (\mathbf{E}^T)^{-1}$ yields a covariance matrix $\boldsymbol{\Sigma}_\mathbf{y} = \boldsymbol{\Lambda}$, which is diagonal. Since $\boldsymbol{\Sigma}_\mathbf{x}$ is symmetric, $\mathbf{E}$ is orthogonal implying that $\mathbf{E}^{-1} = \mathbf{E}^T$, which leads to the final transformation matrix $\mathbf{A} = \mathbf{E}$, such that

$$\mathbf{y} = \mathbf{E}^T\mathbf{x} \tag{3.3}$$

This is an orthogonal transformation which, in essence, *rotates* the data until the components are uncorrelated.

Since the covariance matrix of the transformed data is given by the diagonal eigenvalue matrix, $\boldsymbol{\Lambda}$, the variance of each component of $\mathbf{y}$ is simply an eigenvalue. Furthermore, the total variance is preserved, since

$$\sum_{k=1}^{d} \sigma^2_{\mathbf{x},k} = \mathrm{Tr}(\boldsymbol{\Sigma}_\mathbf{x}) = \sum_{k=1}^{d} \lambda_k, \tag{3.4}$$

where λ_k is the k'th eigenvalue of $\boldsymbol{\Sigma}_\mathbf{x}$, which we have shown is the variance of the k'th component of $\mathbf{y}$.

This kind of transformation is mostly used for dimensionality reduction. The strategy then is to extract the principal components in which the most information is kept[1]. To reduce the dimensionality to $d' < d$ dimensions, you construct a transformation matrix $\mathbf{E}_{d'}$, $d' < d$, which consists of the d' eigenvectors of $\boldsymbol{\Sigma}_\mathbf{x}$ with the *largest corresponding eigenvalues*, such that the total variance of $\mathbf{y}$ is maximized.

[1]In this context, maximize total variance.

3.1.2 Kernel PCA

Kernel Principal Component Analysis (KPCA) [128] is a nonlinear extension of the theory of PCA. The nonlinear property of KPCA is often useful for data with structures which cannot be well represented in a linear subspace. For instance, if we are interested in extracting features as a preprocessing step for classification of nonlinearly separable data, canonical PCA will not be very helpful to aid in discriminating between classes. KPCA, however, may be able to extract features in such a way that the nonlinearly separable data becomes linearly separable.

KPCA uses the theory of *Mercer Kernels* to perform an implicit nonlinear transformation of the data and performs PCA in this (possibly unknown) kernel space. KPCA has been successfully utilized for many important applications. This includes face recognition [84, 159], de-noising [72, 141] and texture classification [83]. One should note that other nonlinear approaches to PCA have been proposed. For instance [81, 86, 92, 130].

Derivation of KPCA As seen in Eq. 3.3, the k'th principal component is given by the *projection* of the data point onto the k'th eigenvector of the covariance matrix. Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathcal{X}$ be a sample and let $\Phi : \mathcal{X} \to \mathcal{H}$ be a nonlinear transformation. Assuming the data is centered in kernel space, the covariance matrix is given by

$$\boldsymbol{\mathcal{R}} = \mathbb{E}\left[\Phi(\mathbf{x})\Phi(\mathbf{x})^T\right],$$

which can be approximated as

$$\mathbf{R} = \frac{1}{N}\sum_{i=1}^{N} \Phi(\mathbf{x}_i)\Phi(\mathbf{x}_i)^T. \tag{3.5}$$

Let $\mathbf{v}$ be an eigenvector of $\mathbf{R}$ with the corresponding eigenvalue λ. Then

$$\mathbf{R}\mathbf{v} = \frac{1}{N}\sum_{i=1}^{N} \Phi(\mathbf{x}_i)\Phi(\mathbf{x}_i)^T\mathbf{v} = \lambda\mathbf{v}, \tag{3.6}$$

such that

$$\mathbf{v} = \frac{1}{\lambda N}\sum_{i=1}^{N}\Phi(\mathbf{x}_i)\Phi(\mathbf{x}_i)^T\mathbf{v} = \sum_{i=1}^{N}\left[\frac{1}{\lambda N}\Phi(\mathbf{x}_i)^T\mathbf{v}\right]\Phi(\mathbf{x}_i) = \sum_{i=1}^{N}a(i)\Phi(\mathbf{x}_i), \tag{3.7}$$

where

$$a(i) = \frac{1}{\lambda N}\Phi(\mathbf{x}_i)^T\mathbf{v}. \tag{3.8}$$

It is clear that $\mathbf{v} \in \text{Span}\{\Phi(\mathbf{x}_i),\ i = 1, 2, \ldots, N\}$. By left multiplying Eq. 3.6 with $\Phi(\mathbf{x}_k)$ and defining the kernel matrix $\mathbf{K} \in \mathbb{R}^{N\times N}$ with elements

$$(\mathbf{K})_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j),$$

and the vector $\mathbf{a} \in \mathbb{R}^N$ with $(\mathbf{a})_i = a(i)$, we can show that

$$\mathbf{K}\mathbf{a} = N\lambda\mathbf{a}. \tag{3.9}$$

That is, $\mathbf{a}$ is an eigenvector of $\mathbf{K}$ with the corresponding eigenvalue $\lambda^* = N\lambda$. The projection of $\Phi(\mathbf{x}_k)$ onto the principal component of $\mathbf{R}$ is given by

$$\begin{aligned}\mathbf{v}^T\Phi(\mathbf{x}_k) &= \sum_{i=1}^{N}a(i)\Phi(\mathbf{x}_i)^T\Phi(\mathbf{x}_k)\\ &= \sum_{i=1}^{N}a(i)\kappa(\mathbf{x}_i, \mathbf{x}_k).\end{aligned} \tag{3.10}$$

This projection is completely defined by the kernel matrix and its eigenvectors. From the theory on PCA, we know that the eigenvectors of the covariance matrix need to be normalized. The squared norm of this eigenvector is given by

$$\|\mathbf{v}\|^2 = \mathbf{v}^T\mathbf{v} = \sum_{i,j=1}^{N}a(i)a(j)\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{a}^T\mathbf{K}\mathbf{a} = \lambda^*\mathbf{a}^T\mathbf{a} = \lambda^*\|\mathbf{a}\|^2$$

by substitution. In order to normalize the eigenvectors, we need

$$\|\mathbf{v}\| = 1 \Leftrightarrow \sqrt{\lambda^*}\|\mathbf{a}\| = 1 \Leftrightarrow \|\mathbf{a}\| = \frac{1}{\sqrt{\lambda^*}}.$$

Thus, $\mathbf{v}$ can be normalized by scaling $\mathbf{a}$. Note that $\lambda^* = N\lambda$ is the corresponding eigenvalue of $\mathbf{a}$, while λ is the corresponding eigenvalue of $\mathbf{v}$. Let

$\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_\ell$ be the ℓ dominant eigenvectors of $\mathbf{K}$, with $\lambda_1^* \geq \lambda_2^* \geq \ldots \geq \lambda_\ell^*$. Then using (3.10) and the normalization trick, the projection of $\Phi(\mathbf{x}_k)$ onto the ℓ principal components of $\mathbf{R}$ can be stored in a vector $\mathbf{z}_k \in \mathbb{R}^\ell$, where

$$(\mathbf{z}_k)_j = \mathbf{v}_j^T \Phi(\mathbf{x}_k) = \frac{1}{\sqrt{\lambda_j^*}} \sum_{i=1}^{N} a_j(i) \kappa(\mathbf{x}_i, \mathbf{x}_k). \tag{3.11}$$

Centering The derivation of KPCA assumed that the data is centered in kernel space, i.e.

$$\mathbb{E}\left[\Phi(\mathbf{x})\right] = \mathbf{0}.$$

This assumption is not necessarily satisfied in practice. However, by instead considering the covariance matrix of $\Phi(\mathbf{x}_i) - \frac{1}{N}\sum_{j=1}^{N} \Phi(\mathbf{x}_j)$, it is possible to show that the same methodology can be used by modifying the kernel matrix. In particular, instead of using the kernel matrix $\mathbf{K}$ directly, we use

$$\widetilde{\mathbf{K}} = \mathbf{K} - \mathbb{1}_N \mathbf{K} - \mathbf{K} \mathbb{1}_N + \mathbb{1}_N \mathbf{K} \mathbb{1}_N, \tag{3.12}$$

where $\mathbb{1}_N \in \mathbb{R}^{N \times N}$ is a matrix with elements $(\mathbb{1}_N)_{ij} = \frac{1}{N}$. For details, see [128].

The question now is whether to center the data or not. This question will not be answered here. The interested reader is directed towards [62] for a review on this topic. However, there are methods where centering does not make sense. For instance Kernel Entropy Component Analysis (KECA) [76].

The Empirical Kernel Space In many cases, the kernel space might be infinite dimensional (e.g. with a Gaussian kernel). As such, an exact finite representation of the kernel representation does not necessarily exist. However, since the inner products are *preserved* when using KPCA, we are able to generate a finite representation with the same inner products as in the possibly infinite dimensional kernel space. Thus, the data is said to be embedded in the *empirical kernel space*.

In-sample KPCA

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbb{R}^p$ be a sample and let $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_N \in \mathbb{R}^\ell$ be the projections of the sample onto the ℓ principal components in kernel feature space. Let $\mathbf{K}$ be a kernel matrix with the ℓ most dominant eigenvectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_\ell$ and the corresponding eigenvalues $\lambda_1^*, \lambda_2^*, \ldots, \lambda_\ell^*$. It is easy to see that if we define the eigenvector matrix $\mathbf{E} = \begin{pmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_\ell \end{pmatrix}$ and the diagonal eigenvalue matrix $\mathbf{\Lambda} = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_\ell)$, a combined calculation of all the projections is done with

$$\begin{aligned} \mathbf{Z} &= \begin{pmatrix} \mathbf{z}_1^T \\ \mathbf{z}_2^T \\ \vdots \\ \mathbf{z}_N^T \end{pmatrix} && (3.13) \\ &= \mathbf{K}^T \mathbf{E} \mathbf{\Lambda}^{-\frac{1}{2}}. && (3.14) \end{aligned}$$

Note that $\mathbf{K}$ is symmetrical, so $\mathbf{K}^T = \mathbf{K}$. Furthermore, $\mathbf{E}$ is orthogonal. This implies that $\mathbf{E}^{-1} = \mathbf{E}^T$. Thus, the kernel matrix can be diagonalized by $\mathbf{K} = \mathbf{E}\mathbf{\Lambda}\mathbf{E}^T$. Using these properties with Eq. 3.13 yields

$$\mathbf{Z} = \mathbf{K}^T \mathbf{E} \mathbf{\Lambda}^{-\frac{1}{2}} = \mathbf{K} \mathbf{E} \mathbf{\Lambda}^{-\frac{1}{2}} = \mathbf{E}\mathbf{\Lambda}\mathbf{E}^T \mathbf{E} \mathbf{\Lambda}^{-\frac{1}{2}} = \mathbf{E}\mathbf{\Lambda}^{\frac{1}{2}}. \qquad (3.15)$$

Thus, all the projections can be done in one matrix operation after constructing the eigenvector matrix and the eigenvalue matrix. We also see that the projection only depends on the eigenvalues and eigenvectors of the kernel matrix, not the kernel function itself. If KPCA is used in combination with for instance classification or regression, we have both training data and test data. In this case, we would need to use the general expression in Eq. 3.11 for the *test* data since the systems are trained based on the projection of the training data. In clustering, however, we can use the expression in Eq. 3.15.

3.2 Clustering

This section describes methods for clustering, which are relevant for Paper I and Paper IV. Clustering in general is a far too comprehensive topic to discuss in detail for this thesis. The interested reader is therefore encouraged to read some of the survey papers available on this topic [75, 149, 157]. The term *clustering* refers to the process of finding natural groups in data, without exploiting label information. In modern days when huge amounts of data are available, clustering is becoming increasingly more important as labelling data requires human interaction, which is time consuming and expensive.

3.2.1 Gaussian Mixture Models

A Gaussian Mixture Model (GMM) is used to model the Probability Density Function (PDF) of the data. The PDF of the data is assumed to be on the form

$$f(\mathbf{x}) = \sum_{k'=1}^{k} w_{k'} \mathcal{N}_d(\mathbf{x} \,|\, \boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'}), \tag{3.16}$$

where $\mathcal{N}_d(\cdot)$ is a multivariate Gaussian distribution with mean vector $\boldsymbol{\mu}_{k'} \in \mathbb{R}^d$ and covariance matrix $\boldsymbol{\Sigma}_{k'} \in \mathbb{R}^{d \times d}$. The weight w_k refers to the prior probability of a datapoint belonging to *mixture component k*. When used for clustering, each mixture component is interpreted as a cluster.

Fitting the model to the data (i.e. finding the clusters) consists of finding mean vectors, covariance matrices and prior probabilities such that the log likelihood is maximized. This optimization problem cannot be solved directly, which means that there is no analytical expression that can be used to calculate the parameters ($\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ and w_k) that maximizes the objective function. Instead, the problem is typically optimized using the Expectation Maximization (EM) algorithm [35], in which one augments the data with a latent variable and alternates between computing the expectation of the complete[2] log likelihood function given the current parameters and computing the parameters which maximizes this expectation. It has been shown

[2]Joint distribution with the data and the hidden variable.

that the objective function will increase for each iteration, such that it may converge to a local optimum [154].

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbb{R}^d$ be a sample and let $\mathbf{z}_i \in \mathbb{R}^k$, $i = 1, 2, \ldots N$ be vectors of latent variables with $(\mathbf{z}_i)_j \geq 0$ and $\langle \mathbb{1}, \mathbf{z}_i \rangle = 1$. In particular, let $(\mathbf{z}_i)_k = 1$ if $\mathbf{x}_i$ is drawn from mixture component k and $(\mathbf{z}_i)_k = 0$ otherwise.

Since w_k is the prior probability of a datapoint being drawn from mixture component k, we have $P((\mathbf{z}_i)_k = 1) = w_k$. The Probability Mass Function of $\mathbf{z}_i$ puts probability mass w_k on $\mathbf{z}_i$ if and only if $(\mathbf{z}_i)_k = 1$. Thus, the probability mass function (PMF) of $\mathbf{z}_i$ is given by

$$f(\mathbf{z}_i) = \prod_{k=1}^{K} w_k^{(\mathbf{z}_i)_k}. \tag{3.17}$$

In order to define the joint distribution of the data and the latent variables, we need the conditional distribution of a data point given the latent variable. If $(\mathbf{z}_i)_k = 1$, the data point $\mathbf{x}_i$ is drawn from a d-variate normal distribution with mean $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$. Thus, we have

$$f(\mathbf{x}_i|\mathbf{z}_i) = \prod_{k'=1}^{k} \left[\mathcal{N}_d(\mathbf{x}_i|\boldsymbol{\mu}_k', \boldsymbol{\Sigma}_k')\right]^{(\mathbf{z}_i)_k}. \tag{3.18}$$

Thus, the joint distribution of the latent variable $\mathbf{z}_i$ and the data $\mathbf{x}_i$ is

$$\begin{aligned} f(\mathbf{x}_i, \mathbf{z}_i) &= f(\mathbf{x}_i|\mathbf{z}_i) f(\mathbf{z}_i) \\ &= \prod_{k'=1}^{k} [\mathcal{N}_d(\mathbf{x}_i|\boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})]^{(\mathbf{z}_i)_{k'}} \prod_{k'=1}^{k} w_{k'}^{(\mathbf{z}_i)_{k'}} \\ &= \prod_{k'=1}^{k} [w_{k'} \mathcal{N}_d(\mathbf{x}_i|\boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})]^{(\mathbf{z}_i)_{k'}}. \end{aligned} \tag{3.19}$$

The log–likelihood of the complete data is given by

$$\begin{aligned} l^c(\boldsymbol{\Theta}|\mathbf{X},\mathbf{Z}) &= \sum_{i=1}^{N} \ln\left[f(\mathbf{x}_i,\mathbf{z}_i)\right] \\ &= \sum_{i=1}^{N} \ln\left[\prod_{k'=1}^{k} \left[w_{k'}\mathcal{N}_d(\mathbf{x}_i|\boldsymbol{\mu}_{k'},\boldsymbol{\Sigma}_{k'})\right]^{(\mathbf{z}_i)_{k'}}\right] \\ &= \sum_{i=1}^{N}\sum_{k'=1}^{k} (\mathbf{z}_i)_{k'} \ln\left[w_{k'}\mathcal{N}_d(\mathbf{x}_i|\boldsymbol{\mu}_{k'},\boldsymbol{\Sigma}_{k'})\right], \end{aligned} \tag{3.20}$$

where $\boldsymbol{\Theta}$ is the set of all the parameters, $\mathbf{X}$ is the set of all data vectors and $\mathbf{Z}$ is the set of all latent variables.

In order to apply the EM–algorithm, we need to compute $Q\left(\boldsymbol{\Theta},\widehat{\boldsymbol{\Theta}}^{(l)}\right) = \mathbb{E}_{\mathbf{Z}}\left[l^c(\boldsymbol{\Theta}|\mathbf{X},\mathbf{Z})|\mathbf{X},\widehat{\boldsymbol{\Theta}}^{(l)}\right]$. This expression is the expectation of the complete log–likelihood with respect to $\mathbf{Z}$ given the data $\mathbf{X}$ and the current estimates of the parameters, $\widehat{\boldsymbol{\Theta}}^{(l)}$. The only stochastic variable in Eq. 3.20 is $(\mathbf{z}_i)_{k'}$ since the data is assumed observed. Thus, we only need to find the expectation of these given the data and parameters. This yields

$$\begin{aligned} \mathbb{E}\left((\mathbf{z}_i)_{k'}\middle|\mathbf{x}_i,\widehat{\boldsymbol{\Theta}}^{(l)}\right) &= 0\cdot P\left((\mathbf{z}_i)_{k'}=0\middle|\mathbf{x}_i,\widehat{\boldsymbol{\Theta}}^{(l)}\right) + 1\cdot P\left((\mathbf{z}_i)_{k'}=1\middle|\mathbf{x}_i,\widehat{\boldsymbol{\Theta}}^{(l)}\right) \\ &= P\left((\mathbf{z}_i)_{k'}=1\middle|\mathbf{x}_i,\widehat{\boldsymbol{\Theta}}^{(l)}\right) \\ &= \frac{P\left((\mathbf{z}_i)_{k'}=1\middle|\widehat{\boldsymbol{\Theta}}^{(l)}\right) f\left(\mathbf{x}_i\middle|(\mathbf{z}_i)_{k'}=1,\widehat{\boldsymbol{\Theta}}^{(l)}\right)}{\sum_{j=1}^{k} P\left((\mathbf{z}_i)_j=1\middle|\widehat{\boldsymbol{\Theta}}^{(l)}\right) f\left(\mathbf{x}_i\middle|(\mathbf{z}_i)_j=1,\widehat{\boldsymbol{\Theta}}^{(l)}\right)} \\ &= \frac{\widehat{w}_{k'}^{(l)}\mathcal{N}_d(\mathbf{x}_i|\widehat{\boldsymbol{\mu}}_{k'}^{(l)},\widehat{\boldsymbol{\Sigma}}_{k'}^{(l)})}{\sum_{j=1}^{k}\widehat{w}_j^{(l)}\mathcal{N}_d(\mathbf{x}_i|\widehat{\boldsymbol{\mu}}_j^{(l)},\widehat{\boldsymbol{\Sigma}}_j^{(l)})} \\ &\equiv \widehat{\gamma}_{ik'}^{(l)}. \end{aligned} \tag{3.21}$$

This can be interpreted as the *posterior* probability of $\mathbf{x}_i$ belonging to component k. Using this result yields

$$\begin{aligned}Q\left(\boldsymbol{\Theta},\widehat{\boldsymbol{\Theta}}^{(l)}\right) &= \sum_{i=1}^{N}\sum_{k'=1}^{k}\widehat{\gamma}_{ik'}^{(l)}\ln\left[w_{k'}\mathcal{N}_d(\mathbf{x}_i|\boldsymbol{\mu}_{k'},\boldsymbol{\Sigma}_{k'})\right]\\ &= \sum_{i=1}^{N}\sum_{k'=1}^{k}\widehat{\gamma}_{ik'}^{(l)}\Big[\ln w_{k'}-\frac{d}{2}\ln(2\pi)-\frac{1}{2}\ln|\boldsymbol{\Sigma}_{k'}|\\ &\quad-\frac{1}{2}(\mathbf{x}_i-\boldsymbol{\mu}_{k'})^T\boldsymbol{\Sigma}_{k'}^{-1}(\mathbf{x}_i-\boldsymbol{\mu}_{k'})\Big].\end{aligned} \tag{3.22}$$

Each iteration of the EM–algorithm is a two–step procedure. 1. Given the current parameters, find $Q\left(\boldsymbol{\Theta},\widehat{\boldsymbol{\Theta}}^{(l)}\right)$ by computing the expectation in Eq. 3.21. 2. Update the parameters by maximizing $Q\left(\boldsymbol{\Theta},\widehat{\boldsymbol{\Theta}}^{(l)}\right)$. The new mean vectors and covariance matrices are easily found by differentiating Eq. 3.22 with respect to $\boldsymbol{\mu}_{k'}$ and $\boldsymbol{\Sigma}_{k'}$ and equating it to zero. This yields

$$\widehat{\boldsymbol{\mu}}_{k'}^{(l)} = \frac{1}{\sum_{i=1}^{N}\widehat{\gamma}_{ik'}^{(l)}}\sum_{i=1}^{N}\widehat{\gamma}_{ik'}^{(l)}\mathbf{x}_i \tag{3.23}$$

$$\widehat{\boldsymbol{\Sigma}}_{k'}^{(l)} = \frac{1}{\sum_{i=1}^{N}\widehat{\gamma}_{ik'}^{(l)}}\sum_{i=1}^{N}\widehat{\gamma}_{ik'}^{(l)}\left(\mathbf{x}_i-\widehat{\boldsymbol{\mu}}_{k'}^{(l)}\right)\left(\mathbf{x}_i-\widehat{\boldsymbol{\mu}}_{k'}^{(l)}\right)^T.a \tag{3.24}$$

Due to the nature of probabilities, the prior probabilities $w_{k'}$ need to be optimized under the constraint $\sum_{k'=1}^{k}w_{k'}=1$. The Lagrangian is given by

$$\mathcal{L}\left(\boldsymbol{\Theta},\widehat{\boldsymbol{\Theta}}^{(l)},\lambda\right) = Q\left(\boldsymbol{\Theta},\widehat{\boldsymbol{\Theta}}^{(l)}\right)-\lambda\left(\sum_{k'=1}^{k}w_{k'}-1\right).$$

Differentiating this with respect to $w_{k'}$ and equating it to zero yields

$$w_{k'} = \frac{\sum_{i=1}^{N}\widehat{\gamma}_{ik'}^{(l)}}{\lambda}. \tag{3.25}$$

Substituting this into the constraint and solving for λ yields $\lambda = N$. Thus,

$$\widehat{w}_{k'}^{(l)} = \frac{\sum_{i=1}^{N}\widehat{\gamma}_{ik}^{(l)}}{N}. \tag{3.26}$$

Note: The GMM presented in this section is the standard GMM. In Paper I, a slightly different variant is used in order to construct a kernel function for missing data.

3.2.2 k–means

The k–means clustering algorithm is arguably the most influential and popular clustering algorithm due to being efficient, easy to implement and easy to understand. Although being linear out of the box, it can be used in conjunction with dimensionality reduction techniques in order to handle non–linear data (like in Spectral Clustering), or alternatively be kernelized with Kernel k–means [47].

Assume that the number of clusters, k, in the dataset is known. Let $\boldsymbol{\theta}_\ell$, $\ell = 1, 2, \dots, k$ be the *cluster representative* for cluster $\mathcal{C}_\ell$, and let $u_{i\ell} \in \{0, 1\}$, $i = 1, 2, \dots, N$ be the cluster assignment for datapoint $\mathbf{x}_i$, where $u_{i\ell} = 1$ if $\mathbf{x}_i \in \mathcal{C}_\ell$ and zero otherwise. The cluster assignments are obtained by minimizing the cost function

$$\mathcal{L}(\mathbf{X}, \boldsymbol{\Theta}, \mathbf{U}) = \sum_{i=1}^{N} \sum_{\ell=1}^{k} u_{i\ell} \|\mathbf{x}_i - \boldsymbol{\theta}_\ell\|^2. \tag{3.27}$$

Since the cluster assignments are discrete, it is not possible to optimize this directly via differentiation. It can, however, be minimized by alternating between minimizing with respect to the cluster assignments and with respect to the cluster representatives. Minimizing Eq. 3.27 with respect to the cluster assignments can easily be done by letting $u_{i\ell} = 1$ if $\|\mathbf{x}_i - \boldsymbol{\theta}_\ell\|^2 < \|\mathbf{x}_i - \boldsymbol{\theta}_m\|^2 \, \forall \, m \neq \ell$ and zero otherwise. That is, $\mathbf{x}_i \in \mathcal{C}_\ell$ if $\boldsymbol{\theta}_\ell$ is the closest cluster representative. Minimizing Eq. 3.27 with respect to the cluster representatives can be done with differentiation. In particular,

$$\frac{\partial \mathcal{L}(\mathbf{X}, \boldsymbol{\Theta}, \mathbf{U})}{\partial \boldsymbol{\theta}_\ell} = -2 \sum_{i=1}^{N} u_{i\ell} (\mathbf{x}_i - \boldsymbol{\theta}_\ell).$$

Equating this to zero yields

$$\boldsymbol{\theta}_\ell = \frac{1}{\sum_{i=1}^{N} u_{i\ell}} \sum_{i=1}^{N} u_{i\ell} \mathbf{x}_i = \frac{1}{|\mathcal{C}_\ell|} \sum_{\mathbf{x}_i \in \mathcal{C}_\ell} \mathbf{x}_i. \tag{3.28}$$

Algorithm 1: k–means clustering

```
input  : Datapoints $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ and number of clusters $k$
output: Cluster assignments $u_{i\ell}$ and cluster representatives
        $\boldsymbol{\theta}_\ell$, $i = 1, 2, \ldots, N$, $\ell = 1, 2, \ldots, k$.
Initialize cluster representatives and cluster assignments;
repeat
    // Update cluster representatives
    for $\ell \leftarrow 1$ to $k$ do
        $\boldsymbol{\theta}_\ell \leftarrow \frac{1}{\sum_{i=1}^N u_{i\ell}} \sum_{i=1}^N u_{i\ell}\mathbf{x}_i$
    end
    // Update cluster assignments
    for $i \leftarrow 1$ to $N$ do
        for $\ell \leftarrow 1$ to $k$ do
            $d_{i\ell} \leftarrow \|\mathbf{x}_i - \boldsymbol{\theta}_\ell\|^2$
        end
        for $\ell \leftarrow 1$ to $k$ do
            $u_{i\ell} \leftarrow 1$ if $d_{i\ell} = \min(d_{i1}, d_{i2}, \ldots, d_{ik})$ else 0
        end
    end
until convergence;
```

Thus, the cluster representative is the *mean* vector of the vectors assigned to cluster $\mathcal{C}_\ell$, which is the origin of the name of the algorithm. This approach does not guarantee a global optimum, but it will converge monotonically [131]. The algorithm is summarized in Alg. 1.

k–means is closely related to GMMs. In particular, if one assumes a spherical covariance structure in a GMM with the variance $\sigma \to 0$, the posterior distributions approaches delta functions with the same assignment rule.

Initialization. Initializing cluster representatives for k–means can, depending on the complexity of the data, be very important for the end result. This has lead to the invention of numerous initialization methods, most notably k–means++ [7]. The procedure can be understood as follows:

1. Randomly select the first cluster representative from the datapoints in the dataset.

2. For all remaining datapoints, compute the distance to its closest cluster representative.

3. Randomly choose the next cluster representative with probability pro-

portional to this distance.

4. Repeat step 2. and 3. until all k cluster representatives have been selected.

This leads to initial cluster representatives evenly spread over the data. It has been shown that by initializing the cluster representatives using k–means++, the clustering procedure converges quickly and improves accuracy [7].

Kernel k–means. Although not directly relevant for the work in this thesis, it is worth mentioning that there exists a kernelized version of k–means. The idea is to map $\mathbf{x}_i \mapsto \phi(\mathbf{x}_i)$ and define cluster representatives in kernel space. The distances between $\phi(\mathbf{x}_i)$ and the cluster representative can then be computed solely using elements from the kernel matrix, such that cluster assignments are possible even without explicitly calculating a cluster representative. Although difficult to initialize properly [36], this allows for clustering data with non–linear group structures without employing costly feature extraction/generation techniques prior to the clustering procedure (e.g. KPCA). Furthermore, it has been shown that a weighted kernel k–means, initialized with special weights, is in fact minimizing the cost function of different spectral clustering techniques [36].

3.2.3 Spectral Clustering

Spectral Clustering (SC) refers to clustering methods in which one exploits properties of the *spectral decomposition* of some matrix in order to generate a representation which can be used to cluster data with intricate non–linear structures. This section describes the Normalized Cut (NCut) [116, 134], a graph based approach to SC which at the time was revolutionary for clustering applications, and is closely related to Markov Chain random walks [102]. While NCut is not used directly in the work presented in this thesis, the general approach of a non–linear spectral embedding and clustering is relevant for Paper I. For a more in-depth introduction to spectral clustering, the interested reader could take a look at [149].

Graph Cut

The goal of the clustering methods is to partition a dataset in K clusters, where the datapoints in the same cluster are *similar*, while datapoints from different clusters are *dissimilar*. A nice way of representing similarity between data points is a similarity graph.

Let $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ be a set of data points that we want to partition into K clusters and let $s_{ij} > 0$ be some similarity measure between $\mathbf{x}_i$ and $\mathbf{x}_j$. The similarity graph $G = (V, E)$ consists of vertices and edges. Each vertex v_i represents a data point $\mathbf{x}_i$. Each edge is weighted by w_{ij}. The weight, $w_{ij} = s_{ij}$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ are connected in the graph and $w_{ij} = 0$ otherwise. The weights of the graph can be stored in the *weight matrix* $\mathbf{W} = \{w_{ij}\}_{i,j=1,2,\ldots,N}$. For an undirected graph, we have $w_{ij} = w_{ji}$, so $\mathbf{W}$ is symmetric.

There are three types of similarity graphs that are commonly used:

1. *ε-neighborhood graph.* In a ε-neighborhood graph, we connect the vertices v_i and v_j with an edge if $\|\mathbf{x}_i - \mathbf{x}_j\| < \varepsilon,\ \varepsilon > 0$. That is, if the Euclidean distance between the data points are lower than some threshold, they are connected in the graph.

2. *k-nearest neighbor (knn) graph.* A k-nearest neighbor graph is constructed by considering the k-nearest neighbors of a data point $\mathbf{x}_i$. Notice that if we connect vertex v_i to v_j by an edge based solely on the k-nearest neighbors of $\mathbf{x}_i$, the graph will become *directed*. To make the graph undirected, we connect v_i to v_j if $\mathbf{x}_j$ is among the k-nearest neighbors of $\mathbf{x}_i$ *or* $\mathbf{x}_i$ is among the k-nearest neighbors of $\mathbf{x}_j$.

3. *Fully connected graph.* In a fully connected graph, all points are connected to each other. This is a useful representation if the similarity measure models local neighborhoods.

The Graph Laplacian The graph Laplacian is an operator that naturally arises in the context of many graph based learning algorithms, including the normalized cut [134]. In this section, the graph Laplacian is defined for future use and essential properties are stated.

In order to provide a definition of the graph Laplacian, it is necessary to first define the *degree matrix.*

Definition 3.1 (Degree matrix)**.** *Let* $\mathbf{W}$ *be a graph weight matrix with elements* $(\mathbf{W})_{ij} = w_{ij}$*. Then the degree matrix is defined as*

$$\mathbf{D} = \operatorname{diag}(d_{11}, d_{22}, \ldots, d_{NN}), \tag{3.29}$$

where $d_{ii} = \sum_{j=1}^{N} w_{ij}$ *is the degree of vertex* i*.*

Definition 3.2 (Graph Laplacian)**.** *Let* $\mathbf{W}$ *be a graph weight matrix with elements* $(\mathbf{W})_{ij} = w_{ij}$*, and let* $\mathbf{D}$ *be the degree matrix as defined in Def. 3.1. Then the graph Laplacian is defined as*

$$\mathbf{L} = \mathbf{D} - \mathbf{W}. \tag{3.30}$$

Property 3.1. *For an undirected graph, the quadratic form of the graph Laplacian can be expressed as*

$$\mathbf{y}^T\mathbf{L}\mathbf{y} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(y_i - y_j)^2 w_{ij}. \tag{3.31}$$

Proof.

$$\begin{aligned}
\mathbf{y}^T\mathbf{L}\mathbf{y} &= \mathbf{y}^T(\mathbf{D} - \mathbf{W})\mathbf{y} \\
&= \mathbf{y}^T\mathbf{D}\mathbf{y} - \mathbf{y}^T\mathbf{W}\mathbf{y} \\
&= \sum_{i=1}^{N} y_i^2 d_i - \sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j w_{ij} \\
&= \frac{1}{2}\left(\sum_{i=1}^{N} y_i^2 d_i - 2\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j w_{ij} + \sum_{j=1}^{N} y_j^2 d_j\right) \\
&= \frac{1}{2}\left(\sum_{i=1}^{N} y_i^2 \sum_{j=1}^{N} w_{ij} - \sum_{i=1}^{N}\sum_{j=1}^{N} 2y_i y_j w_{ij} + \sum_{j=1}^{N} y_j^2 \sum_{i=1}^{N} w_{ji}\right) \\
&= \frac{1}{2}\left(\sum_{i=1}^{N}\sum_{j=1}^{N} y_i^2 w_{ij} - \sum_{i=1}^{N}\sum_{j=1}^{N} 2y_i y_j w_{ij} + \sum_{i=1}^{N}\sum_{j=1}^{N} y_j^2 w_{ij}\right)
\end{aligned}$$

$$= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(y_i^2 - 2y_iy_j + y_j^2\right)w_{ij}$$
$$= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\left(y_i - y_j\right)^2 w_{ij}.$$

□

Property 3.2. *The graph Laplacian is positive semidefinite.*

Proof. Since $w_{ij} \geq 0$ and $(y_i - y_j)^2 \geq 0$, Eq. 3.31 yields $\mathbf{y}^T\mathbf{L}\mathbf{y} \geq 0$, such that $\mathbf{L}$ is positive semidefinite by definition. □

From Prop. 3.2, it follows that the eigenvalues of $\mathbf{L}$ are non–negative. It is also easy to show that any constant vector $c \cdot \mathbb{1}$ lies in the null space of $\mathbf{L}$, such that the smallest eigenvalue of $\mathbf{L}$ is 0 with the associated eigenvector $c \cdot \mathbb{1}$.

The Normalized Cut Clustering on similarity graphs is often done by minimizing *graph cuts*. That is, we want to find a partition of the graph such that the between-cluster weights are as low as possible while the within-cluster weights are high. That means that vertices in different clusters are dissimilar, while vertices within the same cluster are similar. We will consider the situation where we want to partition the dataset into two clusters and then state a generalized algorithm.

Definition 3.3 (Graph cut)**.** *Let A and B be complimentary subsets of the set of vertices V with $A \cup B = V$ and let $\mathbf{W}$ be the graph weight matrix with $(\mathbf{W})_{ij} = w_{ij}$. Then the graph cut is defined as*

$$\operatorname{cut}(A, B) = \sum_{i \in A}\sum_{j \in B} w_{ij}. \qquad (3.32)$$

The graph cut is the sum of the weights from every vertex in A to every vertex in B. This optimization problem makes sense intuitively, since the between–cluster similarity is low if the graph cut is low. Minimizing the graph cut will unfortunately often lead to a partition with one single vertex in one cluster and all the other vertices in the other cluster [134]. This problem is solved by including the *volume* of each cluster, which is defined in Def. 3.4.

Definition 3.4 (Set volume)**.** *Let $A \subseteq V$ and let d_{ii} be the degree of vertex i as defined in Def. 3.1. Then the volume of A is defined as*

$$\mathrm{vol}(A) = \sum_{i \in A} d_{ii}. \tag{3.33}$$

If A consists of a single vertex, the volume will be the sum of all the weights of the outgoing edges of that vertex which is small compared to a partition where A contains more vertices. This leads to the *normalized cut.*

Definition 3.5 (Normalized cut)**.** *The normalized cut is defined as*

$$\mathrm{NCut}(A, B) = \mathrm{cut}(A, B) \left[\frac{1}{\mathrm{vol}(A)} + \frac{1}{\mathrm{vol}(B)} \right]. \tag{3.34}$$

The normalized cut will be minimized if $\mathrm{cut}(A, B)$ is low while at the same time $\mathrm{vol}(A)$ and $\mathrm{vol}(B)$ are high. This ensures that the between-cluster similarity is low and that the within-cluster similarity is high.

Minimizing the Normalized Cut It can be shown that minimizing the normalized cut as defined in (3.34) is NP–complete [134]. However, it is possible to get an appoximate solution by using the graph Laplacian and a spectral relaxation. Consider the label vector $\mathbf{y} = \begin{pmatrix} y_1, & y_2, & \dots, & y_N \end{pmatrix}^T$, where

$$y_i = \begin{cases} \frac{1}{\mathrm{vol}(A)} & \text{if } \mathbf{x}_i \in A \\ -\frac{1}{\mathrm{vol}(B)} & \text{if } \mathbf{x}_i \in B \end{cases}. \tag{3.35}$$

Using this label definition and the fact that $y_i - y_j = 0$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ are assigned to the same cluster, Eq. 3.31 becomes

$$\begin{aligned} \mathbf{y}^T \mathbf{L} \mathbf{y} &= \frac{1}{2} \sum_{i \in A} \sum_{j \in B} \left(\frac{1}{\mathrm{vol}(A)} + \frac{1}{\mathrm{vol}(B)} \right)^2 w_{ij} \\ &= \frac{1}{2} \left(\frac{1}{\mathrm{vol}(A)} + \frac{1}{\mathrm{vol}(B)} \right)^2 \sum_{i \in A} \sum_{j \in B} w_{ij} \end{aligned}$$

$$= \frac{1}{2}\left(\frac{1}{\mathrm{vol}(A)} + \frac{1}{\mathrm{vol}(B)}\right)^2 \mathrm{cut}(A, B).$$

Furthermore,

$$\begin{aligned}
\mathbf{y}^T\mathbf{D}\mathbf{y} &= \sum_{i\in A} y_i^2 d_i + \sum_{j\in B} y_j^2 d_j \\
&= \sum_{i\in A} \frac{1}{\mathrm{vol}^2(A)} d_i + \sum_{j\in B} \frac{1}{\mathrm{vol}^2(B)} d_j \\
&= \frac{1}{\mathrm{vol}^2(A)} \sum_{i\in A} d_i + \frac{1}{\mathrm{vol}^2(B)} \sum_{j\in B} d_j \\
&= \frac{1}{\mathrm{vol}(A)} + \frac{1}{\mathrm{vol}(B)},
\end{aligned}$$

such that

$$\begin{aligned}
\frac{\mathbf{y}^T\mathbf{L}\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}} &= \frac{1}{2} \frac{\left(\frac{1}{\mathrm{vol}(A)} + \frac{1}{\mathrm{vol}(B)}\right)^2 \mathrm{cut}(A, B)}{\frac{1}{\mathrm{vol}(A)} + \frac{1}{\mathrm{vol}(B)}} \\
&\propto \mathrm{NCut}(A, B).
\end{aligned}$$

Thus, the normalized cut can be minimized by

$$\min_{\mathbf{y}} \frac{\mathbf{y}^T\mathbf{L}\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}}. \tag{3.36}$$

By relaxing the constraints on $\mathbf{y}$ in Eq. 3.35 such that any value is allowed, it is possible to show using Lagrangian optimization that an approximate minimization of the normalized cut can be obtained by the generalized eigenvalue problem

$$\mathbf{L}\mathbf{y} = \lambda\mathbf{D}\mathbf{y}, \tag{3.37}$$

were λ is the *second* smallest eigenvalue since the smallest is the trivial solution where $\lambda = 0$. The idea is to compute $\mathbf{y}$ as an eigenvector of $\mathbf{D}^{-1}\mathbf{L}$ and then threshold it to get the cluster labels.

General algorithms In the previous discussion, it was assumed that the data contained two clusters. The theory can be generalized to k–cluster problems by including more eigenvectors. For the general algorithms, it is necessary to define the *normalized* graph Laplacians.

Algorithm 2: Spectral clustering using $\mathbf{L}_{\text{sym}}$

```
input  : Similarity matrix $\mathbf{S}$ and the number of clusters $k$.
output: Cluster assignments.
    1. Construct a similarity graph with the weight matrix $\mathbf{W}$.
    2. Compute the symmetric normalized graph Laplacian $\mathbf{L}_{\text{sym}}$.
    3. Compute the first $k$ eigenvectors of $\mathbf{L}_{\text{sym}}$ corresponding to the $k$ smallest
       eigenvalues and form the matrix $\mathbf{U}$ containing the eigenvectors as columns.
    4. Normalize the rows of $\mathbf{U}$ to unit length.
    5. Let $\mathbf{y}_i,\ i = 1, 2, \dots, N$ be the rows of the row-normalized matrix $\mathbf{U}$. Cluster the
       data points $\mathbf{y}_i$ into $k$ clusters using the $k$-means algorithm.
```

Definition 3.6 (Normalized Laplacian)**.** *The normalized Laplacian is defined as*

$$\mathbf{L}_{rw} = \mathbf{D}^{-1}\mathbf{L}. \tag{3.38}$$

Definition 3.7 (Symmetrically normalized Laplacian)**.** *The symmetrically normalized Laplacian is defined as*

$$\mathbf{L}_{sym} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}. \tag{3.39}$$

The first normalized graph Laplacian is related to random walks, while the second one is symmetrical. It can easily be shown that if $\mathbf{y}$ is an eigenvector of $\mathbf{L}_{\text{rw}}$ with the corresponding eigenvalue λ, then $\mathbf{z}$ is an eigenvector of $\mathbf{L}_{\text{sym}}$ with the corresponding eigenvalue λ. We also have that $\mathbf{y} = \mathbf{D}^{-\frac{1}{2}}\mathbf{z}$.

Alg. 2 and Alg. 3 shows two general spectral clustering algorithms. The idea behind both of these algorithms is to generate a representation of the data using eigenvectors of a Laplacian, and cluster this representation in order to find cluster labels (typically using k–means). The main difference between the algorithms is which normalized Laplacian they use. Alg. 2 [116] uses $\mathbf{L}_{\text{sym}}$, while Alg. 3 [134] uses $\mathbf{L}_{\text{rw}}$. von Luxburg [149] advocates for using Alg. 3 rather than Alg. 2, due to its eigenvectors being interpreted directly as cluster indicators, such that multiplying these eigenvectors with $\mathbf{D}^{\frac{1}{2}}$ (Alg. 2) could lead to undesired artifacts. Furthermore, the row normalization in Alg. 2 could potentially reduce the discriminating properties of the eigenvectors.

```
Algorithm 3: Spectral clustering using $\mathbf{L}_{\text{rw}}$
    input  : Similarity matrix $\mathbf{S}$ and the number of clusters $k$.
    output: Cluster assignments.
        1. Construct a similarity graph with the weight matrix $\mathbf{W}$.
        2. Compute the normalized graph Laplacian $\mathbf{L}_{\text{rw}}$.
        3. Compute the first $k$ eigenvectors of $\mathbf{L}_{\text{rw}}$ corresponding to the $k$ smallest
           eigenvalues and form the matrix $\mathbf{U}$ containing the eigenvectors as columns.
        4. Let $\mathbf{y}_i,\ i = 1, 2, \ldots, N$ be the rows of the matrix $\mathbf{U}$. Cluster the data points $\mathbf{y}_i$
           into $k$ clusters using the $k$-means algorithm.
```

Spectral Clustering using Kernel PCA

The idea behind these SC methods is to use a non–linear feature extraction method in order to generate a beneficial representation of the data and use a simple clustering algorithm on this new representation. In the case of Alg. 3, the feature extractor is the so–called Laplacian Eigenmaps [12]. Another viable option is to use Kernel PCA as the feature extractor, which could potentially transform non–linearly separable data into linearly separable data. Thus, one could for instance use Eq. 3.13 in order to generate a new representation and cluster this representation using a simple clustering algorithm like k–means. Since KPCA exploits the eigenvectors and eigenvalues of a kernel matrix, this is a spectral method. A connection between the Laplacian Eigenmap embedding and the KPCA projections is provided in [14].

3.2.4 Ensemble Clustering

The main idea behind ensemble learning is to construct a composite model by a combination of multiple weaker models which yields an overall strong model. This approach has been used with great success in both supervised learning [19, 44, 55, 165] and unsupervised learning [5, 10, 42]. This section describes ensemble clustering [144], which is relevant for Paper I. For an up–to–date review on ensemble learning in general, the interested reader is referred to the survey paper by Dong et al. [37], and the references therein.

Even though the amount of clustering algorithms is vast, there is no clustering algorithms which will be appropriate to use for every dataset and different algorithms might produce different partitions for the same dataset. Even when applying one clustering algorithm several times to the same dataset with different initial conditions, ambiguous results might arise when the outputs of different instances are compared. Ensemble clustering (also called consensus clustering) refers to clustering methods in which multiple clustering methods, or multiple runs of the same method, are combined in some way in order to produce a partitioning of the data which is *better* in some way than each individual instance. This is often a multi–step procedure, where you

1. Cluster the data multiple times using either multiple clustering algorithms or a single clustering algorithm with different configurations (random initialization, number of clusters etc.)
2. Construct a similarity matrix based on the partitioning from the previous step. This is often referred to as the *consensus-*, *co-association-* or *ensemble* matrix.
3. Use this similarity matrix in order to produce the final partitioning.

In the following, the term co–association matrix is used to denote the similarity matrix.

There are several proposed algorithms to combine clustering results. Fred and Jain [42] suggests using the k-means clustering algorithm several times with random initial conditions. In each instance of the clustering algorithm, the number of clusters, k, is either fixed or chosen randomly in the range $k \in [k_{\min}, k_{\max}]$. The resulting partitions are then used to *vote* in a sense. A $N \times N$ co–association matrix $\mathbf{S} \in \mathbb{R}^{N \times N}$, $(\mathbf{S})_{ij} = s_{ij}$ is constructed by counting the number of times the points $\mathbf{x}_i$ and $\mathbf{x}_j$ are assigned to the same cluster in the M different partitions. Each time these data points are clustered together, it counts as one *vote*. They call this voting process *evidence accumulation*. The elements of $\mathbf{S}$ are then calculated by

$$s_{ij} = \frac{n_{ij}}{M},$$

where n_{ij} is the number of times $\mathbf{x}_i$ and $\mathbf{x}_j$ has been assigned to the same cluster. Since $n_{ij} = n_{ji}$, the co–association matrix is symmetrical.

In the ideal case, the elements of $\mathbf{S}$ take the values

$$s_{ij} = \begin{cases} 1 & \text{if } \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ belong to the same cluster} \\ 0 & \text{otherwise} \end{cases}.$$

This happens when $\mathbf{x}_i$ and $\mathbf{x}_j$ are clustered together in all of the k-means trials. If the data points are ordered according to their final cluster assignment, the co-association matrix is a *block diagonal* matrix. That is, there exists a permutation matrix $\mathbf{Q}$ such that

$$\mathbf{C} = \mathbf{Q}^T \mathbf{S} \mathbf{Q},$$

where $\mathbf{C}$ is a block diagonal matrix. In this case, detecting the cluster structure of the data is trivial. However, in real applications there will be some non-zero matrix $\boldsymbol{\varepsilon}$ in the off-diagonal elements of this matrix and thus, $\mathbf{C}$ is *block diagonally dominant.*

The co-association matrix is a type of similarity matrix. If two data points are clustered together in many of the different instances, they are considered more similar than two data points that are not clustered together as often. This similarity matrix is used in order to obtain a final partition. Fred and Jain [42] suggests a hierarchical algorithm like the single link or average link for this purpose.

In [42], an optimality criteria based on information theory is defined[3], which is used in order to select the number of clusters in the final result. The theory behind this will not be presented in this thesis, but empirical results in [42] suggests that the same number of clusters is chosen when using the longest lifetime based on the dendrogram of the hierarchical clustering algorithm.

While this is the basic idea, other similar approaches have been devised. Monti et al. [108] includes resampling techniques (like Bootstrapping [41]) to simulate a perturbation of the original dataset. Strehl and Ghosh [140] use (amongst other things) hypergraph partitioning to obtain a clustering solution. Hore et al. [64] use centroid based consensus clustering to reduce

[3]The same criterion was also proposed by Strehl and Ghosh [140].

memory complexity for large datasets. Nascimento et al. [115] use spectral clustering theory. Meyer and Wessell [104] employ a stochastic (random walk) approach. Ensemble based approaches have also been proposed for non–vectorial data, like timeseries [106].

3.2.5 Information Theoretic Clustering

This section describes Information Theoretic Quantities used for clustering, which is relevant for Paper IV. As opposed to traditional machine learning techniques, Information Theoretic Learning (ITL) uses entropy measures in order to take into consideration higher order statistics.

The entropy of a statistical distribution describes *uncertainty* of the distribution, and can be quantified using the Rényi α-order entropy given by

$$H(p) = \frac{1}{1-\alpha} \log \left(\int p(\mathbf{x})^{\alpha} \, \mathrm{d}\mathbf{x} \right), \tag{3.40}$$

where $p(\cdot)$ is a probability density function and $\alpha > 0,\ \alpha \neq 1$. In order to compute this quantity for a practical application, it is necessary to either know the true PDF of the data or to estimate the density. A common approach is to estimate the PDF using Parzen window estimation [117] with a Gaussian kernel. That is, given real data $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N \in \mathbb{R}^d$, the value of the PDF at $\mathbf{x}$ is estimated by

$$p(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} W_{\sigma^2}(\mathbf{x}_i - \mathbf{x}), \tag{3.41}$$

where $W_{\sigma^2}(\cdot) = \frac{1}{(2\pi\sigma^2)^{\frac{d}{2}}} e^{-\frac{1}{2\sigma^2}\|\cdot\|^2}$ is a Gaussian PDF with variance σ^2. If we further set $\alpha = 2$, the entropy can be computed explicitly [121]. In particular,

$$H(p|\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N) = -\log(V(p|\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)), \tag{3.42}$$

where $V(p|\cdot)$ is the *information potential*, defined as

$$V(p|\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N) = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} W_{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j). \tag{3.43}$$

In the following, the data points are omitted from the notation for simplicity, such that $H(p) = H(p|\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)$ and $V(p) = V(p|\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)$.

In the particular application of clustering, a more useful quantity is the Cauchý–Schwarz (CS) divergence [77], which measures *dissimilarity* between PDFs. The Cauchy–Schwarz divergence between PDFs p_1 and p_2 is defined as

$$D_{CS}(p_1, p_2) = -\log \left(\frac{\int p_1(\mathbf{x}) p_2(\mathbf{x}) \, \mathrm{d}\mathbf{x}}{\int p_1^2(\mathbf{x}) \, \mathrm{d}\mathbf{x} \int p_2^2(\mathbf{x}) \, \mathrm{d}\mathbf{x}} \right), \tag{3.44}$$

in which we recognize the information potentials $V(p_1)$ and $V(p_2)$ in the denominator. Replacing the PDFs with Parzen window estimates yields [77]

$$D_{CS}(p_1, p_2) = -\log \left(\frac{\sum_{\mathbf{x}_i \in \mathcal{C}_1} \sum_{\mathbf{x}_j \in \mathcal{C}_2} W_{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)}{\sum_{i,j \in \mathcal{C}_1} W_{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j) \sum_{i,j \in \mathcal{C}_2} W_{2\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)} \right). \tag{3.45}$$

The CS divergence expression in Eq. 3.45 can be used as a loss function for clustering [78]. The idea is to let p_1 and p_2 be the estimated PDF of cluster $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively. If the datapoints in $\mathcal{C}_1$ and $\mathcal{C}_2$ are far away from one another, the numerator in Eq. 3.45 will be close to zero. If each cluster is dense, the information potentials in the denominator will be large. Thus, the fraction will be small and the CS divergence will be large. For simplicity, it is common to minimize the argument of the logarithm, instead of maximizing the CS divergence.

When $W(\cdot)$ is a valid kernel function (e.g. an RBF function), the argument of the logarithm in Eq. 3.45 can be interpreted as the cosine of the angle between the cluster mean vectors in kernel space [77]. Thus, the CS divergence obtains its maximum value when the mean vectors in kernel space are orthogonal. This has inspired clustering methods based on Kernel Entropy Component Analysis [53, 76].

3.3 Ranking with the Personalized PageRank

This section describes the Personalized PageRank (PPR), which is relevant for Paper II. The PageRank algorithm [20] is arguably one of the most influential machine learning methods, in terms of its impact on how people interact with the internet. The algorithm uses link information on websites in order to *rank* the importance of web pages when performing a Google search. When it was invented in the late 90's, there were multiple search engines available. Most of them have perished since then, mostly due to Google's superior ability to find relevant web pages, which can be attributed to its ranking algorithm.

Even though the PageRank algorithm was invented for ranking web pages, it has seen a multitude of usages in other applications, including Semi–Supervised Learning [164], image ranking [79], proteins [43], genes [110] and many more. For an extensive review on the application of PageRank outside of web–page ranking, the interested reader is directed towards the review paper by Gleich [48].

Personalized PageRank Let $G = \{V, E\}$ be a graph with vertices V and edges E. In terms of web page ranking, the vertices represent web pages, while the edges represent links between the web pages. The idea behind the Personalized PageRank (PPR) is to construct a *Markov Chain* based on this graph. For each step on the Markov chain random walk, you proceed according to the graph structure with probability $1 - \alpha$ and *restart* according to the *seed* distribution with probability α, $0 < \alpha < 1$. The seed distribution is denoted by $\mathbf{s}$, with $(\mathbf{s})_i \geq 0$, $\mathbf{s}^T \mathbb{1} = 1$. Let p_{ij} be the transition probability from state i to state j and let $\mathbf{P}$ be the transition probability matrix with $(\mathbf{P})_{ij} = p_{ij}$. Then one step of the Markov chain random walk is governed by the difference equation

$$\mathbf{r}_{t+1}^T = (1 - \alpha)\mathbf{r}_t^T \mathbf{P} + \alpha \mathbf{s}^T. \tag{3.46}$$

The procedure to rank the vertices in the graph consists of computing the solution of this difference equation, which in fact is the stationary distribution of a Markov chain with transition probability matrix

$$\mathbf{P}_{\text{ppr}} = (1 - \alpha)\mathbf{P} + \alpha \mathbb{1}\mathbf{s}^T. \tag{3.47}$$

The vertices in the graph are then ranked according to this stationary distribution, such that the vertices with the highest stationary probability are ranked on top. The vertices with the highest stationary probability are the ones which a random walk will visit the most in the long run.

In the original version of the PageRank algorithm, this was used in order to rank web pages. In this situation, each web–page is represented by a vertex in the graph. The idea is to use link–information in order to define transition probabilities in the Markov chain. In particular, if web page i has an out–link to web page j, there is an edge between vertex i and vertex j in the graph. If web page i has a total of ℓ_i out–links, the transition probability p_{ij} is defined as $p_{ij} = \frac{1}{\ell_i}$. In this original formulation, the seed distribution $\mathbf{s}$ is fixed and uniform. While this is necessary in order for the stationary distribution to exist due to highly sparse transition probabilities, it is also the basis for the so–called random surfer model [91]. With probability $1 - \alpha$, a random surfer will follow links on a web page and with probability α, the random surfer will teleport to a different part of the graph. This teleportation is analogous to typing in the address of a web page. Although this model does not accurately model a real surfers behaviour, the PageRank algorithm is arguably one of the most influential algorithms in modern times, and was a large contributor to the initial success of the Google search engine.

Graph induced by a symmetric similarity matrix Various graph based machine learning algorithms need a matrix on the form $\mathbf{D}^{-1}\mathbf{K}$, where $\mathbf{K}$ is a similarity matrix with non–negative elements and $\mathbf{D}$ is the diagonal degree matrix. Matrices on this form are right stochastic due to each row summing to one, and can thus be used in conjunction with the PPR. This section is relevant for Paper II, and is based on work by Chung and Zhao [30], where they use a transition probability matrix on this form and present an explicit solution for the stationary distribution.

Consider the difference equation in Eq. 3.46. Let $\boldsymbol{\pi}(\alpha, s)$ be the solution of this equation. Then

$$\boldsymbol{\pi}(\alpha, s)^T = (1-\alpha)\boldsymbol{\pi}(\alpha, s)^T \mathbf{P} + \alpha \mathbf{s}^T, \tag{3.48}$$

where $\boldsymbol{\pi}(\alpha, s)$ is considered unknown. Solving Eq. 3.48 for $\boldsymbol{\pi}(\alpha, s)$ yields

$$\begin{aligned}
\boldsymbol{\pi}(\alpha, s)^T &= (1-\alpha)\boldsymbol{\pi}(\alpha, s)^T \mathbf{P} + \alpha \mathbf{s}^T \\
\boldsymbol{\pi}(\alpha, s)^T(\mathbf{I} - (1-\alpha)\mathbf{P}) &= \alpha \mathbf{s}^T \\
\boldsymbol{\pi}(\alpha, s)^T(\alpha \mathbf{I} + (1-\alpha)(\mathbf{I} - \mathbf{P})) &= \alpha \mathbf{s}^T \\
\frac{1-\alpha}{\alpha}\boldsymbol{\pi}(\alpha, s)^T \left(\frac{\alpha}{1-\alpha}\mathbf{I} + \mathbf{I} - \mathbf{P}\right) &= \mathbf{s}^T \\
\frac{1}{\beta}\boldsymbol{\pi}(\alpha, s)^T(\beta \mathbf{I} + \mathbf{I} - \mathbf{P}) &= \mathbf{s}^T,
\end{aligned}$$

where $\beta = \frac{\alpha}{1-\alpha}$[4]. If the stochastic matrix $\mathbf{P}$ is on the form $\mathbf{D}^{-1}\mathbf{K}$, where $\mathbf{K}$ is symmetric with non–negative entries we get

$$\begin{aligned}
\frac{1}{\beta}\boldsymbol{\pi}(\alpha, s)^T(\beta \mathbf{I} + \mathbf{I} - \mathbf{D}^{-1}\mathbf{K}) &= \mathbf{s}^T \\
\frac{1}{\beta}\boldsymbol{\pi}(\alpha, s)^T \mathbf{D}^{-\frac{1}{2}}(\beta I + \mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{K})\mathbf{D}^{-\frac{1}{2}})\mathbf{D}^{\frac{1}{2}} &= \mathbf{s}^T \\
\frac{1}{\beta}\boldsymbol{\pi}(\alpha, s)^T \mathbf{D}^{-\frac{1}{2}}(\beta \mathbf{I} + \boldsymbol{\mathcal{L}})\mathbf{D}^{\frac{1}{2}} &= \mathbf{s}^T \\
\frac{1}{\beta}\boldsymbol{\pi}(\alpha, s)^T \mathbf{D}^{-\frac{1}{2}}\boldsymbol{\mathcal{L}}_\beta \mathbf{D}^{\frac{1}{2}} &= \mathbf{s}^T, \qquad (3.49)
\end{aligned}$$

where $\boldsymbol{\mathcal{L}}_\beta = \beta \mathbf{I} + \boldsymbol{\mathcal{L}}$ is the β-normalized Laplacian, and $\boldsymbol{\mathcal{L}}$ is the symmetrically normalized Laplacian as defined in Def. 3.7. This can be solved using a Green's function. In this case, we use a discrete Green's function [29]. A discrete Green's function $\mathbf{G}_\beta$ for $\mathbf{L}_\beta$ satisfies

$$\boldsymbol{\mathcal{G}}_\beta \boldsymbol{\mathcal{L}}_\beta = \boldsymbol{\mathcal{L}}_\beta \boldsymbol{\mathcal{G}}_\beta = \mathbf{I}.$$

Substituting this in Eq. 3.49 and transposing the solution yields

$$\boldsymbol{\pi}(\alpha, s) = \beta \mathbf{D}^{\frac{1}{2}} \boldsymbol{\mathcal{G}}_\beta \mathbf{D}^{-\frac{1}{2}} \mathbf{s}. \tag{3.50}$$

[4]The observant reader will notice that Chung and Zhao [30] use $\beta = \frac{2\alpha}{1-\alpha}$. This is because they introduce a *lazy* random walk, which is not needed when the graph is connected.

Since $\boldsymbol{\mathcal{L}}_\beta$ is symmetric, it can be decomposed as

$$\boldsymbol{\mathcal{L}}_\beta = \sum_{i=1}^{N} \lambda_i \mathbf{e}_i \mathbf{e}_i^T = \mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^T,$$

where λ_i are the eigenvalues of $\boldsymbol{\mathcal{L}}_\beta$ and $\mathbf{e}_i$ are the corresponding orthonormal eigenvectors. Clearly, we have

$$\boldsymbol{\mathcal{G}}_\beta = \sum_{i=1}^{N} \frac{1}{\lambda_i} \mathbf{e}_i \mathbf{e}_i^T = \mathbf{E}\boldsymbol{\Lambda}^{-1}\mathbf{E}^T. \tag{3.51}$$

Thus, the score vector in Eq. 3.50 is found by 1. computing $\boldsymbol{\mathcal{L}}_\beta$, 2. computing the eigenvectors and eigenvalues of $\boldsymbol{\mathcal{L}}_\beta$, 3. computing $\boldsymbol{\mathcal{G}}_\beta$ according to Eq. 3.51 and 4. use this matrix in Eq. 3.50 in order to compute the score vector.

Chapter 4

Neural Networks

This chapter contains a brief overview of neural networks and neural network architectures which are relevant for Paper III, Paper IV and Paper V.

Although neural networks were invented several decades ago, higher computational power, bigger datasets and recent revolutionary advances in neural network architecture and training methods have ensured that neural networks in one form or another are considered state of the art in many applications these days.

A neural network is a layered computational model. Each layer takes an input and processes the input in some way to produce an output. The output of one layer is then fed in as the input to the next layer. This network architecture is typically trained *end–to–end*, such that all processing units in each layer are jointly optimized in order to enable the network to extract complicated patterns from the data. In the following sections, several types of processing units and network architectures will be discussed, along with an optimization procedure.

For the benefit of the reader, most of the theory found in this chapter is presented in a supervised setting, due to the basic processing units and intuition being application agnostic. The difference between supervised and unsupervised methods is typically a combination of the choice of cost function and network architecture. Some relevant unsupervised methods are

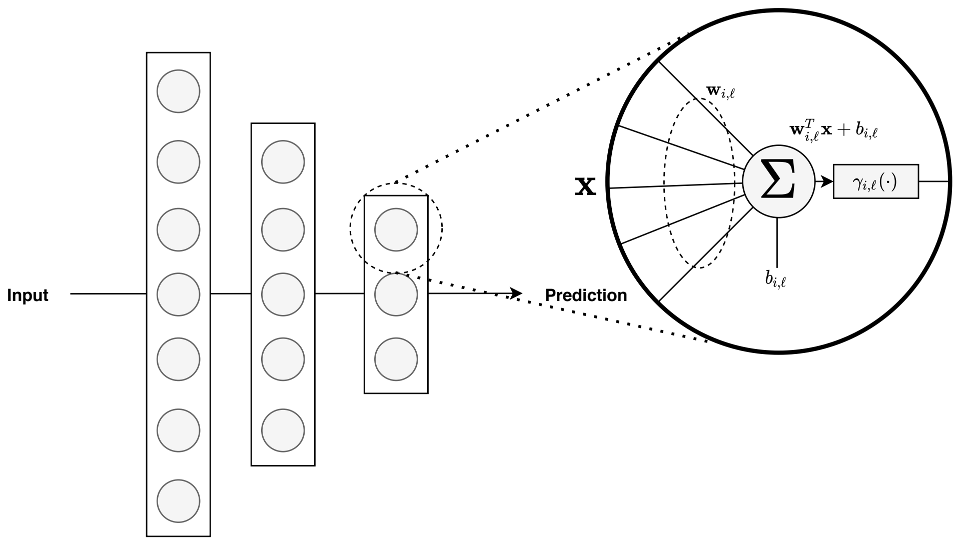


Figure 4.1: Illustration of a Multi–Layer Perceptron. An illustration of the neuron computational model is shown in the top right.

briefly described at the end.

4.1 Multi–Layer Perceptron

The Multi–Layer Perceptron (MLP) is the foundation for neural networks and deep learning. This network architecture is a hierarchical structure of fully–connected layers, from input to output, in which each layer consists of computational units referred to as *neurons*. An illustration of this network architecture is shown in Fig. 4.1. Each neuron is assigned a simple task: based on the input, produce a single number. Formally, we define this mathematical operation as follows. For neuron i in layer ℓ, we compute the output as

$$f(\mathbf{x}|\mathbf{\Theta}_{i,\ell}, \gamma_{i,\ell}) = \gamma_{i,\ell}(\mathbf{w}_{i,\ell}^T\mathbf{x} + b_{i,\ell}), \tag{4.1}$$

where $\mathbf{\Theta}_{i,\ell} = \{\mathbf{w}_{i,\ell}, b_{i,\ell}\}$ are the parameters for the neuron. The vector $\mathbf{w}_{i,\ell}$ is the neuron's associated weight vector, $b_{i,\ell}$ is a bias term and $\gamma_{i,\ell}(\cdot)$ is the *activation function.* The purpose of the activation function is to define the output of the neuron.

There are many possible choices of activation functions. The traditional choice in the hidden layers is a *sigmoid* function, which is a smooth approximation of a binary step function. This is on the form

$$\gamma_{sig}(x|a) = \frac{1}{1+e^{-ax}},$$

where a is a parameter usually equal to 1. In modern neural networks, the activation function of hidden units has to a large degree been replaced with the Rectified Linear Unit (ReLU) [114] due to beneficial properties for deep networks. The ReLU activation function is on the form

$$\gamma_{ReLU}(x) = \max(0, x),$$

and leads to sparse activations and improved gradient flow. The activation function in the output layer depends on the application. The standard choice for classification is a *softmax* function, which ensures that the output is non–negative and sums to 1. This is on the form

$$\gamma_{softmax}((\mathbf{x})_i) = \frac{e^{(\mathbf{x})_i}}{\sum_j e^{(\mathbf{x})_j}}.$$

4.2 Training the network

One of the benefits of a neural network is that the network is trained *end–to–end*, such that all neurons are updated in order to produce a beneficial output for the task at hand. This is done iteratively: (1) The network is presented with data. (2) All parameters/weights are updated layer–by–layer from the output to the input using gradient descent, such that the *loss function* is minimized. These two steps are repeated until convergence or some kind of stopping criterion is met. This is the *back–propagation* algorithm [125], which is essentially a practical implementation of the chain rule. Which loss function to use is a choice that has to be made, and depends on the application. For a classification task, a common choice is the cross–entropy loss, which is defined as

$$\mathcal{L}(\mathbf{X}, \mathbf{Y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{i,k}\log\widehat{y}_{i,k} + (1-y_{i,k})\log(1-\widehat{y}_{i,k}),$$

where $y_{i,k} \in \{0, 1\}$ is the true class membership for datapoint $\mathbf{x}_i$ to class k and $\widehat{y}_i \in [0, 1]$ is the activation of neuron k in the output layer when presented with datapoint $\mathbf{x}_i$.

Dropout Dropout [139] is a regularization technique for neural networks which is designed to reduce over–fitting problems. In modern neural network architectures, the number of trainable parameters is enormous, and the network is almost guaranteed to over–fit to the training data if sufficient regularization techniques are not in place during training.

The idea behind dropout is to randomly deactivate neurons in the network during training in order to prevent co–adaptation on the training data [61]. If neurons are randomly deactivated during training, the other neurons have to learn to accurately predict the output without depending on all neurons in the network.

Formally, given a dropout probability, p, for each iteration, each neuron is dropped with probability $1 - p$ and active with probability p. This is done in practice by sampling from a $Bernoulli(p)$ distribution, one number for each neuron, and multiplying the output of the neuron with this number during training.

This procedure can be interpreted as sampling sub–networks from a larger network during training [139]. At test time, these networks are averaged in order to produce the full network. Since this averaging procedure is computationally infeasible, it is approximated by multiplying each weight by p at test time, such that the actual output at test time is the same as the expected output during training.

Stochastic Gradient Descent During training, the back–propagation algorithm [125] is used in order to compute gradients for the weights in the network. These gradients are then used in conjunction with a gradient descent algorithm in order to update the weights. In the basic gradient descent algorithm, the gradient is calculated based on all the training data. It is then multiplied by the *learning rate*, and subtracted from the current weight. This updates the weight in such a way that the loss function is smaller than it was before the update. *Stochastic* gradient descent is similar, but instead of spending a lot of computational power and memory to calculate the exact gradient, an estimate of the gradient is calculated based on a smaller portion of the data (a *batch*). This not only speeds up the calculation and lowers the memory requirements, but also injects a certain amount of randomness into the gradient. This can help combat getting stuck in local minima.

Most implementations today use more advanced variants of this in order to implement adaptive learning rates for each weight, instead of one global learning rate. This includes AdaGrad [40] and ADAM [85].

Note: In modern deep learning software packages (e.g. Tensorflow [1] or PyTorch [118]), the practical implementation of these training procedures are largely simplified for the end user, as they can automatically differentiate and perform various optimization schemes for a user–specified network architecture and loss function.

4.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) were invented in the 90's, specifically designed to handle image data, and played a massive role in the deep learning revolution in the early 2010's. Akin to classical image processing techniques, CNNs employ *image filters*, which are applied to the images via the convolution operator. As opposed to classical image processing techniques, the filters are not hand–crafted but are instead *learned* from the data in order to best solve the task at hand. Similar to the MLP, the network has a hierarchical structure, where filters are applied one after the other in order to extract higher level features from the data. For classification, the last layers are usually fully–connected which combines the high–level features extracted by the convolutional layers in order to solve the task at hand. Although image data with 2–dimensional convolutions is the most common, CNNs can be applied to any grid–like data with local dependencies, such as temporal data (1–dimensional) and video (3–dimensional).

While MLPs require one weight parameter per neuron per input, the filters,

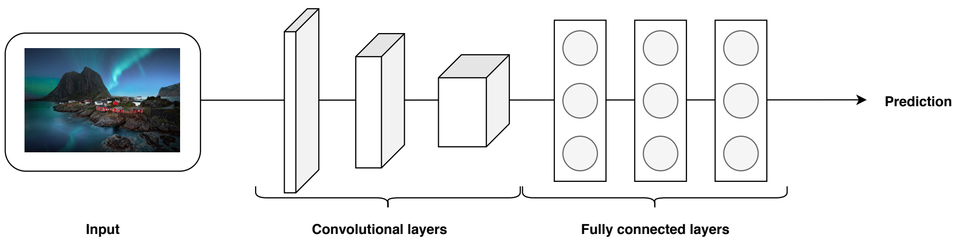


Figure 4.2: Typical CNN architecture with convolutional layers in the beginning of the network and fully connected layers at the end. Stock photo sourced from pixabay.com

and thus the number of weights, in a CNN are usually fairly small compared to the size of the input data. This is possible due to the inner workings of the convolution operator, which in practice slides the filter over the image and computes a weighted sum based on the filter weights and the pixel intensities in the underlying local image patch. Due to the relatively low number of parameters, it is possible to train larger networks without overfitting.

Since the filter weights are fixed during this whole operation, the convolution operator is *translation invariant*, i.e. it does not matter which part of the image an object resides. These two properties synergizes well with image data, since 1. if the number of weights were equal to the number of input features (number of pixels in the image), the network would have far too many weights to train and 2. detecting objects in an image should be possible regardless of the position of the object.

Fig. 4.2 shows a typical CNN architecture with convolutional layers in the beginning and fully connected layers in the end. Each convolutional layer typically contains multiple filters of size $W \times W$. The activation at location (i, j) of convolutional layer ℓ for a given filter is computed as

$$y^{\ell}_{i,j} = \gamma \left(b^{\ell} + \sum_{k=i-W/2}^{i+W/2} \sum_{m=j-W/2}^{j+W/2} w^{\ell-1}_{k,m} y^{\ell-1}_{i+k,j+m} \right), \tag{4.2}$$

where $w^{\ell-1}_{*,*}$ are filter weights and b^{ℓ} is a bias term. After the activation is computed, there is usually a *pooling* operator which computes summarizing statistics for a local region. Common pooling operators include average pooling, max–pooling and min–pooling. These pooling operators compute

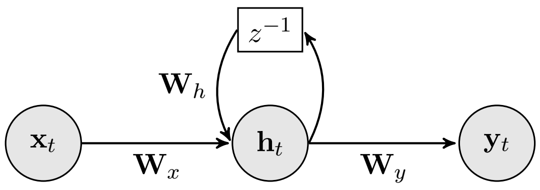


Figure 4.3: Basic RNN model with input $\mathbf{x}_t$, state vector $\mathbf{h}_t$ and output $\mathbf{y}_t$. The z^{-1} block represents a time delay. The weight matrices $\mathbf{W}_x, \mathbf{W}_h$ and $\mathbf{W}_y$ are trainable.

average–, maximum– or minimum values for each small (e.g. 2x2) area of the activation map, which ensures robustness from small local variations in the input. This also leads to reduced size of the input to the next layer.

4.4 Recurrent Neural Networks

This section describes Recurrent Neural Networks (RNN) as a basis for Echo State Networks (ESN), which is relevant for Paper V.

Recurrent Neural Networks (RNN) are used for sequential/temporal data, where there are dependencies between an observation and previous observations. The idea is simple: feed the *state* (hidden representation) of the network at time $t-1$ in with the new observation at time t in order to retain information and account for dependencies from previous time steps. Formally, we have a network with trainable parameters $\mathbf{\Theta} = \{\mathbf{W}_h, \mathbf{W}_x, b_h\}$. We define the state vector at time t as $\mathbf{h}_t = \gamma(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{W}_x\mathbf{x}_t + b_h)$, where $\mathbf{x}_t$ is the observed value at time t and γ is an activation function. This network architecture is illustrated in Fig. 4.3. The state vector can then be fed into an output layer of some sort to produce an output. This can for instance be predicting future values of $\mathbf{x}_t$ (regression).

The RNN is trained using Backpropagation Through Time (BPTT) or Truncated BPTT [111, 152]. The idea is to virtually unroll the network in time as seen in Fig. 4.4, such that it resembles a MLP with one layer for each timestep, and apply the backpropgation algorithm. This method can lead to practical issues, as gradients tend to vanish for deep networks, and the cost of updating parameters will be very high. Truncated BPTT is used in

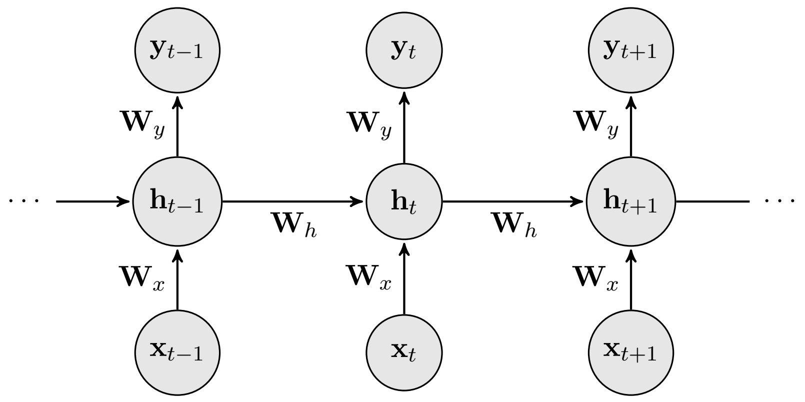


Figure 4.4: Unrolled RNN model.

practice to reduce the severity of vanishing gradients and cost, but in turn it reduces the memory of the network. With Truncated BPTT, one splits the input into several smaller segments and treats each segment as its own training case for BPTT.

4.4.1 RNN architectures

Issues with the standard RNN includes 1. it is difficult to train and 2. its memory of past events tends to be short. There are several variants of RNNs that are used in practice to alleviate these problems, for instance the Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU). These are *gated* networks, which are designed to improve the gradient flow during training such that vanishing gradients are no longer a problem. Furthermore, these gates are used to decide what to keep and what to forget about previous states in order to improve long–term memory when necessary. The GRU solves similar problems to the LSTM, but with fewer gates and thus, fewer trainable parameters.

4.4.2 Echo State Networks

In tandem with the development of the recurrent architectures described above, there has been developments of a secondary methodology for time dependent data. These are the so–called *reservoir* methods, which are de-

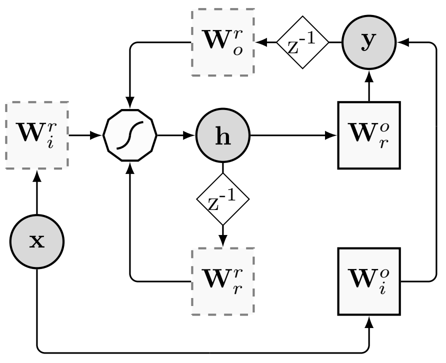


Figure 4.5: Schematic depiction of the ESN architecture. The circles represent input $\mathbf{x}$, state, $\mathbf{h}$, and output, $\mathbf{y}$, respectively. Solid squares $\mathbf{W}_r^o$ and $\mathbf{W}_i^o$, are the trainable matrices of the readout, while dashed squares, $\mathbf{W}_r^r$, $\mathbf{W}_o^r$, and $\mathbf{W}_i^r$, are randomly initialized matrices. The polygon represents the non-linear transformation performed by neurons and z^{-1} is the unit delay operator.

signed to be light–weight in terms of memory usage and computational cost, and can thus be implemented on low–end integrated devices, such as micro–controllers or FPGA. The reservoir methods typically consists of a *reservoir*, a randomized mapping from the input to a high dimensional representation, and a *readout* mechanism which generates an output of the system from this high dimensional representation. The only element that is trained in a reservoir method is the readout. The reservoir is kept fixed. Due to this, the computational cost of training the system is low.

There are numerous types of reservoir computing models, with the most famous being the Liquid–State Machine [100] and Echo State Networks [74]. This section describes Echo State Networks, which are relevant for Paper V.

An Echo State Network (ESN) consists of a large randomized and sparse recurrent layer, which does not require training and a linear readout (output) layer. The idea is that if the network state is high dimensional and random, the network state should contain enough information to perform well on the task at hand as long as the output is trained. A visual representation of an ESN is reported in Fig. 4.5.

The state–update and output of an ESN can be described by the following equations:

$$\mathbf{h}_t = \gamma(\mathbf{W}_r^r \mathbf{h}_{t-1} + \mathbf{W}_i^r \mathbf{x}_t + \mathbf{W}_o^r \mathbf{y}_{t-1} + \xi), \tag{4.3}$$

$$\mathbf{y}_t = \mathbf{W}_i^o \mathbf{x}_t + \mathbf{W}_r^o \mathbf{h}_t, \tag{4.4}$$

where ξ is a small i.i.d. noise term. The reservoir consists of N_r neurons with the activation function $\gamma(\cdot)$, typically implemented as a hyperbolic tangent function. At time t, the network is driven by the input signal $\mathbf{x}_t \in \mathbb{R}^{N_i}$ and it generates the output $\mathbf{y}_t \in \mathbb{R}^{N_o}$, where N_i and N_o is the input and output dimensionality, respectively. The vector $\mathbf{h}_t \in \mathbb{R}^{N_r}$ denotes the ESN internal state. The weight matrices $\mathbf{W}_r^r \in \mathbb{R}^{N_r \times N_r}$ (reservoir connections), $\mathbf{W}_i^r \in \mathbb{R}^{N_r \times N_i}$ (input-to-reservoir), and $\mathbf{W}_o^r \in \mathbb{R}^{N_r \times N_o}$ (output-to-reservoir feedback) contain real values in the $[-1, 1]$ interval, sampled from a uniform distribution.

The reservoir $\mathbf{W}_r^r$ must satisfy the so-called *echo state property* [99]. This guarantees that the effect of a given input on the state of the reservoir vanishes in a finite number of time intervals. A widely used rule-of-thumb suggests to rescale the matrix $\mathbf{W}_r^r$ to have $\rho(\mathbf{W}_r^r) < 1$, where $\rho(\cdot)$ denotes the spectral radius. Several theoretically-founded approaches have been proposed in the literature to properly tune ρ in an ESN driven by a specific input [15, 17, 145].

The weight matrices $\mathbf{W}_i^o$ and $\mathbf{W}_r^o$ are optimized for the task at hand. To determine them, let us consider the training sequence of T_{tr} desired input-outputs pairs given by:

$$(\mathbf{x}_1, y_1) \ldots, (\mathbf{x}_{T_{tr}}, y_{T_{tr}}), \tag{4.5}$$

In the initial phase of training, called *state harvesting*, the inputs are fed to the reservoir in accordance with Eq. 4.3, producing a sequence of internal states $\mathbf{h}_1, \ldots, \mathbf{h}_{T_{tr}}$. Since, by definition, the outputs of the ESN are not available for feedback, according to the *teacher forcing* procedure, the desired output is used instead in Eq. 4.4. States are stacked in a matrix

$\mathbf{S} \in \mathbb{R}^{T_{tr} \times (N_i + N_r)}$ and the desired outputs in a vector $\mathbf{y}^* \in \mathbb{R}^T_{tr}$:

$$\mathbf{S} = \begin{bmatrix} \mathbf{x}_1^T, \ \mathbf{h}_1^T \\ \vdots \\ \mathbf{x}_{T_{tr}}^T, \ \mathbf{h}_{T_{tr}}^T \end{bmatrix}, \mathbf{y}^* = \begin{bmatrix} y_1^* \\ \vdots \\ y_{T_{tr}}^* \end{bmatrix}.$$

The initial D rows $\mathbf{S}$ and $\mathbf{y}^*$ are the washout elements that should be discarded, since they refer to a transient phase in the ESN's behavior.

Since the gain of the sigmoid non-linearity in the neurons is largest around the origin, three coefficients ω_i, ω_o and ω_f are used to scale the input, desired output and feedback signals respectively. In this way, it is possible to control the amount of non-linearity introduced by the processing units.

Training the readout consists of solving a convex optimization problem, for which several closed form solution have been proposed in the literature. The standard procedure to train the readout, originally proposed in [73], consists in a regularized least-square regression, which can be easily computed through the Moore-Penrose pseudo-inverse. To learn a non–linear readout, one can for instance consider using Support Vector Regression (SVR). This is a supervised learning model that can efficiently perform a non-linear separation of data using a kernel function to map the inputs into high-dimensional feature spaces, where they are linearly separable [22].

4.5 Unsupervised Learning

4.5.1 Autoencoders

An autoencoder is a special neural network architecture, which is essentially attempting to learn to produce a compressed representation which contains enough information from the input such that it can be used to approximately reproduce the input. Autoencoders are commonly used in order to determine initial weights prior to training a network [59]. Although many different autoencoder networks exist (e.g. [122, 126, 146, 147]), this section describes the basic autoencoder in its purest form.

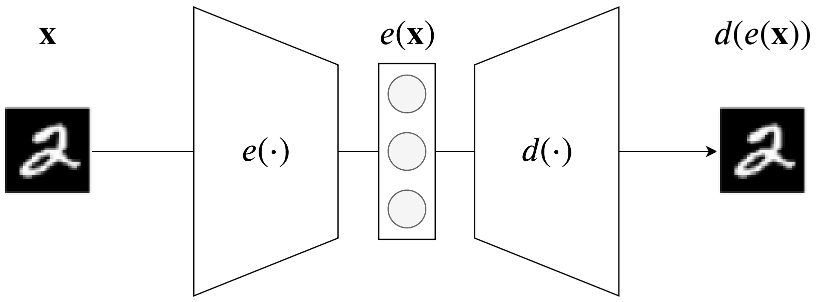


Figure 4.6: Typical autoencoder architecture, with $e(\cdot)$ and $d(\cdot)$ representing the encoder– and decoder network, respectively.

The autoencoder consists of two important network components: an encoder network $e(\cdot)$ and a decoder network $d(\cdot)$. The encoder network produces the code representation of the datapoint, while the decoder network attempts to reproduce the input based on the code representation. Both encoder and decoder networks can consist of fully–connected, convolutional or recurrent layers, depending on the data. A typical auto–encoder is illustrated in Fig. 4.6. As seen in the figure, the dimensionality of the code layer $e(\mathbf{x})$ is typically lower than the rest of the network in order to ensure that the encoder– and decoder network does not simply learn identity mappings [147]. This is not necessary if some sort of regularization scheme is implemented during training (see e.g. [147]).

In its most basic form, the autoencoder is trained to minimize

$$\mathcal{L}(\mathbf{X}) = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}_i - d(e(\mathbf{x}_i))\|^2,$$

the mean squared error of the input and the networks' reproduced version. Since the loss function does not need ground truth labels, this is in fact an unsupervised learning task. This has been shown to be a useful loss function with the goal of reproducing the input, as it is maximizing the lower bound of the mutual information between the input and the code [147], leading to a code representation with as much information from the input as possible.

4.5.2 Clustering

Clustering using neural networks is an emerging trend with great potential, which has until recently been considered a too difficult task due to the extremely high dimensionality of the parameter space and the lack of labels. This task is usually solved through clever use of network architecture and training procedures. This section briefly describes some notable recent methods for clustering using neural networks.

Deep embedded clustering (DEC) [156] is one of the earliest successful clustering methods developed for neural networks. DEC uses a MLP in order to produce an embedding in which the clustering is performed. Similar to k–means, the clustering procedure defines cluster centroids $\boldsymbol{\mu}_{k'}$ in this embedding space. In order to assign a datapoint $\mathbf{x}_i$ to a cluster, Xie et al. [156] define a similarity between its embedding $\mathbf{z}_i$ and a cluster centroid $\boldsymbol{\mu}_{k'}$. This is defined using a Student's t–distribution with 1 degree of freedom. The optimization procedure minimizes the Kullback–Leibler (KL) divergence between this similarity and a target distribution designed to promote pure clusters, strengthen high confidence assignments and prevent collapsing clusters. This procedure is designed to jointly optimize the cluster centroids and the weights in the network, which are initialized using a stacked denoising autoencoder [147]. This idea has since been extended with Improved Deep Embedded Clustering (IDEC) [52] and Deep Clustering Networks (DCN) [158]. The former keeps the decoder after pretraining the network and jointly optimizes the clustering loss and the autoencoder loss. The authors argue that this improves clustering performance by ensuring that the features contain relevant information. The latter introduces a hard cluster assignment to IDEC, which leads to a two step procedure due to cluster assignments being non–differentiable.

Other noteworthy methods include SpectralNet [132] which optimizes a spectral clustering objective, and Generative Adverserial Network (GAN) based clustering methods [112, 138].

Part II

Summary of research and concluding remarks

Chapter 5

Summary of Research

Paper I – Unsupervised Learning using PCKID – A Probabilistic Cluster Kernel for Incomplete Data

In this paper, we develop a novel kernel function designed for vectorial data with missing elements. The kernel function is founded on ensemble methods, leading to an adaptive kernel function which is not sensitive to hyper–parameter choices, and thus is especially suited for unsupervised learning where no ground truth data is available for parameter cross–validation. The kernel function is learned by fitting many Gaussian Mixture Models to the data on different scales (number of mixture components) and with a multitude of initial conditions. The models are inherently capable of dealing with missing data. These mixture models are *under–trained*, in order to provide diversity in the overal ensemble. The pairwise kernel evaluations are constructed as the average of the inner products between posterior distributions, evaluated at two datapoints.

The experiments are performed on spectral clustering and unsupervised ranking, and compared to various methods and kernels, with promising results.

Contributions by the author

- The idea was conceived by me and my co–authors and further developed by me.
- I made all implementations and ran all experiments.
- I wrote the first draft of the manuscript.

Paper II – Kernel Personalized PageRank

In this paper, we investigate the mathematics behind the Personalized PageRank and provide new insights that lead to new methodology. In particular, we show that the score vector of the Personalized PageRank, when the Markov chain transition probability matrix is generated from a kernel similarity function, can be computed using simple projections in the kernel space associated with the *Green's* function of a variant of the graph Laplacian. This naturally leads to a low–rank spectral embedding space in which the scores can easily be approximated both for in–sample data and previously unseen out–of–sample data. This out–of–sample approximation is calculated by a similar expression to out–of–sample Kernel PCA, using a normalized version of the kernel function used to generate the transition probabilities. This result is important, as calculating out–of–sample values of the Green's function is not viable. We provide error approximation bounds and use these to order eigenvectors and eigenvalues such that these error bounds are minimized.

Experiments show quantitatively that our eigenvector sorting method outperforms the naive sorting method(by eigenvalues), which is often used for spectral methods. Furthermore, we show experimentally that using our low–rank embedding can lead to *increased* performance compared to the original PPR.

Contributions by the author

- The idea was conceived by me, and further developed with input from the co–author.
- The mathematical expressions and connections were derived by me.
- I made all implementations and ran all experiments.
- I wrote the first draft of the manuscript.

Paper III – The deep kernelized autoencoder

We develop an auto–encoder that utilizes a kernel–alignment regularization term between inner products (a linear kernel) in code space and a kernel function computed in input space. This leads to inner products in code space that are encouraged to mimic the similarity in input space, which is computed by a kernel function. This regularization term is used during the optimization in order to learn *meaningful* representations in code space.

We demonstrate the usefulness of our code space representation, both qualitatively and quantitatively, by using the this representation for classification, denoising and manifold traversal.

Contributions by the author

- The idea was conceived in joint collaboration with all authors, where some of the main components were my ideas.
- I helped develop the mathematical groundwork.
- I helped debugging the implementation.
- Parts of the first draft of the manuscript was written by me.

Paper IV – Deep divergence–based approach to clustering

We develop a deep clustering network, in which we adapt and incorporate kernel based and information theoretic clustering losses in the training procedure that enforces compactness within clusters and separation between clusters. In particular, we adapt the Cauchy–Schwartz divergence to support non–discrete cluster assignments in order to incorporate it during training of the network. This loss can be interpreted as maximizing the *angle* between cluster mean vectors in kernel space [77]. The loss is combined with geometric regularization constraints in order to avoid degenerate structures in the result. The training procedure is adapted to support batch–wise training, such that the method scales well to large datasets.

We demonstrate our network on a multitude of clustering tasks, with competitive performances compared to other state–of–the art methods at the time.

Contributions by the author

- The idea was conceived in joint collaboration with all authors, where some of the main components were my ideas.
- I helped develop the mathematical groundwork.
- I ran experiments for baseline methods.
- I helped debugging the implementation.
- Parts of the first draft of the manuscript was written by me.

Paper V – Training Echo State Networks with Regularization Through Dimensionality Reduction

In this paper, we introduce unsupervised *dimensionality reduction* (including Kernel PCA) as a regularization to *Echo State Networks*. The dimensionality reduction layer creates a low–rank representation of the large and sparse reservoir, which is fed into the readout layer. It is shown experimentally that this low–rank representation is able to capture the underlying dynamic properties of the time series, and as such it is well suited for the purpose of prediction.

Contributions by the author

- The initial idea was conceived by the second author and further developed by all authors.
- I made the implementation[1].
- I ran experiments, in collaboration with the second author.
- I wrote the first draft of the manuscript in collaboration with the second author.

[1]https://github.com/siloekse/PythonESN

Chapter 6

Concluding Remarks

In this thesis, we have leveraged kernel methods for unsupervised learning. We have proposed a kernel for unsupervised learning with missing data, which inherently learns to adapt to the structure of the data without the need of critical user defined hyper–parameters. Furthermore, we have investigated the Personalized PageRank from a different perspective and shown that this can be interpreted as a kernel method. This lead to a intuitive unsupervised approximate method based on Kernel PCA with an included out–of–sample extension. Theoretical bounds for the approximation error are provided.

In an effort to improve unsupervised deep learning, we have explored injecting kernels during the learning process. In particular, we have used kernels to

1. Align inner products in the code space of an autoencoder to a kernel computed on the input data. This is done in order to learn *meaningful* representations in code space.

2. Develop a deep clustering network in which kernel based information theoretic losses are used during the training procedure in order to enforce cluster friendly structures in the final hidden representation. Clustering is performed simultaneously with training the network, and learning is performed end–to–end without the need of pre–training.

Finally, we have introduced unsupervised dimensionality reduction (including Kernel PCA) as a regularization to Echo State Networks. This is used in order to create a low–rank representation of the reservoir. We have observed that this low–rank representation is able to capture the underlying dynamic properties of the time series.

6.1 Limitations and future work

Paper I Although ensemble based methods work well for many problems, we acknowledge that computational cost might be an issue. In the particular case of the work in Paper I, we try to alleviate this by parallelizing the training phase and by simplifying the models using diagonal covariance matrices. However, pairwise kernel evaluations on a large ensemble of models is still expensive, especially on a large dataset. Runtime complexity could potentially be reduced by incorporating multiple kernel learning in the framework, in order to reduce the number of model evaluations needed at runtime.

Another limitation of the kernel is that although it does not perform badly in a supervised setting, it does not necessarily *outperform* other kernels. Due to the presence of labels, tuning parameters in other kernels for the task at hand is certainly possible, and usually makes other kernels perform on par or better than our kernel. It might be possible to somehow incorporate labels in the kernel evaluation step in order to increase its viability in supervised (or semi–supervised) tasks.

Paper II As with all eigenvector based approaches, performing Kernel PCA to generate our ranking embedding is computationally expensive. This is somewhat alleviated by the fact that we are able to define an out–of–sample extension in our embedding. However, we have yet to find a way to efficiently *update* the basis when a new datapoint enters the dataset without recomputing the eigenvectors of the updated kernel matrix.

Paper III Regularizing an autoencoder in order to control the code space representation is an idea that has been explored in various ways. We decided to do this in such a way that the inner products in code space are imitating a kernel function computed over the input data using a normalized Frobenious distance, or equivalently, a *kernel alignment* [32]. We have recently been made aware of a discussion regarding kernel alignment versus *centered* kernel alignment, where the latter is related to the Hilbert Schmidt Independence Criterion [21]. This has previously been used in Multiple Kernel Learning with good results [31], although it was outperformed by other measures in some tasks when utilized for an *informativeness* measure. Nevertheless, it would be an idea to test out.

In the paper, we performed the experiments using a Probabilistic Cluster Kernel (PCK). The choice of kernel is of course important for the end result, and could be chosen based on the application. For instance, you could potentially use the results from Paper II in order to train an autoencoder in which the code layer is suitable for ranking.

Paper IV Loss surfaces within neural networks are by nature highly irregular with many local minima or saddle points, especially in deep learning where the number of parameters is enormous. This can make optimization difficult. Furthermore, the kernel/information theoretic based loss function used in this work is known to have many local minima, especially when the kernel width parameter is small. Setting this parameter to a sensible value is inherently difficult, especially on a problem that changes during training as with the case of neural networks, and mini batch training. A similar loss function is used in [78]. They solve this problem by employing an *annealing* strategy of the kernel width parameter, in order to start off with a smooth loss surface to approximately find the global minimum, and get a fine–tuned result as the kernel width decreases. Due to the evolving nature of the input data when dealing with deep neural networks and mini–batch training, we opted for using rules–of–thumb for this parameter. Instead of annealing the parameter, we added several regularization terms in order to guide the network to approach a sensible solution. The result of this was a network whose input was expected to have uniformly sized clusters (in terms of the number of datapoints). Furthermore, we experienced that once every now and then, the network would converge to a bad solution. This is the reason why we recommend to train several networks and choose the solution with the lowest loss function value (or perform a voting process). It is unknown to me at this point in time if there are better ways of dealing with this, but it is something that would be interesting to investigate in the future.

Paper V The biggest limitation to this work might be the lack of theoretical guarantees for our findings. We do get experimental results on synthetic data with known dynamical properties that indicate the dimensionality reduction layer is able to capture these properties. If this could be shown theoretically, it would have made the paper much stronger.

Part III

Included papers

Paper I

Unsupervised Learning using PCKID – A Probabilistic Cluster Kernel for Incomplete Data

Sigurd Løkse, Filippo M. Bianchi, Arnt-Børre Salberg and Robert Jenssen.

In submission

Unsupervised Learning using *PCKID* – A Probabilistic Cluster Kernel for Incomplete Data

Sigurd Løkse **Filippo Maria Bianchi** **Arnt–Børre Salberg** **Robert Jenssen**

Abstract

In this paper, we propose *PCKID*, a novel and robust kernel function for unsupervised learning, specifically designed to handle incomplete data. By combining posterior distributions of Gaussian Mixture Models for incomplete data on different scales, we are able to learn a kernel for incomplete data that does not depend on any critical hyperparameters, unlike the commonly used RBF kernel. The kernel is applied to two unsupervised learning tasks, namely spectral clustering and ranking of multi–attribute data with missing elements.

1 Introduction

Unsupervised learning concerns the discovery of patterns in data without access to prior knowledge. Due to an ever-increasing amount of data, and the labeling of data (acquiring ground truth) being resource-intensive, unsupervised learning is of utmost importance in the field of machine learning. Some of the most prominent examples of unsupervised machine learning approaches are e.g. clustering [5, 17] and ranking [4, 14, 39].

Analyzing incomplete datasets (with missing features) poses big challenges within data analysis in general. These are situations which are oft-encountered in real applications. For instance, an entry in the dataset may not be recorded if a sensor is failing or a field in a questionnaire is left unanswered. This problem is especially challenging with unsupervised learning problems where ground truth information cannot be leveraged, as opposed to supervised learning [7, 27, 32]. This is reflected in the numerous approaches for trying to deal with incomplete data in the unsupervised setting, both using statistical models [13, 21, 25], in addition to non–statistical based clustering methods [8, 22]. In general, a common approach is to apply imputation techniques [10] to estimate the missing values for then to proceed with the analysis on the imputed, complete, data set. None of these approaches come without challenges since the best choice of imputation technique is often very dependent on the data, and moreover difficult to evaluate. Furthermore, these methods are specifically designed for a single task (e.g. clustering), and thus not suitable for unsupervised learning in general.

Kernel methods have dominated machine learning research [20] by providing a framework in which one can transform suitable linear methods into non–linear methods via the *kernel trick* [1]. This means that as long as one has access to a suitable kernel function, one can use this kernel for several different tasks. Supervised learning with kernels for missing data has shown promising results [3, 7, 23, 32, 41]. However, in the particular case of kernel methods for unsupervised learning with missing data, very little work has been done [26, 34, 38]. The challenge in this setting is that one often have a user–specified hyper–parameter (or several such parameters) in the kernel function that needs to be tuned in order to get good results. The obtained results are often very sensitive to the choice of this parameter. Due to not having access to ground truth in an unsupervised setting, finding a good value for this parameter is very difficult, especially when combined with e.g. imputation for missing data.

In this paper, we propose as a new approach to integrate in a synergistic manner recent advances in spectral clustering and kernel methods with existing probabilistic methods for dealing with incomplete data. In particular, we further develop the Probabilistic Cluster Kernel (PCK) framework [6, 15, 16, 35], which combines posterior distributions of Gaussian Mixture Models (GMMs) on different scales to learn a robust kernel function, capturing similarities on both a global and a local scale. This kernel function is robust with regards to hyper–parameter choices, since instead of assuming some particular structure in the data, the ensemble of GMMs adapt to the data manifold. We hypothesize that by integrating GMMs specifically designed to handle incomplete data [21] into the PCK framework for unsupervised learning, we will be able to analyze incomplete data sets in a more robust manner compared to existing approaches. The proposed approach for building this new type of kernel matrix to be used for unsupervised learning in our framework, is denoted the *Probabilistic Cluster Kernel for Incomplete Data (PCKID).*

We demonstrate the *PCKID* on numerous clustering and ranking tasks. *PCKID* outperforms the baseline methods in most clustering tasks, over a range of missingness proportions. *PCKID* outperforms the baseline kernels in the ranking tasks, in the sense that the score vector remains relatively unchanged when injecting missingness in the data.

2 Background theory

2.1 Missing data mechanisms

Let $\mathbf{x} = \{x_j\}$ denote a data vector and let $\mathbf{x}^{\mathrm{o}}$ and $\mathbf{x}^{\mathrm{m}}$ denote respectively the observed and the missing features of $\mathbf{x}$. Define $\mathbf{r} = \{r_j\}$, where $r_j = 1$ if $x_j \in \mathbf{x}^{\mathrm{m}}$ and zero otherwise to be the *missing indicator* for $\mathbf{x}$. In order to train a model that accounts for values in the dataset that are not observed, one has to rely on assumptions that describe how missing data occurs. In this section, we describe the three main missing data mechanisms that characterize the structure of $\mathbf{r}$ [32].

2.1.1 Missing completely at random (MCAR)

Features are said to be *missing completely at random* (MCAR) if the features are missing independently from both the observed values $\mathbf{x}^{\mathrm{o}}$ and the missing values $\mathbf{x}^{\mathrm{m}}$. That is,

$$P(\mathbf{r}|\mathbf{x}) = P(\mathbf{r}).$$

This is the missingness assumption on the data that leads to the simplest analysis. However, this assumption is rarely satisfied in practice.

2.1.2 Missing at random (MAR)

If the *features* are missing independently of their *values*, the features are said to be *missing at random* (MAR). Then the missingness of the features are only dependent of the *observed* values, such that

$$P(\mathbf{r}|\mathbf{x}) = P(\mathbf{r}|\mathbf{x}^{\mathrm{o}}).$$

This missing data mechanism is often assumed when working with missing data, since many real world missing data are generated by this mechanism. For instance, a blood test of a patient might be missing if it is only taken given some other test (observed value) exceeds a certain value.

2.1.3 Not missing at random (NMAR)

If the missingness of a feature is dependent on their values, it is said to be not missing at random (NMAR), that is

$$P(\mathbf{r}|\mathbf{x}) = P\left(\mathbf{r}|\mathbf{x}^{\mathrm{m}}\right).$$

For instance, NMAR occurs when a sensor measurement is discarded because it goes beyond the maximum value that the sensor can handle.

2.2 Gaussian Mixture Models for Incomplete Data

In this section, we briefly summarize how to implement Gaussian Mixture Models (GMM) when the data have missing features. This model will be exploited as the foundation for *PCKID* to learn a robust kernel function. For details, we refer the interested reader to [21].

A GMM is used in order to model the probability density function (PDF) for a given dataset. In a GMM, a data point $\mathbf{x}_i$ is assumed to be sampled from a multivariate Gaussian distribution $\mathcal{N}_k(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ with probability π_k and $k \in [1, K]$, where K corresponds to the number of mixture components. Accordingly, the PDF of the data is modeled by a *mixture* of Gaussians, such that

$$f(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{1}$$

The maximum likelihood estimates for the parameters in this model can be approximated through the Expectation Maximization (EM) algorithm.

When the data have missing features, we assume that the elements in a data vector $\mathbf{x}_i$ can be partitioned into two components; one observed part $\mathbf{x}_i^{\mathrm{o}}$ and one missing part $\mathbf{x}_i^{\mathrm{m}}$ as explained in Sec. 2.1. Then, one can construct a binary matrix $\mathbf{O}_i$ by removing the rows from the identity matrix corresponding to the missing elements $\mathbf{x}_i^{\mathrm{m}}$, such that $\mathbf{x}_i^{\mathrm{o}} = \mathbf{O}_i\mathbf{x}_i$. Given the mean vector $\boldsymbol{\mu}_k$ and the covariance matrix $\boldsymbol{\Sigma}_k$ for mixture component k, the mean and covariance matrix for the *observed* part of missingness pattern i is given by

$$\begin{aligned}\boldsymbol{\mu}_{k,i}^{\mathrm{o}} &= \mathbf{O}_i\boldsymbol{\mu}_k\\ \boldsymbol{\Sigma}_{k,i}^{\mathrm{o}} &= \mathbf{O}_i\boldsymbol{\Sigma}_k\mathbf{O}_i^T.\end{aligned}$$

By defining

$$\mathbf{S}_{k,i}^{\mathrm{o}} = \mathbf{O}_i^T {\boldsymbol{\Sigma}_{k,i}^{\mathrm{o}}}^{-1}\mathbf{O}_i,$$

one can show that, under the MAR assumption, the EM procedure outlined in Alg. 1 will find the parameters that maximizes the likelihood function [21].

Note that even though the notation in this paper allows for a unique missingness pattern for each data point $\mathbf{x}_i$, one missingness pattern is usually shared between

Algorithm 1 EM algorithm for incomplete data GMM

```
1: Initialize $\hat{\boldsymbol{\mu}}_k^{(0)}$, $\hat{\boldsymbol{\Sigma}}_k^{(0)}$, $\hat{\pi}_k^{(0)}$ and $\hat{\gamma}_{i,k}^{(0)}$ for $k \in [1, K]$ and $i \in [1, N]$.
2: while not converged do
3:     E-Step: Compute
           $$\hat{\gamma}_{k,i}^{(\ell)} = \frac{\hat{\pi}_k^{(\ell)} \mathcal{N}\left(\mathbf{x}_i^{\mathrm{o}} | \hat{\boldsymbol{\mu}}_{k,i}^{\mathrm{o}(\ell)}, \hat{\boldsymbol{\Sigma}}_{k,i}^{\mathrm{o}(\ell)}\right)}{\sum_{j=1}^{K} \hat{\pi}_j^{(\ell)} \mathcal{N}\left(\mathbf{x}_i^{\mathrm{o}} | \hat{\boldsymbol{\mu}}_{j,i}^{\mathrm{o}(\ell)}, \hat{\boldsymbol{\Sigma}}_{j,i}^{\mathrm{o}(\ell)}\right)},$$
           $$\hat{\mathbf{Y}}_{k,i}^{(\ell)} = \hat{\boldsymbol{\mu}}_k^{(\ell)} + \hat{\boldsymbol{\Sigma}}_k^{(\ell)} \hat{\mathbf{S}}_{k,i}^{\mathrm{o}(\ell)} \left(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k^{(\ell)}\right)$$
4:     M-Step: Compute the next model parameters, given by
           $$\hat{\pi}_k^{(\ell+1)} = \frac{1}{N} \sum_{i=1}^{N} \hat{\gamma}_{k,i}^{(\ell)},$$
           $$\hat{\boldsymbol{\mu}}_k^{(\ell+1)} = \frac{\sum_{i=1}^{N} \hat{\gamma}_{k,i}^{(\ell)} \hat{\mathbf{Y}}_{k,i}^{(\ell)}}{\sum_{i=1}^{N} \hat{\gamma}_{k,i}^{(\ell)}},$$
           $$\hat{\boldsymbol{\Sigma}}_k^{(\ell+1)} = \frac{\sum_{i=1}^{N} \hat{\boldsymbol{\Omega}}_{k,i}^{(\ell)}}{\sum_{i=1}^{N} \hat{\gamma}_{k,i}^{(\ell)}},$$
       where
           $$\boldsymbol{\Omega}_{k,i}^{(\ell)} = \hat{\gamma}_{k,i}^{(\ell)} \Bigg( \left(\hat{\mathbf{Y}}_{k,i}^{(\ell)} - \hat{\boldsymbol{\mu}}_k^{(\ell+1)}\right) \left(\hat{\mathbf{Y}}_{k,i}^{(\ell)} - \hat{\boldsymbol{\mu}}_k^{(\ell+1)}\right)^T + \left(\mathbf{I} - \hat{\boldsymbol{\Sigma}}_k^{(\ell)} \hat{\mathbf{S}}_{k,i}^{o(\ell)}\right) \hat{\boldsymbol{\Sigma}}_k^{(\ell)} \Bigg).$$
5: end while
```

several data points. Thus, to improve efficiency when implementing Alg. 1, one should sort the data points by missingness pattern such that parameters that are common across data points are calculated only once [21].

2.2.1 Diagonal covariance structure assumption.

In some cases, when the dimensionality of the data is large compared to the number of data points, in combination with many missingness patterns, one could consider assuming a diagonal covariance structure for the GMM for computational efficiency and numerical stability when inverting covariance matrices. This will of course limit the models to not encode correlations between dimensions, but for some tasks it provides a good approximation that is a viable compromise when limited computational resources are available. In this case, covariance matrices are encoded in d-dimensional vectors, which simplify the operations in Alg. 1.

Let $\widehat{\boldsymbol{\sigma}}_k$ be the vector of variances for mixture component k and let $\widehat{\mathbf{s}}_{k,i}$ be a vector with elements $\widehat{s}_{k,i}(\ell) = \frac{1}{\sigma_k(\ell)}$ if element ℓ of data point $\mathbf{x}_i$ is observed and $\widehat{s}_{k,i}(\ell) = 0$ otherwise. Define

$$\widehat{\mathbf{y}}_{k,i} = \widehat{\boldsymbol{\mu}}_k + \widehat{\boldsymbol{\sigma}}_k \odot \widehat{\mathbf{s}}_{k,i} \odot (\mathbf{x}_i - \widehat{\boldsymbol{\mu}}_k), \tag{2}$$

and

$$\boldsymbol{\omega}_{k,i} = \hat{\gamma}_{k,i}\big((\widehat{\mathbf{y}}_{k,i} - \widehat{\boldsymbol{\mu}}_k) \odot (\widehat{\mathbf{y}}_{k,i} - \widehat{\boldsymbol{\mu}}_k) + \widehat{\boldsymbol{\sigma}}_k - \widehat{\boldsymbol{\sigma}}_k \odot \widehat{\mathbf{s}}_{k,i} \odot \widehat{\boldsymbol{\sigma}}_k\big) \tag{3}$$

where $\odot$ denotes the Hadamard (element wise) product. Estimating the parameters with an assumption of diagonal covariance structure is then a matter of exchanging $\widehat{\mathbf{Y}}_{k,i}$ and $\boldsymbol{\Omega}_{k,i}$ with $\widehat{\mathbf{y}}_{k,i}$ and $\boldsymbol{\omega}_{k,i}$ respectively in Alg. 1.

3 *PCKID* – A Probabilistic Cluster Kernel for Incomplete Data

In this paper, we propose a novel procedure to construct a kernel matrix based on models learned from data with missing features, which we refer to as *PCKID*. In particular, we propose to learn similarities between data points in an unsupervised fashion by fitting GMMs to the data with different initial conditions $q \in [1, Q]$ and a range of mixture components, $g \in [2, G]$ and combine the results using the posterior probabilities for the data points. That is, we define the kernel function as

$$\kappa_{PCKID}(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{Z} \sum_{q=1}^{Q} \sum_{g=2}^{G} \boldsymbol{\gamma}_i^T(q,g) \boldsymbol{\gamma}_j(q,g), \tag{4}$$

where $\boldsymbol{\gamma}_i(q,g)$ is the posterior distribution for data point $\mathbf{x}_i$ under the model with initial condition q and g mixture components and Z is a normalizing constant. By using Alg. 1 to train the models, we are able to learn the kernel function from the inherent structures of the data, even when dealing with missing features. In this work, we use this kernel for unsupervised learning.

The *PCKID* is able to capture similarities on both a local and a global scale. When a GMM is trained with many mixture components, each mixture component covers a small, *local* region in feature space. On the contrary, when the GMM is trained with a small number of mixture components, each mixture component covers a large, *global* region in feature space. Thus, if two data points are similar under models on all scales, they are likely to be similar, and will have a large value in the *PCKID*. This procedure of fitting models to the data on different scales, ensures robustness with respect to parameters, as long as Q and G are set sufficiently large. Thus, we are able to construct a kernel function that is robust with regards to parameter choice. This way of constructing a robust kernel is similar to the methodology used in ensemble clustering and recent work in spectral clustering [16]. However, such recent methods are not able to explicitly handle missing data.

According to the ensemble learning methodology [28, 40], we build a powerful learner by combining multiple weak learners. Therefore, one does not need to run the EM algorithm until convergence, but instead perform just a few iterations[1]. This also has the positive side-effect of encouraging diversity, providing efficiency and preventing overfitting. To further enforce diversity, it is beneficial to use sub-sampling techniques to train different models on different subsets of the data and evaluate the complete kernel on the full dataset.

3.1 Out of sample

An important property of the GMMs applied in this framework is that the posterior distributions can be calculated for out of sample data. From Alg. 1 it is apparent that the posterior distribution only dependends on the mixture weights, observed values and the mean vector and covariance matrix of the observed dimensions. Thus, by storing the mixture weights, the full mean vector and covariance matrix, one can calculate the posterior distributions for out of sample data. One can even calculate these distributions for data with other missingness patterns than the training data, for instance with fully observed training data and missing values in the test data, which is a realistic scenario.

3.2 Initialization

For each mixture model that is trained, one needs to provide an initialization. Since we are fitting large models to data that in practice does not necessarily fit these models, the initialization needs to be reasonable in order to avoid computational issues when inverting covariance matrices. An initialization procedure that has been validated empirically for the *PCKID* is

1. Use mean imputation to impute missing values.

2. Run *one* k-means iteration (initialized using kmeans++ [2]) in order to get initial cluster assignments and means.

3. Calculate the empirical covariance matrix from each cluster and calculate empirical prior probabilities for the mixture model based on the cluster assignments.

Data with imputed values is only used to be able to calculate initial means and covariances. *When training the model, data without imputed values is used.*

[1]For instance, 10 iterations.

4 Experiments

4.1 Experiment methodology

We demonstrate the performance of *PCKID* on two widely important unsupervised tasks, namely spectral clustering [12, 29, 30, 36, 37] and ranking [9]. Our *PCKID*–based procedures, that we use to solve these two tasks in this paper, are briefly summarized below.

The standard spectral clustering procedure employs a two–stage approach with a non–linear feature generation using the eigenvectors and eigenvalues (or *spectrum*) of a (kernel) matrix for then to cluster this representation using e.g. the k–means clustering algorithm. In this work, we exploit the fact that *PCKID* is a valid kernel and use Kernel PCA [33] in order to generate the features. It can be shown that the k-dimensional representation using Kernel PCA can be computed by

$$\mathbf{Z} = \mathbf{E}_k \mathbf{\Lambda}_k^{\frac{1}{2}}, \tag{5}$$

where $\mathbf{\Lambda}_k \in \mathbb{R}^{k\times k}$ is a diagonal matrix containing the k largest eigenvalues of the kernel matrix along the main diagonal and $\mathbf{E}_k \in \mathbb{R}^{N\times k}$ is a matrix where the k corresponding eigenvectors are placed as columns.

The ranking procedure is based on a particular formulation of the Personalized PageRank [9]. In particular, the transition probability matrix is generated from a similarity matrix (e.g. a kernel matrix) as $\mathbf{P} = \mathbf{D}^{-1}\mathbf{K}_{PCKID},$. where element (i,j) of $\mathbf{K}_{PCKID}$, $(\mathbf{K}_{\text{PCKID}})_{ij} = \kappa_{PCKID}(\mathbf{x}_i, \mathbf{x}_j)$ and $\mathbf{D}$ is the diagonal degree matrix with elements $(\mathbf{D})_{ii} = d_{ii} = \sum_{j=1}^{N} \kappa_{PCKID}(\mathbf{x}_i, \mathbf{x}_j)$. Let α, $0 < \alpha < 1$ be the *restart probability* and let $\mathbf{L}_\beta = \beta\mathbf{I} + \mathbf{L}$ be the β–normalized Laplacian where $\beta = \frac{\alpha}{1-\alpha}$ and $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{K}_{PCKID}\mathbf{D}^{-\frac{1}{2}}$ is the symmetrically normalized Laplacian matrix. Furthermore, let $\mathbf{G}_\beta$ be the Green's function (inverse) of $\mathbf{L}_\beta$, where $\mathbf{G}_\beta = \sum_{i=1}^{N} \frac{1}{\lambda_i}\mathbf{e}_i\mathbf{e}_i^T$ and $\mathbf{e}_i$ and λ_i are the eigenvectors and eigenvalues of $\mathbf{L}_\beta$, respectively. In this particular case, the score vector can be computed by

$$\boldsymbol{\pi}(\alpha, \mathbf{s}) = \beta\mathbf{D}^{\frac{1}{2}}\mathbf{G}_\beta\mathbf{D}^{-\frac{1}{2}}\mathbf{s},$$

where $\mathbf{s}$ is the seed vector with $s_i \geq 0, \sum_{i=1}^{N} s_i = 1$. The seed distribution determines where the Markov chain restarts and is used in order to specify the query.

4.2 Experiment setup

4.2.1 *PCKID* parameters

In order to illustrate that *PCKID* does not need any parameter tuning, the parameters are set to $Q = G = 30$ for all experiments. It has been noted in related works [16, 26] that this type of kernel is robust w.r.t. parameter choice as long as the parameters are large enough. In order to increase diversity, each model in the ensemble is trained on a random subset of 50% of the whole dataset. Once the models are trained, the kernel is evaluated on the full dataset. Each GMM is trained for 10 iterations with a diagonal covariance structure assumption.

4.3 Clustering experiments

4.3.1 Baseline methods

For the baseline methods, missing data is handled with imputation techniques, in particular, i) zero imputation, ii) mean imputation iii) median imputation and (iv) most frequent value imputation. To produce a clustering result, each of these imputation techniques is coupled with i) k-means on the data and ii) spectral clustering using an RBF kernel. Furthermore, we compare our approach with k-POD [8], a recent state-of-the-art clustering method which inherently accounts for incomplete data.

Since no hyperparameters need to be tuned in *PCKID*, the kernel width σ of the RBF is calculated with a rule of thumb. In particular, σ is set to 20% of the median pairwise distances in the dataset, as suggested in [18]. This is in agreement with unsupervised approaches, where labels are not known and cross validation on hyperparameters is not possible.

4.3.2 Performance metric

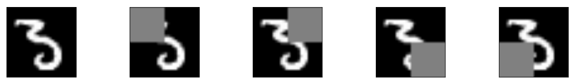
In order to assess the performance of *PCKID*, its supervised clustering accuracy is compared with all baseline models. The supervised clustering accuracy is computed by

$$ACC = \max_{\mathcal{M}} \frac{\sum_{i=1}^{n} \delta\{y_i = \mathcal{M}(\widehat{y}_i)\}}{n}, \qquad (6)$$

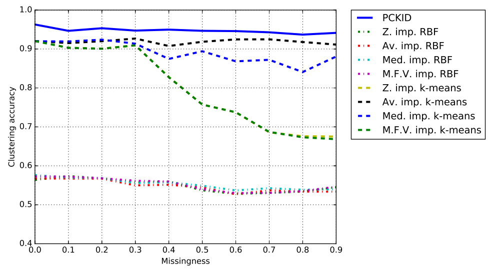
where y_i is the ground truth label, $\widehat{y}_i$ is the cluster label assigned to data point i and $\mathcal{M}(\cdot)$ is the label mapping function that maximizes the matching of the labels. This is computed using the Hungarian algorithm [19]. Note that these labels are only used for calculating the metrics, and not available for during training.

4.3.3 Clustering setup

Spectral clustering with k clusters is performed by mapping the data to a k dimensional empirical kernel space and clustering them with k-means as described in Sec. 4.1. For all methods, k-means is run 100 times. The final clustering is chosen by evaluating the k-means cost function and choosing the partitioning with the lowest cost. The number of clusters, k, is assumed known.



(a)



(b)

Figure 1: (a): Example of missingness patterns. Gray pixels are considered missing. (b): Mean clustering accuracy as a function of the percentage of images with missing values.

4.3.4 MNIST 5 vs. 6

In this experiment, subsets containing 1000 of the MNIST 5 and 6 images are clustered. The subsets consist of a balanced sample, i.e. there are approximately the same amount of images from each class. The images are unraveled to 784 dimensional vectors, which are used as the input to the algorithms. Missing data is generated by randomly choosing a proportion p_{m} of the images and removing one of the four quadrants in the image according to the MAR mechanism. These missingness patterns are illustrated in Fig. 1a. In each test, we consider different probabilities of having missing quadrants, i.e. $p_{\mathrm{m}} \in \{0.0, 0.1, 0.2, \ldots, 0.9\}$, Each method is run 30 times for each value of p_{m}, with a unique random subset of the data for each run. Since there are dimensions in the dataset where there is no variation between images, they are removed before training the GMMs. These are dimensions without information, and causes problems when inverting the covariance matrices. The number of dimensions with variance varies across the runs, since the subset from the dataset and the missingness is randomly sampled for each run. The number of dimensions with variance is approximately 500.

Fig. 1b shows a plot of the mean clustering accuracy over the 30 runs versus the missingness proportion p_{m}. The proposed method outperforms the baseline methods for all p_{m}. Although the clustering accuracy decreases slightly when the p_{m} increases, the results are quite stable.

Table 1: Average clustering accuracy over 30 runs for different combinations of classes in the Hardangervidda dataset. The best results are marked in bold. The baseline methods are: ZI (zero imputation), AI (average imputation), MI (median imputation) and MFVI (most frequent value imputation), combined with either k-means or spectral clustering using an RBF kernel.

Classes	*PCKID*	Spectral clustering, RBF				k-means			
		ZI	AI	MI	MFVI	ZI	AI	MI	MFVI
2-3	0.580	0.610	0.610	0.624	**0.627**	0.601	0.601	0.601	0.605
2-4	0.536	0.663	0.663	0.663	**0.674**	0.591	0.591	0.590	0.597
2-5	0.661	0.589	0.589	0.598	0.605	**0.671**	**0.671**	0.663	0.652
2-6	**0.712**	0.578	0.578	0.571	0.594	0.672	0.672	0.664	0.639
2-7	**0.868**	0.519	0.519	0.516	0.501	0.854	0.854	0.858	0.862
3-4	0.698	0.505	0.505	0.505	0.511	0.697	0.697	0.711	**0.722**
3-5	**0.563**	0.521	0.521	0.511	0.516	0.534	0.534	0.540	0.540
3-6	**0.620**	0.565	0.565	0.562	0.564	0.521	0.521	0.519	0.523
3-7	**0.933**	0.501	0.501	0.726	0.522	0.577	0.577	0.599	0.603
4-5	0.764	0.517	0.517	0.512	0.510	0.839	0.839	0.847	**0.848**
4-6	**0.897**	0.517	0.517	0.547	0.547	**0.897**	**0.897**	0.894	0.880
4-7	**0.931**	0.550	0.550	0.547	0.534	0.687	0.687	0.687	0.718
5-6	**0.740**	0.623	0.623	0.644	0.672	0.554	0.554	0.602	0.606
5-7	**0.956**	0.687	0.687	0.667	0.698	0.706	0.706	0.706	0.706
6-7	**0.970**	0.767	0.767	0.752	0.696	0.759	0.759	0.759	0.670

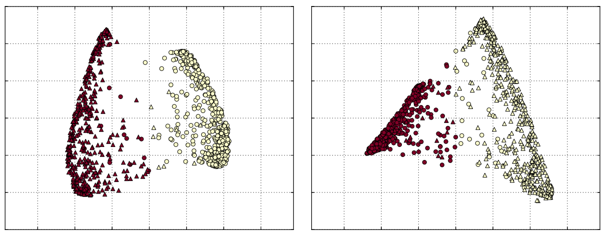

(a) (b)

Figure 2: Example of embedding and clustering in kernel space with (a): No missingness, (b): 90% missingness. The marker indicates the true label, while the color indicates the clustering results.

Fig. 2a–Fig. 2b shows two dimensional representations using kernel PCA on *PCKID* with $p_{\mathrm{m}} = 0$ and $p_{\mathrm{m}} = 0.9$, respectively. The shape of the markers indicates ground truth class, while the color indicate the clustering result. It is interesting to see that although the plot with no missing data has a smoother structure, the overall topology seems to be very similar when $p_{\mathrm{m}} = 0.9$. The two-classes seem to be less separable in the plot with more missing data, which is not surprising, given the numerical clustering results in Fig. 1b.

When considering the approach of k-means directly on data with imputed values, we see that none of the imputation techniques perform as well as *PCKID*, although in this case mean imputation works reasonably well. To explain performance improvements as p_{m} increases, it is possible that the missingness patterns chosen for this experiment introduce some noise that provides a form of regularization that is beneficial to certain imputation techniques, or maybe the balance in the dataset is helping the mean of the observed values to not introduce bias towards one class. With median–, zero– and most frequent value imputation, the clustering accuracy starts to decline around $p_{\mathrm{m}} = 0.3$, with zero imputation and most frequent value imputation following almost exactly the same path. This is likely due to the nature of the data, where many of the dimensions actually contains zeros in most of the images. The most frequent value in most dimensions will then be zero.

Spectral clustering using an RBF kernel completely fails in this experiment, which is probably due to a sub-optimal kernel width. However, this illustrates the difficulty with an unsupervised problem, where no prior information is given, making cross-validation virtually impossible without expertise knowledge on the data.

4.3.5 Land cover clustering

In this experiment, we cluster pixels in high resolution land cover images contaminated with clouds, also used for classification in [31, 32]. The data consists of three Landsat ETM+ images covering Hardangervidda in southern Norway, in addition to elevation and slope information. With 6 bands in each image, the total dimensionality of the data is 20. In this dataset, a value is considered missing if a pixel in an image is contaminated by either clouds or snow/ice. For details on how the dataset is constructed, see [31].

The pixels in the image are labeled as one of 7 classes: 1) *water*, 2) *ridge*, 3) *leeside*, 4) *snowbed*, 5) *mire*, 6) *forest* and 7) *rock*. In the first part of this experiment, we exclude the water class since it is easy to separate from the other classes in the Norwegian mountain vegetation. To investigate how the *PCKID* handle the different combination of classes, we restrict the initial analysis

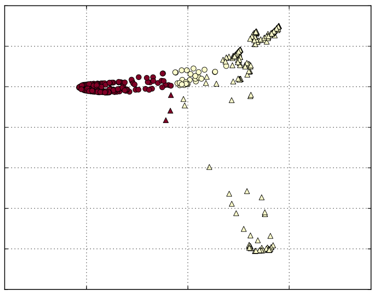


Figure 3: Example of mapping for the *forest–rock* class pair. Colors indicate clustering, while the shape of the marker indicates the ground truth label.

Table 2: Clustering of the full Hardangervidda dataset with 7 clusters. The best results are marked in bold.

	Accuracy		NMI		ARI	
	μ	σ	μ	σ	μ	σ
GMM	4.81E-01	2.90E-02	4.27E-01	1.81E-02	**3.51E-01**	2.50E-02
k-POD	4.57E-01	1.40E-02	3.77E-01	2.00E-02	2.79E-01	1.28E-02
PCK						
AI	3.87E-01	9.56E-03	4.05E-01	5.90E-03	2.60E-01	4.13E-03
MFVI	3.75E-01	1.41E-02	3.01E-01	9.05E-03	2.01E-01	1.63E-02
MI	3.85E-01	4.90E-03	4.00E-01	4.14E-03	2.61E-01	2.86E-03
ZI	3.86E-01	1.32E-02	2.94E-01	6.12E-03	2.03E-01	8.00E-03
RBF						
AI	2.67E-01	1.18E-03	2.53E-01	3.43E-03	2.49E-02	1.31E-03
MFVI	2.69E-01	6.01E-04	2.41E-01	1.13E-03	-7.83E-03	3.70E-03
MI	3.43E-01	9.20E-05	2.53E-01	9.80E-05	7.73E-02	8.55E-05
ZI	2.73E-01	3.31E-04	2.23E-01	1.75E-03	2.10E-02	3.97E-04
k-means						
AI	4.52E-01	1.89E-03	3.97E-01	1.40E-03	2.95E-01	2.17E-03
MFVI	4.72E-01	1.94E-04	3.46E-01	1.11E-04	2.64E-01	1.68E-04
MI	4.49E-01	2.55E-03	3.94E-01	2.13E-03	2.91E-01	2.83E-03
ZI	4.62E-01	1.09E-03	2.79E-01	3.81E-04	2.84E-01	1.04E-03
PCKID	**4.94E-01**	5.84E-03	**4.49E-01**	3.44E-03	**3.52E-01**	4.14E-03

to pairwise classes. Each dimension is standardized on the observed data.

The average clustering accuracy for each combination of the chosen classes is reported in Tab. 1. The average is computed over 30 runs of each algorithm. We see that *PCKID* seems to perform better for most class pairs. Although it might struggle with some classes, most notably class 2. For the class pair 3-5, *PCKID* wins with a clustering accuracy of 0.563, which is not much better than random chance in a two-class problem. It is however worth to note that the classes labels are set according the vegetation at the actual location, which is not necessarily the group structure reflected in the data. The class combinations where *PCKID* really outperforms the other methods seems to be when class 7 (rocks) is present in the data, where we improve performance by up to 25 percentage points compared to the baseline methods.

Fig. 3 shows an example of a mapping for the *forest–rock* class pair, where it seems like the *rock* class, as defined by the ground truth, actually consists of two separate structures in the KPCA embedding using *PCKID*. This demonstrates the power of *PCKID*s ability to adapt to the inherent structures in the data.

Tab. 2 shows a clustering of the full Hardangervidda dataset, where all 7 classes are considered simultaneously. We report both mean and standard deviation of the clustering accuracy, NMI and ARI, computed over 30 runs. *PCKID* outperforms the baseline methods in both accuracy and NMI, while the ARI score is the same for *PCKID* and the GMM. Salberg and Jenssen [32] reports a classification accuracy of up to 83% on this dataset, indicating that this is a challenging dataset, even for supervised methods.

4.3.6 Wine dataset

In this experiment, we cluster the well known Wine dataset from the UCI Machine Learning Repository [11]. We generate missing data by simulating three common missing data mechanisms: missing completely at random (MCAR), missing at random (MAR) and not missing at random (NMAR). Similar to [8], we simulate the MCAR mechanism by removing values at random from the dataset and the MAR mechanism by randomly removing values at random from the 1st, 4th and 7th feature. For the NMAR mechanism, we assume that the sensor collecting each feature is saturated by the true value being larger than the maximum value the sensor can record. To obtain an overall proportion of missing values, p_{m}, we simulate this by randomly choosing to remove the largest $p_{\mathrm{m}} \cdot 100\%$ values of every column in the data matrix. The results are shown in Tab. 3.

For the MCAR and MAR results, *PCKID* outperforms the baseline methods in all instances, except one where PCK with an imputation method performs on par with *PCKID*. For the difficult NMAR problem, *PCKID* does not seem to be the best in most cases, but it is at least the second best in all instances except one. The most stable high performing baseline method in this instance seems to be PCK, combined with an imputation method. However, we would like to stress that only the *best* imputation method is shown in this table, which will vary depending on the mechanism used to generate the missing data.

4.4 Ranking incomplete vectorial data

In this experiment, we use *PCKID* in order to learn Markov Chain transition probabilities for unsupervised ranking of data with missing data akin to [24].

Table 3: Mean and standard deviation of accuracy when clustering the Wine dataset over 30 runs. The percentages indicate the proportion of missing values in the data. For the PCK, k-means and RBF, we only report the best of the four imputation methods. Best results are highlighted in bold. Second best results are highlighted in italic.

	MCAR									
p_{m}	5%		15%		25%		35%		45%	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
GMM	9.56E-01	1.26E-02	*9.45E-01*	1.96E-02	*9.27E-01*	1.77E-02	9.09E-01	5.61E-02	8.75E-01	8.15E-02
k-POD	9.59E-01	1.01E-02	9.33E-01	5.68E-02	9.26E-01	1.93E-02	9.08E-01	1.97E-02	8.69E-01	5.53E-02
PCK (Best)	*9.61E-01*	1.14E-02	9.42E-01	1.35E-02	9.13E-01	1.93E-02	8.95E-01	3.10E-02	8.29E-01	3.31E-02
k-means (Best)	9.60E-01	1.14E-02	9.41E-01	1.31E-02	9.23E-01	1.45E-02	*9.12E-01*	1.61E-02	*8.84E-01*	2.09E-02
RBF (Best)	4.23E-01	3.01E-02	4.32E-01	2.98E-02	4.22E-01	3.33E-02	4.05E-01	2.23E-02	4.11E-01	2.16E-02
PCKID	**9.65E-01**	7.49E-03	**9.55E-01**	1.31E-02	**9.40E-01**	1.51E-02	**9.29E-01**	2.08E-02	**9.08E-01**	2.12E-02
	MAR									
p_{m}	5%		9%		13%		17%		21%	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
GMM	9.39E-01	1.85E-02	9.17E-01	6.13E-02	9.15E-01	5.48E-02	9.07E-01	5.44E-02	9.01E-01	5.91E-02
k-POD	9.41E-01	5.74E-02	9.28E-01	5.53E-02	*9.28E-01*	8.22E-03	9.19E-01	4.49E-03	*9.16E-01*	0.00E-00
PCK (Best)	**9.59E-01**	6.01E-03	*9.42E-01*	1.68E-02	9.25E-01	1.31E-02	9.19E-01	2.06E-02	8.23E-01	6.83E-02
k-means (Best)	*9.47E-01*	9.03E-03	9.39E-01	7.24E-03	*9.26E-01*	8.04E-03	*9.20E-01*	5.01E-03	*9.16E-01*	2.22E-16
RBF (Best)	4.31E-01	3.97E-02	4.27E-01	3.93E-02	4.20E-01	1.86E-02	4.21E-01	2.68E-02	4.28E-01	2.16E-02
PCKID	**9.59E-01**	7.83E-03	**9.48E-01**	9.33E-03	**9.46E-01**	6.74E-03	**9.43E-01**	5.87E-03	**9.38E-01**	3.55E-03
	NMAR									
p_{m}	5%		15%		25%		35%		45%	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
GMM	9.59E-01	6.96E-02	**9.60E-01**	1.40E-03	**9.55E-01**	6.08E-02	8.94E-01	7.61E-02	7.18E-01	1.17E-01
k-POD	*9.66E-01*	3.33E-16	9.49E-01	2.22E-16	8.96E-01	7.26E-02	8.82E-01	5.14E-02	8.58E-01	1.07E-01
PCK (Best)	9.64E-01	2.65E-03	9.38E-01	0.00E-00	*9.51E-01*	3.53E-03	**9.57E-01**	8.96E-03	**9.29E-01**	2.71E-03
k-means (Best)	*9.66E-01*	3.33E-16	9.49E-01	2.22E-16	9.10E-01	3.33E-16	9.18E-01	2.65E-03	8.83E-01	1.39E-02
RBF (Best)	4.25E-01	2.30E-02	4.37E-01	3.04E-02	4.46E-01	4.01E-02	4.31E-01	1.75E-02	4.26E-01	1.66E-02
PCKID	**9.68E-01**	3.81E-03	*9.53E-01*	6.35E-03	9.49E-01	5.98E-03	*9.49E-01*	5.44E-03	*8.99E-01*	8.21E-03

4.4.1 Baseline methods and performance metric

We compare *PCKID* to baseline kernels using the exact same ranking procedure for all instances. The baseline kernels are chosen as RBF kernels with various imputation methods and PCK with the same imputation methods. The experiment is performed as follows: 1. Given a query, compute a ranking score vector for the data without missing values and a score vector for the same query with missing values using all kernels. 2. For a given kernel, compute the cosine between the two score vectors (without and with missing data). This is performed for 100 random queries, with the same query being used with all kernels.

4.4.2 Datasets

The datasets used in this experiment consists of standard UCI machine learning repository datasets (Ecoli, Iris, WDBC and Wine) and a NIPS document dataset. The NIPS document dataset is generated from a bag–of–word representation of 1740 NIPS papers[2]. This is then preprocessed using SVD in order to reduce its dimensionality to 20, a common procedure used for recommender systems with this type of data. Missingness in the data is simulated by the MCAR mechanism.

[2]https://cs.nyu.edu/ roweis/data.html

4.4.3 Results

Let $\boldsymbol{\pi}_b(q)$ and $\boldsymbol{\pi}_{PCKID}(q)$ denote the ranking score vector for a given query q for a baseline kernel and *PCKID* respectively when the dataset has no missing values and let $\boldsymbol{\pi}_b^m(q)$ and $\boldsymbol{\pi}_{PCKID}^m(q)$ denote their score vectors when there are missing values in the data. Fig. 4 shows the performance of the baseline kernels versus *PCKID* for different amounts of missing data for the NIPS document dataset. Each point in the plot represents a specific query. The values of the first axis in the plots are calculated as $\cos(\angle[\boldsymbol{\pi}_b(q), \boldsymbol{\pi}_b^m(q)])$, while the values of the second axis are calculated as $\cos(\angle[\boldsymbol{\pi}_{PCKID}(q), \boldsymbol{\pi}_{PCKID}^m(q)])$. Intuitively, the cosine value is close to 1 if the angle between the score vectors is small. If a point is *above* the dotted diagonal line, the score vector of PCKID with missing data is closer to its non missing data counterpart (in angle) compared to the corresponding baseline kernel. From the plots, it is apparent that *PCKID* outperforms or is on par with the baseline kernels across all missingness percentages shown here. It is also clear that for the baseline methods, the performance might be on par for some queries, but significantly lower for other queries. For instance, PCK with zero– or most–frequent–value imputation, some queries are on (or close to) the diagonal line, while others are far above it.

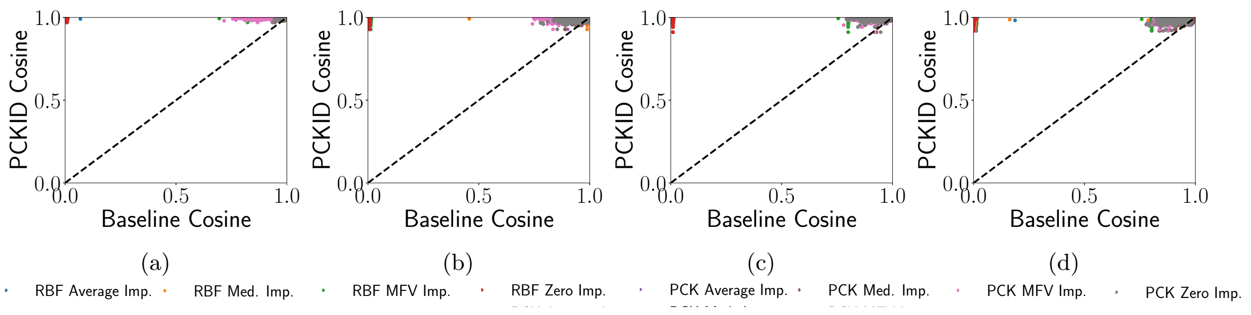


Figure 4: Ranking performance of baseline kernel (first axis) versus *PCKID* (second axis) with (a) 5% (b) 10% (c) 15% and (d) 20% missing data. Each dot represents a query.

Fig. 5 shows a more comprehensive comparison on several datasets. We computed the *mean* cosine value for all combinations of

1. Baseline kernel choice (RBF and PCK)
2. Missingness percentage (5%, 10%, 15% and 20%)
3. Imputation method (Average, Median, Most–frequent–value and Zero imputation)
4. Dataset (NIPS documents, Ecoli, Iris, WDBC and Wine).

These are compared to the respective mean cosine value for *PCKID* in the same setting. Out of the 160 possible configurations, *PCKID* outperformed the baseline kernel in 154 instances. All instances where the baseline kernel outperformed *PCKID* had 5% missingness and a PCK, but the imputation method that worked best varied from dataset to dataset. For the Ecoli dataset with 5% missingness, PCK outperformed *PCKID* with both mean, median and most–frequent–value imputation, while for the other datasets (Iris, NIPS documents and WDBC), the the best imputation method was either mean– or median imputation.

5 Conclusion

In this paper, we have proposed *PCKID*, a novel kernel function for unsupervised learning, designed to i) explicitly handle incomplete data and ii) be robust with regards to parameter choice. By combining posterior distributions of Gaussian Mixture Models for incomplete data on different scales, *PCKID* is able to learn similarities on the data manifold, yielding a kernel function without any *critical* hyperparameters to tune. Experiments have demonstrated the strength of our method, by improved spectral clustering accuracy compared to baseline methods. Furthermore, when applied to ranking tasks, the score vector remains relatively unchanged when injecting missingness into the data, compared to the baseline kernels.

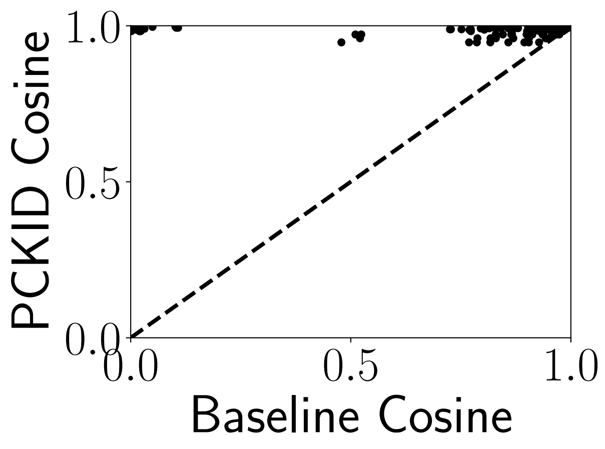


Figure 5: Comparison of datasets for ranking. Points above the dotted diagonal line indicates datasets in which *PCKID* outperforms the baseline.

Acknowledgments

This work was partially funded by the Norwegian Research Council FRIPRO grant no. 239844 on developing the *Next Generation Learning Machines*.

References

[1] Mark A. Aizerman. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, 25:821–837, 1964.

[2] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. *Symposium on Discrete Algorithms (SODA)*, 2007.

[3] Lluís A. Belanche, Vladimer Kobayashi, and Tomàs Aluja. Handling missing values in kernel methods with application to microbiology data. *Neurocomputing*, 141:110 – 116, 2014. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2014.01.047.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, April 1998. ISSN 01697552. doi: 10.1016/S0169-7552(98)00110-X.

[5] M Emre Celebi and Kemal Aydin. *Unsupervised learning algorithms.* Springer, 2016.

[6] Olivier Chapelle, Jason Weston, and Bernhard Schölkopf. Cluster kernels for semi-supervised learning. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 601–608. MIT Press, 2003. URL `http://papers.nips.cc/paper/2257-cluster-kernels-for-semi-supervised-learning.pdf`.

[7] Gal Chechik, Geremy Heitz, Gal Elidan, Pieter Abbeel, and Daphne Koller. Max-margin classification of data with absent features. *JMLR*, 9(Jan):1–21, 2008.

[8] Jocelyn T Chi, Eric C Chi, and Richard G Baraniuk. k-pod: A method for k-means clustering of missing data. *The American Statistician*, 70(1):91–99, 2016.

[9] F. Chung and W. Zhao. Pagerank and random walks on graphs. *Fete of combinatorics and computer science*, pages 1–16, 2010.

[10] John K Dixon. Pattern recognition with partly missing data. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(10):617–621, 1979.

[11] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[12] Maurizio Filippone, Francesco Camastra, Francesco Masulli, and Stefano Rovetta. A survey of kernel and spectral methods for clustering. *Pattern recognition*, 41(1):176–190, 2008.

[13] Zoubin Ghahramani and Michael I Jordan. Supervised learning from incomplete data via an em approach. *Advances in neural information processing systems*, pages 120–120, 1994.

[14] David F Gleich. Pagerank beyond the web. *Siam Review*, 57(3):321–363, 2015.

[15] E. Izquierdo-Verdiguier, L. Gómez-Chova, L. Bruzzone, and G. Camps-Valls. Semisupervised kernel feature extraction for remote sensing image analysis. *IEEE Transactions on Geoscience and Remote Sensing*, 52(9):5567–5578, 2014.

[16] Emma Izquierdo-Verdiguier, Robert Jenssen, Luis Gómez-Chova, and Gustavo Camps-Valls. Spectral clustering with the probabilistic cluster kernel. *Neurocomputing*, 149:1299–1304, 2015.

[17] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

[18] Robert Jenssen. Kernel entropy component analysis. *IEEE transactions on pattern analysis and machine intelligence*, 32(5):847–860, 2010.

[19] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[20] Sun Yuan Kung. *Kernel methods and machine learning.* Cambridge University Press, 2014.

[21] Tsung I Lin, Jack C Lee, and Hsiu J Ho. On fast supervised learning for normal mixture models with missing information. *Pattern Recognition*, 39(6):1177–1187, 2006.

[22] Andrew Lithio and Ranjan Maitra. An efficient k-means-type algorithm for clustering datasets with incomplete records. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 11(6):296–311, 2018.

[23] Tiantian Liu and Yair Goldberg. Kernel machines with missing responses. *arXiv preprint arXiv:1806.02865*, 2018.

[24] Sigurd Lokse and Robert Jenssen. Ranking using transition probabilities learned from multi-attribute data. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2851–2855. IEEE, 2018.

[25] Benjamin M Marlin. *Missing data problems in machine learning.* PhD thesis, University of Toronto, 2008.

[26] Karl **O**yvind Mikalsen, Filippo Maria Bianchi, Cristina Soguero-Ruiz, and Robert Jenssen. Time series cluster kernel for learning similarities between multivariate time series with missing data. *Pattern Recognition*, 76:569–581, 2018.

[27] Majid Mojirsheibani and Zahra Montazeri. Statistical classification with missing covariates. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(5):839–857, 2007.

[28] Stefano Monti, Pablo Tamayo, Jill Mesirov, and Todd Golub. Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data. *Machine learning*, 52(1-2):91–118, 2003.

[29] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856, 2001.

[30] Feiping Nie, Zinan Zeng, Ivor W Tsang, Dong Xu, and Changshui Zhang. Spectral embedded

clustering: A framework for in-sample and out-of-sample spectral clustering. *IEEE Transactions on Neural Networks*, 22(11):1796–1808, 2011.

[31] Arnt-Børre Salberg. Land cover classification of cloud-contaminated multitemporal high-resolution images. *IEEE Transactions on Geoscience and Remote Sensing*, 49(1):377–387, 2011.

[32] Arnt-Børre Salberg and Robert Jenssen. Land-cover classification of partly missing data using support vector machines. *International journal of remote sensing*, 33(14):4471–4481, 2012.

[33] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International Conference on Artificial Neural Networks*, pages 583–588. Springer, 1997.

[34] Kazi Tanzeem Shahid, Akshay Malhotra, Ioannis D. Schizas, and Saibun Tjuatja. Unsupervised kernel correlations based hyperspectral clustering with missing pixels. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 11(6):1799–1810, 2018.

[35] D. Tuia and G. Camps-Valls. Semisupervised remote sensing image classification with cluster kernels. *IEEE Geoscience and Remote Sensing Letters*, 6(2):224–228, 2009.

[36] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[37] Yi Yang, Dong Xu, Feiping Nie, Shuicheng Yan, and Yueting Zhuang. Image clustering using local discriminant models and global integration. *IEEE Transactions on Image Processing*, 19(10):2761–2773, 2010.

[38] Dao-Qiang Zhang and Song-Can Chen. Clustering incomplete data using kernel-based fuzzy c-means algorithm. *Neural processing letters*, 18(3):155–162, 2003.

[39] Dengyong Zhou, Jason Weston, Arthur Gretton, Olivier Bousquet, and Bernhard Schölkopf. Ranking on data manifolds. In *Advances in neural information processing systems*, pages 169–176, 2004.

[40] Arthur Zimek, Matthew Gaudet, Ricardo JGB Campello, and Jörg Sander. Subsampling for efficient and effective unsupervised outlier detection ensembles. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 428–436. ACM, 2013.

[41] Marek Śmieja, Łukasz Struski, Jacek Tabor, and Mateusz Marzec. Generalized rbf kernel for incomplete data. *Knowledge-Based Systems*, 173: 150 – 162, 2019. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2019.02.034.

Paper II

Kernel Personalized PageRank

Sigurd Løkse and Robert Jenssen.

In submission

Kernel Personalized PageRank

Sigurd Løkse
Department of Physics and Technology,
UiT The Arctic University of Norway,
NO-9037 Tromsø,
Norway
sigurd.lokse@uit.no

Robert Jenssen
Department of Physics and Technology,
UiT The Arctic University of Norway,
NO-9037 Tromsø,
Norway
robert.jenssen@uit.no

Abstract

We introduce the Kernel Personalized PageRank (KPPR) by interpreting the Personalized PageRank (PPR) as a linear ranking function in a Reproducing Kernel Hilbert Space (RKHS). This facilitates a low–rank embedding that enables ranking of out–of–sample data points by minimizing an approximation error to the PPR. We show that this error is provably upper bounded for uniform seed distributions. The KPPR provides new insight by disentangling the ranking into a base score and a Markov chain restart component. We perform a range of experiments to highlight these properties and show that the low-rank KPPR may even outperform the PPR.

1 INTRODUCTION

The Google PageRank algorithm (Brin and Page, 1998) has had a tremendous impact on our daily lives. Originally designed to rank web pages, it has also been applied to other types of data/problems (Gleich, 2015; Jing and Baluja, 2008; Klicpera et al., 2019; Zhou et al., 2004b,a). As a more general ranking algorithm, the Personalized PageRank (PPR) (Chung and Zhao, 2010; Kloumann et al., 2017) allows for personalized rankings. This enables the user to specify a seed distribution that alters where the random walk restarts. One limitation of the PPR is that the computation of the exact stationary distribution requires inversion of a $N \times N$ matrix, where N is the size of the dataset. Exact ranking of previously unseen data therefore necessitates a re–inversion of the new (larger) matrix or to perform new power iterations, although approximate iterative methods exist (Bahmani et al., 2010; Lofgren, 2015; Zhang et al., 2016; Zhan et al., 2019).

In this paper, we introduce the Kernel Personalized PageRank (KPPR), based on a novel analysis wherein we show that the PPR is a linear ranking function in a Reproducing Kernel Hilbert Space (RKHS) for data represented by undirected graphs with edge–weights in the form of a symmetric non–negative affinity matrix. This provides new theoretical insight as the KPPR can be disentangled into a base score and a Markov chain restart component which encodes the seed distribution. Operating in a low-rank empirical (spectral) kernel space, we show that the KPPR enables ranking of previously unseen out-of-sample data, without performing new power iterations or re–inverting potentially large matrices. This is done by minimizing an approximation error with respect to the PPR. We show that this error is provably upper-bounded for uniform seed distributions. We validate our theory through experiments, and, finally, we empirically show that the KPPR may even outperform the PPR in terms of ranking quality. Note that all proofs are given in the supplementary material.

Notation Matrices and vectors are denoted by bold fonts, with upper case for matrices and lower case for vectors. Element (i, j) in the matrix $\mathbf{A}$ is denoted by $(\mathbf{A})_{ij}$ and element i of vector $\mathbf{a}$ is denoted by $(\mathbf{a})_i$. $\mathbf{a}^T$ is a transposed vector. $\langle \cdot, \cdot \rangle$ is an inner product. All vectors are column vectors.

2 PERSONALIZED PAGERANK

Consider the difference equation for the PPR

$$\mathbf{r}_{k+1}^T = (1-\alpha)\mathbf{r}_k^T \mathbf{P} + \alpha \mathbf{s}^T, \qquad (1)$$

where $\alpha \in (0, 1)$ is the restart probability and $\mathbf{s} \in \mathbb{R}^N$ is the seed distribution with $(\mathbf{s})_i \geq 0$, $\sum_{i=1}^N (\mathbf{s})_i = 1$. The transition probability matrix is assumed to be on the form $\mathbf{P} = \mathbf{D}^{-1}\mathbf{K}$, where $\mathbf{K}$ is a symmetric affinity matrix with non–negative values and $\mathbf{D} =$

$\text{diag}(d_i)$, $d_i = \sum_{j=1}^{N}(\mathbf{K})_{ij}$ is its diagonal degree matrix. Eq. (1) converges to the stationary distribution $\boldsymbol{\pi}(\alpha, \mathbf{s})$ of the Markov chain associated with the stochastic matrix $\mathbf{P}' = (1-\alpha)\mathbf{P} + \alpha \mathbb{1}\mathbf{s}^T$, in which case $\boldsymbol{\pi}(\alpha, \mathbf{s})^T = (1-\alpha)\boldsymbol{\pi}(\alpha, \mathbf{s})^T\mathbf{P} + \alpha\mathbf{s}^T$. Chung and Zhao (Chung and Zhao, 2010) showed that solving for $\boldsymbol{\pi}(\alpha, \mathbf{s})$ yields

$$\boldsymbol{\pi}(\alpha, \mathbf{s}) = \beta \mathbf{D}^{\frac{1}{2}} \boldsymbol{\mathcal{G}}_\beta \mathbf{D}^{-\frac{1}{2}}\mathbf{s}, \tag{2}$$

where $\beta = \frac{\alpha}{1-\alpha}$, $\boldsymbol{\mathcal{G}}_\beta \boldsymbol{\mathcal{L}}_\beta = \boldsymbol{\mathcal{L}}_\beta \boldsymbol{\mathcal{G}}_\beta = \mathbf{I}$, $\boldsymbol{\mathcal{L}}_\beta = \beta\mathbf{I} + \boldsymbol{\mathcal{L}} = (1+\beta)\mathbf{I} - \boldsymbol{\mathcal{K}}$. Here, $\boldsymbol{\mathcal{G}}_\beta$ denotes the Green's function (Chung and Yau, 2000) of the β-normalized Laplacian $\boldsymbol{\mathcal{L}}_\beta$, $\boldsymbol{\mathcal{L}}$ is the normalized Laplacian and $\boldsymbol{\mathcal{K}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{K}\mathbf{D}^{-\frac{1}{2}}$ is the symmetrically normalized affinity matrix. Since the β-normalized Laplacian is symmetric, it can be written on the form $\boldsymbol{\mathcal{L}}_\beta = \sum_{i=1}^{N} \lambda_i \mathbf{e}_i \mathbf{e}_i^T = \mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^T$, where $\lambda_i, i = 1, 2, \ldots, N$ and $\mathbf{e}_i, i = 1, 2, \ldots N$ are the eigenvalues and orthonormal eigenvectors of $\boldsymbol{\mathcal{L}}_\beta$. Thus, the Green's function is given by $\boldsymbol{\mathcal{G}}_\beta = \sum_{i=1}^{N} \frac{1}{\lambda_i}\mathbf{e}_i\mathbf{e}_i^T = \mathbf{E}\boldsymbol{\Lambda}^{-1}\mathbf{E}^T$.

3 KERNEL PERSONALIZED PAGERANK (KPPR)

Proposition 1. *The Green's function $\boldsymbol{\mathcal{G}}_\beta$ is positive definite for* $0 < \alpha < 1$.

A consequence of Prop. 1, and a key observation for the results in this paper, is that there exists a RKHS where the inner products are given by the Green's function $\boldsymbol{\mathcal{G}}_\beta$ (Mercer, 1909). That is, $(\boldsymbol{\mathcal{G}}_\beta)_{ij} = \mathcal{G}_\beta(\mathbf{x}_i, \mathbf{x}_j) = \langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j)\rangle$. Accordingly, $\mathcal{G}_\beta$ will be referred to as the *Green's kernel* in the remainder of the paper. In the following proposition, we interpret the score of the PPR in terms of linear operations in kernel space.

Proposition 2. *Let $\mathbf{x}_i$ be a datapoint with degree d_i, and let $\pi_i(\alpha, \mathbf{s})$ be its PPR score. Furthermore, let $\boldsymbol{\phi}(\cdot)$ be the mapping from input space to the RKHS defined by $\boldsymbol{\mathcal{G}}_\beta$. Then, the following holds:*

$$\frac{\pi_i(\alpha, \mathbf{s})}{\sqrt{d_i}} \propto \langle \boldsymbol{\phi}(\mathbf{x}_i), \mathbf{w}\rangle, \tag{3}$$

where $\mathbf{w} = \sum_{j=1}^{N} \frac{(\mathbf{s})_j}{\sqrt{d_j}}\boldsymbol{\phi}(\mathbf{x}_j) = \mathbf{m}_s$ is a weighted mean in the RKHS, weighted by the seed.

As seen in Prop. 2, and as a new viewpoint provided in this paper, the PPR[1] is in fact computed by a linear ranking function in kernel space (a common family of ranking functions (Li, 2011)). The weight vector is a weighted mean in kernel space, with weights that are dependent on the seed distribution. As a consequence, datapoints in kernel space that are located furthest away from the origin, *in the direction of the seed–weighted mean* are assigned a large score.

[1]In particular, $\frac{\pi_i(\alpha, \mathbf{s})}{\sqrt{d_i}}$.

3.1 Empirical RKHS and Low–Rank Approximation

In this section, we define a low–rank approximation of the embedding in the empirical RKHS of $\boldsymbol{\mathcal{G}}_\beta$, which is used to approximate the score of a datapoint. We analyze the contribution of the eigenvectors and provide a method for ordering the eigenvectors used in the embedding, in order to minimize the norm of a scaled error vector. Moreover, we show that there exists an equivalent variant of this low rank approximation, whose values depends directly on the *affinity matrix* used to compute $\boldsymbol{\mathcal{G}}_\beta$. This allows for approximating scores of *out–of–sample datapoints* without re–inverting the β–normalized Laplacian.

The empirical embedding to the RKHS induced by $\boldsymbol{\mathcal{G}}_\beta$ is given by $\mathbf{Z}_\beta = \mathbf{E}\boldsymbol{\Lambda}^{-\frac{1}{2}}$ (Williams, 2002), where $\mathbf{E}$ and $\boldsymbol{\Lambda}$ are the orthogonal eigenvector matrix and diagonal eigenvalue matrix of $\boldsymbol{\mathcal{L}}_\beta$. This is a representation in a Euclidean space, in which the inner products contained in $\boldsymbol{\mathcal{G}}_\beta$ are preserved. Moreover, by defining the symmetrically normalized affinity matrix, $\boldsymbol{\mathcal{K}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{K}\mathbf{D}^{-\frac{1}{2}}$, one can easily show that $\boldsymbol{\mathcal{K}}$ has the same eigenvectors as $\boldsymbol{\mathcal{L}}_\beta$, but with shifted eigenvalues $\lambda_i^* = 1 + \beta - \lambda_i$. Therefore, the empirical embedding to the RKHS induced by $\boldsymbol{\mathcal{G}}_\beta$ is given by $\mathbf{Z}_\beta = \mathbf{E}((1+\beta)\mathbf{I} - \boldsymbol{\Lambda}^*)^{-\frac{1}{2}}$, which only depends on the eigenvalues and eigenvectors of $\boldsymbol{\mathcal{K}}$. An interesting observation is that the eigenvectors used in the embedding are in fact similar to the ones used in other graph based methods like spectral clustering, Laplacian eigenmaps, etc. (Belkin and Niyogi, 2003; Jenssen, 2010; Maji et al., 2011; Ng et al., 2001; Shi and Malik, 2000).

Using the previous results, we clearly have $\boldsymbol{\mathcal{G}}_\beta = \mathbf{Z}_\beta\mathbf{Z}_\beta^T$ and $\boldsymbol{\pi}(\alpha, \mathbf{s}) = \beta\mathbf{D}^{\frac{1}{2}}\mathbf{Z}_\beta\mathbf{Z}_\beta^T\mathbf{D}^{-\frac{1}{2}}\mathbf{s}$. In the same manner as in Prop. 2, we get the following relationship between the PPR and the embedded data:

Proposition 3. *Let $\mathbf{x}_i$ be a datapoint with degree d_i, and let $\pi_i(\alpha, \mathbf{s})$ be its PPR score. Furthermore, let $\mathbf{z}_i(\beta)$ be its embedding to the empirical kernel space of $\boldsymbol{\mathcal{G}}_\beta$ (located in row i of $\mathbf{Z}_\beta$). Then, the following holds:*

$$\frac{\pi_i(\alpha, \mathbf{s})}{\sqrt{d_i}} \propto \mathbf{w}^T\mathbf{z}_i(\beta), \tag{4}$$

where $\mathbf{w} = \sum_{j=1}^{N} \frac{(\mathbf{s})_j}{\sqrt{d_j}}\mathbf{z}_j(\beta) = \mathbf{m}_s$ is a weighted mean in the empirical kernel space, weighted by the seed distribution.

By using this result, we can compute the PPR by an inner product between vectors in the empirical kernel space of $\boldsymbol{\mathcal{G}}_\beta$. These derivations provide a new viewpoint to the PPR, and importantly, it opens up for low–rank approximations of $\boldsymbol{\mathcal{G}}_\beta$ which is often useful for e.g. out–of–sample projections.

3.1.1 Eigenvector Contributions

To further analyze what is actually happening in the empirical RKHS, we split the contribution from the eigenvectors to the ranking as shown in the following proposition.

Proposition 4. *Let $\boldsymbol{\pi} = \frac{1}{\mathbb{1}^T \mathbf{D}\mathbb{1}}\mathbf{D}\mathbb{1}$ denote the stationary distribution of the Markov Chain with transition probability matrix $\mathbf{P} = \mathbf{D}^{-1}\mathbf{K}$ (Meyer et al., 2013). Then the PPR score can be decomposed as follows:*

$$\boldsymbol{\pi}(\alpha, \mathbf{s}) = \boldsymbol{\pi} + \boldsymbol{\pi}'(\alpha, \mathbf{s}), \tag{5}$$

where $\boldsymbol{\pi}'(\alpha, \mathbf{s}) = \beta \mathbf{D}^{\frac{1}{2}} \boldsymbol{\mathcal{G}}'_\beta \mathbf{D}^{-\frac{1}{2}}\mathbf{s}$, $\boldsymbol{\mathcal{G}}'_\beta = \boldsymbol{\mathcal{G}}_\beta - \frac{1}{\beta}\mathbf{D}^{\frac{1}{2}}\mathbb{1}(\mathbf{D}^{\frac{1}{2}}\mathbb{1})^T = \sum_{k=2}^N \frac{1}{1+\beta-\lambda_k^*}\mathbf{e}_k\mathbf{e}_k^T$ is the contribution to the PPR score from all eigenvectors of $\boldsymbol{\mathcal{K}}$, except the one with largest eigenvalue.

Prop. 4 states that in the empirical RKHS, the PPR score can be decomposed into two components. 1. The *base score* component, $\boldsymbol{\pi}$, which is the score of the initial Markov Chain without restarts. 2. The *restart* component, $\boldsymbol{\pi}'(\alpha, \mathbf{s})$, which alters the base score according to the seed distribution.

The base score contains all contributions from the first/largest eigenvector, while the restart component is based on contributions from all other eigenvectors. Note that the base score is always known beforehand, since the first eigenvector is on a known form. Therefore, we omit it from the embedding and add its contribution afterwards.

3.1.2 Out–of–Sample Extension

In this section, we propose an out–of–sample extension for the KPPR. Since the Green's kernel is the inverse of the β-normalized Laplacian, it is not trivial to calculate out–of–sample values for the kernel. However, as stated in Thm. 1, one can relate the embedding itself directly to the normalized kernel function. In fact, if $(\mathbf{K})_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ is a kernel function, the embedding is almost identical to Kernel PCA on the normalized kernel $\boldsymbol{\mathcal{K}}$, only differing with respect to the dimensional scaling.

Theorem 1. *Let $\mathbf{z}_i \in \mathbb{R}^N$ be a vector with $(\mathbf{z}_i)_j = \frac{1}{\lambda_j^*\sqrt{1+\beta-\lambda_j^*}}\sum_{k=1}^N \mathcal{K}(\mathbf{x}_i, \mathbf{x}_k)(\mathbf{e}_j)_k$, where $\mathbf{x}_k$, $k = 1, 2, \ldots, N$ are in-sample data points, λ_j^* is the j'th largest eigenvalue of $\boldsymbol{\mathcal{K}}$, $\mathbf{e}_j$ is its corresponding eigenvector and $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_k) = \frac{\kappa(\mathbf{x}_i,\mathbf{x}_k)}{\sqrt{d_k}\sqrt{d_i}}$ is the normalized kernel function evaluated at $\mathbf{x}_i$ and $\mathbf{x}_k$. Then $\mathbf{z}_i$ is the embedding of $\mathbf{x}_i$ to the empirical kernel space of $\boldsymbol{\mathcal{G}}_\beta$.*

From Thm. 1, we see that one can compute the embedding of a datapoint to the empirical kernel space of $\boldsymbol{\mathcal{G}}_\beta$ by only considering the normalized kernel function. Importantly, we can compute out–of–sample values of this kernel and use Thm. 1 to define an out–of–sample embedding.

Definition 1 (Out–of–sample kernel)**.** *Let $\mathbf{x}_i$ be an out–of–sample datapoint and let $\mathbf{x}_k$, $k = 1, 2, \ldots, N$ be the in–sample datapoints. Then, the out–of–sample normalized kernel is defined as*

$$\hat{\mathcal{K}}(\mathbf{x}_i, \mathbf{x}_k) = \frac{\kappa(\mathbf{x}_i, \mathbf{x}_k)}{\sqrt{\hat{d}_i d_k}}, \tag{6}$$

where $\hat{d}_i = \sum_{k=1}^N \kappa(\mathbf{x}_i, \mathbf{x}_k)$. A similar definition is given in (Bengio et al., 2004).

Definition 2 (Out–of–sample embedding)**.** *Let $\mathbf{x}_i$ be an out–of–sample datapoint with the out–of–sample normalized kernel $\hat{\mathcal{K}}(\mathbf{x}_i, \mathbf{x}_k)$ as defined in Def. 1, where $\mathbf{x}_k$, $k = 1, 2, \ldots, N$ are in–sample datapoints. Then, the out–of–sample embedding is defined as $\hat{\mathbf{z}}_i \in \mathbb{R}^N$ with*

$$(\hat{\mathbf{z}}_i)_j = \frac{1}{\lambda_j^*\sqrt{1+\beta-\lambda_j^*}}\sum_{k=1}^N \hat{\mathcal{K}}(\mathbf{x}_i, \mathbf{x}_k)(\mathbf{e}_j)_k, \tag{7}$$

where λ_j^* is the j'th largest eigenvalue of the in–sample normalized kernel $\boldsymbol{\mathcal{K}}$ and $\mathbf{e}_j$, its corresponding eigenvector.

Definition 3 (Out–of–sample score)**.** *Let $\mathbf{x}_i$ be an out–of–sample datapoint with the embedding $\hat{\mathbf{z}}_i$ as defined in Def. 2 and let $\mathbf{z}_k$, $k = 1, 2, \ldots, N$ be the in–sample embeddings for the in–sample datapoints $\mathbf{x}_k$. Then the ranking score for $\mathbf{x}_i$ is given by*

$$\hat{\pi}_i(\alpha, \mathbf{s}) = \beta\sqrt{\hat{d}_i}\hat{\mathbf{z}}_i^T\mathbf{m}_\mathbf{s}, \tag{8}$$

where $\mathbf{m}_\mathbf{s} = \sum_{k=1}^N \frac{(\mathbf{s})_k}{\sqrt{d_k}}\mathbf{z}_k$.

The out–of–sample score in Def. 3 is on the same form as the in–sample score as given in Prop. 3. Moreover, by defining the score in this manner, it can be decomposed in the same way as the in–sample score in Prop. 4. See the supplementary material for details.

3.1.3 Dimensionality Reduction/Low–Rank Approximation

While it is possible to use all eigenvectors in the KPPR, one might benefit from discarding a portion of the eigenvectors to create an embedding with low dimensionality, corresponding to a low-rank approximation. It is well known in the general graph literature that low-rank embeddings may reveal relevant structure in the data (Ng et al., 2001; Shi and Malik, 2000). Moreover, since the normalized affinity matrix will often have eigenvalues which are zero, or close to zero, we can deduce from Def. 2 that dimensionality reduction is in fact *essential* for ranking out–of–sample datapoints, since we need to divide by the eigenvalue. In this section, we define a low–rank embedding for the KPPR and propose a novel criterion used to order the eigenvectors based on minimizing an error norm with respect to the PPR. We define our low–rank embedding as follows.

Definition 4. *Let $\mathbf{e}_1, \ldots, \mathbf{e}_k$ be the first k eigenvectors of $\boldsymbol{\mathcal{K}}$ with the associated eigenvalues $\lambda_1^*, \ldots, \lambda_k^*$, where the eigenvalues and eigenvectors are sorted according to some criterion. Then the low–rank KPPR embedding is defined as $\mathbf{Z}_k = \mathbf{E}_k((1+\beta)\mathbf{I} - \mathbf{\Lambda}_k^*)^{-\frac{1}{2}}$, where $\mathbf{E}_k$ is a matrix with $\mathbf{e}_1, \ldots, \mathbf{e}_k$ in its columns and $\mathbf{\Lambda}_k^*$ is a diagonal matrix with $\lambda_1^*, \lambda_2^*, \ldots, \lambda_k^*$ along the main diagonal.*

Minimizing Error Norm by Eigenvector Ordering In dimensionality reduction methods involving eigenvectors, one needs a mechanism to choose which of the eigenvectors to incorporate. Often, this is chosen based on the value of the eigenvalues (see e.g. (Belkin and Niyogi, 2003; Hotelling, 1933; Schölkopf et al., 1997)). The following proposition reveals a link between a scaled score error vector and the eigenvectors that are discarded in the dimensionality reduction, where the scaling is performed for mathematical convenience.

Proposition 5. *Let $\boldsymbol{\pi}(\alpha, \mathbf{s})$ be the exact PPR and $\hat{\boldsymbol{\pi}}(\alpha, \mathbf{s})$ be an estimate of the PPR using k eigenvectors and define the error vector as $\boldsymbol{\epsilon}(\mathbf{s}) = \boldsymbol{\pi}(\alpha, \mathbf{s}) - \hat{\boldsymbol{\pi}}(\alpha, \mathbf{s})$. Then, the squared norm of the scaled error vector $\frac{1}{\beta}\mathbf{D}^{-\frac{1}{2}}\boldsymbol{\epsilon}(\mathbf{s})$ is given by*

$$\left\|\frac{1}{\beta}\mathbf{D}^{-\frac{1}{2}}\boldsymbol{\epsilon}(\mathbf{s})\right\|^2 = \sum_{i=k+2}^{N}\left[\frac{1}{1+\beta-\lambda_i^*}\mathbf{e}_i^T(\mathbf{D}^{-\frac{1}{2}}\mathbf{s})\right]^2. \tag{9}$$

The scaled error $\left\|\frac{1}{\beta}\mathbf{D}^{-\frac{1}{2}}\boldsymbol{\epsilon}(\mathbf{s})\right\|^2$ is minimized by ordering the eigenvectors $\mathbf{e}_2, \mathbf{e}_3, \ldots, \mathbf{e}_{k+1}$ such that $|c_2| \geq |c_3| \geq \ldots \geq |c_{k+1}|$, where $c_i = \frac{1}{1+\beta-\lambda_i^*}\mathbf{e}_i^T(\mathbf{D}^{-\frac{1}{2}}\mathbf{s})$. Note that this result leads to a eigenvector ordering mechanism similar to (Jenssen, 2010) if $\mathbf{s} \propto \mathbf{D}^{\frac{1}{2}}\mathbb{1}$.

Random Queries and Uniform Seeds Given the general result in Prop. 5, we provide specific expressions for the error for useful seed distributions. Namely, we look at random queries and uniform seed distributions over a subset of the dataset. The latter example can for instance be a uniform seed distribution over a cluster in the dataset which can be useful to analyze its content.

We define a query as a ranking of the data with a single datapoint as the seed. Since the expression for the error would depend on the seed point, we assume that a datapoint is chosen as a seed with probability $\frac{1}{N}$. The expression for the *expected* error is given in the following proposition.

Proposition 6. *Let $\mathbf{s} = \mathbb{1}_j$ with probability $p_j = \frac{1}{N}$, where $\mathbb{1}_j$ has value 1 in element j and zeros everywhere else. Then the expected scaled error for this random query is given by*

$$\mathrm{E}_{\mathbf{s}\sim r.q.}\left(\left\|\frac{1}{\beta}\mathbf{D}^{-\frac{1}{2}}\boldsymbol{\epsilon}(\mathbf{s})\right\|^2\right) = \frac{1}{N}\sum_{i=k+2}^{N}\frac{\|\mathbf{D}^{-\frac{1}{2}}\mathbf{e}_i\|^2}{(1+\beta-\lambda_i^*)^2}. \tag{10}$$

Interestingly, Prop. 6 shows that the optimal eigenvector ordering is in fact different compared to the common method, where eigenvectors are ordered by the associated eigenvalues. In the following proposition, we relate the error when the seed is uniform over a subset of the data with the expected error of a random query.

Proposition 7. *Let $\mathcal{C}_j$ be a subset of the dataset with $N_{\mathcal{C}_j}$ datapoints. If the seed distribution is uniform over $\mathcal{C}_j$, then the following holds:*

$$\left\|\frac{1}{\beta}\mathbf{D}^{-\frac{1}{2}}\boldsymbol{\epsilon}(\mathbf{s})\right\|^2 \leq \frac{N}{N_{\mathcal{C}_j}}\mathrm{E}_{\mathbf{s}\sim r.q.}\left(\left\|\frac{1}{\beta}\mathbf{D}^{-\frac{1}{2}}\boldsymbol{\epsilon}(\mathbf{s})\right\|^2\right). \tag{11}$$

In real applications, one might not know the seed/query beforehand or one might want to perform the ranking with several different seeds. Thus, it is necessary to find an eigenvector ordering that works sufficiently well for several seeds. Because of Prop. 6 and Prop. 7, we propose to order the eigenvectors according to $|c_2| \geq |c_3| \geq \ldots \geq |c_{k+1}|$ with $c_i^2 = \frac{\|\mathbf{D}^{-\frac{1}{2}}\mathbf{e}_i\|^2}{(1+\beta-\lambda_i^*)^2}$ for both random queries and rankings where the seed is uniform over a subset of the data. In the former case, this will minimize the expected error. In the latter case, an upper bound of the error will be minimized.

3.1.4 Efficient Score Computation

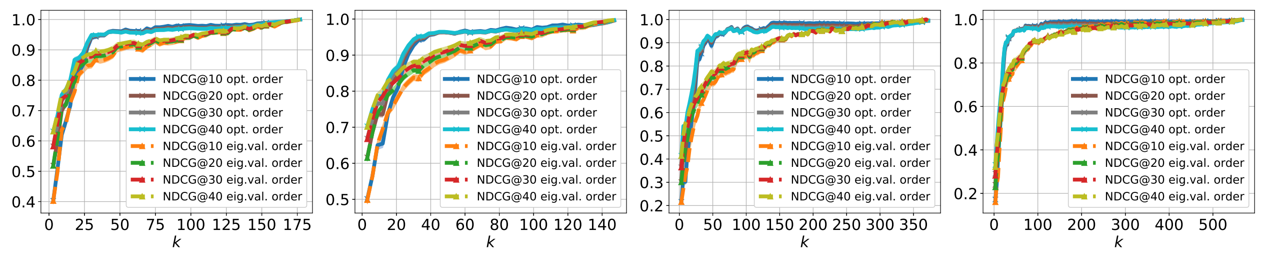
According to the following proposition, once the eigenvectors and eigenvalues are computed, one can compute the score function akin to the representer theorem (Schölkopf et al., 2001).

Proposition 8. *The score function of a datapoint $\mathbf{x}$ is given by*

$$\hat{\pi}(\mathbf{x}|\mathbf{s}) = \sum_{i=1}^{N}\hat{\alpha}_i(\mathbf{s})\kappa(\mathbf{x}, \mathbf{x}_i),$$

where $\hat{\alpha}_i(\mathbf{s}) = \mathbf{m}_\mathbf{s}^T\mathbf{a}_i = \frac{1}{\mathbb{1}^T\mathbf{D}\mathbb{1}} + \mathbf{m}_\mathbf{s}'^T\mathbf{a}_i'$. Here, $\mathbf{m}_\mathbf{s}$ is a weighted mean in the empirical kernel space and $(\mathbf{a}_i)_\ell = \beta\frac{(\mathbf{e}_\ell)_i}{\sqrt{d_i}\lambda_\ell^*\sqrt{1+\beta-\lambda_\ell^*}}$.

Using Prop. 8, the following analysis holds. (i) Given the embedding of the training data $\mathbf{Z}_\beta$, the weighted mean can be computed by $\mathbf{m}_\mathbf{s} = \mathbf{Z}_\beta^T\mathbf{D}^{-\frac{1}{2}}\mathbf{s}$, where $\mathbf{Z}_\beta^T\mathbf{D}^{-\frac{1}{2}} \in \mathbb{R}^{k\times N}$. Thus, if the seed vector is sparse with



(a) (b) (c) (d)

Figure 1: NDCG@k as a function of dimensionality for different eigenvector orderings. (a): Wine, (b): Iris, (c): Two–moon, (d): WDBC

c non–zero elements, computing the weighted mean is $\mathcal{O}(ck)$. (ii) The vectors $\mathbf{a}_1, \mathbf{a}_2, \dots \mathbf{a}_N$ are independent of $\mathbf{x}$ and $\mathbf{s}$, and can be precomputed. (iii) If $\mathbf{m_s}$ is known, computing the coefficients, $\boldsymbol{\alpha} = \mathbf{A}\mathbf{m_s}$ is $\mathcal{O}(Nk)$, where $\mathbf{A}$ is a matrix with $\mathbf{a}_i^T$ as its rows. (iv) If $\mathbf{m_s}$ is unknown, but the seed vector is sparse with c non–zero elements, computing the coefficients, $\boldsymbol{\alpha} = \mathbf{A}\mathbf{Z}_\beta^T\mathbf{D}^{-\frac{1}{2}}\mathbf{s}$ is $\mathcal{O}(Nc)$ provided that the matrix $\mathbf{A}\mathbf{Z}_\beta^T\mathbf{D}^{-\frac{1}{2}} \in \mathbb{R}^{N\times N}$ is precomputed and stored. (v) If the seed is fixed, the coefficients $\hat{\alpha}_i(\mathbf{s})$ can be precomputed and stored such that computing the out–of–sample score is $\mathcal{O}(N)$.

Approximate iterative methods for Personalized PageRank (e.g. (Bahmani et al., 2010)) attempts to maintain scores for a dynamically evolving graph (e.g. social media graphs or web graphs). This is a different setting than ours, where we have a training set consisting of N samples and attempt to approximate scores for out–of–sample data points. Because of this, we do not perform comparisons between the methods in the experiments.

(Bahmani et al., 2010) reports a cost of $\mathcal{O}(\frac{N\log m}{\alpha^2})$ to maintain global PageRank scores when m adversarially chosen edges are added to the graph in random order.

While we are not maintaining scores for existing nodes, computing scores for new nodes with KPPR is $\mathcal{O}(N)$, since the seed is fixed. Furthermore, when using the representer theorem the weights (kernel function values) in the sum is often sparse, such that the effective number of components can potentially be much lower than N.

4 EXPERIMENTS

The experiments are designed for the following purposes: 1. As a sanity check to show that the KPPR provides a good approximation to the exact PPR and that our method for ordering eigenvectors yields a better approximation error than the traditional (order of eigenvalues) method. 2. To show that the KPPR provides meaningful rankings for out–of–sample data, eliminating the need to re–invert matrices or performing new power–iterations. 3. Show that the RKHS approach with low–rank approximations can even yield an improvement in ranking performance compared to the PPR.

We set $\alpha = 0.1$ for all experiments[2]. To show that our theory is applicable regardless of the similarity function used to compute edge weights, we run experiments using both RBF–like functions (sec. 4.3) and the recent Probabilistic Cluster Kernel (PCK) (Izquierdo-Verdiguier et al., 2015) (sec. 4.1–4.2) with default values (not being sensitive to user–specified hyper–parameters). We compare the KPPR to the exact PPR using the Normalized Discounted Cumulative Gain at position k (NDCG@k) (Liu et al., 2009), a ranking metric that is computed by a weighted sum of relevance scores of the top k ranked datapoints. The score of the exact PPR is computed using (2), and is used to provide relevance scores for NDCG@k.

For reproducibility, experiments are performed on well-known UCI benchmark datasets[3], namely Wine, Iris, WDBC and Jain's Two–Moon (Jain and Law, 2005). Moreover, we use a subset of the Caltech–101 dataset (Fei-Fei et al., 2007) and a protein similarity network used in (Weston et al., 2004), described in detail in their respective sections. Additional experiments can be found in the supplementary material.

4.1 Eigenvector Ordering

In this experiment, we verify our method for ordering the eigenvectors. The KPPR embeddings were generated using two different ordering methods: ordering by the result in Prop. 6 and ordering by the eigenvalues.

Fig. 1 shows average NDCG@k, $k = 10, 20, 30, 40$, as a function of the dimensionality of the embedding for

[2]This is chosen somewhat arbitrarily, but is within the range of values used for other PageRank–like methods in the literature.

[3]https://archive.ics.uci.edu/ml/index.php

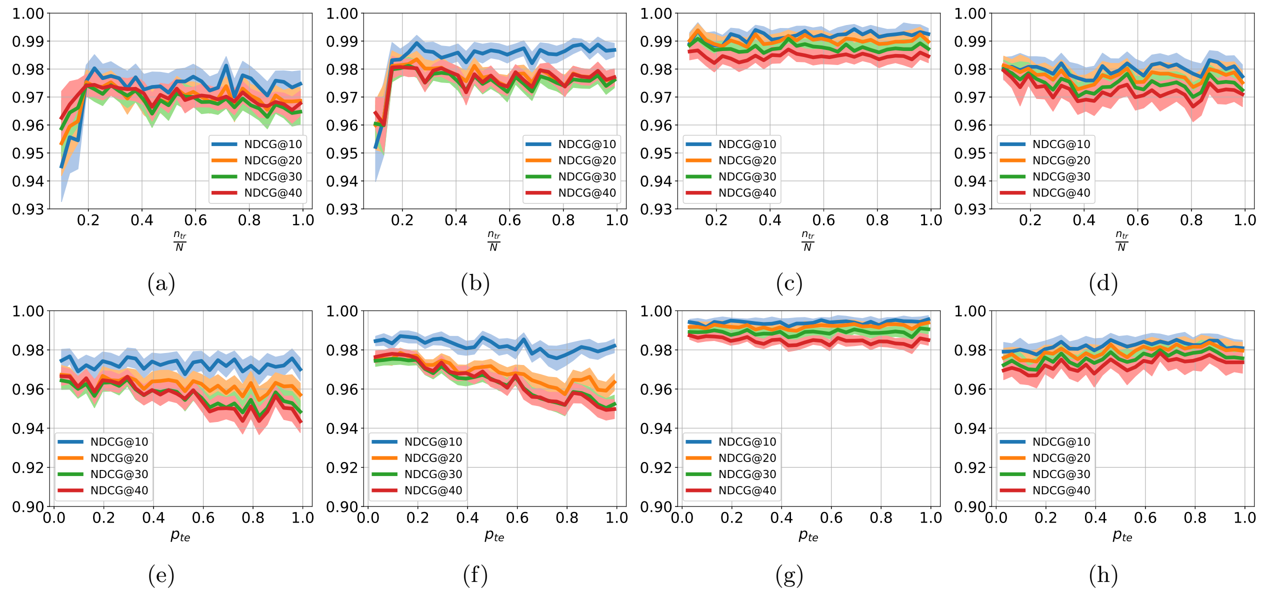


Figure 2: NDCG@k as a function of the training set size with out–of–sample seed (top row) and out–of–sample score with in–sample seed (bottom row) for (a, e): Wine, (b, f): Iris, (c, g): Two–moon, (d, h): WDBC

the benchmark datasets. For each dimensionality, we randomly sampled 100 queries and ranked the data with the two embeddings and using the exact PPR. We observe that in this setting, the embedding generated using Prop. 6 achieves better results relative to the PPR with fewer dimensions compared to the embedding where the eigenvectors are ordered by the eigenvalues.

4.2 Out–of–Sample Ranking

We explore the out–of–sample extension of our method to show that we are able to provide a good approximation of the exact PPR without re–inverting matrices or performing power iterations. We split the experiment in two parts: one where the seed is in the out–of–sample set, and one where the seed is in the in–sample set. The exact PPR is computed on the entire dataset[4]. We choose the dimensionality of our embeddings as the number of eigenvectors whose eigenvalues $\lambda_i^* > 0.01$, since we need to discard eigenvectors with too small eigenvalues to avoid dividing by a small number in Eq. (7).

Out–of–Sample Seed In this experiment, we rank the data with the seed in an out–of–sample dataset. Fig. 2 (top row) shows a 95% confidence interval of NDCG@k as a function of the size of the in–sample set. For each in–sample–set size, 100 different in–sample/out–of–sample sets were randomly sampled. For each random split, we ranked the data using 30 random queries. The seed datapoint was left out of the ranking result. Clearly, the KPPR needs the in–sample set to be of a certain size to get decent results. For the Wine and Iris dataset, an in–sample size larger than around 20% seems to be sufficient to get somewhat stable results. For the other two datasets, the results are good for the whole range of dataset sizes.

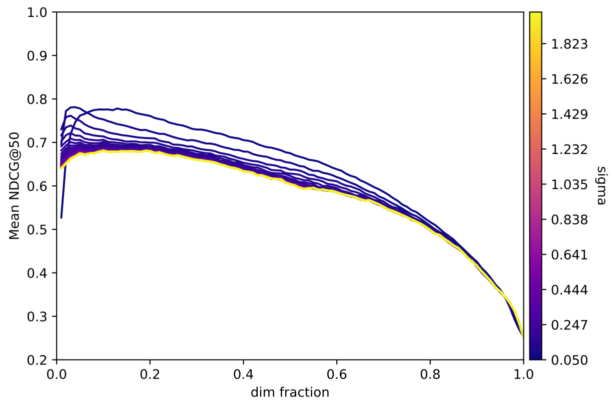
Out–of–Sample Score In this experiment, we let the seed be in the in–sample training set and vary the number of out–of–sample points within the top 40 ranked datapoints. The procedure is as follows: 1. Select a random seed point and calculate the baseline ranking scores using the exact PPR. 2. Select $\sim p_{te} \cdot 40$ random points from the top 40 ranked datapoints to use as an out–of–sample test set. 3. Train the KPPR on the in–sample training set and compute the ranking score for both in–sample and out–of–sample data (discarding the seed point). 4. Compare the ranking to the baseline in step 1. using NDCG@k, $k \in \{10, 20, 30, 40\}$.

Plots of NDGC@k as a function of p_{te} are shown in Fig. 2 (bottom row). We see that for all the datasets, the relevance of the top 10 ranked datapoints is stable. For the Wine and Iris dataset, NDGC@k is decreasing for $k > 10$. However, the top 10 ranked datapoints should still be a good approximation to the exact PPR ranking.

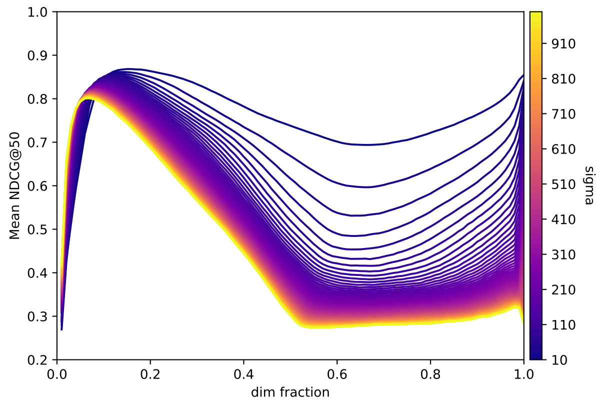
4.3 KPPR Outperforms PPR

In the previous experiments, we showed that the KPPR makes sense relative to the PPR, and that ranking of out-of-sample points are naturally enabled by our

[4]Both in–sample and out–of–sample data



(a)



(b)

Figure 3: NDCG@50 as a function of the number of eigenvectors used set with 100 different values of σ for the (a) Caltech image data (b) SCOP protein data.

RKHS approach. In this section, we show that the KPPR may actually perform better than the PPR relative to a quantitive ranking criterion. However, contrary to before, we will in this experiment examine our method's performance under the oft-encountered setting where the weights are computed by a similarity function in which a user–specified parameter is usually important for good performance. In particular, we construct 100 different graphs for each dataset by generating affinity matrices using similarity functions with 100 different values of the RBF-parameter $\sigma \in [\sigma_{\min}, \sigma_{\max}]$ (see below). We run N queries (one for each datapoint) for each graph and for a varying percentage of eigenvectors used in KPPR (dim fraction $\in \{0.01, 0.02, \ldots, 1.0\}$). Note that the exact PPR is obtained by using all eigenvectors (dim fraction = 1.0). We report mean NDGC@50 for each σ value, for each dim fraction.

The analysis is performed on (i) A subset of the Caltech image dataset and (ii) The SCOP protein network (Weston et al., 2004).

Caltech Images The Caltech image dataset used in this experiment is a subset of the full dataset. In particular, we use all images of transportation vehicles, namely airplanes, cars, helicopters, ferries, motorbikes and wheelchairs. The data is vectorized by generating a bag–of–visual–words representation (Csurka et al., 2004) using SIFT features[5] and clustering the SIFT features into 1024 bins. The edge weights in the graph are calculated using an RBF on the form $(\mathbf{K})_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{1}{2\sigma^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2}$, with 100 σ parameters in the range $[0.05, 2]$. For the purpose of this experiment, we are interested in retrieving objects from the same class as the query. Thus, for query i, datapoint j, the relevance score r_{ij} is set to 1 if $\mathbf{x}_j$ is in the same class as $\mathbf{x}_i$ and zero otherwise. This relevance score is used to calculate NDCG@50, in which case NDCG@50 = 1 if the 50 top ranked points belong to the same class as the query and zero if all the top 50 points belong to the other class.

The results for the Caltech image dataset is shown in Fig. 3a. We see that in this case, KPPR outperforms PPR (dim fraction = 1.0) for all values of σ and independently of the number of dimensions chosen. When choosing the best number of dimensions, we get an average $NDCG@50$ in the range $[0.681, 0.781]$ with the number of eigenvectors in the range $[3, 13]$ percent of the total number of eigenvectors. The best results are obtained with $\sigma = 0.07$, using 4% of the eigenvectors.

The PPR obtains an average $NDCG@50$ in the range $[0.251, 0.253]$. In fact, the PPR performs worse than a randomly ordered list. The analytical expression for the average $NDCG@k$ is given by $\overline{NDCG@k} = \frac{1}{N}\sum_{i=1}^{N} E(NDCG@k_{(i)}) = \sum_{c=1}^{C} P(\omega_c)^2$, where $P(\omega_c)$ is the marginal probability of a datapoint belonging to class c. For details on how to get this expression, see the supplementary material. Using this expression, we get $\overline{NDCG@k} = 0.349$ for a random list in the Caltech dataset, meaning that the PPR performs worse than just ordering the images randomly. However, by discarding most of the eigenvectors, KPPR is able to extract meaningful relationships between datapoints.

SCOP Protein Data We use the protein similarity network from (Weston et al., 2004) to further evaluate our approach. The network is generated by running

[5]Generated using OpenCV and standard parameters.

the PSI–BLAST software on the SCOP database[6] with parameters as described in (Weston et al., 2004), to obtain E values that are used as a *distance* between sequences. The E value between two sequences i and j is then used to weight the edges in a graph, such that the weight between node i and j in the graph is given by $(\mathbf{K})_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{2}\left(e^{-\frac{E_{ij}}{\sigma}} + e^{-\frac{E_{ji}}{\sigma}}\right)$, where σ is a user–specified hyper–parameter. This results in a symmetric affinity matrix. The experiment is performed on the training set from (Weston et al., 2004). Indices for the training examples were obtained from the website of (Weston et al., 2004) using the Wayback Machine[7].

We assign relevance scores between proteins in line with (Weston et al., 2004). Proteins from the same superfamily are assumed to be homologous. Accordingly, we assign a relevance score $r_{ij} = 1$ if proteins i and protein j come from the same superfamily. Furthermore, if proteins i and j are from different folds, they are assumed to be unrelated, such that $r_{ij} = 0$. Protein pairs in the same fold, but different superfamilies, have unknown relationships and are discarded from the evaluation. We evaluate our algorithm with a varying number of discarded eigenvectors and for 100 values of $\sigma \in [10, 1000]$.

Fig. 3 shows mean NDCG@50 for different values of σ and different number of eigenvectors used in KPPR. An interesting observation is that in all cases, there is an optimal number of eigenvectors, in which KPPR outperforms the exact PPR. In other words, there are some components that deteriorates the structures in the embedded data, such that meaningful relevances are not obtainable.

KPPR obtains an average NDCG@50 in the range [0.801, 0.868] when keeping 6%–15% of the eigenvectors. The exact PPR obtains an average NDCG@50 in the range [0.281, 0.855] for the same σ parameters. While the best performance of PPR is nearly as good as the best performance of KPPR when σ is chosen carefully (small σ), KPPR is able to perform well even with a poor σ parameter.

Discussion KPPR outperforming PPR in this setting is a very interesting result. If we study this in terms of Prop. 4, the improved results is a consequence of removing information from the restart component of the score vector. We suspect that by discarding information from the restart component, we are in fact performing some form of de–noising as in the presence of noise, low–rank embeddings on this form have been proven useful for that purpose (Jenssen, 2010; Kwok and Tsang, 2004; Mika et al., 1999). Moreover, it is well known in the graph literature that the eigenvectors used to generate our embedding may reveal structure in the data (Ng et al., 2001; Shi and Malik, 2000). These kind of results are to the authors' best knowledge not found in the current PageRank literature, and will be explored further in future work.

[6]Obtained from http://scop.berkeley.edu/astral/ver=1.59
[7]https://archive.org/web/

5 CONCLUSION

In this paper, we have introduced the Kernel Personalized PageRank based on a novel analysis of the Personalized PageRank from a Reproducing Kernel Hilbert Space perspective. The method requires computing eigenvectors which is expensive. However, we have shown that these interpretations facilitate out–of–sample ranking of previously unseen data without performing new power iterations or re–inverting matrices, allowing for ranking a potentially large dataset using eigenvectors computed on a smaller subset. We provided a method for ordering eigenvectors such that an error criterion is minimized. Experiments show that the KPPR provide a good approximation for the exact PPR, both for in–sample and out–of–sample data. Furthermore, we showed empirically that the low-rank KPPR can actually outperform the PPR.

In the future, we plan on further exploring the empirical kernel space by visualization/geometric interpretations and exploiting the fact that we are able to e.g. perform clustering on the same representation that is used for ranking.

References

Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment*, 4(3):173–184, 2010.

M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

Y. Bengio, O. Delalleau, N. L. Roux, J.-F. Paiement, P. Vincent, and M. Ouimet. Learning Eigenfunctions Links Spectral Embedding and Kernel PCA. *Neural Computation*, 16(10):2197–2219, 2004.

S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, April 1998. ISSN 01697552. doi: 10.1016/S0169-7552(98)00110-X.

F. Chung and W. Zhao. Pagerank and random walks on graphs. *Fete of combinatorics and computer science*, pages 1–16, 2010.

Fan Chung and S.-T. Yau. Discrete Green's Functions.

Journal of Combinatorial Theory, Series A, 91(1-2): 191–214, July 2000. ISSN 00973165.

Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22, 2004.

Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer vision and Image understanding*, 106(1):59–70, 2007.

David F Gleich. Pagerank beyond the web. *SIAM Review*, 57(3):321–363, 2015.

H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417–441, September 1933.

Emma Izquierdo-Verdiguier, Robert Jenssen, Luis Gómez-Chova, and Gustavo Camps-Valls. Spectral clustering with the probabilistic cluster kernel. *Neurocomputing*, 149, Part C(0):1299–1304, 2015.

Anil K Jain and Martin HC Law. Data clustering: A user's dilemma. In *International conference on pattern recognition and machine intelligence*, pages 1–10. Springer, 2005.

R. Jenssen. Kernel Entropy Component Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5):847–860, 2010. ISSN 0162-8828. doi: 10.1109/TPAMI.2009.100.

Y. Jing and S. Baluja. VisualRank: applying PageRank to large-scale image search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11): 1877–90, November 2008. ISSN 1939-3539.

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=H1gL-2A9Ym`.

Isabel M Kloumann, Johan Ugander, and Jon Kleinberg. Block models and personalized pagerank. *Proceedings of the National Academy of Sciences*, 114 (1):33–38, 2017.

JT-Y Kwok and IW-H Tsang. The pre-image problem in kernel methods. *IEEE transactions on neural networks*, 15(6):1517–1525, 2004.

Hang Li. A short introduction to learning to rank. *IEICE TRANSACTIONS on Information and Systems*, 94(10):1854–1862, 2011.

Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

Peter Lofgren. Efficient algorithms for personalized pagerank. *arXiv preprint arXiv:1512.04633*, 2015.

Subhransu Maji, Nisheeth K Vishnoi, and Jitendra Malik. Biased normalized cuts. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2057–2064. IEEE, 2011.

J. Mercer. Functions of Positive and Negative Type, and their Connection with the Theory of Integral Equations. *Philosophical Transactions of The Royal Society*, A(209):415–446, 1909.

C. D. Meyer, S. Race, and K. Valakuzhy. Determining the Number of Clusters via Iterative Consensus Clustering. In *SDM*, pages 94–102. SIAM, 2013. ISBN 978-1-61197-262-7. doi: 10.1137/1.9781611972832.11.

Sebastian Mika, Bernhard Schölkopf, Alex J Smola, Klaus-Robert Müller, Matthias Scholz, and Gunnar Rätsch. Kernel pca and de-noising in feature spaces. In *Advances in neural information processing systems*, pages 536–542, 1999.

A. Y. Ng, M. I. Jordan, and Y. Weiss. On Spectral Clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, pages 849–856, 2001.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International conference on artificial neural networks*, pages 583–588. Springer, 1997.

Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer, 2001.

J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000. ISSN 0162-8828.

Jason Weston, Andre Elisseeff, Dengyong Zhou, Christina S Leslie, and William Stafford Noble. Protein ranking: from local to global structure in the protein similarity network. *Proceedings of the National Academy of Sciences*, 101(17):6559–6563, 2004.

Christopher KI Williams. On a connection between kernel pca and metric multidimensional scaling. *Machine Learning*, 46(1-3):11–19, 2002.

Zexing Zhan, Ruimin Hu, Xiyue Gao, and Nian Huai. Fast incremental pagerank on dynamic networks. In *International Conference on Web Engineering*, pages 154–168. Springer, 2019.

Hongyang Zhang, Peter Lofgren, and Ashish Goel. Approximate personalized pagerank on dynamic graphs.

In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1315–1324. ACM, 2016.

D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf. Ranking on Data Manifolds. *Advances in Neural Information Processing Systems*, 16:169–176, 2004a.

Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. Learning with Local and Global Consistency. *Advances in Neural Information Processing Systems*, 16(16):321–328, 2004b.

Kernel Personalized PageRank: Supplementary Material

Proof of Proposition 1

Proof. Let $\mathbf{e}_i$ be an eigenvector of $\boldsymbol{\mathcal{L}}$ with the corresponding eigenvalue λ_i'. Then the eigenvalue of $\boldsymbol{\mathcal{L}}_\beta$ is given by $\lambda_i = \lambda_i' + \beta$. Since $\boldsymbol{\mathcal{G}}_\beta = \sum_{i=1}^N \frac{1}{\lambda_i}\mathbf{e}_i\mathbf{e}_i^T$, we get

$$\begin{aligned}\mathbf{y}^T\boldsymbol{\mathcal{G}}_\beta\mathbf{y} &= \mathbf{y}^T\sum_{i=1}^N \frac{1}{\lambda_i'+\beta}\mathbf{e}_i\mathbf{e}_i^T\mathbf{y}\\ &= \sum_{i=1}^N \frac{1}{\lambda_i'+\beta}\mathbf{y}^T\mathbf{e}_i\mathbf{e}_i^T\mathbf{y}\\ &= \sum_{i=1}^N \frac{1}{\lambda_i'+\beta}a_i^2,\ a_i = \mathbf{y}^T\mathbf{e}_i\\ &> 0\end{aligned}$$

for $\frac{1}{\lambda_i'+\beta} > 0,\ i = 1, 2, \ldots, N$. Since $\boldsymbol{\mathcal{L}}$ is positive semidefinite, $\lambda_i' \geq 0$. Consequently, $\frac{1}{\lambda_i'+\beta} > 0$ for $\beta = \frac{\alpha}{1-\alpha} > 0$, which is satisfied for $0 < \alpha < 1$. □

Proof of Proposition 2

Proof. Recall that $(\boldsymbol{\mathcal{G}}_\beta)_{ij} = \mathcal{G}_\beta(\mathbf{x}_i, \mathbf{x}_j) = \langle\boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j)\rangle$. By combining this with (2), we get

$$\begin{aligned}\frac{\pi_i(\alpha,\mathbf{s})}{\sqrt{d_i}} &= \beta\sum_{j=1}^N \frac{(\mathbf{s})_j}{\sqrt{d_j}}\langle\boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j)\rangle\\ &= \beta\left\langle\boldsymbol{\phi}(\mathbf{x}_i), \sum_{j=1}^N \frac{(\mathbf{s})_j}{\sqrt{d_j}}\boldsymbol{\phi}(\mathbf{x}_j)\right\rangle \propto \langle\boldsymbol{\phi}(\mathbf{x}_i), \mathbf{w}\rangle,\end{aligned}$$

where $\mathbf{w} = \sum_{j=1}^N \frac{(\mathbf{s})_j}{\sqrt{d_j}}\boldsymbol{\phi}(\mathbf{x}_j)$. □

Proof of Proposition 3

Proof. Given $\mathbf{Z}_\beta = \begin{pmatrix}\mathbf{z}_1(\beta) & \mathbf{z}_2(\beta) & \cdots & \mathbf{z}_N(\beta)\end{pmatrix}^T$ and $\boldsymbol{\pi}(\alpha,\mathbf{s}) = \beta\mathbf{D}^{\frac{1}{2}}\mathbf{Z}_\beta\mathbf{Z}_\beta^T\mathbf{D}^{-\frac{1}{2}}\mathbf{s}$, we have $\mathbf{Z}_\beta^T\mathbf{D}^{-\frac{1}{2}}\mathbf{s} = \sum_{j=1}^N \frac{(\mathbf{s})_j}{\sqrt{d_j}}\mathbf{z}_j(\beta)$. Thus, element i of $\boldsymbol{\pi}(\alpha,\mathbf{s})$ can be computed by $\pi_i(\alpha,\mathbf{s}) = \beta\sqrt{d_i}\sum_{j=1}^N \frac{(\mathbf{s})_j}{\sqrt{d_j}}\mathbf{z}_j(\beta)^T\mathbf{z}_i(\beta)$, or equivalently $\frac{\pi_i(\alpha,\mathbf{s})}{\sqrt{d_i}} \propto \mathbf{w}^T\mathbf{z}_i(\beta)$, where $\mathbf{w} = \sum_{j=1}^N \frac{(\mathbf{s})_j}{\sqrt{d_j}}\mathbf{z}_j(\beta) = \mathbf{m}_s$ is a weighted mean in the empirical kernel space, weighted by the seed distribution. □

Proof of Proposition 4

Proof.

$$\begin{aligned}\boldsymbol{\pi}(\alpha,\mathbf{s}) &= \beta\mathbf{D}^{\frac{1}{2}}\boldsymbol{\mathcal{G}}_\beta\mathbf{D}^{-\frac{1}{2}}\mathbf{s}\\ &= \beta\mathbf{D}^{\frac{1}{2}}\left(\sum_{i=1}^N \frac{1}{\lambda_i}\mathbf{e}_i\mathbf{e}_i^T\right)\mathbf{D}^{-\frac{1}{2}}\mathbf{s}\\ &= \beta\mathbf{D}^{\frac{1}{2}}\left(\frac{1}{\lambda_1}\mathbf{e}_1\mathbf{e}_1^T + \sum_{i=2}^N \frac{1}{\lambda_i}\mathbf{e}_i\mathbf{e}_i^T\right)\mathbf{D}^{-\frac{1}{2}}\mathbf{s}.\end{aligned}$$

Using the fact that $\lambda_1 = 1 + \beta - \lambda_1^* = \beta$ and $\mathbf{e}_1 = \frac{1}{\sqrt{\mathbb{1}^T\mathbf{D}\mathbb{1}}}\mathbf{D}^{\frac{1}{2}}\mathbb{1}$, we get

$$\begin{aligned}\boldsymbol{\pi}(\alpha,\mathbf{s}) &= \beta\frac{1}{\beta}\frac{1}{\mathbb{1}^T\mathbf{D}\mathbb{1}}\mathbf{D}^{\frac{1}{2}}\mathbf{D}^{\frac{1}{2}}\mathbb{1}\mathbb{1}^T\mathbf{D}^{\frac{1}{2}}\mathbf{D}^{-\frac{1}{2}}\mathbf{s}\\ &+ \beta\mathbf{D}^{\frac{1}{2}}\left(\sum_{i=2}^N \frac{1}{\lambda_i}\mathbf{e}_i\mathbf{e}_i^T\right)\mathbf{D}^{-\frac{1}{2}}\mathbf{s}\\ &= \frac{1}{\mathbb{1}^T\mathbf{D}\mathbb{1}}\mathbf{D}\mathbb{1}\mathbb{1}^T\mathbf{s} + \beta\mathbf{D}^{\frac{1}{2}}\boldsymbol{\mathcal{G}}_\beta{}'\,\mathbf{D}^{-\frac{1}{2}}\mathbf{s} = \boldsymbol{\pi} + \boldsymbol{\pi}'(\alpha,\mathbf{s}),\end{aligned}$$

since $\mathbb{1}^T\mathbf{s} = 1$. □

Proof of Theorem 1

Proof. Consider the in–sample embedding $\mathbf{Z}_\beta = \mathbf{E}\boldsymbol{\Lambda}^{-\frac{1}{2}}$, where $\mathbf{E}$ is the orthogonal eigenvector matrix of $\boldsymbol{\mathcal{L}}_\beta$ and $\boldsymbol{\Lambda}$ is its diagonal eigenvalue matrix. Recall that the normalized kernel matrix $\boldsymbol{\mathcal{K}}$ has the same eigenvectors, but with shifted eigenvalues $\boldsymbol{\Lambda}^* = (1+\beta)\mathbf{I} - \boldsymbol{\Lambda}$, implying $\boldsymbol{\mathcal{K}} = \mathbf{E}\boldsymbol{\Lambda}^*\mathbf{E}^T$, where $\mathbf{E}^T\mathbf{E} = \mathbf{I}$. Assuming that ${\boldsymbol{\Lambda}^*}^{-1}$ exists, we get

$$\begin{aligned}\mathbf{Z}_\beta = \mathbf{E}\boldsymbol{\Lambda}^{-\frac{1}{2}} &= \mathbf{E}\boldsymbol{\Lambda}^*\mathbf{E}^T\mathbf{E}{\boldsymbol{\Lambda}^*}^{-1}\boldsymbol{\Lambda}^{-\frac{1}{2}} = \boldsymbol{\mathcal{K}}\mathbf{E}{\boldsymbol{\Lambda}^*}^{-1}\boldsymbol{\Lambda}^{-\frac{1}{2}}\\ &= \boldsymbol{\mathcal{K}}\mathbf{E}{\boldsymbol{\Lambda}^*}^{-1}\left[(1+\beta)\mathbf{I} - \boldsymbol{\Lambda}^*\right]^{-\frac{1}{2}}\end{aligned}$$

The result follows. □

Out–of–sample decomposition

Let $\mathbf{x}_i$ be an out–of–sample datapoint with out–of–sample embedding $\hat{\mathbf{z}}_i$ as defined in Def. 2 and out–of–sample score $\hat{\pi}_i(\alpha, \mathbf{s})$ according to Def. 3. Then

$$\hat{\pi}_i(\alpha, \mathbf{s}) = \bar{\pi}_i + \beta\sqrt{\hat{d}_i}\hat{\mathbf{z}}_i'^{\,T}\mathbf{m_s}',$$

where $\hat{\mathbf{z}}_i'$ is the out–of–sample embedding of $\mathbf{x}_i$ when discarding the first dimension (trivial eigenvector), $\bar{\pi}_i \equiv \frac{\hat{d}_i}{\mathbb{1}^T\mathbf{D}\mathbb{1}}$ and $\hat{d}_i$ is defined as in Def. 1

Proof. The out–of–sample score is given by

$$\begin{aligned}\hat{\pi}_i(\alpha, \mathbf{s}) &= \beta\sqrt{\hat{d}_i}\hat{\mathbf{z}}_i^T\mathbf{m_s}\\ &= \beta\sqrt{\hat{d}_i}\hat{z}_{i,1}m_{\mathbf{s},1} + \beta\sqrt{\hat{d}_i}\hat{\mathbf{z}}_i'^{\,T}\mathbf{m_s}',\end{aligned}$$

where $\hat{z}_{i,1}$ and $m_{\mathbf{s},1}$ are the contributions from the trivial eigenvector $\mathbf{e}_1 = \frac{1}{\sqrt{\mathbb{1}^T\mathbf{D}\mathbb{1}}}\mathbf{D}^{\frac{1}{2}}\mathbb{1}$ for $\hat{\mathbf{z}}_i$ and $\mathbf{m_s}$ respectively. Note that the first eigenvalue is given by $\lambda_1^* = 1$. Using these values in Def. 2 yields

$$\begin{aligned}\hat{z}_{i,1} &= \frac{1}{\sqrt{\beta}}\sum_{k=1}^{N}\frac{\kappa(\mathbf{x}_i, \mathbf{x}_k)}{\sqrt{\hat{d}_i d_k}}\sqrt{\frac{d_k}{\mathbb{1}^T\mathbf{D}\mathbb{1}}}\\ &= \frac{1}{\sqrt{\beta\hat{d}_i\mathbb{1}^T\mathbf{D}\mathbb{1}}}\sum_{k=1}^{N}\kappa(\mathbf{x}_i, \mathbf{x}_k)\\ &= \sqrt{\frac{\hat{d}_i}{\beta\mathbb{1}^T\mathbf{D}\mathbb{1}}},\end{aligned}$$

since $\hat{d}_i = \sum_{k=1}^{N}\kappa(\mathbf{x}_i, \mathbf{x}_k)$ according to Def. 1.Furthermore,

$$\begin{aligned}\hat{m}_{i,1} &= \sum_{j=1}^{N}\frac{(\mathbf{s})_j}{\sqrt{d_j}}z_{i,1}\\ &= \sum_{j=1}^{N}\frac{(\mathbf{s})_j}{\sqrt{d_j}}\frac{1}{\sqrt{\beta}}\sum_{k=1}^{N}\frac{\kappa(\mathbf{x}_j, \mathbf{x}_k)}{\sqrt{d_j d_k}}\sqrt{\frac{d_k}{\mathbb{1}^T\mathbf{D}\mathbb{1}}}\\ &= \frac{1}{\sqrt{\beta\mathbb{1}^T\mathbf{D}\mathbb{1}}}\sum_{j=1}^{N}\frac{(\mathbf{s})_j}{d_j}\sum_{k=1}^{N}\kappa(\mathbf{x}_j, \mathbf{x}_k)\\ &= \frac{1}{\sqrt{\beta\mathbb{1}^T\mathbf{D}\mathbb{1}}}\sum_{j=1}^{N}(\mathbf{s})_j\\ &= \frac{1}{\sqrt{\beta\mathbb{1}^T\mathbf{D}\mathbb{1}}},\end{aligned}$$

since $d_j = \sum_{k=1}^{N}\kappa(\mathbf{x}_j, \mathbf{x}_k)$ and $\sum_{j=1}^{N}(\mathbf{s})_j = 1$. Finally, the contribution from the first eigenvector is given by

$$\begin{aligned}\beta\sqrt{\hat{d}_i}\hat{z}_{i,1}m_{\mathbf{s},1} &= \beta\sqrt{\hat{d}_i}\sqrt{\frac{\hat{d}_i}{\beta\mathbb{1}^T\mathbf{D}\mathbb{1}}}\frac{1}{\sqrt{\beta\mathbb{1}^T\mathbf{D}\mathbb{1}}}\\ &= \frac{\hat{d}_i}{\mathbb{1}^T\mathbf{D}\mathbb{1}}\\ &\equiv \bar{\pi}_i.\end{aligned}$$

The result follows. □

Proof of Proposition 5

Proof.

$$\begin{aligned}\left\|\frac{1}{\beta}\mathbf{D}^{-\frac{1}{2}}\boldsymbol{\epsilon}\right\|^2 &= \Bigg\|\frac{1}{\beta}\mathbf{D}^{-\frac{1}{2}}\boldsymbol{\pi} + \sum_{i=2}^{N}\frac{1}{\lambda_i}\mathbf{e}_i\mathbf{e}_i^T\mathbf{D}^{-\frac{1}{2}}\mathbf{s}\\ &\quad - \left(\frac{1}{\beta}\mathbf{D}^{-\frac{1}{2}}\boldsymbol{\pi} + \sum_{i=2}^{k+1}\frac{1}{\lambda_i}\mathbf{e}_i\mathbf{e}_i^T\mathbf{D}^{-\frac{1}{2}}\mathbf{s}\right)\Bigg\|^2\\ &= \left\|\sum_{i=k+2}^{N}\frac{1}{\lambda_i}\mathbf{e}_i\mathbf{e}_i^T\mathbf{D}^{-\frac{1}{2}}\mathbf{s}\right\|^2 = \left\|\sum_{i=k+2}^{N}c_i\mathbf{e}_i\right\|^2,\end{aligned}$$

where $c_i = \frac{1}{\lambda_i}\mathbf{e}_i^T\mathbf{D}^{-\frac{1}{2}}\mathbf{s}$. Then,

$$\begin{aligned}\left\|\sum_{i=k+2}^{N}c_i\mathbf{e}_i\right\|^2 &= \sum_{i=k+2}^{N}c_i\mathbf{e}_i^T\sum_{j=k+2}^{N}c_j\mathbf{e}_j\\ &= \sum_{i=k+2}^{N}\sum_{j=k+2}^{N}c_ic_j\mathbf{e}_i^T\mathbf{e}_j = \sum_{i=k+2}^{N}c_i^2\\ &= \sum_{i=k+2}^{N}\left[\frac{1}{\lambda_i}\mathbf{e}_i^T(\mathbf{D}^{-\frac{1}{2}}\mathbf{s})\right]^2,\end{aligned}$$

since $\mathbf{e}_i^T\mathbf{e}_j = 0,\ i \neq j$. □

Proof of Proposition 6

Proof. Let $\mathbb{1}_j$ be a vector with 1 in element j and zeros everywhere else. If datapoint $\mathbf{x}_j$ is chosen as a query with probability $p_j = \frac{1}{N}$, then the expected error norm is given by

$$\begin{aligned}&\mathrm{E}_{\mathbf{s}\sim\text{r.q.}}\left(\left\|\frac{1}{\beta}\mathbf{D}^{-\frac{1}{2}}\boldsymbol{\epsilon}(\mathbf{s})\right\|^2\right) = \sum_{j=1}^{N}p_j\left\|\frac{1}{\beta}\mathbf{D}^{-\frac{1}{2}}\boldsymbol{\epsilon}(\mathbb{1}_j)\right\|^2\\ &= \sum_{j=1}^{N}\frac{1}{N}\sum_{i=k+2}^{N}\left[\frac{1}{\lambda_i}\mathbf{e}_i^T(\mathbf{D}^{-\frac{1}{2}}\mathbb{1}_j)\right]^2\\ &= \frac{1}{N}\sum_{i=k+2}^{N}\frac{1}{\lambda_i^2}\sum_{j=1}^{N}\left(\frac{(\mathbf{e}_i)_j}{\sqrt{d_j}}\right)^2 = \frac{1}{N}\sum_{i=k+2}^{N}\frac{\|\mathbf{D}^{-\frac{1}{2}}\mathbf{e}_i\|^2}{\lambda_i^2}.\end{aligned}$$

□

Proof of Proposition 7

Proof. Let the seed be given by $\mathbf{s} = \frac{1}{N_{\mathcal{C}_j}} \mathbb{1}_{\mathcal{C}_j}$, where $\mathbb{1}_{\mathcal{C}_j}$ is a vector with value 1 in element ℓ if $\mathbf{x}_\ell \in \mathcal{C}_j$ and zero otherwise. Then

$$\left\| \frac{1}{\beta} \mathbf{D}^{-\frac{1}{2}} \boldsymbol{\epsilon} \right\|^2 = \sum_{i=k+2}^{N} \left[\frac{1}{\lambda_i} \mathbf{e}_i^T \mathbf{D}^{-\frac{1}{2}} \frac{1}{N_{\mathcal{C}_j}} \mathbb{1}_{\mathcal{C}_j} \right]^2 = \frac{1}{N_{\mathcal{C}_j}^2} \sum_{i=k+2}^{N} \frac{1}{\lambda_i^2} \langle \mathbf{D}^{-\frac{1}{2}} \mathbf{e}_i, \mathbb{1}_{\mathcal{C}_j} \rangle^2,$$

where $\langle \cdot, \cdot \rangle$ denotes an inner product. By the triangle inequality, we have

$$\langle \mathbf{D}^{-\frac{1}{2}} \mathbf{e}_i, \mathbb{1}_{\mathcal{C}_j} \rangle^2 \leq \|\mathbf{D}^{-\frac{1}{2}} \mathbf{e}_i\|^2 \|\mathbb{1}_{\mathcal{C}_j}\|^2 = N_{\mathcal{C}_j} \|\mathbf{D}^{-\frac{1}{2}} \mathbf{e}_i\|^2.$$

The result follows. □

Proof of Proposition 8

Proof. Let $\mathbf{z}$ be the image of $\mathbf{x}$ in the empirical kernel space of $\boldsymbol{\mathcal{G}}_\beta$. Recall that $(\mathbf{z})_j = \frac{1}{\lambda_j^* \sqrt{1+\beta-\lambda_j^*}} \sum_{k=1}^{N} \hat{\mathcal{K}}(\mathbf{x}, \mathbf{x}_k)(\mathbf{e}_j)_k$. The PPR of $\mathbf{x}$ is given by

$$\pi(\mathbf{x}|\mathbf{s}) = \hat{\pi} + \beta \sqrt{d(\mathbf{x})} \mathbf{z}'^T \mathbf{m}'_\mathbf{s},$$

where $\hat{\pi} = \frac{d(\mathbf{x})}{\mathbb{1}^T \mathbf{D} \mathbb{1}}$, $d(\mathbf{x}) = \sum_{i=1}^{N} \kappa(\mathbf{x}, \mathbf{x}_i)$ is the degree of $\mathbf{x}$, $\mathbf{z}'$ and $\mathbf{m}'_s$ are vectors in the empirical kernel space where the first component is discarded. Then

$$\beta \sqrt{d(\mathbf{x})} \mathbf{z}'^T \mathbf{m}'_\mathbf{s} = \beta \sqrt{d(\mathbf{x})} \sum_{j=2}^{k+1} \frac{1}{\lambda_j^* \sqrt{1+\beta-\lambda_j^*}} \sum_{i=1}^{N} \hat{\mathcal{K}}(\mathbf{x}, \mathbf{x}_i)(\mathbf{e}_j)_i (\mathbf{m}_\mathbf{s})_j.$$

Since $\hat{\mathcal{K}}(\mathbf{x}, \mathbf{x}_i) = \frac{\kappa(\mathbf{x}, \mathbf{x}_i)}{\sqrt{d(\mathbf{x})}\sqrt{d_i}}$, we get

$$\beta \sqrt{d(\mathbf{x})} \mathbf{z}'^T \mathbf{m}'_\mathbf{s} = \sum_{i=1}^{N} \left[\sum_{j=2}^{k+1} \beta \frac{(\mathbf{e}_j)_i}{\sqrt{d_i} \lambda_j^* \sqrt{1+\beta-\lambda_j^*}} (\mathbf{m}_\mathbf{s})_j \right] \kappa(\mathbf{x}, \mathbf{x}_i) = \sum_{i=1}^{N} \mathbf{m}'^T_\mathbf{s} \mathbf{a}'_i \kappa(\mathbf{x}, \mathbf{x}_i),$$

where $(\mathbf{a}_i)_j = \beta \frac{(\mathbf{e}_j)_i}{\sqrt{d_i} \lambda_j^* \sqrt{1+\beta-\lambda_j^*}}$. Finally, substituting this in the PPR score, we get

$$\begin{aligned} \pi(\mathbf{x}|\mathbf{s}) &= \hat{\pi} + \beta \sqrt{d(\mathbf{x})} \mathbf{z}'^T \mathbf{m}'_\mathbf{s} \\ &= \frac{d(\mathbf{x})}{\mathbb{1}^T \mathbf{D} \mathbb{1}} + \sum_{i=1}^{N} \mathbf{m}'^T_\mathbf{s} \mathbf{a}'_i \kappa(\mathbf{x}, \mathbf{x}_i) \\ &= \frac{\sum_{i=1}^{N} \kappa(\mathbf{x}, \mathbf{x}_i)}{\mathbb{1}^T \mathbf{D} \mathbb{1}} + \sum_{i=1}^{N} \mathbf{m}'^T_\mathbf{s} \mathbf{a}'_i \kappa(\mathbf{x}, \mathbf{x}_i) \\ &= \sum_{i=1}^{N} \left[\frac{1}{\mathbb{1}^T \mathbf{D} \mathbb{1}} + \mathbf{m}'^T_\mathbf{s} \mathbf{a}'_i \right] \kappa(\mathbf{x}, \mathbf{x}_i) \\ &= \sum_{i=1}^{N} \hat{\alpha}_i(\mathbf{s}) \kappa(\mathbf{x}, \mathbf{x}_i), \end{aligned}$$

where $\hat{\alpha}_i(\mathbf{s}) = \frac{1}{\mathbb{1}^T \mathbf{D} \mathbb{1}} + \mathbf{m}'^T_\mathbf{s} \mathbf{a}'$. This concludes the proof. □

Expected NDCG@k

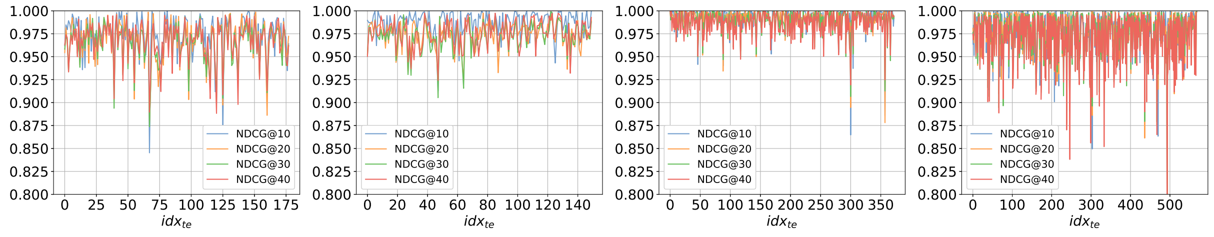
In this section, we derive the average expected NDCG@k for a dataset with class structures, with a randomly ordered list. For simplicity, we assume that $k \leq \min(N_1, N_2, \ldots, N_C)$, where N_c, $c = 1, 2, \ldots, C$ are the number of datapoints in class ω_c. Additionally, we assume that the relevance of datapoint $\mathbf{x}_j$ in query i is given by $r_{ij} = 1$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ belong to the same class and $r_{ij} = 0$ otherwise. The expected NDCG@k for query i is given by

$$E(NDCG@k_{(i)}) = E\left(\frac{DCG@k_{(i)}}{DCG@k_{\text{opt}}} \right).$$

Let $r_i(k')$ be the relevance of the datapoint in position k' in an ordered list for query i, i.e. $r_i(k') = r_{ij}$ if $\mathbf{x}_j$ is in position k' in the ordered list. Since $k \leq \min(N_1, N_2, \ldots, N_C)$ the optimal $DCG@k$ is given by $DCG@k_{\text{opt}} = \sum_{k'=1}^{k} \frac{r_i(k')^{\text{opt}}}{\log_2(k'+1)} = \sum_{k'=1}^{k} \frac{1}{\log_2(k'+1)}$, which is constant. Then,

$$\begin{aligned} E(NDCG@k_{(i)}) &= \frac{1}{DCG@k_{\text{opt}}} E(DCG@k_{(i)}) \\ &= \frac{1}{DCG@k_{\text{opt}}} \sum_{k'=1}^{k} \frac{E(r_i(k'))}{\log_2(k'+1)}. \end{aligned}$$

For a randomly ordered list, any datapoint can come at any position in the list and $r_i(k'), k' = 1, 2, \ldots, N$



(a) (b) (c) (d)

Figure 1: Leave–one–out NDCG@k as a function of the test/seed index. (a): Wine, (b): Iris, (c): Two–moon, (d): WDBC

Table 1: Leave–one–out results

	NDGC@10		NDGC@20		NDGC@30		NDGC@40	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Wine	0.973	0.023	0.967	0.023	0.965	0.021	0.967	0.021
Iris	0.987	0.013	0.975	0.015	0.973	0.015	0.976	0.015
Two moons	0.993	0.013	0.992	0.012	0.989	0.012	0.987	0.012
WDBC	0.979	0.025	0.976	0.025	0.973	0.026	0.970	0.028

are i.i.d. The theorem of double expectation yields

$$\begin{aligned} E(r_i(k')) &= E(E(r_i(k')|\omega(\mathbf{x}_i))) \\ &= \sum_{c=1}^{C} P(\mathbf{x}_i \in \omega_c) E(r_i(k')|\mathbf{x}_i \in \omega_c) \\ &= \sum_{c=1}^{C} P(\mathbf{x}_i \in \omega_c)(1 \cdot P(\mathbf{x}_{k'} \in \omega_c) \\ &+ \sum_{c' \neq c} 0 \cdot P(\mathbf{x}_{k'} \in \omega_{c'})) \\ &= \sum_{c=1}^{C} P(\omega_c)^2, \end{aligned}$$

where $P(\omega_c), c = 1, 2, \ldots, C$ are the marginal probabilities of a datapoint belonging to class ω_c. Notice that this is independent of the query. Thus,

$$\begin{aligned} E(NDCG@k_{(\mathrm{i})}) &= \frac{1}{DCG@k_{\mathrm{opt}}} \sum_{k'=1}^{k} \frac{\sum_{c=1}^{C} P(\omega_c)^2}{\log_2(k'+1)} \\ &= \sum_{c=1}^{C} P(\omega_c)^2 \frac{DCG@k_{\mathrm{opt}}}{DCG@k_{\mathrm{opt}}} \\ &= \sum_{c=1}^{C} P(\omega_c)^2. \end{aligned}$$

Thus, the expected $NDCG@k$ is given by

$$E(E(NDCG@k_{(i)})) = \sum_{c=1}^{C} P(\omega_c)^2.$$

Additional experiments

Out–of–sample, leave–one–out results

In addition to the experiments in sec. 4.3 in the main paper, we ran a leave–one–out like scheme, where we computed the in–sample embedding based on all datapoints except one. We then used our out–of–sample extension to compute the embedding for this datapoint, and ranked the data using the left–out datapoint as a seed. Fig. 1 shows NDCG@k as a function of the datapoint which was left out and Tab. 1 shows the mean and standard deviation of NDGC@k. We see that our method is, on average, able to retrieve relevant objects, especially in the top 10 results. However, there are examples of datapoints where NDCG@k is low as seen in Fig. 1. These datapoints might be outliers that are located far away from the other datapoints in input space.

Ranking and clustering: Seal image data

In this experiment, we perform ranking and clustering using the seal image data.

The dataset contains aerial RGB images of hooded and harp seal pups. Vectorial representations of the images were generated by feeding the images through a neural network using the AlexNET architecture (Krizhevsky et al., 2012) and storing the input to the classification layer. Using publically available pretrained weights[1],

[1]Trained on the ImageNET database

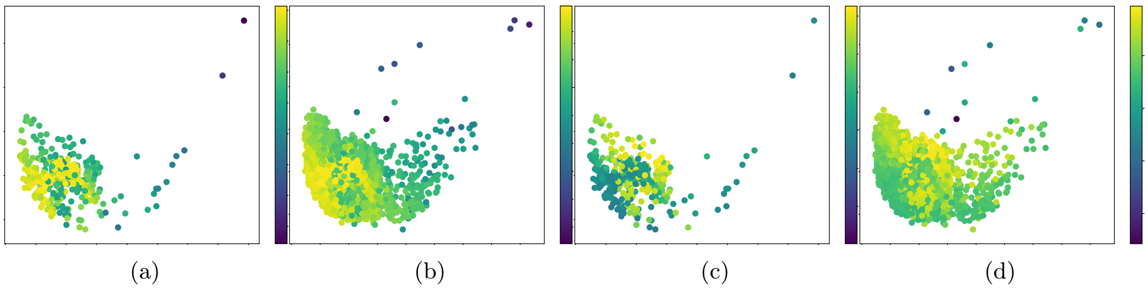


Figure 2: In–sample (left) and out–of–sample (right) embedding for the seals dataset with uniform seed over in–sample data in (a)–(b) cluster 1, (c)–(d) cluster 2. Color indicates ranking score.

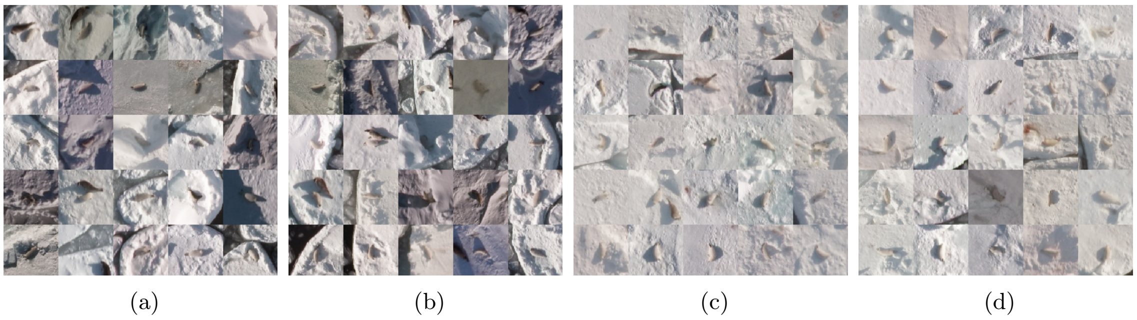


Figure 3: Top ranked images from in–sample (left) and out–of–sample (right) data with a uniform seed over in–sample data in (a)–(b) cluster 1, (c)–(d) cluster 2.

we finetuned the network on the seal images. The data used in our experiments are independent on the data used to finetune the network. For more information on the data, see e.g. (Kampffmeyer et al., 2019).

We used a small subset of the data to train the PCK (100 datapoints) and split the remainder of the dataset into an in–sample and out–of–sample set with 500 and 6312 datapoints, respectively. Because of the high dimensionality of the input data (4096), we reduced the dimensionality of the vectorial data to 100 prior to training the PCK.

Using our method, we transformed the in–sample data into a 5 dimensional representation and clustered the data using the k–means clustering algorithm with $k = 2$. The remaining data was transformed using our out–of–sample extension.

Using a uniform seed over each of the clusters, we ranked both in–sample and out–of–sample data. Fig. 2 shows the first two dimensions of the KPPR representation of the data. The left column corresponds to the in–sample data, while the right column corresponds to the out–of–sample data. Each row corresponds to a ranking of the data using one of the two seeds. The color of a data point corresponds to its score (yellow=high, blue=low). This plot shows clearly that the out–of–sample embedding holds the general shape of the in–sample embedding.

Fig. 3 shows the top ranked images using both seeds[2]. Other than noticing that the top ranked out–of–sample images look similar to the top ranked in–sample images, one can notice that the top ranked images from the different seeds are indeed very different. Using this ranking information, we see that one cluster seems to contain clean images of white seals. The other cluster seems to contain images of black seals and images that are "contaminated" by shadows or water.

References

Michael Kampffmeyer, Sigurd Løkse, Filippo M. Bianchi, Lorenzo Livi, Arnt-Børre Salberg, and Robert Jenssen. Deep divergence-based approach to clustering. *Neural Networks*, 113:91 – 101, 2019. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2019.01.015. URL `http://www.sciencedirect.com/science/article/pii/S0893608019300292`.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2]The layout is equal to the one in Fig. 2

Paper III

The deep kernelized autoencoder

Michael Kampffmeyer, Sigurd Løkse, Filippo M. Bianchi, Robert Jenssen and Lorenzo Livi.

Applied Soft Computing, 2018

Applied Soft Computing 71 (2018) 816–825

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

The deep kernelized autoencoder

Michael Kampffmeyer[a,*], Sigurd Løkse[a], Filippo M. Bianchi[a], Robert Jenssen[a,b], Lorenzo Livi[c]

[a] *Machine Learning Group, UiT – The Arctic University of Norway, Norway*[1]
[b] *Norwegian Computing Center, Oslo, Norway*
[c] *Department of Computer Science, University of Exeter, UK*

ARTICLE INFO

Article history:
Received 15 February 2018
Received in revised form 4 June 2018
Accepted 7 July 2018
Available online 18 July 2018

Keywords:
Autoencoders
Kernel methods
Deep learning
Representation learning

ABSTRACT

Autoencoders learn data representations (codes) in such a way that the input is reproduced at the output of the network. However, it is not always clear what kind of properties of the input data need to be captured by the codes. Kernel machines have experienced great success by operating via inner-products in a theoretically well-defined reproducing kernel Hilbert space, hence capturing topological properties of input data. In this paper, we enhance the autoencoder's ability to learn effective data representations by aligning inner products between codes with respect to a kernel matrix. By doing so, the proposed *kernelized* autoencoder allows learning similarity-preserving embeddings of input data, where the notion of similarity is explicitly controlled by the user and encoded in a positive semi-definite kernel matrix. Experiments are performed for evaluating both reconstruction and kernel alignment performance in classification tasks and visualization of high-dimensional data. Additionally, we show that our method is capable to emulate kernel principal component analysis on a denoising task, obtaining competitive results at a much lower computational cost.

1. Introduction

Autoencoders (AEs) are a class of neural networks that gained increasing interest in recent years [53,30,44,51,28]. AEs are used for unsupervised learning of *effective* latent representations of data [18,4]. However, what an *effective* representation consists of is highly dependent on the target task, such as clustering and classification [5]. In standard AEs, representations are derived by training the network to reconstruct inputs through either a bottleneck layer, thereby forcing the network to learn how to compress inputs, or through an over-complete representation. It can be shown that training autoencoders using a reconstruction error corresponds to maximizing the lower bound of the mutual information between input and the learned representation [53]. Regularization methods are commonly employed for enforcing sparseness, improving robustness to noise, avoiding trivial identity mappings, or penalizing sensitivity of the representation to small changes in inputs [5]. Nonetheless, regularization alone provides limited control over the nature of the hidden representation.

In this paper, we propose a method to learn representations that preserve desired similarities in input space with an AE. In our approach, similarities are encoded in form of a kernel matrix, which is used as a prior to be reproduced by inner products of the hidden representations learned by the AE. This allows us to learn data representations with specified pairwise relationships. The training loss minimizes a combination of reconstruction error and a term quantifying the misalignment of the prior and the inner products of the hidden representations; the misalignment is computed by means of the normalized Frobenius norm. We note that this process acts as a regularization for the hidden representations and resembles the well-known kernel alignment procedure [54]. Our contribution is in principle related to other well-established methods like those from the family of multidimensional scaling [7], where an explicit embedding of the data is computed by minimizing a measure of distortion based on inner products. Further, we will experimentally show that the proposed regularization method allows mitigating a problem often observed in non-regularized AEs, where codes for similar images are not similar themselves and the underlying manifold is disconnected [37].

1.1. Related works

The proposed model, called *deep kernelized autoencoder*, is related to recent attempts to incorporate kernel and information

* Corresponding author.
E-mail address: michael.c.kampffmeyer@uit.no (M. Kampffmeyer).
[1] http://site.uit.no/ml/.

https://doi.org/10.1016/j.asoc.2018.07.029
1568-4946/© 2018 Elsevier B.V. All rights reserved.

theoretic learning methods within neural network architectures [55,9]. Specifically, it is connected to works on interpreting neural networks from a kernel perspective [39] and the Information Theoretic-Learning Auto-Encoder [44], which imposes a prior distribution over the hidden representation in a variational autoencoder [30]. Achille and Soatto [1] proposed a regularization method exploiting information dropout, an information-theoretic generalization of dropout [48] for neural networks and show that an AE trained with such a regularization for a specific parameter setting simplifies to the variational autoencoder objective. Other information-theoretic learning concepts, such as the information bottleneck [49], have also recently emerged in the deep learning literature [47]. In [2] variational inference is used to optimize the lower bound on the information bottleneck to learn representations that maximize the mutual information between learned representation and output while minimizing the mutual information between input and hidden representation. Computing the information bottleneck is difficult, especially with high-dimensional data. Chalk et al. [8] proposed an efficient variational scheme for maximizing a lower bound of the original information bottleneck formulation, which also allows for non-linear mappings between input and compressed representation via kernel functions. Beside dimensionality reduction, neural networks utilizing kernel and information theoretic concepts have also been used to perform clustering [26].

In our work, we exploit kernel alignment to match the inner products of the learned representations with a similarity measure in the input space encoded as a kernel matrix. A recent related work in this direction by Horn and Müller [21] attempts to learn representations that preserve pairwise similarity by means of AEs. The authors specifically focus on dimensionality reduction, showing the possibility to approximate the pairwise data similarity in input space in linear fashion from the learned low-dimensional representation. In practice, given an input data point, the network is trained to recreate the related row of the similarity matrix. Recently, Chu and Cai [10] propose a similarity-preserving AE based on clustering data in input space. Hidden representations are learned in such a way that data points belonging to the same cluster are similar also in the hidden representation.

Another recent approach consists in integrating Wasserstein Generative Adversarial Neural Networks into the AE framework [28]. Similarly, Tolstikhin et al. [51] propose the Wasserstein Autoencoder, which is based on a novel regularization technique minimizing the Wasserstein distance between the model distribution and a target distribution.

1.2. Contribution and paper organization

In addition to providing more control over hidden representations, our method also has several benefits that compensate for important drawbacks of traditional kernel methods. By means of an end-to-end training procedure, we learn an explicit approximate mapping function from the input to a kernel space, as well as the associated back-mapping to the input space. Once the mapping is learned, it can be applied to inputs and operations performed in kernel space can then be explicitly simulated by means of linear operations in code space, thus in practice allowing to perform non-linear operations in input space. Mini-batch training is used in the proposed method in order to lower the computational complexity inherent to traditional kernel methods and, especially, spectral methods [45,6,24]. Furthermore, our method can be used with arbitrary kernel functions, even those computed with an algorithmic procedure, i.e., where inner products in kernel space are not expressed by an analytic function. To stress this fact, in our experiments we consider the probabilistic cluster kernel (PCK), a kernel function that is the result of a feature generation procedure. PCK is robust with respect to hyperparameter choices and has been shown to often outperform counterparts such as the radial basis function (RBF) kernel [23].

A preliminary version of this method appeared in [25]. Here we extend our work by:

- providing a thorough literature background discussion, placing our work into a broader context;
- extending the experimental evaluation to additional datasets, namely (i) the image dataset CIFAR-10, (ii) the text dataset Reuters, and (iii) the remote sensing dataset Cloud;
- experimentally analyzing the effectiveness of the learned representations for classification tasks and visualizing high-dimensional data, and for generating new data samples beyond those seen during training.

The paper is structured as follows. Section 2 provides the reader with a discussion of the relevant background, such as AEs and kernel methods; notably in Section 2.3 we introduce PCK, adopted here for obtaining kernel matrices to be used in our method. Section 3 describes the proposed methodology. Experimental results are discussed in Section 4 and Section 5. Finally, Section 6 draws conclusions and points to future research directions.

2. Background

2.1. Autoencoders and stacked autoencoders

AEs simultaneously learn two functions. The first one, the *encoder*, provides a mapping from an input domain, X, to a code domain, C, i.e., the hidden representation. The second function, the *decoder*, maps from C back to X. For a single hidden layer AE, the encoding function $E(\cdot)$ and the decoding function $D(\cdot)$ are defined as

$$\begin{aligned}\mathbf{h} &= E(\mathbf{x}) = \sigma(\mathbf{W}_E\mathbf{x} + \mathbf{b}_E)\\ \tilde{\mathbf{x}} &= D(\mathbf{h}) = \sigma(\mathbf{W}_D\mathbf{h} + \mathbf{b}_D),\end{aligned} \tag{1}$$

where $\sigma(\cdot)$ denotes a suitable transfer function (e.g., a sigmoid applied component-wise), $\mathbf{x}$, $\mathbf{h}$, and $\tilde{\mathbf{x}}$ denote, respectively, a sample from the input space, its hidden representation also called *code*, and its reconstruction; finally, $\mathbf{W}_E$ and $\mathbf{W}_D$ are the weights, and $\mathbf{b}_E$ and $\mathbf{b}_D$ the bias of encoder and decoder, respectively.

In order to minimize the discrepancy between the original data and its reconstruction, model parameters in Eq. (1) are learned by minimizing, usually through stochastic gradient descent (SGD), a reconstruction loss of the form

$$L_r(\mathbf{x}, \tilde{\mathbf{x}}) = \|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2. \tag{2}$$

Differently from Eq. (1), a stacked autoencoder (sAE) consists of several hidden layers [18]. Deep architectures are capable of learning complex representations by transforming input data through multiple layers of nonlinear processing [5]. The optimization of the weights is harder in this case and pretraining is beneficial, as it is often easier to learn intermediate representations, instead of training the whole architecture end-to-end [4]. A common application of pre-trained sAE is the initialization of layers in deep neural networks [53]. Pretraining is performed in different phases, each of which consists of training a single AE layer. After the first AE has been trained, its encoding function $E(\cdot)$ is kept fixed and is applied to the input and the resulting representation is used to train the next AE in the stacked architecture. Each layer, being trained independently, aims at capturing more abstract features by trying to reconstruct the representation in the previous layer. Once all individual AEs are trained, their hidden layers (encoding and decoding functions) are extracted and stacked on each other, yielding a pre-trained sAE.

2.2. A brief introduction to kernel methods

Kernel methods process data in a reproducing kernel Hilbert space (RKHS) K associated with an input space X through an implicit (non-linear) mapping $\phi : X \to K$. There, data are more likely to become separable by linear methods [11], which produces results that are otherwise only obtainable by nonlinear operations in the input space. Explicit computation of the mapping $\phi(\cdot)$ and its inverse $\phi^{-1}(\cdot)$ is, in practice, not required. In fact, operations in the kernel space are expressed through inner products (kernel trick), which are computed as Mercer kernel functions in input space: $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle$.

As a major drawback, kernel methods scale poorly with the number of samples n: traditionally, memory requirements of these methods scale with $O(n^2)$ and computation with $O(n^2 \times d)$, where d is the input dimension [13]. For example, kernel principal component analysis (kPCA) [45], a common dimensionality reduction technique that projects data into the subspace that preserves the maximal amount of variance in kernel space, requires to compute the eigendecomposition of a kernel matrix $\mathbf{K} \in \mathbb{R}^{n\times n}$, with $K_{ij} = \kappa(x_i, x_j), x_i, x_j \in X$, with computational and memory costs scaling as $O(n^3)$ and $O(n^2)$, respectively. For this reason, kPCA is not applicable to large-scale problems. The availability of efficient (approximate) mapping functions, however, would reduce the complexity, thereby enabling these methods to be applicable to larger datasets [9]. In this direction, Rahimi and Recht [41] and Vedaldi and Zisserman [52] proposed approximate mappings preserving the dot product structure by using low-dimensional randomized features, hence allowing the use of fast linear methods in an explicit way. Furthermore, finding an explicit inverse mapping from K to the input domain is a central problem in several applications, such as image denoising performed with kPCA, also known as the pre-image problem [3,20].

Our proposed method instead, attempts to approximate the operations in the kernel space using an AE architecture that scales to large datasets, provides an implicit inverse mapping, and, once trained, can process new samples efficiently.

2.3. Probabilistic cluster kernel

The Probabilistic Cluster Kernel (PCK) [23] is a robust kernel function, which automatically adapts to the inherent structures in the data. Its robustness comes from the fact that it does not depend on any critical user-specified hyperparameters, like the width in Gaussian kernels. The PCK is trained by fitting multiple Gaussian Mixture Models (GMMs) to the input data using the EM algorithm and combining these models to generate a single kernel. In particular, GMMs are trained using different number of mixture components $g = 2, 3, \ldots, G$, each with different randomized initial conditions $q = 1, 2, \ldots, Q$. Let $\boldsymbol{\pi}_i(q, g)$ denote the *posterior distribution* for data point $\mathbf{x}_i$ under a GMM with g mixture components and initial condition q. The PCK is then defined as

$$\kappa_{\text{PCK}}(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{Z}\sum_{q=1}^{Q}\sum_{g=2}^{G}\boldsymbol{\pi}_i^T(q, g)\boldsymbol{\pi}_j(q, g), \tag{3}$$

where Z is a normalizing constant.

Intuitively, the posterior distribution under a mixture model contains probabilities that a given data point belongs to a certain mixture component in the model. Thus, the inner products in Eq. (3) are large if data pairs often belong to the same mixture component. By averaging these inner products over a range of g values, the kernel function has a large value if these data points are similar on both global scale (small $g \to$ large mixture components) and local scale (large $g \to$ small mixture components).

The PCK has previously been used for semi-supervised learning [22] and spectral clustering [23]. Additionally, variations of the method for handling missing data have been proposed for both time series [38] and vectorial data [35].

3. Deep kernelized autoencoders

In this section, we describe our contribution, which is a method combining deep AEs with kernel methods: the deep kernelized AE (dkAE). A dkAE is trained by minimizing the following loss function

$$L = (1 - \lambda)L_r(\mathbf{x}, \tilde{\mathbf{x}}) + \lambda L_c(\mathbf{C}, \mathbf{P}), \tag{4}$$

where $L_r(\cdot, \cdot)$ is the reconstruction loss in Eq. (2). $L_c(\cdot, \cdot)$ is the code loss, a distance measure between two matrices, $\mathbf{P} \in \mathbb{R}^{n\times n}$, the kernel matrix given as prior, and $\mathbf{C} \in \mathbb{R}^{n\times n}$, the inner product matrix of codes associated to the input data. The objective of $L_c(\cdot, \cdot)$ is to enforce the similarity between $\mathbf{C}$ and the kernel matrix $\mathbf{P}$. λ is a hyperparameter ranging in [0, 1], which weights the importance of the two objectives in Eq. (4); for $\lambda = 0$, the loss function simplifies to the traditional AE loss in Eq. (2). A depiction of the training procedure is reported in Fig. 1.

We implement $L_c(\cdot, \cdot)$ as the normalized Frobenius distance between $\mathbf{C}$ and $\mathbf{P}$. Each matrix element C_{ij} in $\mathbf{C}$ is given by $C_{ij} = E(\mathbf{x}_i) \cdot E(\mathbf{x}_j)$ and the code loss is computed as

$$L_c(\mathbf{C}, \mathbf{P}) = \left\| \frac{\mathbf{C}}{\|\mathbf{C}\|_F} - \frac{\mathbf{P}}{\|\mathbf{P}\|_F} \right\|_F. \tag{5}$$

It is worth noting that minimizing the normalized Frobenius distance between the kernel matrices is equivalent to maximizing the traditional kernel alignment cost, since

$$\left\| \frac{\mathbf{C}}{\|\mathbf{C}\|_F} - \frac{\mathbf{P}}{\|\mathbf{P}\|_F} \right\|_F = \sqrt{2 - 2A(\mathbf{C}, \mathbf{P})}, \tag{6}$$

where $A(\mathbf{C}, \mathbf{P}) = \frac{\langle \mathbf{C}, \mathbf{P}\rangle_F}{\|\mathbf{C}\|_F\|\mathbf{P}\|_F}$ is exactly the kernel alignment cost function [12,54]. Note that the distance in Eq. (6) can be implemented also with more advanced differentiable measures of (dis)similarity between positive-definite matrices, such as divergence and mutual information [32,14]. However, these options are not explored in this paper and are left for future research.

In this paper, the prior kernel matrix $\mathbf{P}$ is computed by means of the PCK algorithm introduced in Section 2.3, such that $\mathbf{P} = \mathbf{K}_{\text{PCK}}$. However, our approach is general and *any* kernel matrix can be used as prior in Eq. (5).

Note, that the kernel alignment also acts as a regularization, discouraging the learning of trivial mappings. Furthermore, we also employ tied weights in the encoder and decoder as additional regularization following [27].

3.1. Mini-batch training

We use mini batches of k samples to train the dkAE, thereby avoiding the computational restrictions of kernel and especially spectral methods outlined in Section 2.2. In particular, the memory complexity of the algorithm can be reduced to $O(k^2)$, where $k \ll n$. Finally, we note that the computational complexity scales linearly with regard to the network parameters. Given a mini batch of k samples, the dkAE loss function is defined by taking the average of the per-sample reconstruction cost

$$L_{\text{batch}} = \frac{1 - \lambda}{kd}\sum_{i=1}^{k} L_r(\mathbf{x}_i, \tilde{\mathbf{x}}_i) + \lambda \left\| \frac{\mathbf{C}_k}{\|\mathbf{C}_k\|_F} - \frac{\mathbf{P}_k}{\|\mathbf{P}_k\|_F} \right\|_F, \tag{7}$$

where d is the dimensionality of the input space, $\mathbf{P}_k$ is a subset of $\mathbf{P}$ that contains only the k rows and columns related to the current

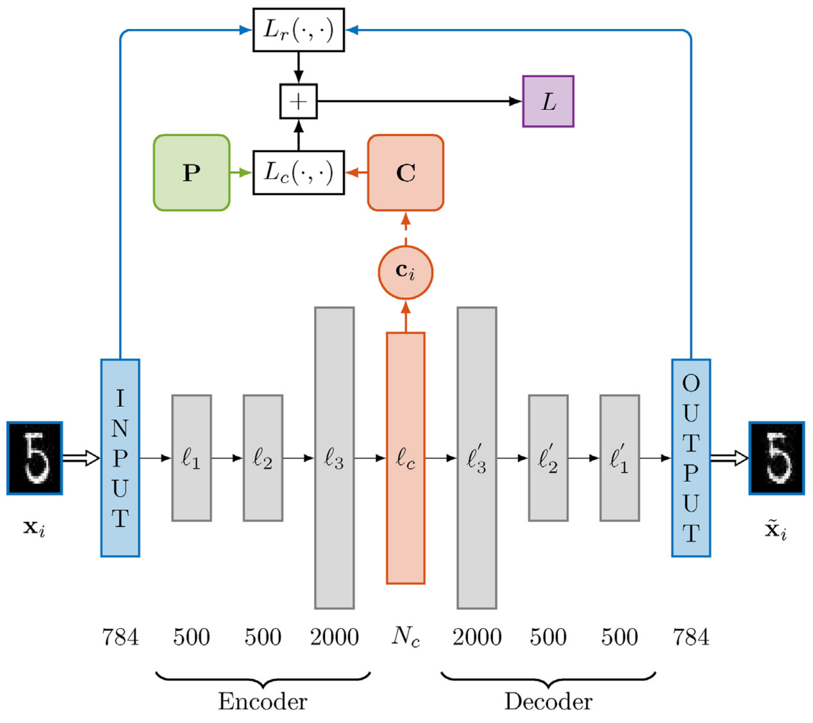


Fig. 1. Schematic illustration of dkAE architecture. Loss function L depends on two terms. First, $L_r(\cdot, \cdot)$, is the reconstruction error between the true input $\mathbf{x}_i$ and the output of the dkAE, $\tilde{\mathbf{x}}_i$. The second term, $L_c(\cdot, \cdot)$, is the distance measure between matrices $\mathbf{C}$ (computed as inner products of codes $\{\mathbf{c}_i\}_{i=1}^n$) and the target prior kernel matrix $\mathbf{P}$. For mini-batch training the matrix $\mathbf{C}$ is computed over the codes of the data in the mini-batch and that distance is compared to the submatrix of $\mathbf{P}$ related to the current mini-batch.

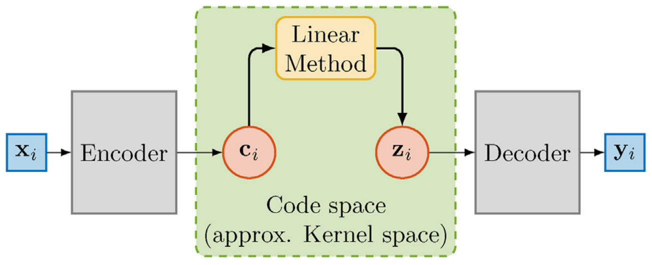


Fig. 2. The encoder maps input $\mathbf{x}_i$ to $\mathbf{c}_i$, which lies in code space. In dkAEs, the code domain approximates the space associated to the prior kernel $\mathbf{P}$. A linear method receives input $\mathbf{c}_i$ and produces output $\mathbf{z}_i$. The decoder maps $\mathbf{z}_i$ back to input space. The result $\mathbf{y}_i$ can be seen as the output of a non-linear operation on $\mathbf{x}_i$ in input space.

mini-batch, and $\mathbf{C}_k$ contains the inner products of the codes related to the mini-batch. Note that $\mathbf{C}_k$ is re-computed for each mini batch ($O(k^2)$), while $\mathbf{P}_k$ is obtained by means of indexing operations with cost $O(k)$.

3.2. *Operations in code space*

Linear operations in code space can be performed as shown in Fig. 2. The encoding scheme of the proposed dkAE implicitly approximates $\phi(\cdot)$, mapping an input $\mathbf{x}_i$ onto the kernel space. In particular, in dkAEs, the feature vector $\phi(\mathbf{x}_i)$ is approximated by the code $\mathbf{c}_i$. Our non-linear encoder maps the inputs into a space where they are more likely to be linearly separable, as there the code vectors preserve a non-linear similarity computed in the input space. A linear operation on $\mathbf{c}_i$ produces a result in the code space, $\mathbf{z}_i$, relative to the input $\mathbf{x}_i$. Unlike other kernel methods where the explicit mapping back to the input space is not defined, we can map codes back by means of a decoder, which in our case approximates the inverse mapping $\phi(\cdot)^{-1}$ from the kernel space back to the input domain. This enables dkAEs to provide visualization and interpretation of the results in the original space; we further explore these perspectives in the experiments.

4. Analysis of dkAE

In this section, we perform an analysis of the proposed method by considering three experiments. Section 4.1 delineates the experimental setting. In Section 4.2, we evaluate the sensitivity of the two terms in the objective function (Eq. (7)) when varying the λ hyperparameter (in Eq. (4)) and the size of the code layer (i.e., number of neurons in the innermost hidden layer). Successively, in Section 4.3 we evaluate the reconstruction accuracy and kernel alignment performance implemented by dkAEs. Further, in Section 4.4 we compare dkAEs approximation accuracy of the prior kernel matrix with kPCA as the number of retained principal components increases.

4.1. *Experimental setting*

The analysis is performed on the MNIST dataset, which consists of 60,000 handwritten digit images [33]. We use a subset of 20,000 samples due to the computational restrictions imposed by the PCK, which we use to illustrate dkAEs ability to learn arbitrary kernels, even if they originate from an ensemble procedure.

We train PCK by fitting GMMs on a subset of 200 training samples using parameters $Q = G = 30$. These parameters are sufficiently large to ensure robust results [35]. Once trained, the GMM models are applied to the remaining data to calculate the whole kernel matrix. We use 70%, 15% and 15% of the data for training, validation, and testing, respectively.

The network architecture used in the experiments is d–500–500–2000–N_c (see Fig. 1), which has been demonstrated to perform well on several datasets, including MNIST, for both supervised and unsupervised tasks [36,19]. Here, N_c refers to the dimensionality of the code layer. Training was performed using

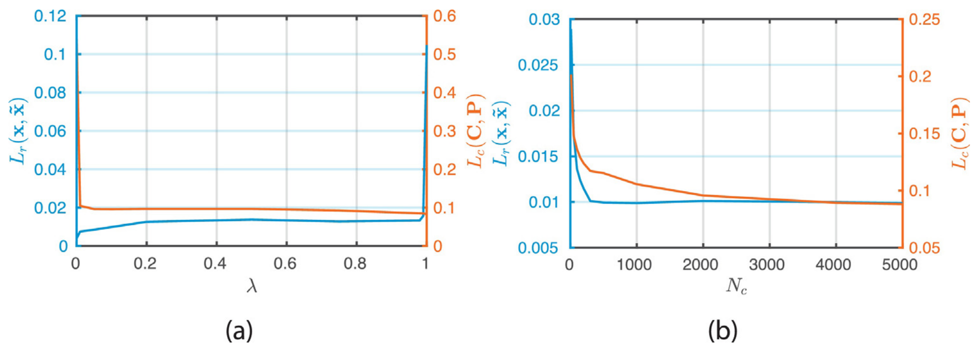


Fig. 3. (a) Tradeoff when choosing λ. High λ values result in low L_c, but high reconstruction cost, and vice versa. (b) Both L_c and reconstruction costs decrease when code dimensionality N_c increases.

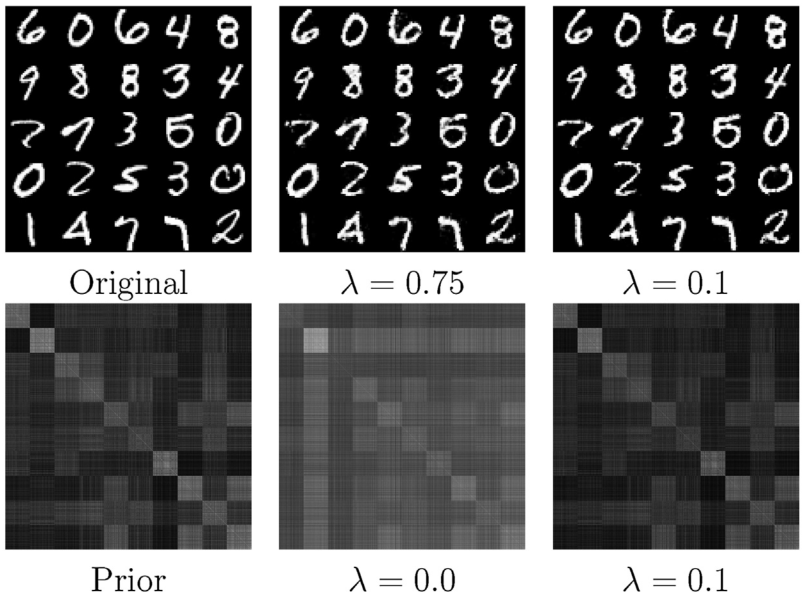


Fig. 4. Illustrating the reconstruction error and kernel alignment trade-off in for different λ values. We note that the reconstruction for a small λ is generally better (see also Fig. 3(a)), but that small λ yields high L_c.

the sAE pretraining approach outlined in Section 2.1. To avoid learning the identity mapping on each individual layer, we applied a common [27] regularization technique where the encoder and decoder weights are tied, i.e., $W_E = W_D^T$. This is done during pretraining and fine-tuning. Unlike in traditional sAEs, to account for the kernel alignment objective, the code layer is optimized according to Eq. (4) *also* during pretraining.

Size of mini-batches for training was chosen to be $k = 200$ randomly, independently sampled data points; in our experiments, an epoch consists of processing $(n/k)^2$ batches. Pretraining is performed for 30 epochs per layer and the final architecture is fine-tuned for 100 epochs using gradient descent based on Adam [29]. The dkAE weights are randomly initialized according to Glorot et al. [15].

4.2. Sensitivity analysis of hyperparameter λ and size N_c of code layer

Here, we evaluate the influence of the two main hyperparameters influencing the resulting model. Note that the experiments shown in this section are performed by training the dkAE on the training set and evaluating the performance on the validation set. We evaluate both the out-of-sample reconstruction L_r and L_c. This is done in order to select the optimal parameters for evaluating the test set in the successive experiments. Fig. 3(a) illustrates the effect of λ for a fixed value $N_c = 2000$ of neurons in the code layer. It can be observed that the reconstruction loss L_r increases as more and more focus is put on minimizing L_c (obtained by increasing λ). This quantifies empirically the trade-off in optimizing the reconstruction performance and the kernel alignment at the same time. By inspecting the results, specifically the near constant losses for λ in range [0.1, 0.9] the method appears robust to changes in hyperparameter λ.

Analyzing the effect of varying N_c given a fixed $\lambda = 0.1$ (Fig. 3(b)), we observe that both losses decrease as N_c increases. This could suggest that an even larger architecture, characterized by more layers and more neurons w.r.t. the architecture adopted here might work well, as the dkAE does not seem to overfit; due also to the regularization effect provided by the kernel alignment.

4.3. Reconstruction error and kernel alignment

By considering the previous results, in the following experiments we set $\lambda = 0.1$ and $N_c = 2000$. Fig. 4 illustrates the results in Section 4.2 qualitatively by displaying a set of original images from our test set and their reconstruction error for the chosen λ value and a non-optimal one. Similarly, the prior kernel (rows/columns sorted by class in the figure, to ease the visualization) and the dkAEs

Table 1
We compute L_c with respect to the ideal kernel matrix $\mathbf{K}_I$ for our test dataset (10 classes) and compare the relative improvement for the three kernels in Fig. 4. It can be seen that the kernel matrix produced by dkAE (**C**) is quantitatively comparable to the prior kernel (**P**) with regard to its distance from the ideal kernel matrix and outperforms the traditional sAE ($\mathbf{K}_{AE}$).

Kernel	Improvement [%] vs.			$L_c(\cdot, \mathbf{K}_I)$
	P	$\mathbf{K}_{AE}$	**C**	
P	0	12.7	−0.2	1.0132
$\mathbf{K}_{AE}$	−11.3	0	−11.4	1.1417
C	0.2	12.9	0	1.0115

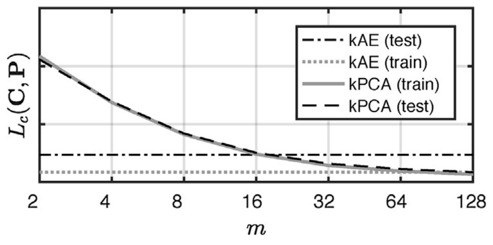


Fig. 5. Comparing dkAEs approximation of the kernel matrix to kPCA for an increasing number of components. The plot shows that dkAE reconstruction is more accurate for low number (i.e., $m < 16$) of components.

approximated kernel matrices, relative to test data, are displayed for two different λ values. Note that, to illustrate the difference to a traditional sAE, one of the two λ values is set to zero. It can be clearly seen that, for $\lambda = 0.1$, both the reconstruction error and kernel matrix closely resemble the original, which agrees with the plots in Fig. 3(a).

Inspecting the kernels obtained in Fig. 4, we compare the distance between the kernel matrices, **C** and **P**, and the ideal kernel matrix, obtained by considering supervised information. We build the ideal kernel matrix $\mathbf{K}_I$, where $K_I(i, j) = 1$ if elements i and j belong to same class, otherwise $K_I(i, j) = 0$. Table 1 illustrates that the kernel approximation produced by dkAE outperforms a traditional sAE with regard to kernel alignment with the ideal kernel. Additionally, it can be seen that the kernel approximation **C** actually is more similar to the ideal kernel than the kernel prior, which we hypothesize is due to the reconstruction objective, which allows the codes to capture additional information (w.r.t. to PCK) about the structure of the input space.

4.4. Approximation of kernel matrix given as prior

In order to quantify the kernel alignment performance, we compare dkAE to the approximation provided by kPCA when varying the number of retained principal components. For this test, we take the kernel matrix **P** of the training set and compute its eigendecomposition. We then select an increasing number of components m (with $m \geq 1$ components related to the largest eigenvalues) to project the input data as follows: $\mathbf{Z}_m = \mathbf{E}_m \boldsymbol{\Lambda}_m^{1/2}$, $d = 2, \ldots, N$. The approximation of the original kernel matrix (prior) is then given by $\mathbf{K}_m = \mathbf{Z}_m \mathbf{Z}_m^T$. We compute the distance between $\mathbf{K}_m$ and **P** following Eq. (6) and compare it to the dissimilarity between **P** and **C**. For evaluating the out-of-sample performance, we use the Nyström approximation for kPCA [45] and compare it to the dkAE kernel approximation on the test set.

Fig. 5 shows that the approximation obtained by means of dkAEs achieves a more accurate reconstruction then kPCA when using a small number of components, i.e., $m < 16$. Note that it is common in spectral methods to chose a number of components equal to the number of classes in the dataset [40], in which case, for the 10 classes in MNIST, dkAE would outperform kPCA. As expected, when the number of selected components increases, the approximation provided by kPCA is better. However, as shown in the previous experiment (Section 4.3), this does not mean that the approximation performs better with regard to the ideal kernel. In fact, in that experiment the kernel approximation of dkAE actually performed at least as well as the prior kernel (kPCA with all components taken into account).

5. Applications of dKAEs in classification, denoising, and visualization of high-dimensional data

In this section, we evaluate the effectiveness of dkAEs learned representations on multiple tasks. In Section 5.1, we compare classification performance on different benchmarks and illustrate how dkAEs can be used also for visualization of high-dimensional data. In Section 5.2, we present an application of our method for image denoising, where we apply PCA in dkAE code space C to remove noise.

For our classification experiments, apart from MNIST, we consider also the following datasets:

- CIFAR-10, which consists of 60,000 32 × 32 color images belonging to 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck) [31]. Similar to the MNIST dataset, we consider a subset of 20,000 samples.
- Cloud, a dataset containing three multispectral satellite images captured over Spain and France. Each pixel in the images is represented by 19 dimensions, where 13 dimensions represent spectral bands from the MEdium Resolution Imaging Spectrometer (MERIS) instrument on board the Environmental Satellite (ENVISAT) [42], while the remaining six dimensions are related to physical features [16]. Each pixel is labeled according to the presence of a cloud in that particular area. This is a binary classification task, where the goal is to identify areas in the image which are obscured by clouds. The dataset is identical to the one previously used in [17]. Similar to the MNIST dataset, we consider a subset of 20,000 samples, where the training set consists of pixels sampled from one image, the validation set is sampled from a different image and the test set is sampled from the remaining image.
- Reuters, which consists of 800,000 news stories that have been manually categorized into a category tree [34]. Similar to [56] we choose the four root categories as labels and remove stories that are labeled with multiple root categories. To represent each news story we compute feature vectors consisting of the Term Frequency-Inverse Document Frequency (TF-IDF) of the 2000 most frequently occurring word stems and then use a Singular Value Decomposition (SVD) to produce 20 dimensional vectors prior to training. The SVD is performed on the training set, with out-of-sample transformations for validation and test sets.

5.1. Visualization and classification in code space

In order to evaluate the learned representation and illustrate the use of our method on an independent test set, we evaluate the classification performance of the learned representation. Here we make use of a linear support vector machine (SVM) operating in the code space and compare it to a linear and a non-linear kernel SVM (kSVM) operating directly in input space. The dKAE is trained on the training dataset, the SVMs model parameters are optimized on the validation set and the final accuracy is shown on the test dataset. Table 1 shows that linear SVM trained in the code space (cSVM) outperforms the SVM models operating in input space on all datasets.

Table 2
Quantitative analysis of the learned feature representation of dkAE for classification tasks. A linear SVM operating in code space (cSVM) is compared with a linear SVM and a kernel SVM (kSVM) operating directly in input space. We also considered a linear SVM operating in code space where the prior **P** for the alignment is given by the outer product of class labels (scSVM).

Method	MNIST	CLOUD	CIFAR-10	Reuters
SVM	90.60	99.50	36.60	91.40
kSVM	93.80	99.60	36.93	93.23
cSVM	94.80	99.63	38.17	93.77
scSVM	96.23	99.70	42.73	94.17

As a consequence of the fact that our code representation is controlled by an arbitrary kernel matrix, we can also extend our work to learn representations in a supervised manner by aligning the code matrix **C** with the ideal kernel matrix. Similar to the experiments for the unsupervised representation, we train a linear SVM in the code space representation that has been learned (by exploiting supervised information) and provide the achieved accuracy (scSVM) in Table 2. As expected, when exploiting supervised information to learn representations, improvements are observed for all datasets. To illustrate the robustness of our approach with respect to architectural choices, we make use of the same architecture for all datasets, namely the one described in Section 4. Note, however, to avoid overfitting to the training data when training the supervised representation for the CLOUD dataset, the architecture for this particular dataset was reduced to d–50–50–200–200 for all experiments.

Now we assess the capability to visualize high-dimensional data. Fig. 6 shows the visualization of a low-dimensional representation learned by dkAE for the MNIST dataset; here, we consider 2000-dimensional codes. We utilize PCA to map the learned codes to two-dimensional vectors. We take into account also the low-dimensional representation learned by four alternative methods, namely an autoencoder without the use of kernel alignment, a denoising autoencoder (DAE) [53] with 20% masking noise, and kernel entropy component analysis (KECA) [24] as well as ISOMAP [50], two popular non-linear dimensionality reduction methods. Note, that the visualization here is presented for the test set and not the training data. For KECA we utilize an RBF kernel with σ being set to 15 percent of the median pairwise euclidean distances between data points, following a rule of thumb from [24]. We use KECA to

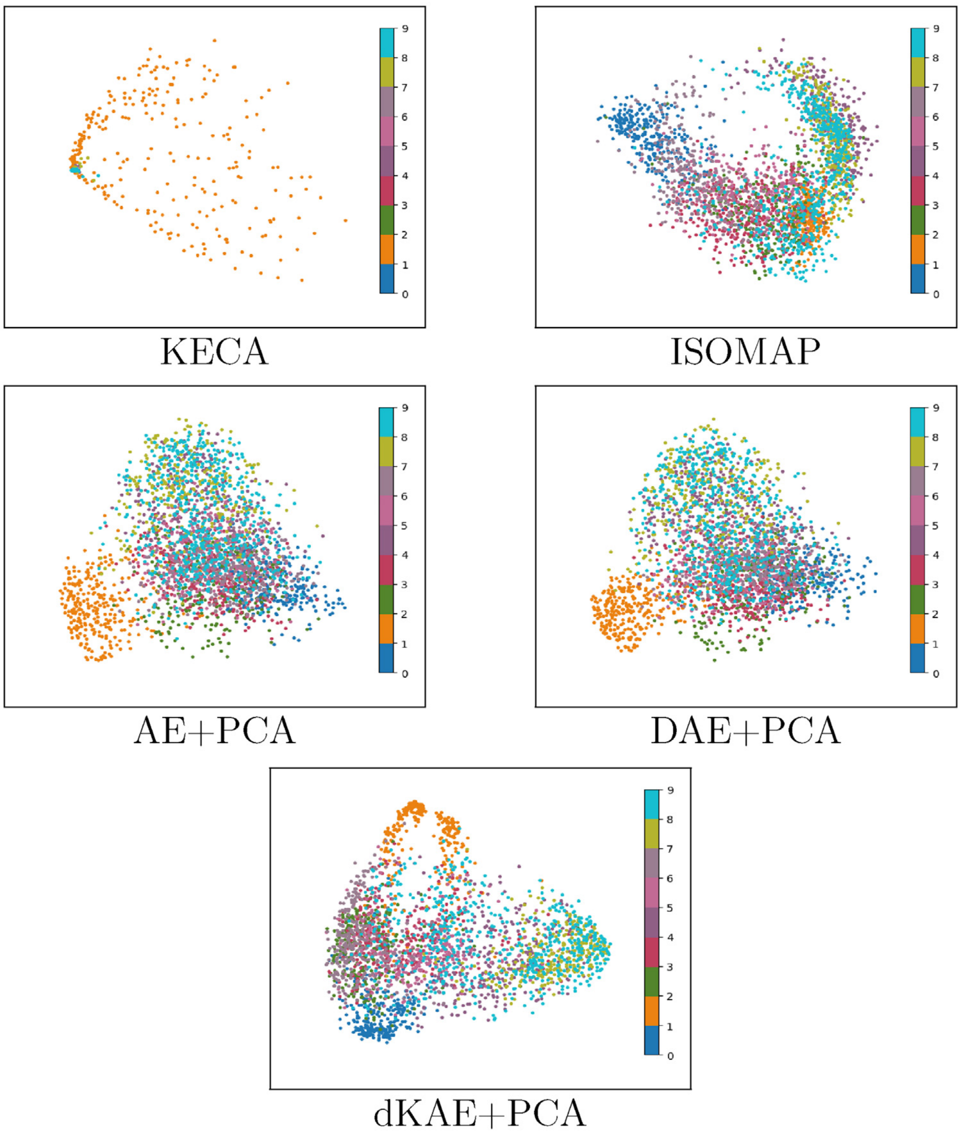


Fig. 6. MNIST data. Two dimensional embedding of the code space obtained using standard AEs, DAEs and our dKAE. The codes are projected to two dimensions using PCA. We compare their preformance to non-linear dimensionality reduction techniques KECA and ISOMAP.

Table 3
1-Nearest neighbor classification accuracy on representations shown in Fig. 6. The overall best result is highlighted in bold.

KECA	ISOMAP	AE + PCA	DAE + PCA	dkAE + PCA
29.5	36.8	30.5	31.2	**39.6**

reduce the dimensionality to 10 dimensions, the number classes in the dataset, before using PCA to reduce it further down to 2. In order to provide a quantitative evaluation of the visualizations, we consider the generalization error on a 1-Nearest Neighbor classification task following the example of [43]. Results are shown in Table 3, which demonstrate the superior performance obtained by means of dkAE.

5.2. Denoising and visualizing code space traversal in input space

Here, we highlight the potential of performing explicit operations in code space as initially described in Section 3.2. We try to emulate kPCA by performing PCA in our learned code space and evaluate the performance on a denoising task. Denoising is a task that requires both a mapping to the kernel space, as well as a back-projection to the input space. Traditional kernel methods cannot perform back-projection explicitly; approximate solutions have been proposed in the literature [3,20]. We choose the method proposed by Bakir et al. [3], where they use kernel ridge regression, such that a different kernel (in our case an RBF) can be used for back-mapping. Due to the challenge of finding a good σ for the RBF kernel that works on all MNIST numbers, we performed this test on the 5 and 6 class only. The regularization parameter and the σ required for the back-projection where found via grid search, where the best regularization parameter according to mean squared error (MSE) reconstruction was found to be 0.5 and σ as the median of the Euclidean distances between the projected feature vectors.

Both models are fitted on the training set and additive Gaussian noise is added to the test set. For both methods, 32 principal components are used. Table 4 shows that dkAE + PCA outperforms kPCAs reconstruction in terms of MSE. However, as MSE is not necessarily a good measure for denoising [3], we also visualize the results in Fig. 7. It can be seen that dkAE yields sharper images in the denoising task. We further compare the results to a denoising autoencoder

Table 4
MSE of reconstruction.

Noise std.	kPCA	DAE + PCA	dkAE + PCA
0.25	0.0427	0.0173	0.0358

(DAE + PCA). We observe that the denoising autoencoder is able to outperform the dkAE with regard to the MSE measure as it is explicitly trained for the denoising task. Qualitatively, however, we observe in Fig. 7 that the qualitative difference between these two is small, with DAE outperforming the dkAE on some images while producing more washed out images on others. For example, the reconstruction of the first image in the first row is better reconstructed using the DAE, while the second and fifth image in the first row are better reconstructed by the dKAE.

dkAE allows to explicitly explore the code space beyond the image of the dataset at hand and accordingly generate new instances with related representations in input space. To this end, we visualize the effect of movements in code space, illustrated in Fig. 8. For this experiment, we perform k-means clustering in code space and chose the number of clusters to be equal to the number of classes in MNIST. We select pairs of cluster centroids at random and interpolate between two centroids following a straight path in code space; in future works, we will consider also non-linear methods to obtain a smoother interpolation between the centroids [46]. The first and last image in Fig. 8 correspond to the cluster centers. The intermediate images are generated by mapping points along the aforementioned path in code space back to the input space by means of the trained decoder. In the first two panels, we observe a smooth transition of an 8 and a 7 to a 0. The third panel, instead, illustrates that k-means found two clusters in the 1s class, one for the far leaning ones and one for the straight ones. Interpolating between these two allows us to generate numbers with a varying degree of leaning to the right.

6. Conclusions

We proposed a novel model for autoencoders, dubbed deep kernelized autoencoders, that exploits information provided by a user-defined kernel matrix to learn similarity-preserving data representations. The proposed model is trained end-to-end in an

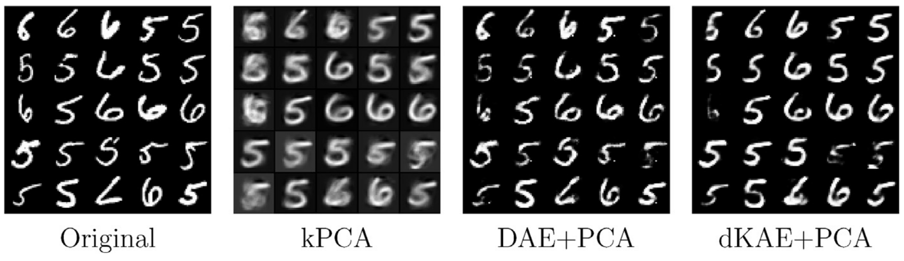


Fig. 7. Denoising with kPCA in input space and PCA in code space.

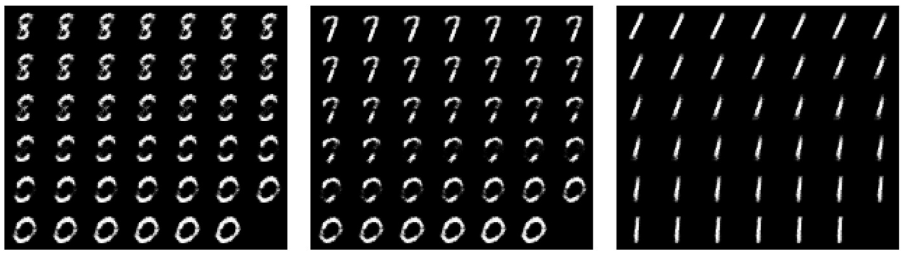


Fig. 8. First and the last image of each panel show two k-means centroids in code space obtained on the MNIST dataset. Additional numbers are generated by "walking" on interpolated points between the two centroids.

unsupervised way. By means of a parameter-free kernel alignment procedure based on inner products between codes, we are able to approximate arbitrary kernel functions defined in input space. This allows us to learn an explicit mapping from the input space to the code space, as well as the backward mapping. We evaluated the learned data representations on classification tasks and illustrated how the learned backmapping can be used to visualize operations performed directly in code space. In addition, the proposed autoencoder enables us to emulate well-known kernel methods for unsupervised learning, such as kernel PCA; however, our approach scales well with the number of data points as it is not based on eigendecomposition procedures.

In future work, we will continue to investigate this line of research by exploring alternative loss functions for kernel alignment, beyond those based on Frobenius norm. In particular, we will investigate the use of information-theoretic divergence measures and formulations based on mutual information between positive semi-definite matrices.

Acknowledgments

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the GPU used for this research. This work was partially funded by the Norwegian Research Council FRIPRO grant no. 239844 on developing the *Next Generation Learning Machines*.

References

[1] A. Achille, S. Soatto, Information Dropout: Learning Optimal Representations Through Noise, 2017 arXiv:1611.01353.
[2] A.A. Alemi, I. Fischer, J.V. Dillon, K. Murphy, Deep Variational Information Bottleneck, 2017 arXiv:1612.00410.
[3] G.H. Bakir, J. Weston, B. Schölkopf, Learning to find pre-images, in: Advances in Neural Information Processing Systems, 2004, pp. 449–456.
[4] Y. Bengio, Learning deep architectures for AI, Found. Trends Mach. Learn. 2 (1) (2009) 1–127.
[5] Y. Bengio, A. Courville, P. Vincent, Representation learning: a review and new perspectives, IEEE Trans. Pattern Anal. Mach. Intell. 35 (August (8)) (2013) 1798–1828, http://dx.doi.org/10.1109/TPAMI.2013.50, ISSN 0162-8828.
[6] B.E. Boser, I.M. Guyon, V.N. Vapnik, A training algorithm for optimal margin classifiers, Proceedings of the Fifth Annual Workshop on Computational Learning Theory (1992) 144–152.
[7] A.M. Bronstein, M.M. Bronstein, R. Kimmel, Generalized multidimensional scaling: a framework for isometry-invariant partial surface matching, Proc. Natl. Acad. Sci. U. S. A. 103 (5) (2006) 1168–1172, http://dx.doi.org/10.1073/pnas.0508601103.
[8] M. Chalk, O. Marre, G. Tkacik, Relevant sparse codes with variational information bottleneck, in: D.D. Lee, M. Sugiyama, U.V. Luxburg, I. Guyon, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Currant Associates, Inc., 2016, pp. 1957–1965.
[9] Y. Cho, L.K. Saul, Kernel methods for deep learning, in: Y. Bengio, D. Schuurmans, J.D. Lafferty, C.K.I. Williams, A. Culotta (Eds.), Advances in Neural Information Processing Systems, Curran Associates, 2009, pp. 342–350.
[10] W. Chu, D. Cai, Stacked similarity-aware autoencoders, in: Proceedings of the 26th International Joint Conference on Artificial Intelligence, AAAI Press, 2017, pp. 1561–1567.
[11] T.M. Cover, J.A. Thomas, Elements of Information Theory, Wiley, New York, 1991.
[12] N. Cristianini, A. Elisseeff, J. Shawe-Taylor, J. Kandola, On kernel-target alignment, in: Advances in Neural Information Processing Systems, 2001.
[13] B. Dai, B. Xie, N. He, Y. Liang, A. Raj, M.-F.F. Balcan, L. Song, Scalable kernel methods via doubly stochastic gradients, in: Advances in Neural Information Processing Systems, 2014, pp. 3041–3049.
[14] L.G.S. Giraldo, M. Rao, J.C. Principe, Measures of entropy from data using infinitely divisible kernels, IEEE Trans. Inf. Theory 61 (November (1)) (2015) 535–548, http://dx.doi.org/10.1109/TIT.2014.2370058.
[15] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, Proceedings of the International Conference on Artificial Intelligence and Statistics (May 2010) 249–256.
[16] L. Gómez-Chova, G. Camps-Valls, J. Calpe-Maravilla, L. Guanter, J. Moreno, Cloud-screening algorithm for ENVISAT/MERIS multispectral images, IEEE Trans. Geosci. Remote Sens. 45 (12) (2007) 4105–4118.
[17] L. Gómez-Chova, R. Jenssen, G. Camps-Valls, Kernel entropy component analysis for remote sensing image clustering, IEEE Geosci. Remote Sens. Lett. 9 (2) (2012) 312–316.
[18] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507, http://dx.doi.org/10.1126/science.1127647, ISSN 0036-8075.
[19] G.E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (7) (2006) 1527–1554.
[20] P. Honeine, C. Richard, A closed-form solution for the pre-image problem in kernel-based machines, J. Signal Process. Syst. 65 (3) (2011) 289–299.
[21] F. Horn, K.-R. Müller, Learning Similarity Preserving Representations with Neural Similarity Encoders, 2017 arXiv:1702.01824.
[22] E. Izquierdo-Verdiguier, L. Gomez-Chova, L. Bruzzone, G. Camps-Valls, Semisupervised kernel feature extraction for remote sensing image analysis, IEEE Trans. Geosci. Remote Sens. 52 (9) (2014) 5567–5578.
[23] E. Izquierdo-Verdiguier, R. Jenssen, L. Gómez-Chova, G. Camps-Valls, Spectral clustering with the probabilistic cluster kernel, Neurocomputing 149 (2015) 1299–1304.
[24] R. Jenssen, Kernel entropy component analysis, IEEE Trans. Pattern Anal. Mach. Intell. 32 (5) (2010) 847–860.
[25] M. Kampffmeyer, S. Løkse, F.M. Bianchi, R. Jenssen, L. Livi, Deep kernelized autoencoders, in: P. Sharma, F.M. Bianchi (Eds.), 20th Scandinavian Conference on Image Analysis, Springer International Publishing, Tromsø, Norway, June 2017, pp. 419–430, http://dx.doi.org/10.1007/978-3-319-59126-1_35.
[26] M. Kampffmeyer, S. Løkse, F.M. Bianchi, L. Livi, A.-B. Salberg, R. Jenssen, Deep divergence-based clustering, IEEE International Workshop on Machine Learning for Signal Processing (September 2017) 1–8, http://dx.doi.org/10.1109/MLSP.2017.8168158.
[27] H. Kamyshanska, R. Memisevic, The potential energy of an autoencoder, IEEE Trans. Pattern Anal. Mach. Intell. 37 (6) (2015) 1261–1273, http://dx.doi.org/10.1109/TPAMI.2014.2362140.
[28] J. Zhao, Y. Kim, K. Zhang, A. Rush, Y. LeCun, Adversarially Regularized Autoencoders, Proceedings of the 35th International Conference on Machine Learning (2018).
[29] D. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014 arXiv:1412.6980.
[30] D.P. Kingma, M. Welling, Auto-encoding Variational Bayes, 2013 arXiv:1312.6114.
[31] A. Krizhevsky, Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.
[32] B. Kulis, M.A. Sustik, I.S. Dhillon, Low-rank kernel learning with Bregman matrix divergences, J. Mach. Learn. Res. 10 (Feb) (2009) 341–376.
[33] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[34] D.D. Lewis, Y. Yang, T.G. Rose, F. Li, RCV1: a new benchmark collection for text categorization research, J. Mach. Learn. Res. 5 (Apr) (2004) 361–397.
[35] S. Løkse, F.M. Bianchi, A.-B. Salberg, R. Jenssen, Spectral clustering using PCKID – a probabilistic cluster kernel for incomplete data, in: Scandinavian Conference on Image Analysis, Springer, 2017, pp. 431–442.
[36] L. Maaten, Learning a parametric embedding by preserving local structure, International Conference on Artificial Intelligence and Statistics (2009) 384–391.
[37] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, B. Frey, Adversarial Autoencoders, 2015 arXiv:1511.05644.
[38] K.Ø. Mikalsen, F.M. Bianchi, C. Soguero-Ruiz, R. Jenssen, Time series cluster kernel for learning similarities between multivariate time series with missing data, Pattern Recognit. 76 (2018) 569–581.
[39] G. Montavon, M.L. Braun, K.-R. Müller, Kernel analysis of deep networks, J. Mach. Learn. Res. 12 (2011 Nov) 2563–2581, ISSN 1532-4435.
[40] A.Y. Ng, M.I. Jordan, Y. Weiss, et al., On spectral clustering: analysis and an algorithm, in: Advances in Neural Information Processing Systems, 2001, pp. 849–856.
[41] A. Rahimi, B. Recht, Random features for large-scale kernel machines, in: J.C. Platt, D. Koller, Y. Singer, S.T. Roweis (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc., 2008, pp. 1177–1184.
[42] M. Rast, J. Bezy, S. Bruzzi, The ESA medium resolution imaging spectrometer MERIS a review of the instrument and its mission, Int. J. Remote Sens. 20 (9) (1999) 1681–1702.
[43] G. Sanguinetti, Dimensionality reduction of clustered data sets, IEEE Trans. Pattern Anal. Mach. Intell. 30 (3) (2008) 535–540.
[44] E. Santana, M. Emigh, J.C. Principe, Information Theoretic-Learning Auto-encoder, 2016 arXiv:1603.06653.
[45] B. Schölkopf, A. Smola, K.-R. Müller, Nonlinear component analysis as a kernel eigenvalue problem, Neural Comput. 10 (5) (1998) 1299–1319.
[46] H. Shao, A. Kumar, P.T. Fletcher, The Riemannian Geometry of Deep Generative Models, 2017 arXiv:1711.08014.
[47] R. Shwartz-Ziv, N. Tishby, Opening the Black Box of Deep Neural Networks via Information, 2017 arXiv:1703.00810.
[48] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.
[49] S. Still, Information bottleneck approach to predictive inference, Entropy 16 (2) (2014) 968–989, http://dx.doi.org/10.3390/e16020968.
[50] J.B. Tenenbaum, V. De Silva, J.C. Langford, A global geometric framework for nonlinear dimensionality reduction, Science 290 (5500) (2000) 2319–2323, http://dx.doi.org/10.1126/science.290.5500.2319.
[51] I. Tolstikhin, O. Bousquet, S. Gelly, B. Schoelkopf, Wasserstein Auto-encoders, 2017 arXiv:1711.01558.

[52] A. Vedaldi, A. Zisserman, Efficient additive kernels via explicit feature maps, IEEE Trans. Pattern Anal. Mach. Intell. 34 (March (3)) (2012) 480–492, http://dx.doi.org/10.1109/TPAMI.2011.153, ISSN 0162-8828.
[53] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion, J. Mach. Learn. Res. 11 (2010) 3371–3408.
[54] T. Wang, D. Zhao, S. Tian, An overview of kernel alignment and its applications, Artif. Intell. Rev. 43 (2) (2015) 179–192.
[55] A.G. Wilson, Z. Hu, R. Salakhutdinov, E.P. Xing, Deep kernel learning, Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (2016) 370–378.
[56] J. Xie, R. Girshick, A. Farhadi, Unsupervised deep embedding for clustering analysis, Proceedings of the 33rd International Conference on Machine Learning, vol. 48 (2016) 478–487, JMLR.org.

Paper IV

Deep divergence–based approach to clustering

Michael Kampffmeyer, Sigurd Løkse, Filippo M. Bianchi, Lorenzo Livi, Anrt-Børre Salberg and Robert Jenssen.

Neural Networks, 2019

Neural Networks 113 (2019) 91–101

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Deep divergence-based approach to clustering

Michael Kampffmeyer [a,*], Sigurd Løkse [a], Filippo M. Bianchi [a], Lorenzo Livi [b,c], Arnt-Børre Salberg [d], Robert Jenssen [a,d]

[a] *Machine Learning Group, UiT the Arctic University of Norway, Norway* [1]
[b] *Department of Computer Science, University of Exeter, UK*
[c] *Departments of Computer Science and Mathematics, University of Manitoba, Canada*
[d] *Norwegian Computing Center, Oslo, Norway*

A R T I C L E I N F O

Article history:
Received 8 June 2018
Received in revised form 14 January 2019
Accepted 29 January 2019
Available online 8 February 2019

Keywords:
Deep learning
Clustering
Unsupervised learning
Information-theoretic learning
Divergence

A B S T R A C T

A promising direction in deep learning research consists in learning representations and simultaneously discovering cluster structure in unlabeled data by optimizing a discriminative loss function. As opposed to supervised deep learning, this line of research is in its infancy, and how to design and optimize suitable loss functions to train deep neural networks for clustering is still an open question. Our contribution to this emerging field is a new deep clustering network that leverages the discriminative power of information-theoretic divergence measures, which have been shown to be effective in traditional clustering. We propose a novel loss function that incorporates geometric regularization constraints, thus avoiding degenerate structures of the resulting clustering partition. Experiments on synthetic benchmarks and real datasets show that the proposed network achieves competitive performance with respect to other state-of-the-art methods, scales well to large datasets, and does not require pre-training steps.

1. Introduction

Deep neural networks (Goodfellow, Bengio, & Courville, 2016; Krizhevsky, Sutskever, & Hinton, 2012) excel at hierarchical representation learning (Bengio, Courville, & Vincent, 2013), and yield state-of-the-art performance in image classification (Krizhevsky et al., 2012), object detection (Ren, He, Girshick, & Sun, 2015), segmentation (Kampffmeyer, Salberg, & Jenssen, 2016; Long, Shelhamer, & Darrell, 2015), time series prediction (Bianchi, Maiorino, Kampffmeyer, Rizzi, & Jenssen, 2017) and speech recognition (Graves, Mohamed, & Hinton, 2013), to name a few. However, deep networks are usually trained in a supervised manner, hence requiring a large amount of labeled data. This is a challenge in many application domains.

Clustering (Jain, 2010; Von Luxburg, 2007), one of the fundamental areas in machine learning, aims at categorizing unlabeled data into groups (clusters). A promising direction in deep learning research is to learn representations and simultaneously discover cluster structure in unlabeled data by optimizing a discriminative loss function. Deep Embedded Clustering (DEC) (Xie, Girshick, & Farhadi, 2016) exemplifies this line of work and represents, to the best of our knowledge, the state-of-the-art. DEC is based on an optimization strategy in which a neural network is pre-trained by means of an autoencoder and then fine-tuned by jointly optimizing cluster centroids in output space and the underlying feature representation. Another example is (Yang, Fu, Sidiropoulos and Hong, 2016), where the authors propose a joint optimization for dimensionality reduction using a neural network and k-means clustering. Alternative approaches to unsupervised deep learning based on adversarial networks have recently been proposed (Goodfellow et al., 2014). These approaches are different in spirit but can also be used for clustering (Makhzani, Shlens, Jaitly, Goodfellow, & Frey, 2015; Springenberg, 2015).

In this work, we propose what we called the Deep Divergence-based Clustering (DDC) algorithm. Our method takes inspiration from the vast literature on traditional clustering techniques that optimize discriminative loss functions based on information-theoretic measures (Dhillon, Mallela, & Kumar, 2003; Jenssen, Erdogmus, Hild, Principe, & Eltoft, 2007; Tishby & Slonim, 2001; Vikjord & Jenssen, 2014). The main motivation for this choice is that the divergence, as a measure of dissimilarity between clusters represented by their probability density functions, builds on two fundamental objectives (Fig. 1): the separation between clusters and the compactness within clusters. These are desirable properties to increase identifiability of nonparametric mixtures (Aragam, Dan, Ravikumar, & Xing, 2018). Our new divergence-based loss function for deep clustering supports end-to-end learning and explicitly exploits knowledge about the geometry of the output space during the optimization. DDC achieves state-of-the-art performance without requiring hand-crafted feature design, reducing also the importance of a pre-training phase.

* Corresponding author.
E-mail address: michael.c.kampffmeyer@uit.no (M. Kampffmeyer).
1 http://machine-learning.uit.no/.

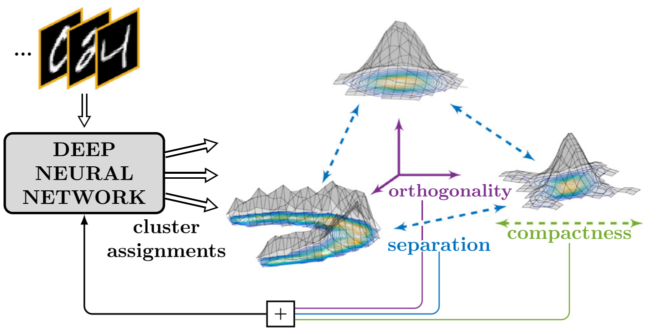


Fig. 1. Our approach takes advantage of the power of deep learning to extract features and perform clustering in an end-to-end manner. The proposed loss function is rooted in two fundamental objectives of clustering: separation and compactness of clusters.

A preliminary version of this paper appeared in Kampffmeyer, Løkse, Bianchi, Livi, Salberg and Jenssen (2017). The preliminary version was targeted towards image clustering combining a convolutional neural network architecture with our proposed clustering loss function. Here, we extend our work by (i) modifying the proposed architecture such that it can also handle textual data; (ii) conducting experiments and comparisons on additional datasets (including textual data — Reuters dataset); (iii) providing a thorough analysis of the proposed cost function and its components via ablation experiments; (iv) illustrating and discussing the functioning of the method in controlled settings; (v) interpreting predictions of the network by means of guided backpropagation (Springenberg, Dosovitskiy, Brox, & Riedmiller, 2014); and finally (vi) providing a more thorough literature background discussion, placing our work into a broader context.

This paper is structured as follows. Section 2 provides an overview of related works. Section 3 presents the proposed methodology for performing clustering with deep networks. In Section 4, we show the experimental results on several datasets and analyze the proposed cost function in detail. Finally, in Section 5 we draw conclusions and point to future directions.

2. Related work

Common approaches to unsupervised deep learning include methods based on deep belief networks, autoencoders, and generative adversarial networks (Bengio et al., 2013; Goodfellow et al., 2014). These methods have been mainly used for unsupervised pre-training (Erhan, Bengio, Courville, Manzagol, Vincent, & Bengio, 2010). Deep belief networks were the first of these models to be proposed and consist of stacked restricted Boltzmann machines that are trained in a greedy fashion (Hinton, Osindero, & Teh, 2006). Once trained, deep belief networks can be used to initialize neural networks.

Although several types of autoencoders have been proposed, all share a common underlying architecture consisting of an encoding and a decoding layer. The encoder is responsible for producing a hidden representation; the decoder re-generates inputs from the hidden representation. Both can efficiently be learned using backpropagation, by minimizing the reconstruction loss between original input and decoder output. Variations include, among others, denoising autoencoders (Vincent, Larochelle, Bengio, & Manzagol, 2008), which regularize the original autoencoder model by adding noise to inputs and then changing the objective to both include reconstruction and denoising, contractive autoencoders (Rifai, Vincent, Muller, Glorot, & Bengio, 2011), and more recently autoencoders that are regularized by preserving similarities in input space (Kampffmeyer, Løkse, Bianchi, Jenssen and Livi, 2017). Variational autoencoders (Kingma & Welling, 2013) have been used recently for several unsupervised tasks, such as image generation (Gregor, Danihelka, Graves, Rezende, & Wierstra, 2015) and segmentation (Sohn, Lee, & Yan, 2015). This approach assumes that data are generated from directed graphical models and uses a variational approach to learn latent representations.

Adversarial generative models (Goodfellow et al., 2014) are more recent approaches to unsupervised deep learning. Here, two networks are trained: one is responsible for discriminating between real and generated images; the other is responsible for generating realistic-enough images to confuse the first network.

Clustering is a classic information processing problem, particularly important in machine learning (Bianchi, Livi, & Rizzi, 2016; Jain, 2010; Myhre, Mikalsen, Løkse, & Jenssen, 2018; Nie, Tian, & Li, 2018; Rodriguez & Laio, 2014). Countless approaches exist for clustering, with mean shift (Comaniciu & Meer, 2002), k-means and expectation–maximization algorithms (Aggarwal & Reddy, 2013), being some of the most well-known ones. In the last decade, *spectral clustering* played a prominent role in the field, see for instance (Jenssen, 2010; Ng, Jordan, Weiss, et al., 2002; Nie, Zeng, Tsang, Xu, & Zhang, 2011; Von Luxburg, 2007; Yang, Xu, Nie, Yan, & Zhuang, 2010). Spectral clustering exploits the spectrum of similarity matrices to partition input data. Although these methods have demonstrated good performance in complex problems, they suffer from lack of scalability with respect to the number of input data points; cubic computational complexity for eigensolvers and quadratic complexity in terms of memory occupation. Attempts to solve these problems have been made by designing approximations or employing different optimization techniques (Dhillon, Guan, & Kulis, 2004; Han & Filippone, 2016; Yan, Huang, & Jordan, 2009).

Only a few methods have been proposed to exploit deep learning architectures for clustering, thereby taking advantage of hierarchical feature representations learned by such networks. CatGAN (Springenberg, 2015) and AAE (Makhzani et al., 2015) are based on the idea of adversarial networks. CatGAN is a method for learning a discriminative model, trained by optimizing a loss function implementing two different objectives. The first accounts for mutual information and predicted categorical distribution of classes in the data. The second objective maximizes the robustness of the discriminative network against an adversarial generative model. AAE instead assumes that data are generated from two latent variables, one associated with a categorical distribution and the other with a Gaussian distribution, and uses two adversarial networks to impose these distributions on the data representation. In a recent contribution (Bojanowski & Joulin, 2017), the authors propose an unsupervised training algorithm for CNNs and test its performance on image classification problems. The idea is to deal with the so-called "feature collapse problem" by mapping the learned features on random targets uniformly distributed on a d-dimensional unit sphere. A combination of recurrent and convolutional networks has also been used to perform image clustering by interpreting agglomerative clustering as a recurrent process (Yang, Parikh and Batra, 2016). Another recent approach to clustering based on the idea of hierarchical feature representation learning is provided by Zhang (2018), who proposes a multilayer bootstrap network where each layer performs multiple mutually independent k-centroids clusterings. Each layer gets trained individually in a bottom-up fashion and the input of consecutive layers is an indicator vector of which centroids are closest to a given input. Unlike the previously discussed methods, the multilayer bootstrap network does not offer end-to-end training.

To the best of our knowledge, DEC (Xie et al., 2016) is the method that is most closely related to our approach, as it is also founded on traditional clustering approaches. DEC simultaneously learns a feature representation as well as a cluster assignment in a two-step procedure. In the first step, soft assignments are computed between the data and cluster centroids based on a Student's t-distribution. Then, the parameters are optimized by matching soft assignments to a target distribution, which expresses confidence in assignments. The matching is performed by minimizing the Kullback–Leibler divergence. However, the effectiveness of DEC depends on a pre-training step implemented with autoencoders and does require explicit feature design to handle complex image data, e.g., Histogram of Oriented Gradients (HOG) features (Dalal & Triggs, 2005).

3. Deep clustering

We first describe, in Section 3.1 the proposed clustering loss function and present a description of the overall algorithm in Section 3.2. Successively, in Section 3.3, we discuss the deep network architectures that we propose to use for clustering problems, namely one that is based on convolutional layers for image clustering and one based on fully connected layers for vectorial data. Finally, we discuss scalability in Section 3.4.

Inspired by recent successes of introducing companion losses (Lee, Xie, Gallagher, Zhang, & Tu, 2015) to supervised deep learning models, we propose a loss function for clustering that includes terms computed over several network layers. The details of the loss function are outlined in what follows.

3.1. The loss function for clustering

The design of a loss function that allows the network to learn via gradient descent the intrinsic cluster structure in the input data is a fundamental part of this work. As illustrated in Fig. 2 and explained below, in addition to exploiting the geometry of the output space induced by the softmax activation, we adopt a kernel-based approach to estimate the divergence between clusters.

3.1.1. Loss term based on information-theoretic divergence measures

An information-theoretic divergence measure computes the dissimilarity between probability density functions (PDFs). For a clustering application, one would model each cluster by its PDF and optimize cluster assignments such that the divergence between their PDFs is maximized. Several different formulations of divergence measures exist in the literature (Basseville, 2013), many of which are suitable for clustering. In this work, we focus on one particular divergence measure that has been proven useful for clustering in the past, namely the Cauchy-Schwarz (CS) divergence (Jenssen, Principe, Erdogmus, & Eltoft, 2006; Vikjord & Jenssen, 2014), also referred to as the Information Cut in a graph clustering perspective (Jenssen et al., 2007). The CS divergence can be used in multi-cluster problems (i.e., problems with more than two clusters) by averaging the pairwise divergence over all pairs of cluster PDFs.

Considering $k \geq 2$ distinct PDFs, the CS divergence is defined as

$$D_{\text{cs}}(p_1, \ldots, p_k) = -\log\left(\frac{1}{k}\sum_{i=1}^{k-1}\sum_{j>i}\frac{\int p_i(\mathbf{x})p_j(\mathbf{x})d\mathbf{x}}{\sqrt{\int p_i^2(\mathbf{x})d\mathbf{x}\int p_j^2(\mathbf{x})d\mathbf{x}}}\right). \tag{1}$$

For a pair of PDFs, p_i and p_j, we have $0 \leq D_{\text{cs}}(p_1, p_2) < \infty$, where we obtain the minimum value iff $p_i = p_j$. Thus, in order to maximize cluster separation and compactness, we want the divergence to be as large as possible. Since the logarithm is a monotonic function, maximizing (1) is in practice equivalent to minimizing the argument of the logarithm. We observe that the minimum is obtained when the numerator is small and the denominator is large. Intuitively, this fact implies that the similarity between samples in different clusters is small (numerator) and similarity of samples within the same cluster is large (denominator).

In this paper, we make use of the divergence in (1) to measure the distance between clusters. Since the underlying true densities p_i, p_j are unknown, we follow a data-driven approach and approximate the PDFs using a Parzen window estimator, configured with a Gaussian kernel having bandwidth σ. We define the matrix $\mathbf{K} \in \mathbb{R}^{n\times n}$ that encodes the similarities between n input data. The matrix element $k_{i,j}$ stores the value $\exp(-d(\mathbf{x}_i, \mathbf{x}_j)^2/(2\sigma^2))$, where $d(\mathbf{x}_i, \mathbf{x}_j)$ is the Euclidean distance between data point $\mathbf{x}_i$ and $\mathbf{x}_j$. Using the Parzen window estimator, the CS divergence can be expressed as (Jenssen et al., 2006)

$$D_{\text{cs}} = -\log\left(\frac{1}{k}\sum_{i=1}^{k-1}\sum_{j>i}\frac{\sum_{q\in C_i}\sum_{l\in C_j}k_{q,l}}{\sqrt{\sum_{q,q'\in C_i}k_{q,q'}\sum_{l,l'\in C_j}k_{l,l'}}}\right). \tag{2}$$

Note that this estimate in (2) of the CS divergence can also be interpreted as measuring the cosine of the angle between cluster means in a Reproducing Kernel Hilbert Space (Jenssen et al., 2006) and is closely related to maximum mean discrepancy (Gretton, Borgwardt, Rasch, Schölkopf, & Smola, 2012). Assume that we have a $n \times k$ cluster assignment matrix $\mathbf{A} = [\alpha_{q,i}]$, with elements $\alpha_{q,i} \in \{0, 1\}$ that represent the crisp cluster assignment of data point q to cluster C_i. Thus, each data point q is represented by a one-hot vector. Then,

$$\sum_{q\in C_i}\sum_{l\in C_j}k_{q,l} = \sum_{q,l=1}^{n}\alpha_{q,i}\alpha_{l,j}k_{q,l} = \boldsymbol{\alpha}_i^T\mathbf{K}\boldsymbol{\alpha}_j,$$

where $\boldsymbol{\alpha}_i$ is the ith column of $\mathbf{A}$. The CS-divergence becomes $D_{\text{cs}} = -\log(d_\alpha)$, where

$$d_\alpha = \frac{1}{k}\sum_{i=1}^{k-1}\sum_{j>i}\frac{\boldsymbol{\alpha}_i^T\mathbf{K}\boldsymbol{\alpha}_j}{\sqrt{\boldsymbol{\alpha}_i^T\mathbf{K}\boldsymbol{\alpha}_i\boldsymbol{\alpha}_j^T\mathbf{K}\boldsymbol{\alpha}_j}}. \tag{3}$$

The formulation of the CS-divergence in (2) generalizes to soft cluster assignments, $\alpha_{q,i}$, preserving the differentiability of the loss function. In our architecture, the soft cluster assignments correspond to the softmax outputs and thereby the probability of a data point q to belong to cluster C_i.

The similarity values in $\mathbf{K}$ depend on the data representation. In particular, as data are processed by the neural network, several non-linear transformations map inputs onto different feature spaces, representing different levels of abstraction. The kernel bandwidth σ is computed based on the statistics of the learned representations. More details can be found in Section 4.6. To take advantage of the different representations and use the idea of companion losses for restricting the intermediate representations of the network, we use the hidden representation computed by the last fully connected layer before the output layer in addition to the soft cluster assignments produced by the softmax output layer. We do this by computing a (kernel) similarity matrix $\mathbf{K}_{\text{hid}}$, which, by considering the corresponding $d_{\text{hid},\alpha}$ in (3), yields a similarity score.

3.1.2. Loss term based on the geometry of the output space

The output space has a fixed number of dimensions (corresponding to the number of output neurons/clusters) and a precise geometry induced by the softmax activations used in the output layer (whose elements sum to 1), which we exploit in our algorithm:

1. The output space is a simplex in $\mathbb{R}^k$;
2. A data points degree of membership to a given cluster is maximized if the cluster assignment lies in a corner of the simplex (i.e., $\alpha_{q,i} = 1$ if data point q is fully assigned to cluster C_i);

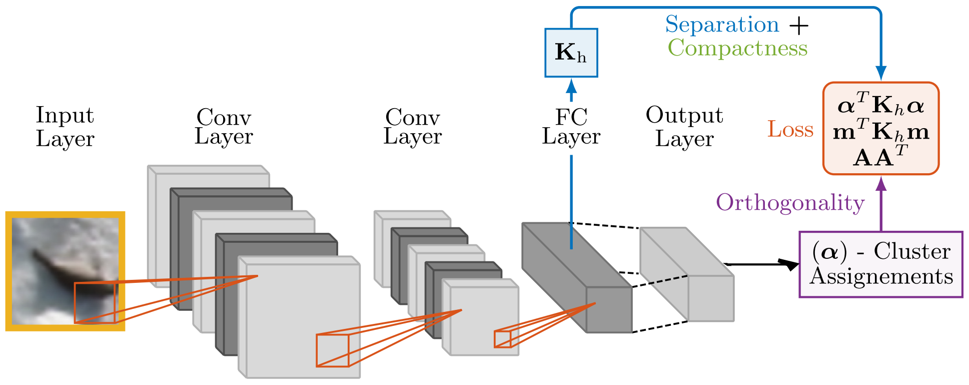


Fig. 2. Schematic depiction of the proposed architecture for image datasets and details of the loss function. The network consists of two convolution layers (each one followed by a pooling layer, not depicted in the figure) and a fully connected (FC) layer. Each layer is followed by a non-linear ReLU transformation. Finally, a fully connected output layer implements a logistic function (softmax). The unsupervised loss function operates on the kernel matrix $\mathbf{K}_h$ encoding data similarities evaluated on the hidden representation, and the values of the cluster assignment returned by the softmax function. The orthogonality constraint is derived from cluster assignments, while separation and compactness constraints come from the Cauchy-Schwarz divergence, computed on the similarity matrix (weighted by cluster assignments). The convolutional layers are replaced when non-image data are considered.

3. Following from the previous point, cluster assignment vectors of data points assigned to different clusters, in the optimal case, should be orthogonal to each other.

This intuition about the geometry enables us to introduce a term that avoids degenerate solutions by addressing the aforementioned problem of collapsing features/clusters and encourages diversity in cluster assignment. For a given cluster assignment matrix, **A**, the strictly upper triangular elements of $\mathbf{A}\mathbf{A}^T$, denoted by triu($\mathbf{A}\mathbf{A}^T$), consists of inner products between cluster assignment vectors. Unless explicitly stated, triu($\mathbf{A}\mathbf{A}^T$) will denote the sum of these elements. Note that we do not include the elements on the diagonal. Further, $\mathbf{A}\mathbf{A}^T$ will consist entirely of non-negative elements because **A** is non-negative; cluster assignment vectors are orthogonal if and only if these inner products are zero. Thus, our criterion consists of enforcing low values in the upper triangular elements. This also has the effect of a regularization term if the number of clusters is smaller than the number of input data points. Indeed, not all data points in the restricted space can be orthogonal to each other, forcing data points to repel each other, thereby acting against the terms that try to improve similarity. This term also encourages a balanced distribution of data points in the different classes, which makes our loss ideal for problems with balanced classes. Alternative regularization methods that are not based on the balanced class assumption will be investigated in future work.

The fact that cluster assignment vectors are orthogonal, however, does not imply that such vectors are embedded in a corner of the simplex. As an example, assume that $\alpha_{q,i} = 1$ and $\alpha_{l,i} = 0$. Due to the restrictions of the simplex geometry, it follows that $\alpha_{q,k} = 0, \ k \neq i$ and therefore $\boldsymbol{\alpha}_q^T \boldsymbol{\alpha}_l = 0$ independently of the values of $\alpha_{l,k}, \ k \neq i$. Thus, l is not restricted to a simplex corner. Therefore, in order to enforce closeness to a corner of the simplex, we define an additional term for the loss function that reads

$$m_{q,i} = \exp(-\|\boldsymbol{\alpha}_q - \mathbf{e}_i\|^2),$$

where $\mathbf{e}_i \in \mathbb{R}^k$ is a vector denoting the ith corner of the simplex; representing cluster C_i. This exponential evaluates to one only when $\boldsymbol{\alpha}_q$ is located in a corner of the simplex. We make use of this fact by defining a third similarity term $d_{\mathrm{hid,m}}$, where $\mathbf{m}_i = [m_{q,i}] \in \mathbb{R}^n$ takes the place of $\boldsymbol{\alpha}_i$ in (3).

3.1.3. *The final clustering loss function*

The weights in the neural network architecture described in Section 3.3 can then be trained by minimizing the sum of the three different terms discussed in the previous section:

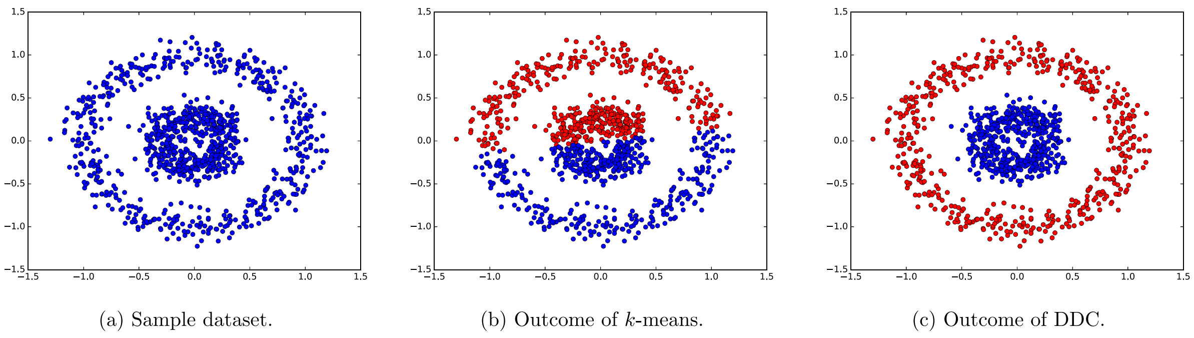
$$\begin{aligned} L &= d_{\mathrm{hid},\alpha} + \mathrm{triu}(\mathbf{A}\mathbf{A}^T) + d_{\mathrm{hid,m}} \\ &= \frac{1}{k}\sum_{i=1}^{k-1}\sum_{j>i}\frac{\boldsymbol{\alpha}_i^T \mathbf{K}_{hid}\boldsymbol{\alpha}_j}{\sqrt{\boldsymbol{\alpha}_i^T \mathbf{K}_{hid}\boldsymbol{\alpha}_i \boldsymbol{\alpha}_j^T \mathbf{K}_{hid}\boldsymbol{\alpha}_j}} + \mathrm{triu}(\mathbf{A}\mathbf{A}^T) \\ &\quad + \frac{1}{k}\sum_{i=1}^{k-1}\sum_{j>i}\frac{\mathbf{m}_i^T \mathbf{K}_{hid}\mathbf{m}_j}{\sqrt{\mathbf{m}_i^T \mathbf{K}_{hid}\mathbf{m}_i \mathbf{m}_j^T \mathbf{K}_{hid}\mathbf{m}_j}} \end{aligned} \quad (4)$$

As a proof of concept, we illustrate the functioning of our clustering method on a classical synthetic dataset where one class is represented by a small circle and the other class is a ring. Note, that we use a fully-connected architecture (described in Section 3.3) for these experiments as we are considering non-image data. The dataset is shown in Fig. 3a. Figs. 3b and 3c illustrate the clustering outcome using k-means and the proposed method, respectively. It can be observed that the proposed method captures the highly nonlinear structure in the data and is able to discover clusters of non-spherical shapes, a highly desirable quality of clustering algorithms (Rodriguez & Laio, 2014).

Further, in Fig. 4 we visualize the output space for a three-cluster experiment. We chose three classes of the MNIST dataset (for dataset details see Section 4) and visualize the output space configuration during three different stages of the optimization process. As expected, the proposed clustering loss function (4) attempts to separate the data points by grouping similar points in ideal cluster centers located at the corners of the simplex. Note, we are using the convolutional architecture described in Section 3.3 as we are considering images in this experiment.

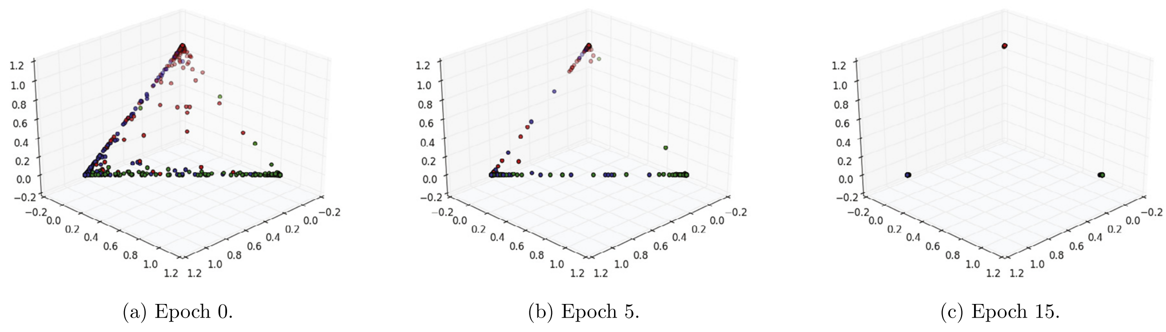
3.2. *Description of complete algorithm*

In this section, we summarize the proposed algorithm. For a given data batch, the assignment vectors and the hidden representation are obtained via a single forward pass. Based on the hidden representation, the kernel bandwidth, σ, is estimated and the kernel matrix is computed. From the assignment vectors, the distance to each of the ideal cluster centers (simplex corners) is computed, obtaining $\boldsymbol{m}_q$ for each data point q in a given minibatch. Using Eq. (4), we then can compute the loss based on the kernel matrix, the assignment vector $\boldsymbol{\alpha}_q$, and $\boldsymbol{m}_q$. Finally, all weights in the network are updated using Adam (Kingma & Ba, 2014). The full algorithm is outlined in Algorithm 1.



(a) Sample dataset. (b) Outcome of k-means. (c) Outcome of DDC.

Fig. 3. Illustration of DDC clustering outcome on a synthetic dataset, showing the capability of learning non-linear structures.



(a) Epoch 0. (b) Epoch 5. (c) Epoch 15.

Fig. 4. Illustration of DDC output space for three class MNIST training. Colors indicate class information of data points.

Algorithm 1 Deep Divergence-Based Clustering

```
Input: $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^n$
Output: Cluster assignments $\mathcal{A} = \{\boldsymbol{\alpha}_i\}_{i=1}^n$
1: Randomly initialize network parameters $\boldsymbol{\Theta}$
2: while not converged do
3:     Sample data batch $\mathcal{X}^{(b)}$ from $\mathcal{X}$
4:     Obtain assignment vectors $\mathcal{A}^{(b)}$ and hidden representations $\mathcal{H}^{(b)}$
5:     Compute $m_{q,i} = \exp(-\|\boldsymbol{\alpha}_q - \mathbf{e}_i\|^2),\ \forall\, \boldsymbol{\alpha}_q \in \mathcal{A}^{(b)}$
6:     Estimate kernel bandwidth $\sigma$ and compute $\mathbf{K}_{\text{hid}}^{(b)}$ from $\mathcal{H}^{(b)}$
7:     Compute loss with (4) and update $\boldsymbol{\Theta}$ with gradient descent
8: end while
```

3.3. Architecture overview

The network architecture is a design choice, and as such there are many options. In this paper, we choose an approach based on convolutional neural networks to process different image datasets, using a LeNet-inspired architecture (LeCun, Bottou, Bengio, & Haffner, 1998). We selected LeNet since it is a well-known benchmark network that supports end-to-end learning and has been widely used for image classification. The architecture is depicted in Fig. 2. It consists of two convolutional layers: the first one with 32 and the second one with 64 5×5 filters, each of them followed by a 2×2 max pooling layer and a ReLU activation. The last convolutional layer is followed by a fully connected layer with 100 nodes, another ReLU nonlinearity and, finally, the softmax output layer, whose dimensionality corresponds to the number of desired clusters. Batch-normalization (Ioffe & Szegedy, 2015) is applied in the fully connected layer. This design choice was made to increase the models' robustness and is explained in Section 4.6.

Our approach can also be applied to cluster data that are not images simply by replacing the convolutional and pooling layers with fully connected layers. In particular, for the experiments conducted on non-image data, we use an architecture with four fully connected layers of size $200 - 200 - 500 - C$. The 500 unit fully connected layer includes batch-normalization and the C unit layer corresponds to the softmax output layer with dimensionality equal to the number of clusters.

Recently, theoretical advances in the theory of neural networks (Giryes, Sapiro, & Bronstein, 2016) highlighted how the metric structure of input data is preserved by deep neural networks with random i.i.d Gaussian weights. One restriction is the fact that this is only true in the case where the intrinsic dimensionality of the data is proportional to the network width. However, Giryes et al. (2016) proved that the intrinsic dimensionality of the data does not increase as the data propagate through the network, which suggests that the width of the network (the number of hidden units per layer) that we consider for our experiments should suffice. This theoretical property supports the design choice behind the proposed loss function, which estimates the divergence over the hidden representation, rather than in input space.

3.4. Main memory footprint

Using gradient-based optimization in neural networks allows us to process large datasets, overcoming well-known limitations of spectral methods mentioned in the introduction with regards to memory requirements. The memory cost of our approach is kept low by the use of mini-batch training and scales linearly with the number of input data points, n, compared to the quadratic or super quadratic complexities encountered in spectral methods. The proposed method scales quadratically with the mini-batch size m as the kernel matrix is computed over the hidden representation for a given mini-batch; however, this is generally not an issue as $m \ll n$.

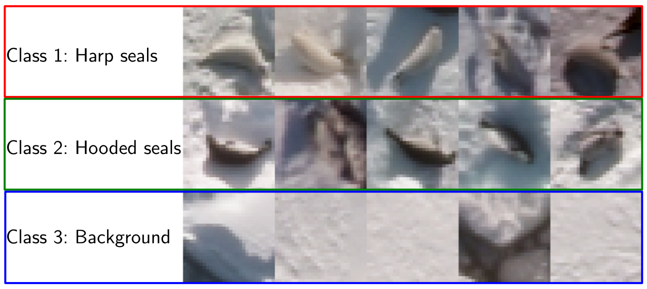


Fig. 5. Examples from the SEALS-3 dataset. The top row displays examples from the harp seal class, the middle row from the hooded seal, and the bottom row from the background class.

4. Experiments

We evaluate DDC on the MNIST handwritten image dataset as it represents a well-known benchmark dataset in the literature. In addition, we evaluate our algorithm on two more challenging real-world datasets: one dataset for detection of seal pups in aerial images here referred to as the SEALS-dataset and the Reuters dataset for news story clustering. In the results, we compare our method to four alternative clustering approaches.

4.1. MNIST dataset

The MNIST dataset contains 70 000 handwritten images of the digits 0 to 9 (LeCun et al., 1998) and consists of images that were originally in the National Institute of Standards and Technology (NIST) dataset. The images are grayscale with the digits centered in the 28×28 images.

4.2. SEALS dataset

The SEALS-3 dataset consists of several thousand aerial RGB images acquired during surveys in the West Ice east of Greenland in 2007 and 2012 and east of New Foundland, Canada, in 2012. The images are acquired from approximately 300m altitude, and the pixel spacing is about 3 cm (depending on the exact flight altitude). A typical image size is $11\,500 \times 7500$ pixels. From these images 64×64 image crops of harp seal pups, hooded seal pups and background (non-seals) were extracted and down-sampled to 28×28 to fit our chosen architecture. As the smallest class consists of 1000 images, we select a reduced set of 1000 images from each class to create a balanced dataset. Fig. 5 shows example images for the three classes in the SEALS-3 dataset.

As the background class contains a large variety of images, such as white snow and black water images, unsupervised algorithms are likely to partition these instances into different clusters. Therefore, we additionally created and tested another dataset (SEALS-2), where the background class was not included.

4.3. Reuters dataset

The Reuters dataset consists of 800 000 news stories that have been manually categorized into a category tree (Lewis, Yang, Rose, & Li, 2004). In this work, similarly to Xie et al. (2016), we chose the four root categories as labels, removed stories that are labeled with multiple root categories and represent each news story as a feature vector consisting of the Term Frequency-Inverse Document Frequency (TF-IDF) of the 2000 most frequently occurring word stems. As done for the SEALS dataset, we select 54 000 datapoints from each class in order to balance the dataset.

4.4. Performance measures

To evaluate the partition quality obtained after training, we consider two different supervised measures. The first measure is the Normalized mutual information (NMI), defined as

$$NMI = \frac{I(l, c)}{\frac{1}{2}[H(l) + H(c)]}\,, \tag{5}$$

where $I(\cdot, \cdot)$ and $H(\cdot)$ denote mutual information and entropy functionals, respectively. The second evaluation metric is the unsupervised clustering accuracy

$$ACC = \max_{\mathcal{M}} \frac{\sum_{i=1}^{n} \delta(l_i = \mathcal{M}(c_i))}{n}\,, \tag{6}$$

where l_i refers to the ground truth label, c_i to the assigned cluster of data point i, and $\delta(\cdot)$ is the Dirac delta. $\mathcal{M}$ is the mapping function that corresponds to the optimal one-to-one assignment of clusters to label classes implemented by means of the Hungarian algorithm (Kuhn, 1955), which solves the linear assignment problem of assigning a cluster to its label class in polynomial time.

4.5. Baseline methods

As methods for comparison, we use k-means (with the so-called k-means++ initialization Arthur & Vassilvitskii, 2007) as a well-known baseline and a hierarchical information theoretic clustering approach (Vikjord & Jenssen, 2014) based on implicit cluster density estimation using (1) a k-NN approach (ITC-kNN) and (2) a parzen window approach (ITC-parzen). Further we compare our approach to a representative subset of state-of-the-art methods in clustering, namely *Deep Embedded Clustering* (DEC) (Xie et al., 2016),[2] *Spectral Embedded Clustering* (SEC) (Nie et al., 2011), and *Local Discriminant Models and Global Integration* (LDMGI) (Yang et al., 2010). SEC and LDMGI are spectral clustering algorithms based on the foundations discussed in Ng et al. (2002). In SEC, the authors jointly optimize the normalized cut loss function and a linear transformation from input to the embedding space for spectral clustering, such that the transformed data is close to the embedded data. The similarity is modeled using a Gaussian kernel. LDMGI optimizes a similar objective function, but the Laplacian matrix is learned by exploiting manifold structure and discriminant information, in contrast to most spectral clustering methods where the Laplacian is calculated by using a Gaussian kernel. Both of these methods use *spectral rotation* (Yu & Shi, 2003) to obtain the final cluster assignments instead of k-means, which is common for many spectral clustering methods. To the authors best knowledge, these two methods represent the current state-of-the-art in spectral clustering, outperforming conventional spectral clustering methods in a wide variety of clustering tasks (Nie et al., 2011; Yang et al., 2010).

Following the experiment setup of Xie et al. (2016), the parameters in the baseline models are set according to the suggestions in their respective papers, varying their hyperparameters over 9 possible choices. For each one, we run the baseline models 20 times. The best result is shown in the experiments. Due to the lack of hyperparameters in k-means (except the number of clusters k, which is fixed in our experiments), the accuracy for the best run from 20 different random initializations is reported.

4.6. Implementation

The proposed network model is trained end-to-end by using Adam (Kingma & Ba, 2014) and implemented using the Theano framework (Theano Development Team, 2016). For each image

[2] Caffe version of DEC publicly available: https://github.com/piiswrong/dec.

datasets we used the same convolutional architecture and for each vectorial datasets we used the same fully connected architecture. Training is performed on mini-batches of size 100. By avoiding a fine-tuning for each problem at hand, we show the robustness of our architecture. Training is performed with a learning rate of 0.001 for the convolutional neural network and 10^{-5} for the fully connected network. The network is trained for 70 000 iterations and the ordering of the mini-batches was reshuffled at each epoch. Weights of the network are initialized following He, Zhang, Ren, and Sun (2015). Following the rule-of-thumb in Jenssen (2010), σ of the Gaussian kernel was chosen to be 15% of the median pairwise Euclidean distances between the feature representation produced by the first fully-connected layer, which produced satisfying results for all datasets. The median is adaptive and recomputed as part of the cost function evaluation. In our experiments, we observed that this rule-of-thumb benefited considerably from activation rescaling through batch-normalization.

As DDC is prone to get stuck in local minima, a common problem for unsupervised deep architectures, we run DDC for 20 runs and report the accuracy of the run with the lowest value of our unsupervised loss function. We also report the results of a voting scheme of the top three runs according to our unsupervised loss function. Following Strehl and Ghosh (2002) and Vikjord and Jenssen (2014), clustering results of the best performing run are used as a starting point and the clustering results of the other two runs are aligned to it via the mapping function provided by the Hungarian algorithm in an unsupervised manner. Once the results are aligned, we combine them via a simple voting procedure and compute the final unsupervised clustering accuracy using (6). In the following, this network ensemble is referred to as DDC-VOTE. Note that the voting procedure is completely unsupervised and is commonly used in ensemble approaches.

Table 1
Clustering accuracy for DDC, DDC-VOTE, and the baseline models. Best results are highlighted in bold.

Datasets	Method	NMI	ACC[%]
MNIST	K-means	0.50	53.33
	ITC (parzen)	–	–
	ITC (kNN)	–	–
	SEC	0.77	68.82
	LDMGI	0.81	83.03
	DEC	0.81	84.31
	DDC	0.83	86.58
	DDC-VOTE	**0.87**	**88.49**
SEALS-3	K-means	0.13	51.33
	ITC (parzen)	0.003	35.30
	ITC (kNN)	0.10	53.95
	SEC	0.15	49.00
	LDMGI	0.13	50.43
	DEC	**0.17**	50.33
	DDC	0.14	**55.97**
	DDC-VOTE	0.13	53.30
SEALS-2	K-means	0.015	56.85
	ITC (parzen)	0.003	51.55
	ITC (kNN)	0.020	57.20
	SEC	0.021	58.15
	LDMGI	0.018	57.85
	DEC	0.005	54.04
	DDC	0.15	72.05
	DDC-VOTE	**0.18**	**74.65**
Reuters	K-means	0.38	60.5
	ITC (parzen)	–	–
	ITC (kNN)	–	–
	SEC	–	–
	LDMGI	–	–
	DEC	0.38	63.79
	DDC	0.50	73.06
	DDC-VOTE	**0.51**	**77.62**

4.7. Results

We compare DDC and DDC-VOTE to the baseline algorithms on the MNIST and SEALS datasets and observe that they outperform the baseline methods on all datasets. The results can be found in Table 1. Due to very high computational complexity, the ITC algorithm could not be evaluated on MNIST and large datasets in general. This also highlights an important advantage of our formulation with regards to previous clustering approaches based on the CS divergence. Unlike cluster algorithms that estimate the optimal number of clusters from data, our method and the baseline approaches require the user to specify the number of clusters beforehand. By following a common practice, we have chosen the number of clusters equal to the number of classes in the corresponding datasets. It can be observed that DDC-VOTE generally outperforms DDC, except in the case of the SEALS-3 dataset, where all tested clustering algorithms perform poorly due to the high variation characterizing the background class. Methods based on adversarial networks, namely AAE and CatGAN, have shown to perform well on the MNIST clustering task (the only real dataset analyzed in these papers), by clustering the dataset into a large number of groups ($\geq$ 16), and mapping these into the 10 original classes in a post-processing step. A similar approach could potentially be employed by DDC to boost performance and will be explored in future work.

In what follows, we qualitatively analyze obtained clustering performance. Fig. 6 displays clustering results for the SEALS-3 dataset, where each row corresponds to the top ten scoring images for each of the three clusters. It is possible to note that the clustering result for the second and third cluster corresponds to a mix of background and seal images, with the third one containing white background images and harp seals and the second cluster containing black background images and hooded seals. As clustering is an unsupervised task and does not necessarily agree with the available supervised information, this result is not unexpected due to the fact that the background class includes large variations.

To further illustrate the clusters found in the dataset, we increased the number of clusters for the SEALS-3 dataset to 10. Fig. 7a shows an overview over the learned clusters. It can be observed that DDC generally finds reasonable clusters, for example by grouping water (dark patches) in one cluster and white background images in another. Also, it is possible to note that DDC generally groups the two different seal classes into separate clusters and assigns images containing both water and snow to a specific class.

Fig. 7b illustrates clustering results on MNIST. Interestingly, each cluster represents a distinct number. However, it can be observed that the 7's and 9's are mixed, which is expected due to their shape similarity. Furthermore, the MNIST dataset contains 1's that are straight and 1's that are rotated. Our results indicate that some of the far leaning 1's are clustered together with the 2 class, which has a similar diagonal line.

The results for the SEALS-3 dataset coincide with our intuition that the background class is likely to be divided into different classes. For the SEALS-2 dataset, we observe that DDC outperforms the competitor algorithms by a large margin (Table 1). As the SEALS dataset is less structured and more challenging datasets, we observe that methods that operate directly in pixel space (i.e., raw input space) perform poorly, stressing the importance of extracting higher-level features for clustering. Fig. 8 shows clustering results for DDC, where it can be clearly observed that hooded seals and harp seals are separated into two distinct clusters.

The proposed clustering cost function is not dependent on the convolutional architecture used in the previous two experiments and can also be used for training fully connected neural networks — which are used when working with vectorial data. For this purpose, we consider the Reuters dataset and substitute convolutional layers with fully connected ones as described in Section 3.3. Results

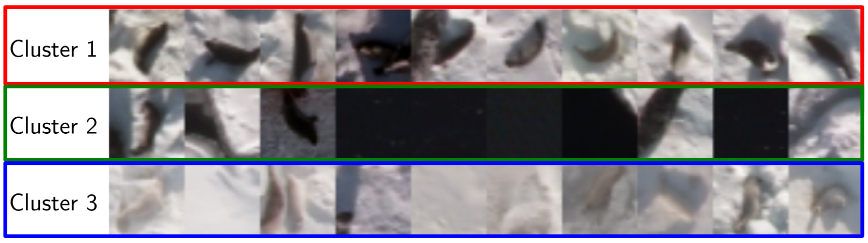


Fig. 6. Clustering result for the three classes in the SEALS-3 dataset. The first cluster appears to correspond to hooded seals, whereas the other two clusters correspond to a mix of background and seal images.

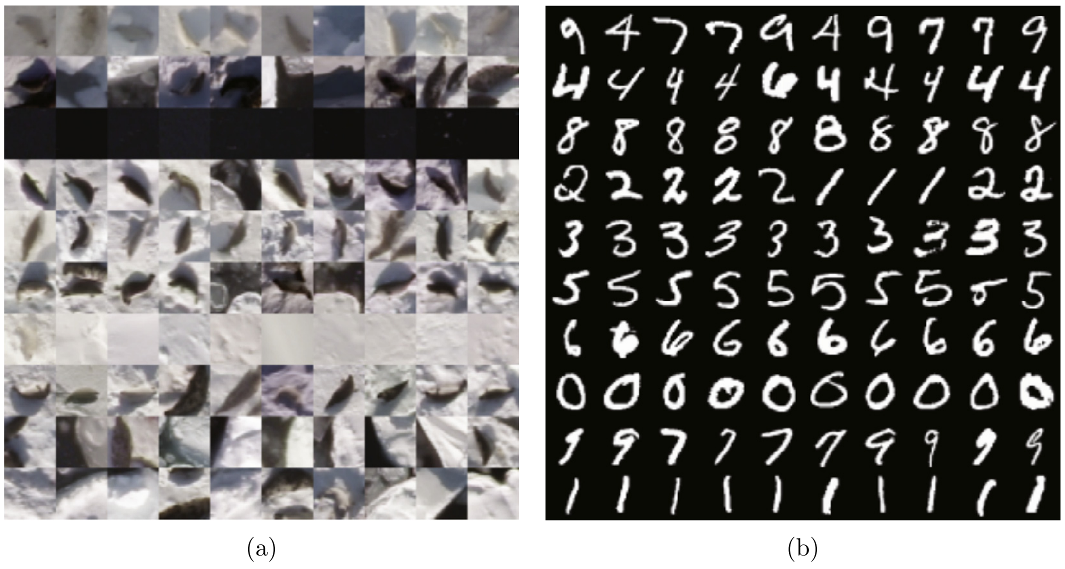


Fig. 7. (a) Results for the SEALS-3 dataset when clustering into ten clusters. (b) Clustering result for the ten-class MNIST dataset.

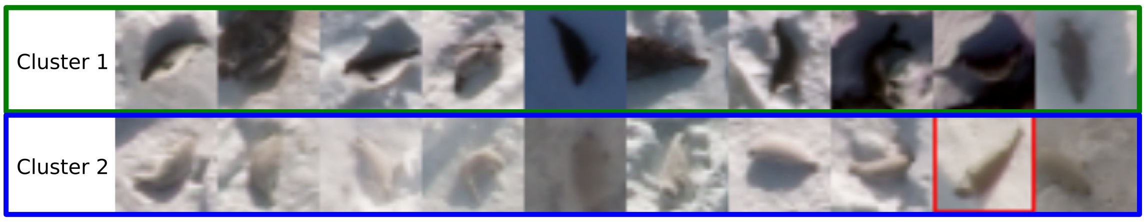


Fig. 8. Clustering result for the two classes in the SEALS-2 dataset. DDC groups the two seal types in distinct clusters. The dark seals in the top row corresponding to hooded seals and the light seals in the bottom row corresponding to harp seals. The red box indicates a mismatch.

are shown in Table 1, where it is possible to observe that DDC outperforms the competitors. Note that, due to the size of the Reuters dataset, running LDMGI and SEC was impossible as a consequence of their memory requirements discussed in Section 3.4. Note that, from the results presented in Xie et al. (2016), we can see that DEC still performs well when handling the imbalanced Reuters dataset, where the balanced assumption of DDC does not hold.

4.8. *Loss function*

In the following, we analyze the proposed loss function (4), providing empirical evidence of the importance of the different terms; moreover, we evaluate whether the different terms in the loss are related to the performance of the model. Fig. 9a shows the different terms in the loss function during the training phase as we monitor the accuracy of the best run on MNIST. It is clearly possible to observe that all terms (and also the overall loss) agree reasonably well with the overall clustering accuracy.

Considering that the network architecture is identical to networks used in supervised approaches and the availability of labels in our datasets, we can also monitor the terms of the loss function during supervised training. Fig. 9b shows how each term in the loss function and the overall loss decrease as we perform supervised training on MNIST using a cross-entropy loss function. Again, it is possible to notice that the individual terms have a similar decreasing trend. Note that large variations in the second loss term correspond to the aforementioned regularization effect (see Section 3.1.2).

In order to further analyze our method, we perform an ablation experiment (Nguyen, Yosinski, & Clune, 2015) to investigate the effect of the different terms on the clustering result. To this end, we recompute clustering accuracy on MNIST and Reuters for all different combinations of cost function terms. We repeat the experiments five times (20 runs each), compute the overall accuracy for the run with the lowest cost function each time and report the mean, standard deviation and the maximum accuracy values over these five results. The maximum accuracy value is

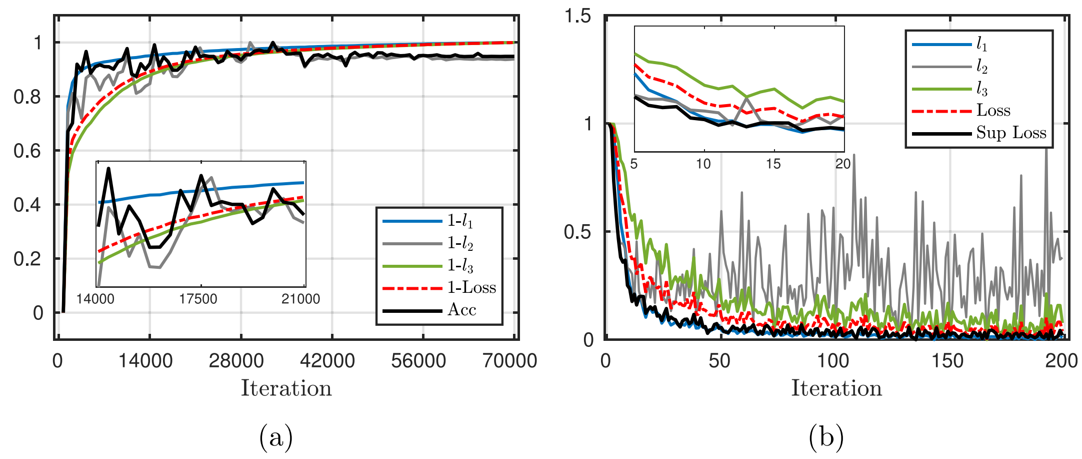


Fig. 9. (a) Comparison of the supervised accuracy with the unsupervised loss function (Loss) and its three terms ($l_1 = d_{\text{hid},\alpha}$, $l_2 = \text{triu}(\mathbf{AA}^T)$, and $l_3 = d_{\text{hid,m}}$) during training. (b) Values of the loss function and its three terms when training the network using a supervised loss function (Sup Loss). Note, the losses have been rescaled to range [0,1].

Table 2
Results of the ablation experiment for the MNIST and the Reuters datasets, illustrating the effect of the three different terms ($l_1 = d_{\text{hid},\alpha}$, $l_2 = \text{triu}(\mathbf{AA}^T)$, and $l_3 = d_{\text{hid,m}}$) composing the loss function (4).

Cost	MNIST		Reuters	
	Mean ± std	Max	Mean ± std	Max
l1	26.1 ± 2.9	31.5	39.3 ± 4.8	48.7
l2	48.9 ± 1.5	51.8	41.1 ± 2.9	46.0
l3	74.0 ± 2.9	77.0	70.2 ± 6.4	78.8
l1+l2	71.9 ± 8.0	83.5	66.0 ± 8.2	75.9
l1+l3	84.7 ± 5.1	88.6	64.8 ± 5.2	72.7
l2+l3	74.8 ± 5.5	84.3	70.3 ± 4.4	75.4
l1+l2+l3	80.8 ± 5.8	87.5	69.8 ± 7.3	82.6

reported solely in order to illustrate the maximum potential of the proposed method. In practice, the strategy from Section 4.6 would be used, wherein the best models according to the unsupervised cost function from each run are combined, denoted as DDC-Vote. Note, this differs from the results reported in Table 1 for DDC, where we report the accuracy only over 20 runs. Results for the MNIST and Reuters datasets are reported in Table 2. Our results illustrate that, by using all three terms together, we generally obtain better performance. However, the contribution of each term to the final performance is not consistent over all datasets. For instance, we observe that the $l2$ regularization term does not improve the overall result on the MNIST dataset, but does have a positive effect on the Reuters dataset.

The three terms in the loss function (4) were equally weighted in all experiments. However, better performance might be achieved by weighing such terms according to the data properties. For instance, decreasing the importance on $l_2 = \text{triu}(\mathbf{AA}^T)$ might allow our method to better handle imbalanced datasets. The analysis of more advanced weighting schemes is left for future work.

We further conducted experiments where we replaced the CS divergence (2) with a symmetrized Kullback–Leibler (KL) divergence, the divergence used by DEC. However, in our experiments we noticed that the performance of the proposed method drops considerably. On MNIST DDC with KL divergence only obtains an accuracy of 53.66%. Further, using a cosine similarity together with the KL divergence for DDC obtains 35.48% accuracy. We hypothesize that this is mainly due to the fact that the KL divergence encourages separation of clusters, but does not necessarily enforce their compactness. A more thorough analysis of alternative divergence measures is left for future work.

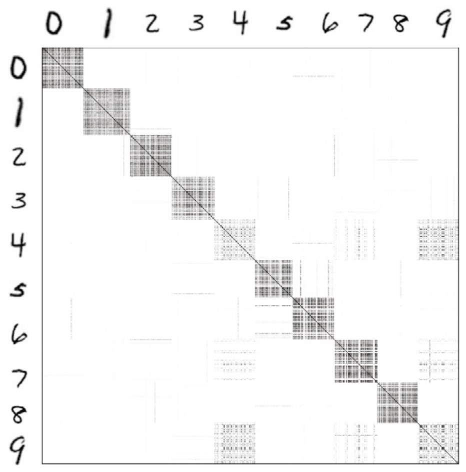


Fig. 10. Kernel matrix computed over the learned hidden representation. White colors correspond to low values in the kernel matrix, whereas dark values indicate large values.

4.9. *Learned representation*

Fig. 10 illustrates the final kernel matrix **K** computed over the hidden layer for the best MNIST run. Here, unlike in the case of training, where data points are fed to the model in a random order, the data points have been sorted according to their class labels. A clear block structure is evident from the figure. White and black values indicate low and high similarity, respectively. However, especially the 4 and 9 class show high in-between class similarity, which is not surprising due to their closeness in shape.

4.10. *Interpretability of neural network predictions*

A recent trend in deep learning is the development of methods to interpret predictions of neural networks trained with supervised information (Montavon, Lapuschkin, Binder, Samek, & Müller, 2017; Springenberg et al., 2014). However, interpretability is not only a problem in supervised settings. It could be argued that it is even more essential for unsupervised training, where learning is task-driven. One such approach is the guided backpropagation (Springenberg et al., 2014), which allows visualizing pixels in the image that are most influential for a given output decision. We use this technique to visualize what the model learns to recognize MNIST images. Fig. 11 illustrates the results for 10 random

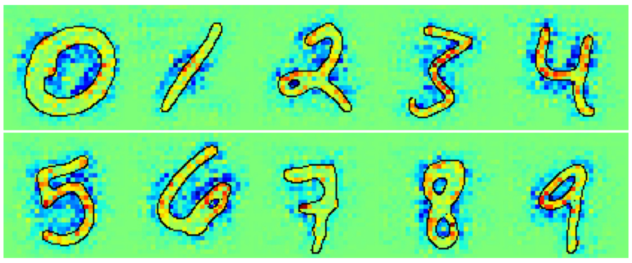


Fig. 11. Interpretability results for a random subset of MNIST examples using the guided backpropagation technique.

numbers of the MNIST dataset. Red pixels indicate pixels positively correlated with the output, in our case the input to the softmax layer (the unnormalized class score) and lowering the value of the red pixels will lead to a reduced class score. We observe that our method focuses on features that are unique for a specific number. This includes, for instance, the loop for the 6 and the top part of the 4. Rendering of the interpretability results overlaying the MNIST number was inspired by Lapuschkin, Binder, Montavon, Müller, and Samek (2016).

5. Conclusion and future work

In this paper, we proposed a novel approach to clustering, dubbed DDC, which (i) takes advantage of the power of deep learning architectures, (ii) is trainable end-to-end in a fully unsupervised way, (iii) does not require pre-training or complex feature design such as HOG (Dalal & Triggs, 2005) and SIFT (Lowe, 2004), and finally (iv) achieves results that outperform or are comparable with state-of-the-art methods on two real-world image datasets and a news story text dataset. We have also evaluated the performance of an ensemble DDC approach, which generally outperformed a single DDC model in the considered benchmarks. Overall, experimental results presented here are promising and stress the importance of unsupervised learning in modern deep learning methods.

In future works, we intend to study the robustness of our method. Further, we will explore alternative loss function formulations, including approaches that are not based on divergence measures and information-theoretic learning. Finally, it would be interesting to explore the use of the proposed method for related clustering settings, such as for instance multi-view clustering (Nie et al., 2018) and constraint clustering (Li et al., 2018).

Acknowledgments

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the GPU used for this research. This work was partially funded by the Research Council of Norway FRIPRO grant no. 239844 on developing the *Next Generation Learning Machines* and IKTPLUSS grant no. 270738 *Deep Learning for Health*. Thanks to the Research Council of Norway (UAVSEAL project, no. 234339) for funding, and the Institute of Marine Research, Norway, and Northwest Atlantic Fisheries Centre, Canada, for providing images and ground truth information about the location of the seal pups in the images.

References

Aggarwal, C. C., & Reddy, C. K. (2013). *Data clustering: Algorithms and applications*. Boca Raton, Florida, US: CRC Press.
Aragam, B., Dan, C., Ravikumar, P., & Xing, E. P. (2018). Identifiability of nonparametric mixture models and Bayes optimal clustering, arXiv preprint arXiv:1802.04397.
Arthur, D., & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics.
Basseville, M. (2013). Divergence measures for statistical data processing–an annotated bibliography. *Signal Processing*, *93*(4), 621–633. http://dx.doi.org/10.1016/j.sigpro.2012.09.003.
Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828. http://dx.doi.org/10.1109/TPAMI.2013.50.
Bianchi, F. M., Livi, L., & Rizzi, A. (2016). Two density-based k-means initialization algorithms for non-metric data clustering. *Pattern Analysis and Applications*, *19*(3), 745–763.
Bianchi, F. M., Maiorino, E., Kampffmeyer, M. C., Rizzi, A., & Jenssen, R. (2017). An overview and comparative analysis of recurrent neural networks for short term load forecasting, arXiv preprint arXiv:1705.04378.
Bojanowski, P., & Joulin, A. (2017). Unsupervised learning by predicting noise, arXiv preprint arXiv:1704.05310.
Comaniciu, D., & Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(5), 603–619.
Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer vision and pattern recognition, 2005. CVPR 2005. IEEE computer society conference on, Vol. 1* (pp. 886–893). IEEE.
Dhillon, I. S., Guan, Y., & Kulis, B. (2004). Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 551–556). ACM.
Dhillon, I. S., Mallela, S., & Kumar, R. (2003). A divisive information-theoretic feature clustering algorithm for text classification. *Journal of Machine Learning Research (JMLR)*, *3*(Mar), 1265–1287.
Erhan, D., Bengio, Y., Courville, A., Manzagol, P. -A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research (JMLR)*, *11*(Feb), 625–660.
Giryes, R., Sapiro, G., & Bronstein, A. M. (2016). Deep neural networks with random gaussian weights: A universal classification strategy? *IEEE Transactions on Signal Processing*, *64*, 3444–3457.
Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press, http://www.deeplearningbook.org.
Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., et al. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).
Graves, A., Mohamed, A. -r., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing* (pp. 6645–6649). IEEE.
Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., & Wierstra, D. (2015). Draw: A recurrent neural network for image generation, arXiv preprint arXiv:1502.04623.
Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., & Smola, A. (2012). A kernel two-sample test. *Journal of Machine Learning Research (JMLR)*, *13*(Mar), 723–773.
Han, Y., & Filippone, M. (2016). Mini-batch spectral clustering, arXiv preprint arXiv:1607.02024.
He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026–1034).
Hinton, G. E., Osindero, S., & Teh, Y. -W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*(7), 1527–1554.
Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167.
Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, *31*(8), 651–666.
Jenssen, R. (2010). Kernel entropy component analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(5), 847–860. http://dx.doi.org/10.1109/TPAMI.2009.100.
Jenssen, R., Erdogmus, D., Hild, K. E., Principe, J. C., & Eltoft, T. (2007). Information cut for clustering using a gradient descent approach. *Pattern Recognition*, *40*(3), 796–806.
Jenssen, R., Principe, J. C., Erdogmus, D., & Eltoft, T. (2006). The Cauchy–Schwarz divergence and Parzen windowing: Connections to graph theory and Mercer kernels. *Journal of the Franklin Institute*, *343*(6), 614–629.
Kampffmeyer, M., Løkse, S., Bianchi, F. M., Jenssen, R., & Livi, L. (2017). Deep kernelized autoencoders. In *Scandinavian conference on image analysis* (pp. 419–430). Springer.
Kampffmeyer, M., Løkse, S., Bianchi, F. M., Livi, L., Salberg, A. -B., & Jenssen, R. (2017). Deep divergence-based clustering. In *Proceedings of the IEEE workshop on machine learning for signal processing*. Tokyo, Japan.
Kampffmeyer, M., Salberg, A. -B., & Jenssen, R. (2016). Semantic segmentation of small objects and modeling of uncertainty in urban remote sensing images using deep convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops* (pp. 1–9).

Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes, arXiv preprint arXiv:1312.6114.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, *2*(1–2), 83–97.

Lapuschkin, S., Binder, A., Montavon, G., Müller, K. -R., & Samek, W. (2016). The lrp toolbox for artificial neural networks. *Journal of Machine Learning Research (JMLR)*, *17*(114), 1–5, URL http://jmlr.org/papers/v17/15-618.html.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Lee, C. -Y., Xie, S., Gallagher, P., Zhang, Z., & Tu, Z. (2015). Deeply-supervised nets. In *AISTATS, Vol. 2* (p. 6).

Lewis, D. D., Yang, Y., Rose, T. G., & Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research (JMLR)*, *5*(Apr), 361–397.

Li, Z., Nie, F., Chang, X., Nie, L., Zhang, H., & Yang, Y. (2018). Rank-constrained spectral clustering with flexible embedding. *IEEE Transactions on Neural Networks and Learning Systems*.

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3431–3440).

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, *60*(2), 91–110.

Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., & Frey, B. (2015). Adversarial autoencoders, arXiv preprint arXiv:1511.05644.

Montavon, G., Lapuschkin, S., Binder, A., Samek, W., & Müller, K. -R. (2017). Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, *65*, 211–222.

Myhre, J. N., Mikalsen, K. Ø., Løkse, S., & Jenssen, R. (2018). Robust clustering using a knn mode seeking ensemble. *Pattern Recognition*, *76*, 491–505.

Ng, A. Y., Jordan, M. I., Weiss, Y., et al. (2002). On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, *2*, 849–856.

Nguyen, A., Yosinski, J., & Clune, J. (2015). Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 427–436).

Nie, F., Tian, L., & Li, X. (2018). Multiview clustering via adaptively weighted procrustes. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 2022–2030). ACM.

Nie, F., Zeng, Z., Tsang, I. W., Xu, D., & Zhang, C. (2011). Spectral embedded clustering: A framework for in-sample and out-of-sample spectral clustering. *IEEE Transactions on Neural Networks*, *22*(11), 1796–1808.

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems (NIPS)*.

Rifai, S., Vincent, P., Muller, X., Glorot, X., & Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (pp. 833–840).

Rodriguez, A., & Laio, A. (2014). Clustering by fast search and find of density peaks. *Science*, *344*(6191), 1492–1496.

Sohn, K., Lee, H., & Yan, X. (2015). Learning structured output representation using deep conditional generative models. In *Advances in neural information processing systems* (pp. 3483–3491).

Springenberg, J. T. (2015). Unsupervised and semi-supervised learning with categorical generative adversarial networks, arXiv preprint arXiv:1511.0639.

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net, arXiv preprint arXiv:1412.6806.

Strehl, A., & Ghosh, J. (2002). Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research (JMLR)*, *3*(Dec), 583–617.

Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions, arXiv e-prints abs/1605.02688. URL http://arxiv.org/abs/1605.02688.

Tishby, N., & Slonim, N. (2001). Data clustering by Markovian relaxation and the Information Bottleneck Method. In *Advances in neural information processing systems, Vol. 13* (pp. 640–646).

Vikjord, V. V., & Jenssen, R. (2014). Information theoretic clustering using a k-nearest neighbors approach. *Pattern Recognition*, *47*(9), 3070–3081. http://dx.doi.org/10.1016/j.patcog.2014.03.018.

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P. -A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning* (pp. 1096–1103). ACM.

Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, *17*(4), 395–416.

Xie, J., Girshick, R., & Farhadi, A. (2016). Unsupervised deep embedding for clustering analysis. In *Proceedings of the 33rd international conference on international conference on machine learning, Vol. 48* (pp. 478–487). JMLR.org.

Yan, D., Huang, L., & Jordan, M. I. (2009). Fast approximate spectral clustering. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 907–916). ACM.

Yang, B., Fu, X., Sidiropoulos, N. D., & Hong, M. (2016). Towards k-means-friendly spaces: Simultaneous deep learning and clustering, arXiv preprint arXiv:1610.04794.

Yang, J., Parikh, D., & Batra, D. (2016). Joint unsupervised learning of deep representations and image clusters. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 5147–5156).

Yang, Y., Xu, D., Nie, F., Yan, S., & Zhuang, Y. (2010). Image clustering using local discriminant models and global integration. *IEEE Transactions on Image Processing*, *19*(10), 2761–2773.

Yu, S. X., & Shi, J. (2003). Multiclass spectral clustering. In *Computer vision, 2003. Proceedings. Ninth IEEE international conference on* (pp. 313–319). IEEE.

Zhang, X. -L. (2018). Multilayer bootstrap networks. *Neural Networks*, *103*, 29–43.

Paper V

Training Echo State Networks with Regularization Through Dimensionality Reduction

Sigurd Løkse, Filippo M. Bianchi and Robert Jenssen.

Cognitive Computation, 2017

Cogn Comput
DOI 10.1007/s12559-017-9450-z

Training Echo State Networks with Regularization Through Dimensionality Reduction

Sigurd Løkse[1] · Filippo Maria Bianchi[1] · Robert Jenssen[1]

Received: 7 September 2016 / Accepted: 2 January 2017

Abstract In this paper, we introduce a new framework to train a class of recurrent neural network, called Echo State Network, to predict real valued time-series and to provide a visualization of the modeled system dynamics. The method consists in projecting the output of the internal layer of the network on a lower dimensional space, before training the output layer to learn the target task. Notably, we enforce a regularization constraint that leads to better generalization capabilities. We evaluate the performances of our approach on several benchmark tests, using different techniques to train the readout of the network, achieving superior predictive performance when using the proposed framework. Finally, we provide an insight on the effectiveness of the implemented mechanics through a visualization of the trajectory in the phase space and relying on the methodologies of nonlinear time-series analysis. By applying our method on well-known chaotic systems, we provide evidence that the lower dimensional embedding retains the dynamical properties of the underlying system better than the full-dimensional internal states of the network.

✉ Filippo Maria Bianchi
filippo.m.bianchi@uit.no

Sigurd Løkse
sigurd.lokse@uit.no

Robert Jenssen
robert.jenssen@uit.no

1 Machine Learning Group, Department of Physics and Technology, University of Tromsø - The Arctic University of Norway, Tromsø, Norway

Keywords Echo state network · Nonlinear time-series analysis · Dimensionality reduction · Time-series prediction

Introduction

In order to solve a multitude of complex tasks, neurons in the prefrontal and parietal cortices participate to multiple ensembles, showing each time with different behaviors in their activity [64]. Beside this heterogeneity in neurons responses, to further increase the dimensionality of the firing rate space, neurons activities are related through non-linear dependencies. This is analog to the blessing of dimensionality in computational learning theory and it is formalized by Cover's theorem [65], which states that inputs implicitly mapped by non-linear functions (kernels) into high-dimensional space become linearly separable. Accordingly, when external stimuli are mapped into such a high dimensional representation, a large number of task-related responses can be solved by using a linear readout [66]. Consider now that the input data naturally lay on a low-dimensional manifold, in this high-dimensional response space and assume the presence of noise. This latter produces small variations along each possible direction, with a consequent increment of the minimum number of dimensions necessary to embed the data manifold. Also, due to its non-repeatability, noise generates additional components, which are useless to obtain the desired separability on different realizations. This unwanted increment in dimensionality reduces the discriminability along the directions where noise introduces these small variations and it must be contrasted with a form of regularization [64]. Therefore, the brain must perform two opposing tasks, which are projecting the inputs into a high dimensional space where they are separable and reducing the dimensionality to get rid of non relevant factors. To implement and to study the effectiveness of this strategy, we analyze a biologically inspired, artificial, recurrent neural network, called Echo State Network (ESN). ESNs belong to the class of computational dynamical systems, implemented according to the so-called reservoir computing approach [40]. An input signal is fed to a large, recurrent, and randomly connected dynamic hidden layer, the *reservoir*, whose outputs are combined by a memory-less layer called *readout* to solve a specified task. Contrary to most computationally expensive approaches, which demand long training procedures to learn model parameters through an optimization algorithm [28], ESN is characterized by a very fast learning phase that usually consists in solving a convex optimization problem.

Standard ESNs and their variants [42, 44] have been adopted in a variety of different contexts, such as static classification [1], time-series classification [41], speech recognition [55], intrusion detection [23], adaptive control [24] harmonic distortion measurements [46] and, in general, for modeling of various kinds of non-linear dynamical systems [25]. The application field where ESN has been employed the most is the prediction of real valued time-series relative, for example, to telephonic or electric load, where the forecast is usually performed 1 and 24 h ahead [6, 15, 16, 49, 59]. Outstanding results have also been achieved by ESN in prediction of chaotic time-series [32, 37], which highlighted the capability of these neural networks to learn amazingly accurate models to forecast a chaotic process from almost noise-free training data.

Although a large reservoir provides a greater descriptive power and it could capture the dynamics of the underlying system more accurately, the resulting model is more complex and its internal dynamics become more redundant. This could lead to overfit the training data, with a consequent decrement in the generalization capabilities of the model. Additionally, having a large number of reservoir neurons means that high-dimensional data have to be handled in training the readout layer. This could raise the well-known curse of dimensionality issue, which causes increments in both the computational requirements and the resource needed, as the classification/regression model requires a higher amount of data to be trained [7]. To damper the high variance of such models characterized by a large complexity, regularization techniques are required.

Dimensionality reduction is a form of regularization which, beside being implemented by biological brains [67], has been adopted in several signal processing and machine learning applications, to evaluate regression functions [63], performing classification [14], or finding neighbors [29]. In fact, by operating on a space with reduced dimensionality, it is possible in many cases to maintain meaningful distance relationships between original data and to constraint the complexity of the model the same time. Through dimensionality reduction, redundant features are removed, noise can be filtered, and algorithms that are unfit to deal with a large number of dimensions become applicable.

In the ESN literature, different regularization methods have been proposed to increase the generalization capability of the network. For example, in [17], the authors introduce a bias in the model by shrinking the weights of the connections from the reservoir to the readout layer. In [52], by pruning some connections from the reservoir to the readout layer, better generalization capabilities are achieved along with some insight on which neurons are actually useful for the output, providing clues on how to create a good reservoir.

Inspired by the strategy of biological neural networks that allow to achieve better generalization by enforcing invariance to inputs that should not influence the desired response, we propose a novel framework for training an ESN. We introduce an additional computational block to process the output of the internal reservoir, before being fed into the readout layer. In particular, the internal state of the network is mapped to a lower dimensional subspace, using both linear and nonlinear transformations. In this way, we are able to use a large reservoir to accurately model the dynamics of the underlying system, while maintaining the generalization capabilities of the network, due to regularization constraints provided by dimensionality reduction on its internal phase space.

Even if additional operations are introduced to compute the reduced dimensionality embedding, training the readout layer becomes less demanding, especially in regression methods whose computational complexity depends on input dimension [18]. We stress that the dimensionality reduction treated here is intended to deal with sparsity in the state space, i.e., the actual number of parameters which are necessary to describe the evolution of the dynamics of the system. We do not focus on dimensionality reduction in the input space, since it is not an issue in ESNs. In fact, to better separate the data, the ESN maps the input into a high-dimensional kernel space, which is even more sparse. Through this mapping, dependencies in the data are more likely to become linearly separable, even in the original input space, the dynamics are linked through complex, nonlinear relationships.

In this paper, we show how through the proposed procedure we improve the generalization capabilities of an ESN, achieving better results on well-known benchmarking problems with respect to the standard ESN architecture. Additionally, in cases where data can be mapped to spaces with 2 or 3 dimensions, internal network dynamics can be visualized precisely and relevant patterns can be detected. To justify the results obtained and to understand the mechanisms which determines the effectiveness of the proposed system, we provide a theoretical study based on methods coming from the field of nonlinear time-series analysis. To the best of the authors' knowledge, the coupling of dimensionality reduction with the ESN architecture has not been explored before. However, a recent work is pointing in this direction, while focusing more on memory properties [13].

The remainder of the paper is organized as follows. In the "Background Material" section, we describe the ESN structure along with existing approaches for its training and we review the dimensionality reduction methods adopted in this work. In the "Proposed Architecture" section, we discuss the details of the proposed architecture. In the "Experiments" section, we describe the datasets used to test our system, the experimental settings adopted and the performance reached on several prediction problems. In the "Discussion" section, we analyze the results and the functioning of our system through the perspective of nonlinear time-series analysis. Finally, in the "Conclusions and Future Directions" section, we draw our conclusions.

Background Material

In the following, we shortly review the methodologies adopted in our framework. Initially, we describe the classic ESN architecture and two effective approaches adopted for its training. Successively, we summarize two well-known methods used for reducing the dimensionality of the data and for mapping them in a smaller subspace.

Echo State Network

An ESN consists of a large, untrained recurrent layer of non-linear units and a linear, memory-less readout layer, usually trained with a linear regression. A visual representation of an ESN is reported in Fig. 1.

The equations describing the ESN state-update and output are, respectively, defined as follows:

$$\mathbf{h}[k] = \phi(\mathbf{W}_r^r\mathbf{h}[k-1] + \mathbf{W}_i^r\mathbf{x}[k] + \mathbf{W}_o^r\mathbf{y}[k-1] + \xi), \quad (1)$$

$$\mathbf{y}[k] = \mathbf{W}_i^o\mathbf{x}[k] + \mathbf{W}_r^o\mathbf{h}[k], \quad (2)$$

where ξ is a small i.i.d. noise term. The reservoir consists of N_r neurons characterized by a transfer/activation function $\phi(\cdot)$, typically implemented as a hyperbolic tangent function. At time instant k, the network is driven by the input signal $\mathbf{x}[k] \in \mathbb{R}^{N_i}$ and it generates the output $\mathbf{y}[k] \in \mathbb{R}^{N_o}$, being N_i and N_o the dimensionality of input and output, respectively. The vector $\mathbf{h}[k] \in \mathbb{R}^{N_r}$ describes the ESN (instantaneous) state. The weight matrices $\mathbf{W}_r^r \in \mathbb{R}^{N_r \times N_r}$ (reservoir connections), $\mathbf{W}_i^r \in \mathbb{R}^{N_r \times N_i}$ (input-to-reservoir), and $\mathbf{W}_o^r \in \mathbb{R}^{N_r \times N_o}$ (output-to-reservoir feedback) contain real values in the $[-1, 1]$ interval, sampled from a uniform distribution.

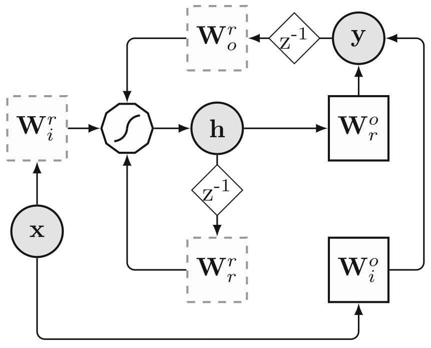


Fig. 1 Schematic depiction of the ESN architecture. The *circles* represent input **x**, state, **h**, and output, **y**, respectively. *Solid squares* $\mathbf{W}_r^o$ and $\mathbf{W}_i^o$, are the trainable matrices of the readout, while *dashed squares*, $\mathbf{W}_r^r$, $\mathbf{W}_o^r$, and $\mathbf{W}_i^r$, are randomly initialized matrices. The *polygon* represents the nonlinear transformation performed by neurons and z^{-1} is the unit delay operator

According to the ESN theory, the reservoir $\mathbf{W}_r^r$ must satisfy the so-called echo state property (ESP) [40]. This guarantees that the effect of a given input on the state of the reservoir vanish in a finite number of time intervals. A widely used rule-of-thumb suggests to rescale the matrix $\mathbf{W}_r^r$ to have $\rho(\mathbf{W}_r^r) < 1$, where $\rho(\cdot)$ denotes the spectral radius, but several theoretically founded approaches have been proposed in the literature to properly tune ρ in an ESN driven by a specific input [8, 9, 60].

The weight matrices $\mathbf{W}_i^o$ and $\mathbf{W}_r^o$ instead are optimized for the task at hand. To determine them, let us consider the training sequence of T_{tr} desired input-outputs pairs given by

$$(\mathbf{x}[1], y^*[1]) \ldots, (\mathbf{x}[T_{tr}], y[T_{tr}]), \quad (3)$$

In the initial phase of training, called *state harvesting*, the inputs are fed to the reservoir in accordance with Eq. 1, producing a sequence of internal states $\mathbf{h}[1], \ldots, \mathbf{h}[T_{tr}]$. Since, by definition, the outputs of the ESN are not available for feedback, according to the *teacher forcing* procedure, the desired output is used instead in Eq. 2. States are stacked in a matrix $\mathbf{S} \in \mathbb{R}^{T_{tr} \times N_i + N_r}$ and the desired outputs in a vector $\mathbf{y}^* \in \mathbb{R}^{T}_{tr}$:

$$\mathbf{S} = \begin{bmatrix} \mathbf{x}^T[1], & \mathbf{h}^T[1] \\ & \vdots \\ \mathbf{x}^T[T_{tr}], & \mathbf{h}^T[T_{tr}] \end{bmatrix}, \; \mathbf{y}^* = \begin{bmatrix} y^*[1] \\ \vdots \\ y^*[T_{tr}] \end{bmatrix}.$$

The initial D rows $\mathbf{S}$ and $\mathbf{y}^*$ are the washout elements that should be discarded, since they refer to a transient phase in the ESN's behavior.

Since the gain of the sigmoid nonlinearity in the neurons is the largest around the origin, three coefficients ω_i, ω_o, and ω_f are used to scale the input, desired output, and feedback signals, respectively. In this way, it is possible to control the amount of nonlinearity introduced by the processing units.

The training of the readout consists in solving a convex optimization problem, for which several closed form solution have been proposed in the literature. The standard procedure to train the readout, originally proposed in [30], consists in a regularized least-square regression, which can be easily computed through the Moore–Penrose pseudo-inverse. However, to learn the optimal readout, we also consider the support vector regression (SVR), a supervised learning model that can efficiently perform a nonlinear separation of data using a kernel function to map the inputs into high-dimensional feature spaces, where they are linearly separable [11].

Ridge Regression to train the readout with a linear regressor, we adopted ridge regression, whose solution can be computed by solving the following regularized least-square problem:

$$\begin{aligned}\mathbf{W}^*_{\text{ls}} &= \underset{\mathbf{W}\in\mathbb{R}^{N_i+N_r}}{\arg\min}\ \frac{1}{2}\|\mathbf{S}\mathbf{W}-\mathbf{y}^*\|^2+\frac{\lambda}{2}\|\mathbf{W}\|^2\\ &= \left(\mathbf{S}^T\mathbf{S}+\lambda\mathbf{I}\right)^{-1}\mathbf{S}^T\mathbf{y}^*\,,\end{aligned}\tag{4}$$

where $\mathbf{W}=\left[\mathbf{W}_i^o\ \mathbf{W}_r^o\right]^T$ and $\lambda\in\mathbb{R}^+$ is the L$_2$ regularization coefficient.

Support Vector Regression we adopt a ν-SVR [54] with a Gaussian kernel, initially proposed in [7] as method for readout training. In this case, the ESN acts as a preprocessor for a ν-SVR kernel and their combination can be seen as an adaptive kernel, capable of learning a task-specific time dependency. The state $\mathbf{s}_i=\left[\mathbf{x}^T[i]\ \mathbf{h}^T[i]\right]^T$ is projected to a higher dimensional feature space $\phi(\mathbf{s}_i)$, and the ν-SVR is applied on the resulting space. The dual optimization problem can be written as:

$$\mathbf{W}^*_{\text{svr}}=\begin{cases}\underset{\boldsymbol{\alpha},\boldsymbol{\alpha}^*\in\mathbb{R}^{T_{tr}}}{\min} & \frac{1}{2}\left(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*\right)\mathbf{K}\left(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*\right)+\mathbf{y}^{*T}\left(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*\right)\\ \text{subject to} & \mathbf{1}^T\left(\boldsymbol{\alpha}-\boldsymbol{\alpha}^*\right)=0\,,\\ & \mathbf{1}^T\left(\boldsymbol{\alpha}+\boldsymbol{\alpha}^*\right)\le\lambda\nu\,,\\ & 0\le\alpha_i,\alpha_i^*\le\frac{\lambda}{T_{tr}},\ i=\ldots,T_{tr}\end{cases}\tag{5}$$

where each entry K_{ij} is given by $\mathcal{K}\left(\mathbf{s}_i,\mathbf{s}_j\right)$, with $\mathcal{K}(\cdot,\cdot)$ being a reproducing Gaussian kernel associated with the feature mapping, given by $\mathcal{K}(\mathbf{s}_i,\mathbf{s}_j)=\exp\left\{-\gamma\|\mathbf{s}_i-\mathbf{s}_j\|^2\right\}$, where γ is denoted as the scale parameter.

By an extension of the representer's theorem, the output of the ESN at a generic time-instant t in this case is given by

$$y[t]=\sum_{i=1}^{T_{tr}}\left(\alpha_i-\alpha_i^*\right)\mathcal{K}\left(\mathbf{s}_i,\mathbf{s}_t\right)\,,\tag{6}$$

where α_i and α_i^* are the entries of the optimal solution to problem Eq. 5, and they are non-zero only for patterns that are support vectors.

Dimensionality Reduction Methods

In the following, we describe the dimensionality reduction techniques that we implemented in our framework. We underline that several approaches can be followed for reducing the dimensionality of the data and to learn underlying manifold on a subspace of the data space [34, 35, 43, 58]. In this work, we limit our analysis to the well-known and effective, yet simple procedures, namely principal component analysis (PCA) [27] and kernel principal component analysis (kPCA) [53]. We stress that the dimensionality reduction method to be included in our framework must implement out of sample extensions. This is not guaranteed by the original formulation of more complex methods, such as locally linear embedding, Laplacian eigenmaps, and multidimensional scaling. For these approaches, one has to rely on specific workarounds [5]. Additionally, we believe that using more advanced/complicated methods could hinder the interpretability of the results, as they could be dependent on some complex, hidden mechanism in the dimensionality reduction method.

PCA is a statistically motivated method, which projects the data onto an orthonormal basis that preserves most variance in the input signal, while ensuring that the individual components are uncorrelated. These basis vectors are called the *Principal Components*. Let $\mathbf{X}\in\mathbb{R}^p$ be a random vector and let $\mathbf{\Sigma}_X=\mathbf{E}\mathbf{\Lambda}\mathbf{E}^T$ be its covariance matrix, where $\mathbf{E}=\left(\mathbf{e}_1\mathbf{e}_2\cdots\mathbf{e}_p\right)$ and $\mathbf{\Lambda}=\operatorname{diag}\left(\lambda_i\right)$ is the orthogonal eigenvector matrix and the diagonal eigenvalue matrix, respectively. Then, the linear transformation $\mathbf{Y}=\mathbf{E}^T\mathbf{X}$ ensures that the covariance matrix of $\mathbf{Y}$ is $\mathbf{\Sigma}_Y=\mathbf{\Lambda}$, which clearly implies that the components of $\mathbf{Y}$ are uncorrelated. We also see that

$$\sum_{i=1}^p VarX_i=\sum_{i=1}^p VarY_i=\sum_{i=1}^p\lambda_i.\tag{7}$$

To reduce the dimensionality to d dimensions, we project the data onto the d eigenvectors with the largest eigenvalues. That is,

$$\widehat{Y}=\mathbf{E}_d^T\mathbf{X},$$

where $\mathbf{E}_d=\left(\mathbf{e}_1\quad\mathbf{e}_2\cdots\mathbf{e}_d\right)$ is the *truncated* eigenvector matrix associated with the eigenvalues $\lambda_1\ge\lambda_2\ge\cdots\ge\lambda_d$. According to Eq. 7, this ensures that $\widehat{Y}$ preserves most of the variance of $\mathbf{X}$.

Kernel Principal Component Analysis (kPCA) is a nonlinear extension of PCA. Given a valid positive semidefinite (psd) Mercer Kernel

$$\mathcal{K}(\mathbf{h}[i], \mathbf{h}[j]) = \langle \Phi(\mathbf{h}[i]), \Phi(\mathbf{h}[j]) \rangle_{\mathcal{H}},$$

where Φ is some nonlinear mapping from feature space to a Hilbert space $\mathcal{H}$, Kernel PCA implicitly performs PCA in $\mathcal{H}$.

Let $\mathbf{K} = \{K_{ij}\}_{N \times N}$, where $K_{ij} = \mathcal{K}(\mathbf{h}[i], \mathbf{h}[j])$ be the *kernel matrix* and let $\mathbf{E}$ and $\boldsymbol{\Lambda}$ be its eigenvector and eigenvalue matrix respectively with the eigenvalues sorted in descending order. Then, the projection of the in-sample data onto the principal components in $\mathcal{H}$ is given by

$$\bar{\mathbf{H}} = \mathbf{E}\boldsymbol{\Lambda}^{\frac{1}{2}}. \tag{8}$$

The out-of-sample approximation for the projection of a data point $\mathbf{h}[k]$ onto the ℓth principal component is given by

$$\bar{h}_\ell[k] = \frac{1}{\sqrt{\lambda_\ell}} \sum_{i=1}^{N} e_\ell(i) \mathcal{K}(\mathbf{h}[i], \mathbf{h}[k]). \tag{9}$$

Just like canonical PCA, to perform dimensionality reduction with kPCA, one need to use the truncated eigenvector- and eigenvalue matrix with Eqs. 8 and 9.

The kernel function that is commonly used in practice is the *Gaussian kernel* which is given by $\mathcal{K}(\mathbf{h}[i], \mathbf{h}[j]) = \exp\left\{-\gamma \|\mathbf{h}[i] - \mathbf{h}[j]\|^2\right\}$, where γ controls the width of the kernel.

Both PCA and kPCA methods admit an out-of-sample extension, a feature which is required in our framework, as discussed later.

Proposed Architecture

In this section, we provide the details of the architecture of the framework proposed.

The large size of the reservoir, specified by the amount N_r of hidden neurons, is one of the main features that determines the effectiveness of the reservoir computing paradigm. Due to the high quantity of neurons, the internal recurrent connections in the reservoir are capable of generating a rich and heterogeneous dynamic to solve complex memory-dependent tasks. However, as the size of the reservoir increases, also the complexity of the model grows, with a consequent risk of overfitting and a reduced generalization capability [4]. Dimensionality reduction and manifold learning are techniques that allow to diminish the variance in the data and to introduce a bias, which can reduce the expected value on the prediction error [20]. In the architecture proposed, we use a large reservoir in order to capture the dynamic of the underlying unknown process and then, through a dimensionality reduction procedure, we enforce regularization constraints to increase the generalization capability of our model. Another important consequence that follows from reducing the dimensionality of the reservoir is that complex regression methods can benefit from a reduced computational complexity if the internal states are described by a lower number of variables. Details on the computational complexity for different configurations of the architecture are given in Table 1. Additionally, several methods used to identify, in an unsupervised way, the configurations of hyperparameters which maximize the computational capabilities of the network, require computational demanding procedures of analysis [8, 9, 39]. These procedures would greatly benefit from the simplification offered by our proposed architecture.

At each time step t, the vector $\mathbf{h}[t] \in \mathbb{R}^{N_r}$ that represents the internal state of the reservoir is mapped into a lower dimensional space by a projector $\mathcal{P} : \mathbb{R}^{N_r} \to \mathbb{R}^d$. The new d-dimensional state vector $\bar{\mathbf{h}}[t] = \mathcal{P}(\mathbf{h}[t])$ is then processed by the readout to compute the predicted value $\mathbf{y}[t]$.

To train our system, the time-series is split in three contiguous parts, namely the training $\{\mathbf{X}_{tr}, \mathbf{Y}_{tr}\}$, validation $\{\mathbf{X}_{vs}, \mathbf{Y}_{vs}\}$ and test set $\{\mathbf{X}_{ts}, \mathbf{Y}_{ts}\}$. Since we deal with time-series prediction problems, each set contains coupled real values, which represent the input value and the ground truth of the associated prediction. For example, in the training set, we have $\{\mathbf{x}[t], \mathbf{y}[t]\}_{t=1}^{T_{tr}}$, where $\mathbf{y}[t]$ is the predicted value of $\mathbf{x}[t]$. The regression function in the readout is implemented according to one of the two procedures proposed in the

Table 1 Comparison of computational complexity (C.C.) and memory complexity (M.C.). We indicate the dominating operation which determines the complexity

Model	C.C. —Dimensionality reduction	C.C. —Training readout	M.C.
Ridge reg. ESN	–	Pseudo-inverse $\mathcal{O}(N_r^3)$	Empirical covariance matrix $\mathcal{O}(N_r^2)$
Ridge reg. PCA	Eigendecomposition $\mathcal{O}(N_r^3)$	Pseudo-inverse $\mathcal{O}(d^3)$	Empirical covariance matrix $\mathcal{O}(N_r^2)$
Ridge reg. kPCA	Eigendecomposition $\mathcal{O}(T_{tr}^3)$	Pseudo-inverse $\mathcal{O}(d^3)$	Kernel matrix $\mathcal{O}(T_{tr}^2)$
ν-SVR ESN	–	QP Solver $\mathcal{O}(T_{tr}^3)$	Kernel matrix $\mathcal{O}(T_{tr}^2)$

The parameters are as follows: The number of neurons in the network (N_r), the dimensionality after the dimensionality reduction layer (d), and the number of samples in the training set (T_{tr}).

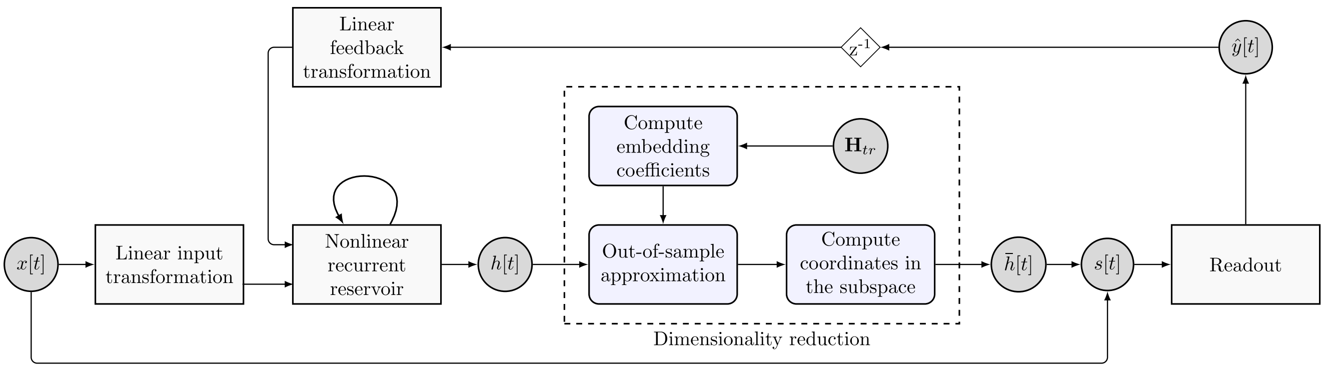

Linear feedback transformation
z^{-1}
$\hat{y}[t]$
Compute embedding coefficients
$\mathbf{H}_{tr}$
$x[t]$
Linear input transformation
Nonlinear recurrent reservoir
$h[t]$
Out-of-sample approximation
Compute coordinates in the subspace
$\bar{h}[t]$
$s[t]$
Readout
Dimensionality reduction

Fig. 2 When a new element $\mathbf{x}[t]$ is fed into the network, the internal state of the ESN is updated and its new value is stored in $\mathbf{h}[t]$. Such state vector is then projected on a subspace, computed during the training on the state matrix $\mathbf{H}_{tr}$ and the vector of reduced dimensionality in this subspace $\bar{\mathbf{h}}[t]$ is evaluated. At this point, the predicted output value $\hat{\mathbf{y}}[t]$ is computed by the ESN readout

"Echo State Network" section and the model parameters are learned on the training data. The system depends on several hyperparameters, which affects the network behavior and they must be carefully tuned on the specific problem at hand by performing a cross-validation procedure on the validation set, with a method whose details are provided in the next section.

Once the model has been trained, a new test element $\mathbf{x}[t]$ of the test set is processed and the relative internal reservoir state $\mathbf{h}[t]$ is generated. Successively, the projection $\bar{\mathbf{h}}[t]$ in the subspace with reduced dimensionality is evaluated using a suitable out of sample approximation. In the case of PCA, this can be done by projecting $\mathbf{h}[t]$ on the basis defined by the covariance matrix computed on the T_{tr} states relative to the elements in training set, which are collected in the matrix $\mathbf{H}_{tr}$ during the training phase. For kPCA, it is possible to use the Nÿstrom approximation [2], which specifies an interpolating function for determining the values of out-of-sample data points.

A schematic representation of the whole procedure is depicted in Fig. 2.

Hyperparameter Optimization

The set of hyperparameters $\boldsymbol{\theta}$ that are used to control the architecture of the ESN, the regression in the readout, and the dimensionality reduction procedure are optimized by minimizing a loss function $L(\cdot)$, defined as

$$L(\boldsymbol{\theta}_i) = (1-\alpha)Err(\mathbf{Y}_{vs}) + \alpha\theta_i^{(\mathrm{d})}, \tag{10}$$

where $\theta^{(\mathrm{d})} = \frac{d}{N_r}$ is the hyperparameter that defines the number of dimensions, d, of the new subspace. In order to lower the complexity of the model, $L(\cdot)$ jointly penalizes prediction error on the validation set and the number of dimensions retained after the dimensionality reduction.

The loss function is minimized using a standard genetic algorithm with Gaussian mutation, random crossover,

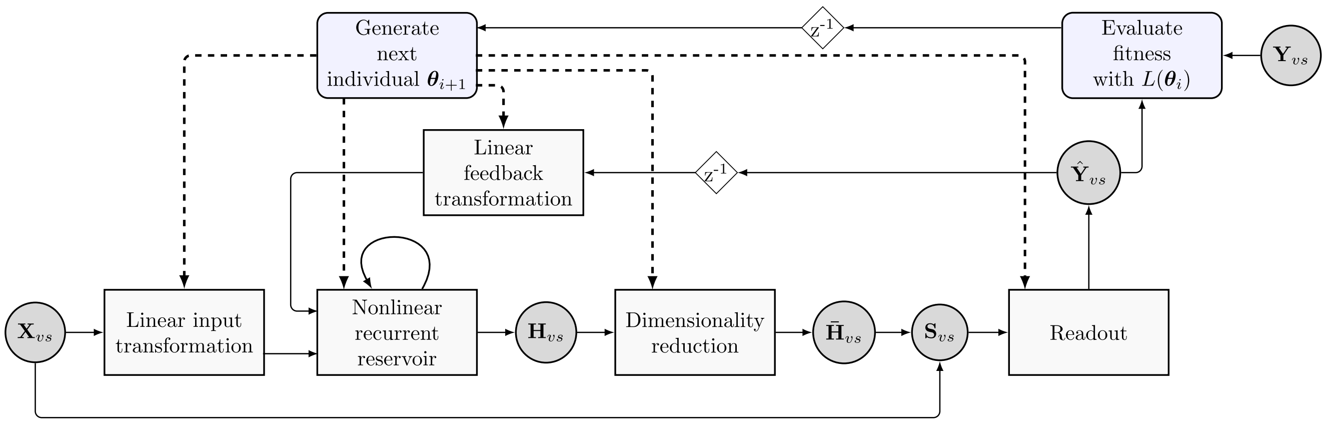

Generate next individual $\boldsymbol{\theta}_{i+1}$
z^{-1}
Evaluate fitness with $L(\boldsymbol{\theta}_i)$
$\mathbf{Y}_{vs}$
Linear feedback transformation
z^{-1}
$\hat{\mathbf{Y}}_{vs}$
$\mathbf{X}_{vs}$
Linear input transformation
Nonlinear recurrent reservoir
$\mathbf{H}_{vs}$
Dimensionality reduction
$\bar{\mathbf{H}}_{vs}$
$\mathbf{S}_{vs}$
Readout

Fig. 3 Overview of the hyperparameters optimization in the proposed architecture. At the i-th iteration, the input elements of the validation set $\mathbf{X}_{vs}$ are processed by the ESN configured with the hyperparemeters in $\boldsymbol{\theta}_i$, which is the i-th individual generated by the GA. The predicted output $\hat{\mathbf{Y}}_{vs}$ produced by the network is matched against the ground truth $\mathbf{Y}_{vs}$, the resulting similarity (prediction error) is used to compute the fitness of $\boldsymbol{\theta}_i$ with the loss function $L(\boldsymbol{\theta}_i)$. In the next iteration, a new individual $\boldsymbol{\theta}_{i+1}$ is generated, depending on results obtained so far and on the policies of the GA

Table 2 Each hyperparameter is searched by the GA in the interval [*min*, *max*] with resolution σ

	$\mathrm{N_r}$	$\boldsymbol{\xi}$	$\boldsymbol{\omega_i}$	$\boldsymbol{\omega_o}$	$\boldsymbol{\omega_f}$	$\boldsymbol{\rho}$	$\boldsymbol{\theta}^{(\mathrm{d})}$	$\boldsymbol{\gamma}$	$\boldsymbol{\lambda}$	**C**	$\boldsymbol{\nu}$
min	100	0.0	0.1	0.1	0.0	0.5	0.001	0.001	0.001	0.001	0.001
max	500	0.1	0.9	0.9	0.6	1.4	1.0	0.1	1.0	10.0	1.0
σ	5	0.01	0.08	0.08	0.06	0.09	0.1	0.01	0.1	1.0	0.1

The fields in the table are the following: spectral radius of the reservoir (ρ), neurons in the reservoir (N_r), noise in ESN state update (ξ), scaling of input, teacher and feedback weights (ω_i, ω_o, ω_f), embedding dimension $\left(\theta^{(\mathrm{d})} = \frac{d}{N_\mathrm{r}}\right)$, $\mathrm{L_2}$ norm regularization factor (λ), and ν-SVR parameters (C, γ, ν).

elitism, and tournament selection [56]. While the hyperparameter optimization is performed on the validation set, the best individual found is stored and it is successively used to configure the network during the training phase. A schematic description of the training procedure is depicted in Fig. 3.

Experiments

The component of the loss function (Eq. 10) relative to the error on the given task is implemented by the normalized root mean squared error (NRMSE):

$$\mathrm{NRMSE} = \sqrt{\frac{\langle \|\mathbf{y}[k] - \mathbf{y}^*[k]\|^2\rangle}{\langle \|\mathbf{y}[k] - \langle \mathbf{y}^*[k]\rangle\|^2\rangle}},$$

where $\mathbf{y}[k]$ is the ESN prediction and $\mathbf{y}^*[k]$ the desired/teacher output.

The GA uses a population size of 50 individuals and evaluates 20 generations. The individuals are mutated and bred at each generation with a mutation probability of $P_{\mathrm{mut}} = 0.2$ and a crossover probability of $P_{\mathrm{cx}} = 0.5$. The individuals in the next generation are selected by a tournament strategy with a tournament size of four individuals. The bounds for all parameters are shown in Table 2. The weight parameter α in the loss function (Eq. 10) is set to 0.1.

Due to the stochastic nature of the ESN, which is a consequence of the random initialization of the weight matrices $\mathbf{W}_i^r$, $\mathbf{W}_r^r$ and $\mathbf{W}_o^r$, each individual is evaluated on the validation set using five networks initialized with different weight parameters. The fitness is then given by the NRMSE, averaged over these five networks. Once the optimal set of parameters $\boldsymbol{\theta}^*$ has been found, we predict values for the test set using 32 randomly initialized networks, using the same set of optimal parameters.

Datasets Description

To test our system, we consider three benchmark tasks commonly used in time-series forecasting, namely the prediction of Mackey-Glass time-series, of multiple superimposed oscillator and of the NARMA signal. The forecasting problems that we consider have a different level of difficulty, given by the nature of the signal and the complexity of the prediction task. Accordingly to a commonly used approach [33], in each prediction task, we set the forecast step τ_f by computing a statistic that measures the independence of τ_f-separated points in the time series. One usually wants the smallest τ_f that guarantees the measurements to be decorrelated. Hence, we considered the first zero of the autocorrelation function of the time series, which yields the smallest τ_f that maximizes the linear independence between the samples. Alternatively, it is possible to choose the forecast step by considering more general forms of independence, such as the first local minimum on the average mutual information [19] or on the correlation sum [38].

Mackey-Glass Time-series the input signal is generated from the Mackey-Glass (MG) time-delay differential system, described by the following equation:

$$\frac{dx}{dt} = \frac{\alpha x(t-\tau_{\mathrm{MG}})}{1 + x(t-\tau_{\mathrm{MG}})^{10}} - \beta x(t).$$

We generated a time-series of 150,000 time-steps using $\tau_{\mathrm{MG}} = 17$, $\alpha = 0.2$, $\beta = 0.1$, initial condition $x(0) = 1.2$, and 0.1 as integration step for (4.1).

NARMA Signal the non-linear autoregressive moving average (NARMA) task, originally proposed in [31], consists in modeling the output of the following r-order system:

$$y(t+1) = 0.3y(t) + 0.05y(t)\left[\sum_{i=0}^{r} y(t-i)\right] + 1.5x(t-r)x(t) + 0.1.$$

The input to the system $x(t)$ is a uniform random noise in [0, 1], and the model is trained to reproduce $y(t+1)$. The NARMA task is known to require a memory of at least r past time-steps, since the output is determined by the current input and outputs relative to the last r time-steps. In our test, we set $r = 20$.

Multiple Superimposed Oscillator The prediction of a sinusoidal signal is a relatively simple task, which demands a minimum amount of memory to determine the next network output. However, superimposed sine waves with not integer frequencies are much harder to predict, since the wavelength can be extremely long. The signal we consider is the multiple superimposed oscillator (MSO), studied in [32] and defined as

$$\begin{aligned} y(t) = & \, sin(0.2t) + sin(0.311t) + sin(0.42t) + sin(0.51t) \\ & + sin(0.63t) + sin(0.74t) \end{aligned}$$

ESN struggles to solve this task, since neurons in the reservoir tends to couple, while the task requires the simultaneous existence of multiple decoupled internal states [61].

Results

The averaged prediction results and the standard deviations are reported in Table 3. The convergence rate during the optimization of the hyperparameters for each method, expressed as the NRMSE error on the validation set, is depicted in Fig. 4.

The prediction of MG is a quite simple task and each model manages to achieve high forecast accuracy. However, by applying a dimensionality reduction on the states of the reservoir, it is possible to lower the error by one or more order of magnitude. Also, the standard deviation of the prediction error decreases, especially in the models using kPCA. The best results are achieved by ν-SVR + PCA and ν-SVR + kPCA, while using ν-SVR without reducing the dimensions of the reservoir demonstrated to be less effective. This means that nonlinearities benefit the training, but without enforcing the regularization constraint, the complexity of the model is too high to fit well testing points. As we can see, in every case, the number of dimensions d retained by both PCA and kPCA is much lower than the optimal number of neurons N_r identified. This underlines the effectiveness of the regularization conveyed by our architecture. From Fig. 4a results that the model implementing ridge regression + kPCA achieves the lowest convergence rate during the cross-validation step. However, thanks to the generalization power provided by the nonlinear dimensionality reduction, the test error is lower than the other models, whose readout is trained with ridge regression.

In NARMA prediction task, the best result is achieved by training the readout function with ν-SVR on a reservoir output, whose dimensionality is reduced by kPCA. NARMA is a more complex task which requires a higher amount on nonlinearity to be solved. This is clearly reflected by the results, which improve as more nonlinearity is introduced to learn the function, both in the readout training and in the dimensionality reduction procedure. At the same time, the bias introduced by the regularization enhance the generalization capability of the network significantly. For what concerns the number of dimensions of the optimal subspace, it is higher than in MG task, except for the model implemented with ridge regression + PCA. In this latter cases, however, we obtain the worst performance. Interestingly, from Fig. 4b, we observe that kPCA has the lower convergence rate, even if this is the best performing model in the testing phase. In this case, the dimensionality reduction introduces a bias, which prevents the model to overfit on the validation data and to develop a high predictive power. On the other hand, the model with ν-SVR and no dimensionality reduction overfits on the validation data with a consequent poor performance in the test phase.

Finally, in the MSO task, the model with the highest prediction performance is ν-SVR without the dimensionality reduction. In this case, the signal to predict has an infinitely long periodicity, which benefits from a network model characterized by a high complexity, a large amount of memory. Hence, the compression of the information through the dimensionality reduction could hamper the memory capacity of the network. Furthermore, due to the long periodicity, the slice of time-series used to train the network can be quite different from the slice to be predicted in the test. Consequently, test points are projected in a subspace which is not optimal, as the basis is learned from the training data. As expected, the number of dimensions kept after the dimensionality reduction is larger than in the other tasks. The need of a high degree of complexity is also denoted by the poor results obtained by using ridge regression in the readout training. From Fig. 4c, we observe the convergence rate to be faster in models equipped with ν-SVR, which obtain better results both in validation and in testing phase. This symmetry on performances on test and validation reflects the scarce effectiveness of the regularization constraints for this task.

Discussion

To understand the mechanics and the effectiveness of the proposed architecture, we analyze the results through the theory of nonlinear time-series analysis, which offer powerful methods to retrieve dynamical information from time-ordered data [10]. The objective of time-series analysis is to reconstruct the full dynamics of a complex nonlinear dynamical system, starting from a measurement of only one of its variables. In fact, in many cases, it is possible to observe only a subset of the components necessary to determine the time-evolution law which governs the dynamical system.

Table 3 Average prediction results obtained on the test set

	RT	DR	$\mathbf{N_r}$	$\boldsymbol{\rho}$	$\boldsymbol{\xi}$	ω_i	ω_o	ω_f	$\boldsymbol{\theta}^{(\mathbf{d})}$	$\boldsymbol{\gamma}_{\mathrm{k}}$	$\boldsymbol{\lambda}$	**C**	$\boldsymbol{\nu}$	$\boldsymbol{\gamma}_{\mathrm{r}}$	Error
MG	ridge reg.	–	378	1.22	0.0	0.55	0.212	0.25	–	–	0.625	–	–	–	$3.064E{-}2 \pm 1.648E{-}4$
	ridge reg.	PCA	318	1.214	0.042	0.807	0.840	0.355	89	–	0.294	–	–	–	$6.483E{-}3 \pm 9.126E{-}3$
	ridge reg.	*k*PCA	445	1.148	0.0	0.339	0.202	0.422	33	0.050	0.521	–	–	–	$4.283E{-}3 \pm 1.893E{-}4$
	ν-SVR	–	490	1.051	0.0	0.873	0.588	0.568	–	–	–	0.234	5.346	0.868	$1.700E{-}1 \pm 5.171E{-}2$
	ν-SVR	PCA	474	1.044	0.0	0.604	0.807	0.350	3	–	–	2.535	0.340	0.825	**3.210E−5 ± 3.122E−5**
	ν-SVR	*k*PCA	466	0.738	0.0	0.373	0.664	0.069	14	0.059	–	7.975	0.448	0.299	$4.902E{-}4 \pm 9.619E{-}6$
NARMA	ridge reg.	–	406	1.031	0.053	0.19	0.231	0.194	–	–	0.163	–	–	–	$3.759E{-}1 \pm 1.409E{-}3$
	ridge reg.	PCA	409	0.934	0.016	0.135	0.127	0.073	22	–	0.963	–	–	–	$3.791E{-}1 \pm 4.887E{-}4$
	ridge reg.	*k*PCA	342	0.887	0.018	0.167	0.407	0.0	225	0.008	0.001	–	–	–	$1.024E{-}1 \pm 1.542E{-}3$
	ν-SVR	–	440	0.928	0.015	0.129	0.603	0.031	–	–	–	4.332	0.271	0.017	$7.298E{-}2 \pm 7.901E{-}4$
	ν-SVR	PCA	433	0.952	0.0	0.107	0.207	0.267	274	–	–	4.099	0.134	0.028	$6.438E{-}2 \pm 6.254E{-}4$
	ν-SVR	*k*PCA	460	0.962	0.002	0.1	0.302	0.037	163	0.028	–	4.281	0.752	0.420	**5.852E−2 ± 1.475E−3**
MSO	ridge reg.	–	298	1.148	0.008	0.345	0.147	0.045	–	–	0.438	–	–	–	$9.427E{-}1 \pm 1.675E{-}2$
	ridge reg.	PCA	499	1.141	0.017	0.187	0.184	0.309	438	–	0.601	–	–	–	$7.642E{-}1 \pm 1.189E{-}1$
	ridge reg.	*k*PCA	454	1.053	0.003	0.137	0.169	0.184	407	0.040	0.117	–	–	–	$5.959E{-}1 \pm 3.233E{-}2$
	ν-SVR	–	444	1.0	0.001	0.114	0.1	0.09	–	–	–	3.714	0.282	0.037	**2.353E−1 ± 1.609E−2**
	ν-SVR	PCA	459	1.147	0.001	0.174	0.144	0.276	225	–	–	6.373	0.38	0.601	$7.091E{-}1 \pm 2.182E{-}2$
	ν-SVR	*k*PCA	480	1.027	0.032	0.1	0.627	0.011	135	0.002	–	3.529	0.204	0.495	$4.860E{-}1 \pm 2.082E{-}2$

The table contains the following fields: method for readout training (RT), dimensionality reduction procedure (DM), spectral radius of the reservoir (ρ), neurons in the reservoir (N_r), noise in ESN state update (ξ), scaling of input, teacher and feedback weights (ω_i, ω_o, ω_f), dimensionality (d), kPCA kernel width (γ_{k}), L_2 norm regularization factor (λ), and ν-SVR parameters (C, ν, γ_{r}). Best results are highlighted in bold.

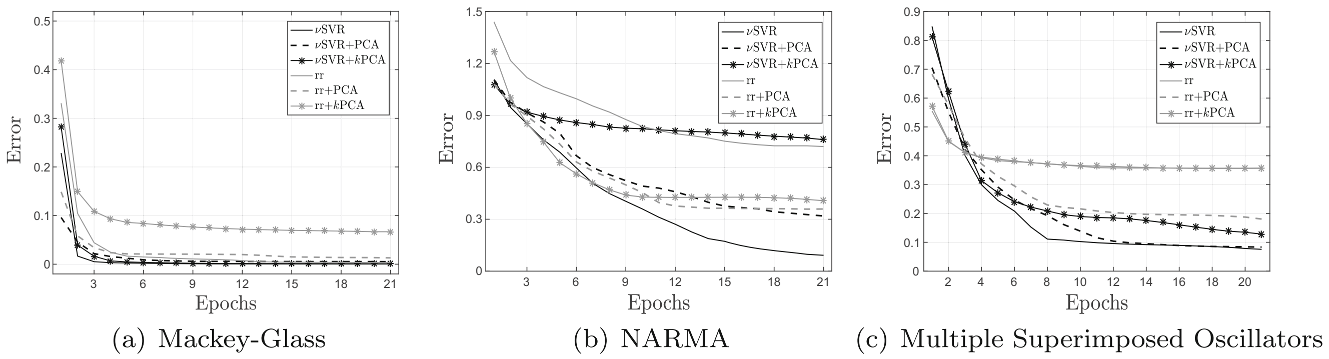


Fig. 4 Convergence of the error on the validation set in hyperparameters optimization with the GA. Black lines represent models whose readout is trained with ν-SVR, models trained with ridge regression are depicted with gray lines

The main idea which inspires this analysis is that a dynamic system is completely described by the time-dependent trajectory in its phase space. Hence, a recurrent neural network that is capable of reconstructing with a high degree of accuracy the dynamic attractor can calculate future states assumed by the system, given a state at any particular moment.

A frequently used method for phase space reconstruction is the *delay-coordinate embedding*, which provides an estimation of the attractor that is topologically identical to the true one. From this reconstruction, it is possible to infer several properties of the hidden dynamical system, which are invariant under diffeomorhpism. We refer to these measures as the *dynamical invariants* of the system. The most commonly studied are the *fractal dimension* of the attractor, the *Lyapuanov exponents*, and the *Rényi entropy*. In the following, we briefly introduce the delay-coordinate embedding procedure and two approaches used to estimate the aforementioned dynamical invariants. We refer the interested reader to [21, 36] for a comprehensive overview of these methods and many other aspects of time-series analysis.

Delay-coordinate Embedding a dynamical system is characterized by a time-evolution law, which determines its trajectory in the phase space. Each specific state of the system at time t is defined by a d-dimensional vector in the state space: $\mathbf{s}(t) = [s_1(t), \dots, s_d(t)]^T$, being d the number of variables of the system. The delay-coordinate embedding method allows to reconstruct such state vectors from a discrete time measurement of only one generic smooth function of the state space [47]. Given a time-series $\mathbf{x} = \{x(i\Delta t)\}_{i=1}^{N}$ evenly sampled at rate Δt, the embedding is defined as

$$\hat{s}(i) = \sum_{j=1}^{m} x(i + (j-1)\tau_e)\mathbf{e}_j, \tag{11}$$

where m is the embedding dimension, τ_e is the time delay, and $\mathbf{e}_j$ form an orthonormal basis in $\mathbb{R}^m$.

With a proper choice of embedding parameters m and τ_e, Taken theorem guarantees the existence of a diffeomorphism between the real and reconstructed dynamic [57]. A sufficient condition for a correct reconstruction is $m \geq 2d + 1$. The value of m is usually computed with the false nearest-neighbors algorithm [51], which provides an estimation of the smallest sufficient embedding dimension. On the other hand, a suitable time-delay τ_e can be estimated looking at the first zero of the autocorrelation function of $\mathbf{x}$ or by relying on nonlinear time dependencies, such as the mutual information [12].

Correlation Dimension dimension is an invariant property under diffeomorphism that allows to quantify the similarity of geometrical objects. Attractors of dissipative chaotic systems often exhibit complicated geometries (hence the name *strange*) which are contained in a fractal dimension D_q, called Rényi dimension [50]. An efficient estimator of fractal dimensions is Grassberger-Procaccia algorithm [22], which computes the correlation dimension D_2 through the correlation sum C_2:

$$C_2(m, \epsilon) = \frac{1}{2N_\epsilon(N_\epsilon - \tau_c)} \sum_{i} \sum_{j < i - \tau_c} \mathbf{\Theta}\left(\epsilon - \|x(i) - x(j)\|\right). \tag{12}$$

The temporal spacing parameter τ_c is chosen to ensure temporal independence between samples, $\mathbf{\Theta}$ is the Heaviside function, and ϵ is the dimension of a set of N_ϵ small boxes used to cover the geometric shape of the attractor. If $m \geq D_2$, $C_2(m, \epsilon) \propto \epsilon^{D_2}$. The correlation dimension D_2 is computed as the slope of the log-log scaling between C_2 and box size, ϵ.

Lyapuanov Exponent the Lyapuanov spectrum $\{\lambda_1, \dots, \lambda_d\}$ is another invariant measure that characterizes the predictability of a dynamical system. Lyapuanov exponents quantify the rate of separability of two trajectories, which are infinitesimally close at the initial time instant. Such exponents are related to the second order Rényi entropy

K_2, as described by the Peisin identity: $K_2 \leq \sum_{\lambda_i>0} \lambda_i$. This quantity measures the number of possible trajectories that the system can take for a given number of time steps in the future. A perfectly non-chaotic, deterministic system can only evolve along one possible trajectory and hence $K_2 = 0$. In contrast, for purely stochastic systems, the number of possible future trajectories increases to infinity, so $K_2 \to \infty$. Chaotic systems are characterized by a finite value of K_2, as the number of possible trajectories diverges but not as fast as in the stochastic case.

The largest Lyapuanov exponent (LLE) λ_1 is a good estimate of K_2, and its sign determines whether a system is chaotic or not. The so-called *direct methods* can be used to compute λ_1 by estimating the divergent motion of the reconstructed space, without fitting a model to the data [48, 62]. In particular, the average exponential growth of the distance of neighboring orbits can be studied on a logarithmic scale by monitoring the prediction error $p(t)$:

$$p(t) = \frac{1}{N}\sum_{k=1}^{N} \log_2 \left(\frac{\|\mathbf{x}[k+t] - \mathbf{x}_{\text{nn}}^{[\text{k}]}[t]\|}{\|\mathbf{x}[k] - \mathbf{x}_{\text{nn}}^{[\text{k}]}\|} \right), \qquad (13)$$

being $\mathbf{x}_{\text{nn}}^{[\text{k}]}$ the nearest neighbor of $\mathbf{x}$ at time k. The LLE is estimated as $\lambda_1 \propto p(t)/T$ with $t \in [1, T]$, where T is the forecast horizon within which the divergence of the trajectory in the phase space is evaluated.

ESN Phase Space Reconstruction

In the following, we analyze two chaotic time-series generated by the Lorenz and the Moore–Spiegel system respectively. We evaluate the accuracy of the phase space reconstruction performed with our ESN by comparing the topological properties of the true attractor of the dynamic, with the one obtained by applying a dimensionality reduction to the network reservoir. The equivalence of attractors geometries are computed by measuring the dynamical invariants, estimated through the correlation sum and the largest Lyapuanov exponent.

In the following, we refer to *true attractor*, as the trajectory in the phase space generated directly by the differential equations of the dynamic system. With *delay-embedding attractor*, we refer at the trajectory described by the embedding, generated with the delay-coordinate procedure. Finally, *ESN attractor* is the trajectory spanned by the component of the multivariate vector $\bar{\mathbf{h}}$. The latter is the output of the dimensionality reduction procedure applied to the multivariate vector $\mathbf{h}$, which contains the sequence of the states of the reservoir (see the "Proposed Architecture" section). For these tests, we considered only the component of the loss function relative to the prediction error, by setting $\alpha = 0$ in Eq. 10, and we fixed the number of dimensions in PCA and kPCA to 3. Finally, to further empathize the effectiveness of the architecture proposed, we also consider the phase space reconstruction obtained directly from $\mathbf{h}$, in the case where the reservoir contains only 3 neurons ($N_r = 3$).

Lorenz the system is governed by the following ordinary differential equations:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y-x), \quad \frac{\mathrm{d}y}{\mathrm{d}t} = x(\rho - z) - y, \quad \frac{\mathrm{d}z}{\mathrm{d}t} = xy - \beta z, \quad (14)$$

where variables x, y and z define the state of the system, while σ, ρ and β are system parameters. In this work, we set $\sigma = 10$, $\beta = 8/3$, and $\rho = 28$, values for which the system exhibits chaotic behavior.

Figure 5 depicts the geometric shapes of the true attractor, the delay-embedding attractor, the two ESN attractors, generated using a dimensionality reduction or a reservoir with three neurons. As it is possible to observe visually, both the embedding and ESN with dimensionality reduction manage to reconstruct well the trajectory described by the differential equations of the dynamic system. To quantify formally this similarity, we compute on the dynamical invariants previously introduced each attractor. In Table 4, we report for each phase space trajectory the estimated correlation dimension and the largest Lyapuanov exponent, which as previously discussed, represents a good approximation of the K_2 entropy. Due to the stochastic nature of the approaches adopted for estimating these quantities, we repeated the procedure 10 different times and we report their average values and the standard deviations. As we can see from the results, both the trajectories described by $\bar{\mathbf{h}}$ in the subspace computed using PCA and kPCA generate an attractor whose dynamic invariants are well approximated. In particular, the accuracy of the reconstruction is comparable to the one obtained by the classic time-delay embedding method and in some case, it is even better. The standard deviations in the measurements of both correlation dimension and LLE are very small, which indicates a high degree of reliability on both measurements. For what concerns the ESN with three neurons, the trajectory described is more "flat," as it can be seen in the figure. This is confirmed by the estimated correlation dimension and LLE, whose values are much lower than in the other cases. This denotes that the reconstructed dynamic is not rich enough, a symptom that the complexity and the memory of the network is not sufficient to model the underlying system.

Moore–Spiegel this dynamical systems manifests interesting synchronization properties, generated by complicated patterns of period-doubling, saddle-node, and homoclinic

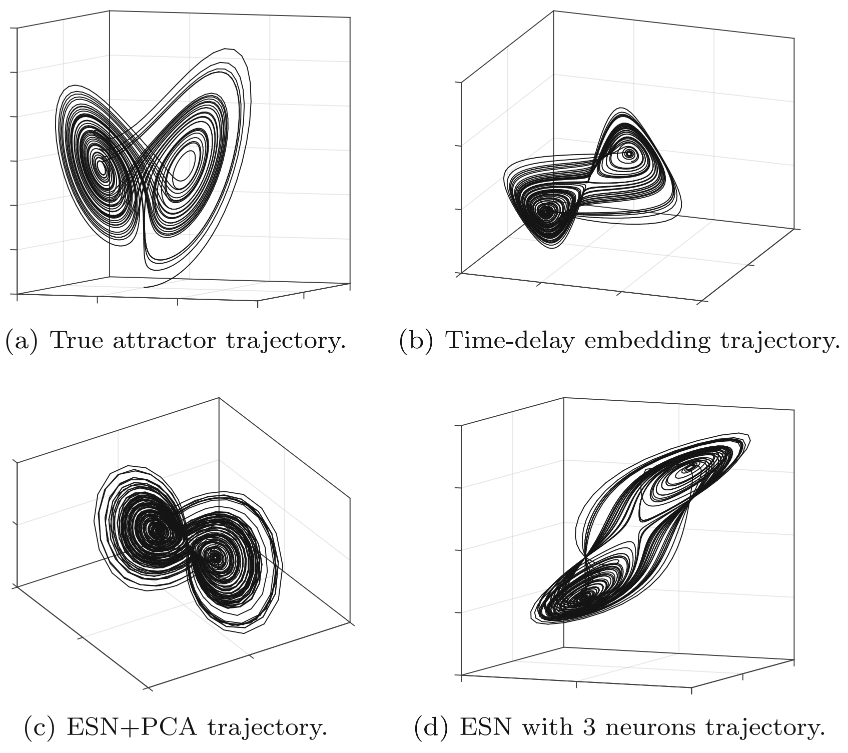


Fig. 5 Trajectory of the attractors of the Lorenz dynamical system in the phase space. In **a**, the true trajectory, which is computed directly from the ordinary differential equations of the system. In **b**, the trajectory reconstructed using time-delay embedding. In **c**, the trajectory generated by the internal state of ESN internal state, on the subspace defined by the first three components of the PCA. In **d**, the trajectory described by the internal state of an ESN with a small reservoir with 3 neurons

bifurcations [3]. The differential equations which governs system dynamics are the following:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = y, \quad \frac{\mathrm{d}y}{\mathrm{d}t} = z, \quad \frac{\mathrm{d}z}{\mathrm{d}t} = -z - (t - r + rx^2)y - tx, \quad (15)$$

where x, y, and z form the state of the system and r and t are the parameters of the model. In this study, we set $r = 100$, $b = 10$, and $c = 14$, for which the dynamics of the system exhibits a chaotic behavior.

In Fig. 6, we show the shape of the attractors of the dynamic, evaluated directly on the differential equations of the system, on the time-delay embedding, on the internal state of the ESN reduced through PCA, and on the state of the ESN with three neurons. In this second test, the reconstructed trajectories of the Moore–Spiegel system are more jagged and irregular, with respect to the original one. This suggest a poorer approximation of the true dynamic of the system and is confirmed by the results in Table 4. Compared to the Lorenz case, the dynamical invariants estimated on the time-delay embedding and on ESN state trajectories approximate with less accuracy the real ones. The reconstructed attractors have a lower correlation dimension, which usually denotes a poor embedding [45]. The results in prediction accuracy are reasonably good because of the simplicity of the task. However, the performance is inferior to the one obtained with the Lorenz system. It is worth to notice that the two attractors reconstructed by the ESN+PCA and ESN+kPCA have a larger C_2 value than the time-delay embedding and hence they approximate better the true dynamics. For what concerns the LLE, the estimated value in each reconstructed dynamic is larger than in the original one. This means that both the time-delay embedding and the ESNs generate a more chaotic dynamic, as is also reflected by the jagged trajectories in Fig. 6.

Table 4 Correlation dimension (D_2) and largest Lyapuanov exponent (LLE) of the attractors of Lorenz and Moore-Spiegel dynamical systems

System	Invariant	True	Emb	ESN+PCA	ESN+*k*PCA	ESN small
Lorenz	D_2	$2.068 \pm 4E{-}6$	$1.8871 \pm 8E{-}6$	$2.1722 \pm 3E{-}6$	$1.8614 \pm 5E{-}6$	$1.6044 \pm 1E{-}6$
	LLE	$0.9056 \pm 5E{-}4$	$0.9181 \pm 6E{-}4$	$1.0397 \pm 5E{-}4$	$0.91496 \pm 8E{-}4$	$0.76138 \pm 3E{-}5$
Moore–Spiegel	D_2	$1.9802 \pm 1E{-}6$	$0.83499 \pm 4E{-}6$	$0.87619 \pm 3E{-}6$	$0.95588 \pm 1E{-}6$	$0.63507 \pm 2E{-}7$
	LLE	$0.00708 \pm 7E{-}4$	$0.7003 \pm 4E{-}4$	$0.51611 \pm 4E{-}4$	$0.54784 \pm 4E{-}4$	$0.75421 \pm 2E{-}5$

Each invariant is estimated on the trajectories generated by: the ordinary differential equations (True); the dime-delay embedding (Emb); the ESN reservoir state, whose dimensionality is reduced using PCA (ESN+PCA) or k-PCA (ESN+kPCA); the internal state of an ESN with a small reservoir with 3 neurons (ESN small)

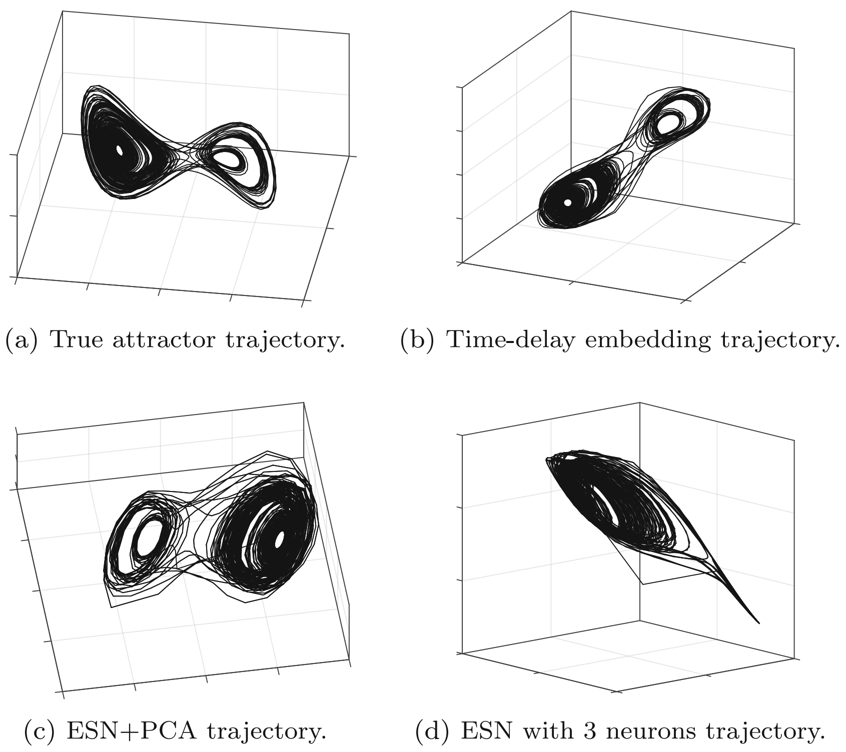


Fig. 6 Trajectory of the attractors of the Moore–Spiegel dynamical system in the phase space. In **a**, the true trajectory, which is computed directly from the ordinary differential equations of the system. In **b**, the trajectory reconstructed using time-delay embedding. In **c**, the trajectory generated by the internal state of ESN internal state, on the subspace defined by the first three components of the PCA. In **d**, the trajectory described by the internal state of an ESN with a small reservoir with three neurons

Even in this case, however, LLE is better approximated by ESN+PCA and ESN+kPCA than by the time-delay embedding. Like before, the standard deviations of the estimates of the two dynamical invariants is very small, which provides a high degree of confidence on the measurements. For what concerns the trajectory described by the ESN state with a small reservoir of three neurons, the geometric properties of the reconstruct attractor are even more different from the real ones. This confirms that also in this case, such a small amount of neurons cannot catch the dynamic properties of the system to be modeled.

As a concluding remark, it is important to understand another aspect of the utility of the ESN in reproducing the attractor of the system dynamic. In fact, this provides a valid alternative to the standard approach based on the time-delay embedding for reconstructing the phase of the system, which presents several caveats and pitfalls [10]. This a fundamental tool for a wide set of applications, where an accurate estimation of the phase space of the system is required [36].

Conclusions and Future Directions

In this work, we have presented a new framework for training an Echo State Network, which enhances its generalization capabilities through the regularization constraints introduced by the smoothing effect of a dimensionality reduction procedure. Through a series of test on benchmark dataset, we have demonstrated how the proposed architecture can achieve better prediction performance in different contexts. Successively, we provided a theoretically grounded explanation of the functioning of the proposed architecture, based on the theory of nonlinear time-series analysis. By studying the dynamical properties of the network under this novel perspective, we showed that through an ESN, it is possible to reconstruct the phase space of the dynamic system; this offers a solid, yet simple alternative to the time-delay embedding procedure.

We believe that this work could be useful not only to enhance the prediction capabilities of an ESN, but also provide a new tool for analysis of dynamical systems. As a follow-up of a recent work focused on identifying the edge of criticality of an ESN by evaluating the Fisher information on the state matrix [39], we plan to study the criticality using more reliable Fisher Information Matrix estimators, which are capable of working only on space with few dimensions (e.g., [26]). We also plan on investigating other dimensionality reduction methods, manifold learning, and semi-supervised learning approaches to shrink and regularize the output of the network recurrent layer [4, 5]. Finally, as a future work, we propose to use different dimensionality reduction techniques in parallel and combine their result through a single reservoir to produce the final result.

Compliance with Ethical Standards

Conflict of Interests The authors declare that they have no conflict of interest.

Informed Consent All procedures followed were in accordance with the ethical standards of the responsible committee on human experimentation (institutional and national) and with the Helsinki Declaration of 1975, as revised in 2008 (5). Additional informed consent was obtained from all patients for which identifying information is included in this article.

Human and Animal Rights This article does not contain any studies with human or animal subjects performed by any of the authors.

References

1. Alexandre LA, Embrechts MJ, Linton J. Benchmarking reservoir computing on time-independent classification tasks. IJCNN International Joint Conference on Neural Networks, 2009. IEEE; 2009. p. 2009.
2. Baker CT. The numerical treatment of integral equations. Clarendon Press, Israel Program for Scientific Translations, 1973. ISBN 019853406X.
3. Balmforth N, Craster R. Synchronizing moore and spiegel. Chaos: An Interdisciplinary Journal of Nonlinear Science. 1997;7(4):738–752.
4. Belkin M, Niyogi P, Sindhwani V. Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. J Mach Learn Res. 2006;7:2399–2434.
5. Bengio Y, Paiement J-F, Vincent P, Delalleau O, Le Roux N, Ouimet M. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. Adv Neural Inf Proces Syst. 2004;16:177–184.
6. Bianchi FM, De Santis E, Rizzi A, Sadeghian A. Short-term electric load forecasting using echo state networks and PCA decomposition. IEEE Access. 2015a;3:1931–1943. ISSN 2169-3536. doi:10.1109/ACCESS.2015.2485943.
7. Bianchi FM, Scardapane S, Uncini A, Rizzi A, Sadeghian A. Prediction of telephone calls load using Echo State Network with exogenous variables. Neural Netw. 2015b;71:204–213. doi:10.1016/j.neunet.2015.08.010.
8. Bianchi FM, Livi L, Alippi C. Investigating echo state networks dynamics by means of recurrence analysis. 2016. arXiv:1601.07381.
9. Boedecker J, Obst O, Lizier JT, Mayer NM, Asada M. Information processing in echo state networks at the edge of chaos. Theory Biosci. 2012;131(3):205–213.
10. Bradley E, Kantz H. Nonlinear time-series analysis revisited. Chaos: An Interdisciplinary Journal of Nonlinear Science. 2015;25(9):097610.
11. Burges CJ. A tutorial on support vector machines for pattern recognition. Data Min Knowl Disc. 1998;2(2):121–167.
12. Cao L. Practical method for determining the minimum embedding dimension of a scalar time series. Physica D: Nonlinear Phenomena. 1997;110(1):43–50.
13. Charles A, Yin D, Rozell C. Distributed sequence memory of multidimensional inputs in recurrent networks. 2016. arXiv:1605.08346.
14. Davenport MA, Duarte MF, Wakin MB, Laska JN, Takhar D, Kelly KF, Baraniuk RG. The smashed filter for compressive classification and target recognition. Electronic Imaging 2007, pages 64980H–64980H. International Society for Optics and Photonics; 2007.
15. Deihimi A, Showkati H. Application of echo state networks in short-term electric load forecasting. Energy. 2012;39(1):327–340.
16. Deihimi A, Orang O, Showkati H. Short-term electric load and temperature forecasting using wavelet echo state networks with neural reconstruction. Energy. 2013;57:382–401.
17. Dutoit X, Schrauwen B, Campenhout JV, Stroobandt D, Brussel HV, Nuttin M. Pruning and regularization in reservoir computing. Neurocomputing. 2009;72(7–9):1534 – 1546. ISSN 0925-2312. doi:10.1016/j.neucom.2008.12.020 Advances in Machine Learning and Computational Intelligence16th European Symposium on Artificial Neural Networks 200816th European Symposium on Artificial Neural Networks 2008.
18. Fodor IK. A survey of dimension reduction techniques Technical report. 2002.
19. Fraser AM, Swinney HL. Independent coordinates for strange attractors from mutual information. Phys Rev A. 1986;33(2):1134.
20. Friedman JH. On bias, variance, 0/1—loss, and the curse-of-dimensionality. Data Min Knowl Disc. 1997;1(1):55–77.
21. Gao J, Cao Y, Tung W-w, Hu J. Multiscale analysis of complex time series: integration of chaos and random fractal theory, and beyond: John Wiley & Sons; 2007. ISBN 978-0-471-65470-4.
22. Grassberger P, Procaccia I. Measuring the strangeness of strange attractors. The Theory of Chaotic Attractors. Springer; 2004. p. 170–189.
23. Hai-yan D, Wen-jiang P, Zhen-ya H. A multiple objective optimization based echo state network tree and application to intrusion detection. Proceedings of 2005 IEEE International Workshop on VLSI Design and Video Technology, 2005; 2005. p. 443–446. doi:10.1109/IWVDVT.2005.1504645.
24. Han S, Lee J. Fuzzy echo state neural networks and funnel dynamic surface control for prescribed performance of a nonlinear dynamic system. IEEE Trans Ind Electron. 2014a;61(2):1099–1112. ISSN 0278-0046. doi:10.1109/TIE.2013.2253072.
25. Han SI, Lee JM. Fuzzy echo state neural networks and funnel dynamic surface control for prescribed performance of a nonlinear dynamic system. IEEE Trans Ind Electron. 2014b;61(2):1099–1112.
26. Har-Shemesh O, Quax R, Miñano B, Hoekstra AG, Sloot PMA. Nonparametric estimation of Fisher information from real data. Phys Rev E. 2016;93(2):023301. doi:10.1103/PhysRevE.93.023301.
27. Hotelling H. Analysis of a complex of statistical variables into principal components. J Educ Psychol. 1933;24(6):417–441.
28. Huang C-M, Huang C-J, Wang M-L. A particle swarm optimization to identifying the armax model for short-term load forecasting. IEEE Trans Power Syst. 2005;20(2):1126–1133.
29. Indyk P, Motwani R. Approximate nearest neighbors: towards removing the curse of dimensionality. Proceedings of the thirtieth annual ACM symposium on Theory of computing. ACM; 1998. p. 604–613.
30. Jaeger H. The echo state approach to analysing and training recurrent neural networks-with an erratum note. Bonn, Germany: German National Research Center for Information Technology GMD Technical Report. 2001;148:34.
31. Jaeger H. Adaptive nonlinear system identification with echo state networks. Advances in neural information processing systems; 2002. p. 593–600.
32. Jaeger H, Haas H. Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication. science. 2004;304(5667):78–80.
33. Jan van Oldenborgh G, Balmaseda MA, Ferranti L, Stockdale TN, Anderson DL. Did the ecmwf seasonal forecast model outperform statistical enso forecast models over the last 15 years? J Clim. 2005;18(16):3240–3249.
34. Jenssen R. Kernel entropy component analysis. IEEE Trans Pattern Anal Mach Intell. 2010;32(5):847–860. ISSN 0162-8828. doi:10.1109/TPAMI.2009.100.

35. Jenssen R. Entropy-relevant dimensions in the kernel feature space: cluster-capturing dimensionality reduction. IEEE Signal Process Mag. 2013;30(4):30–39. ISSN 1053-5888. doi:10.1109/MSP.2013.2249692.
36. Kantz H, Schreiber T, Vol. 7. Nonlinear time series analysis: Cambridge university press; 2004. ISBN 9780511755798. doi:10.1017/CBO9780511755798.
37. Li D, Han M, Wang J. Chaotic time series prediction based on a novel robust echo state network. IEEE Transactions on Neural Networks and Learning Systems. 2012;23(5):787–799.
38. Liebert W, Schuster H. Proper choice of the time delay for the analysis of chaotic time series. Phys Lett A. 1989;142(2-3):107–111.
39. Livi L, Bianchi FM, Alippi C. Determination of the edge of criticality in echo state networks through fisher information maximization. 2016. arXiv:1603.03685.
40. Lukoševičius M, Jaeger H. Reservoir computing approaches to recurrent neural network training. Computer Science Review. 2009;3(3):127–149. doi:10.1016/j.cosrev.2009.03.005.
41. Ma Q, Shen L, Chen W, Wang J, Wei J, Yu Z. Functional echo state network for time series classification. Inf Sci. 2016;373:1 – 20. ISSN 0020-0255. doi:10.1016/j.ins.2016.08.081.
42. Malik ZK, Hussain A, Wu J. Novel biologically inspired approaches to extracting online information from temporal data. Cogn Comput. 2014;6(3):595–607. ISSN 1866-9964. doi:10.1007/s12559-014-9257-0.
43. Malik ZK, Hussain A, Wu J. An online generalized eigenvalue version of laplacian eigenmaps for visual big data. Neurocomputing. 2016a;173(2):127 – 136. ISSN 0925-2312. doi:10.1016/j.neucom.2014.12.119.
44. Malik ZK, Hussain A, Wu QJ. Multilayered echo state machine: A novel architecture and algorithm. 2016b.
45. Marwan N, Romano MC, Thiel M, Kurths J. Recurrence plots for the analysis of complex systems. Phys Rep. 2007;438(5):237–329.
46. Mazumdar J, Harley R. Utilization of echo state networks for differentiating source and nonlinear load harmonics in the utility network. IEEE Trans Power Electron. 2008;23(6):2738–2745. ISSN 0885-8993. doi:10.1109/TPEL.2008.2005097.
47. Packard NH, Crutchfield JP, Farmer JD, Shaw RS. Geometry from a time series. Phys Rev Lett. 1980;45(9):712.
48. Parlitz U. Nonlinear Time-Series Analysis. Boston, MA: Springer US; 1998, pp. 209–239. ISBN 978-1-4615-5703-6. doi:10.1007/978-1-4615-5703-6_8.
49. Peng Y, Lei M, Li J-B, Peng X-Y. A novel hybridization of echo state networks and multiplicative seasonal ARIMA model for mobile communication traffic series forecasting. Neural Comput & Applic. 2014;24(3-4):883–890.
50. Rényi A. On the dimension and entropy of probability distributions. Acta Mathematica Academiae Scientiarum Hungarica. 1959;10(1-2):193–215.
51. Rhodes C, Morari M. The false nearest neighbors algorithm: An overview. Comput Chem Eng. 1997;21:S1149–S1154.
52. Scardapane S, Comminiello D, Scarpiniti M, Uncini A. Significance-Based Pruning for Reservoir's Neurons in Echo State Networks: Springer International Publishing, Cham; 2015, pp. 31–38. ISBN 978-3-319-18164-6. doi:10.1007/978-3-319-18164-6_4.
53. Schölkopf B, Smola A, Müller K-R. Kernel principal component analysis. International Conference on Artificial Neural Networks. Springer; 1997. p. 583–588.
54. Schölkopf B, Smola AJ, Williamson RC, Bartlett PL. New support vector algorithms. Neural Comput. 2000;12(5):1207–1245.
55. Skowronski MD, Harris JG. Automatic speech recognition using a predictive echo state network classifier. Neural Netw. 2007;20(3):414–423.
56. Srinivas M, Patnaik LM. Genetic algorithms: a survey. Computer. 1994;27(6):17–26. ISSN 0018-9162. doi:10.1109/2.294849.
57. Takens F. Detecting strange attractors in turbulence. Berlin, Heidelberg: Springer Berlin Heidelberg; 1981, pp. 366–381. ISBN 978-3-540-38945-3. doi:10.1007/BFb0091924.
58. Van Der Maaten L, Postma E, Van den Herik J. Dimensionality reduction: a comparative. J Mach Learn Res. 2009;10:66–71.
59. Varshney S, Verma T. Half Hourly Electricity Load Prediction using Echo State Network. International Journal of Science and Research. 2014;3(6):885–888.
60. Verstraeten D, Schrauwen B. On the quantification of dynamics in reservoir computing. Artificial Neural Networks – ICANN 2009. In: Alippi C, Polycarpou M, Panayiotou C, and Ellinas G, editors. Heidelberg: Springer Berlin; 2009. p. 985–994. ISBN 978-3-642-04273-7. doi:10.1007/978-3-642-04274-4_101.
61. Wierstra D, Gomez FJ, Schmidhuber J. Modeling systems with internal state using evolino. Proceedings of the 7th annual conference on Genetic and evolutionary computation. ACM; 2005. p. 1795–1802.
62. Wolf A, Swift JB, Swinney HL, Vastano JA. Determining lyapunov exponents from a time series. Physica D: Nonlinear Phenomena. 1985;16(3):285–317.
63. Zhou S, Lafferty J, Wasserman L. Compressed and privacy-sensitive sparse regression. IEEE Trans Inf Theory. 2009;55(2): 846–866.
64. Fusi S, Miller EK, Rigotti M. Why neurons mix: high dimensionality for higher cognition. Curr Opin Neurobiol. 2016;37:66–74. ISSN 0959-4388. doi:10.1016/j.conb.2016.01.010.
65. Cover TM. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. IEEE Transactions on Electronic Computers. 1965;EC-14(3):326–334. ISSN 0367-7508. doi:10.1109/PGEC.1965.264137.
66. Mante V, Sussillo D, Shenoy KV, Newsome WT. Context-dependent computation by recurrent dynamics in prefrontal cortex. Nature. 2013;503(7474):78–84. ISSN 0028-0836. doi:10.1038/nature12742.
67. DiCarlo JJ, Cox DD. Untangling invariant object recognition. Trends Cogn Sci. 2007;11(8):333–341. ISSN 1364-6613. doi:10.1016/j.tics.2007.06.010.

Part IV

Appendix

Appendix A

Missing data

The term *missing data* refers to data in which elements are missing, and is a common occurrence in real world applications. Examples of this include a censor that fails/saturates or questions on a questionnaire that is left unanswered. No matter which mechanism under which the data is missing, this requires the analyst to take extra steps in order to analyze the data. Note that the terms *incomplete data* and *missing data* mean the same thing and are often used interchangeably.

A.1 Missing data mechanisms

There are essentially three mechanisms defined under which missing data is generated, namely *Missing Completely At Random* MCAR, *Missing At Random* MAR and *Not Missing At Random* NMAR [45]. These all refer to different assumptions on how/why the elements in the data are missing. In the following, these assumptions will be defined mathematically.

Let $\mathbf{X} \in \mathbb{R}^d$ be a random vector and let $\mathbf{R} \in \mathbb{R}^d$ be a vector denoting which elements are missing from $\mathbf{X}$. That is, $(\mathbf{R})_k = 1$ if entry $(\mathbf{X})_k$ is missing and 0 otherwise. Furthermore, let $\mathbf{X}^o$ and $\mathbf{X}^m$ denote the observed and missing portion of $\mathbf{X}$, respectively. The data is said to be MCAR if the elements being missing does not depend on the data, i.e.

$$p(\mathbf{R}|\mathbf{X}) = p(\mathbf{R}).$$

This is the least strict assumption, and can easily be handled. An example of this is if, in the examples above, questions in the questionnaire is left unanswered randomly because the participants forgets to answer them.

MAR refers to when the data being missing only depends on the observed value, such that

$$p(\mathbf{R}|\mathbf{X}) = p(\mathbf{R}|\mathbf{X}^o).$$

This is more strict, but can still be handled by the analyst if extra care is taken. From the questionnaire example, this might happen if the questionnaire is designed in such a way that parts of the questionnaire is left unanswered based on previous answers.

NMAR refers to when the data being missing depends on the missing data, such that

$$p(\mathbf{R}|\mathbf{X}) = p(\mathbf{R}|\mathbf{X}^m).$$

This is the most strict assumption, and is difficult to deal with. From the sensor example, this will happen if the sensor is *saturated*. That is, the value is not recorded because the true value is outside of the range the sensor can handle.

A.2 Examples of methods dealing with missing data

This section describes a few of the common approaches for handling missing data. The methods described here are mainly application agnostic, meaning that they are not incorporated in the machine learning method itself, but rather used as a pre–processing step of the data. For a comprehensive overview of existing missing data approaches, the interested reader is directed towards other sources on the topic [45, 94, 119].

Complete– and available case analysis Complete case analysis methods refer to the act of analysing only the complete cases in the dataset by manipulating the data, such that one obtains a rectangular complete–data format [45]. This is done by simply ignoring all datapoints (cases) with missing data. With this approach, one assumes that the data is MCAR [119], which is more likely to happen when the amount of missing data in the dataset is small ($\sim 5\%$ [45]).

Available case analysis is often used when one is interested in computing model parameters/statistics, and is utilizing all observed data to do so. In this situation, one would compute e.g. the mean of a variable based on all cases in which that variable is available. This approach is usually not applicable for machine learning methods, as the focus is not necessarily on model parameters/statistics, but rather on generating a complete dataset which can be analyzed further with machine learning methods.

Data imputation Data imputation is used in order to estimate values of missing elements. This is used as a pre–processing step before further analysis. There are several methods used in order to impute missing values. The most common ones are mean imputation, median imputation and zero imputation. These imputation methods are performed by replacing all missing values by these values, where the mean and median are computed based on all observed values in the dataset of the variable in question. These are examples of the so–called *single imputation* methods, where missing values are replaced by a single value. There are also *multiple imputation* methods [124], in which one computes M plausible values, generating M complete datasets. This facilitates uncertainty estimates for the missing values.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] Mark A. Aizerman. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, 25:821–837, 1964.

[3] Ahmed Alaoui and Michael W. Mahoney. Fast randomized kernel ridge regression with statistical guarantees. In *Advances in Neural Information Processing Systems*, pages 775–783, 2015.

[4] Hany Alashwal, Mohamed El Halaby, Jacob J. Crouse, Areeg Abdalla, and Ahmed A. Moustafa. The application of unsupervised clustering methods to alzheimer's disease. *Frontiers in computational neuroscience*, 13:31, 2019.

[5] Tahani Alqurashi and Wenjia Wang. Clustering ensemble method.

International Journal of Machine Learning and Cybernetics, 10(6): 1227–1246, 2019.

[6] Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950.

[7] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. *Symposium on Discrete Algorithms (SODA)*, 2007.

[8] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[9] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. In *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4945–4949. IEEE, 2016.

[10] Liang Bai, Jiye Liang, and Yike Guo. An ensemble clusterer of multiple fuzzy *k*-means clusterings to recognize arbitrarily shaped clusters. *IEEE Transactions on Fuzzy Systems*, 26(6):3524–3533, 2018.

[11] Guha Balakrishnan, Amy Zhao, Mert R. Sabuncu, John Guttag, and Adrian V. Dalca. An unsupervised learning model for deformable medical image registration. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9252–9260, 2018.

[12] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15 (6):1373–1396, 2003.

[13] Asa Ben-Hur and William Stafford Noble. Kernel methods for predicting protein–protein interactions. *Bioinformatics*, 21(suppl_1):i38–i46, 2005.

[14] Yoshua Bengio, Olivier Delalleau, Nicolas Le Roux, Jean-François Paiement, Pascal Vincent, and Marie Ouimet. Learning eigenfunctions links spectral embedding and kernel pca. *Neural computation*, 16(10):2197–2219, 2004.

[15] Filippo Maria Bianchi, Lorenzo Livi, and Cesare Alippi. Investigating echo-state networks dynamics by means of recurrence analysis. *IEEE transactions on neural networks and learning systems*, 29(2):427–439, 2016.

[16] Filippo Maria Bianchi, Enrico Maiorino, Michael C. Kampffmeyer, Antonello Rizzi, and Robert Jenssen. *Recurrent neural networks for short-term load forecasting: an overview and comparative analysis.* Springer, 2017.

[17] Joschka Boedecker, Oliver Obst, Joseph T. Lizier, N. Michael Mayer, and Minoru Asada. Information processing in echo state networks at the edge of chaos. *Theory in Biosciences*, 131(3):205–213, 2012.

[18] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

[19] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[20] Sergey Brin and Larry Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, April 1998. ISSN 01697552. doi: 10.1016/S0169-7552(98)00110-X.

[21] Austin J. Brockmeier, Tingting Mu, Sophia Ananiadou, and John Y. Goulermas. Quantifying the informativeness of similarity measurements. *The Journal of Machine Learning Research*, 18(1):2592–2652, 2017.

[22] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[23] Christopher J.C. Burges. *Dimension reduction: A guided tour.* Now Publishers Inc, 2010.

[24] Andrea Caponnetto and Ernesto De Vito. Optimal rates for the regularized least-squares algorithm. *Foundations of Computational Mathematics*, 7(3):331–368, 2007.

[25] Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 77–87, 2017.

[26] Xiangyu Chang, Shao-Bo Lin, and Ding-Xuan Zhou. Distributed semi-supervised learning with kernel ridge regression. *The Journal of Machine Learning Research*, 18(1):1493–1514, 2017.

[27] Olivier Chapelle, Jason Weston, and Bernhard Schölkopf. Cluster kernels for semi-supervised learning. In *Advances in neural information processing systems*, pages 601–608, 2003.

[28] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2018.

[29] Fan Chung and S.-T. Yau. Discrete Green's Functions. *Journal of Combinatorial Theory, Series A*, 91(1-2):191–214, July 2000. ISSN 00973165.

[30] Fan Chung and Wenbo Zhao. Pagerank and random walks on graphs. *Fete of combinatorics and computer science*, pages 1–16, 2010.

[31] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning Research*, 13(1):795–828, 2012.

[32] Nello Cristianini, John Shawe-Taylor, Andre Elisseeff, and Jaz S. Kandola. On kernel-target alignment. In *Advances in neural information processing systems*, pages 367–373, 2002.

[33] John P. Cunningham and Zoubin Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *The Journal of Machine Learning Research*, 16(1):2859–2900, 2015.

[34] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[35] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[36] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 551–556, 2004.

[37] Xibin Dong, Zhiwen Yu, Wenming Cao, Yifan Shi, and Qianli Ma. A survey on ensemble learning. *Frontiers of Computer Science*, pages 1–18, 2020.

[38] Harris Drucker, Christopher J.C. Burges, Linda Kaufman, Alex J. Smola, and Vladimir Vapnik. Support vector regression machines. In *Advances in neural information processing systems*, pages 155–161, 1997.

[39] Liang Du, Peng Zhou, Lei Shi, Hanmo Wang, Mingyu Fan, Wenjian Wang, and Yi-Dong Shen. Robust multiple kernel k-means using l21-norm. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

[40] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

[41] Bradley Efron and Robert J. Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[42] Ana L. N. Fred and Anil K. Jain. Combining multiple clusterings using evidence accumulation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(6):835–850, 2005.

[43] Valerio Freschi. Protein function prediction from interaction networks using a random walk ranking algorithm. In *2007 IEEE 7th International Symposium on BioInformatics and BioEngineering*, pages 42–48. IEEE, 2007.

[44] Jerome H. Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[45] Pedro J. García-Laencina, José-Luis Sancho-Gómez, and Aníbal R. Figueiras-Vidal. Pattern classification with missing data: a review. *Neural Computing and Applications*, 19(2):263–282, 2010.

[46] Peter Vincent Gehler. *Kernel learning approaches for image classification*. PhD thesis, Citeseer, 2009.

[47] Mark Girolami. Mercer kernel-based clustering in feature space. *IEEE Transactions on Neural Networks*, 13(3):780–784, 2002.

[48] David F. Gleich. Pagerank beyond the web. *Siam Review*, 57(3): 321–363, 2015.

[49] Shawn M. Gomez, William Stafford Noble, and Andrey Rzhetsky. Learning to predict protein–protein interactions from protein sequences. *Bioinformatics*, 19(15):1875–1881, 2003.

[50] Mehmet Gönen and Ethem Alpaydin. Multiple kernel learning algorithms. *Journal of machine learning research*, 12(64):2211–2268, 2011.

[51] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[52] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. Improved deep embedded clustering with local structure preservation. In *IJCAI*, pages 1753–1759, 2017.

[53] Luis Gómez-Chova, Robert Jenssen, and Gustavo Camps-Valls. Kernel entropy component analysis for remote sensing image clustering. *IEEE Geoscience and Remote Sensing Letters*, 9(2):312–316, March 2012. ISSN 1558-0571. doi: 10.1109/LGRS.2011.2167212.

[54] Mark S. Handcock, Adrian E. Raftery, and Jeremy M. Tantrum. Model-based clustering for social networks. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 170(2):301–354, 2007.

[55] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

[56] David Haussler. Convolution kernels on discrete structures. Technical report, Technical report, Department of Computer Science, University of California . . . , 1999.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[58] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[59] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[60] Geoffrey E. Hinton, Li Deng, Dong Yu, George E. Dahl, Abdelrahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[61] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[62] Paul Honeine. An eigenanalysis of data centering in machine learning, July 2014.

[63] John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[64] Prodip Hore, Lawrence O. Hall, and Dmitry B. Goldgof. A scalable framework for cluster ensembles. *Pattern recognition*, 42(5):676–688, 2009.

[65] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[66] Seyedmehdi Hosseinimotlagh and Evangelos E. Papalexakis. Unsupervised content-based identification of fake news articles with tensor

decomposition ensembles. In *Proceedings of the Workshop on Misinformation and Misbehavior Mining on the Web (MIS2)*, 2018.

[67] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[68] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4-5):411–430, 2000.

[69] Ryo Inokuchi and Sadaaki Miyamoto. Lvq clustering and som using a kernel function. In *2004 IEEE International Conference on Fuzzy Systems (IEEE Cat. No. 04CH37542)*, volume 3, pages 1497–1500. IEEE, 2004.

[70] Emma Izquierdo-Verdiguier, Luis Gómez-Chova, Lorenzo Bruzzone, and Gustavo Camps-Valls. Semisupervised kernel feature extraction for remote sensing image analysis. *IEEE Transactions on Geoscience and Remote Sensing*, 52(9):5567–5578, 2014.

[71] Emma Izquierdo-Verdiguier, Robert Jenssen, Luis Gómez-Chova, and Gustavo Camps-Valls. Spectral clustering with the probabilistic cluster kernel. *Neurocomputing*, 149:1299–1304, 2015.

[72] A. M. Jade, B. Srikanth, V. K. Jayaraman, B. D. Kulkarni, J. P. Jog, and L. Priya. Feature extraction and denoising using kernel {PCA}. *Chemical Engineering Science*, 58(19):4441–4448, 2003. ISSN 0009-2509.

[73] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.

[74] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

[75] Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

[76] Robert Jenssen. Kernel Entropy Component Analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(5):847–860, 2010. ISSN 0162-8828. doi: 10.1109/TPAMI.2009.100.

[77] Robert Jenssen, Jose C. Principe, Deniz Erdogmus, and Torbjørn Eltoft. The cauchy–schwarz divergence and parzen windowing: Connections to graph theory and mercer kernels. *Journal of the Franklin Institute*, 343(6):614 – 629, 2006. ISSN 0016-0032. doi: https://doi.org/10.1016/j.jfranklin.2006.03.018.

[78] Robert Jenssen, Deniz Erdogmus, Kenneth E. Hild II, Jose C. Principe, and Torbjørn Eltoft. Information cut for clustering using a gradient descent approach. *Pattern Recognition*, 40(3):796–806, 2007.

[79] Yushi Jing and Shumeet Baluja. Visualrank: Applying pagerank to large-scale image search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1877–1890, 2008.

[80] Motonobu Kanagawa, Philipp Hennig, Dino Sejdinovic, and Bharath K. Sriperumbudur. Gaussian processes and kernel methods: A review on connections and equivalences. *arXiv preprint arXiv:1807.02582*, 2018.

[81] Juha Karhunen and Jyrki Joutsensalo. Representation and separation of signals using nonlinear {PCA} type learning. *Neural Networks*, 7 (1):113–127, 1994. ISSN 0893-6080.

[82] Melody Y. Kiang, Michael Y. Hu, and Dorothy M. Fisher. An extended self-organizing map network for market segmentation—a telecommunication example. *Decision Support Systems*, 42(1):36–47, 2006.

[83] K. I. Kim, S. H. Park, and H. J. Kim. Kernel principal component analysis for texture classification. *Signal Processing Letters, IEEE*, 8 (2):39–41, February 2001. ISSN 1070-9908.

[84] K. I. Kim, K. Jung, and H. J. Kim. Face recognition using kernel principal component analysis. *Signal Processing Letters, IEEE*, 9(2): 40–42, February 2002. ISSN 1070-9908.

[85] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[86] Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[87] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[88] Brian Kulis, Sugato Basu, Inderjit Dhillon, and Raymond Mooney. Semi-supervised graph clustering: a kernel approach. *Machine learning*, 74(1):1–22, 2009.

[89] Sun Yuan Kung. *Kernel methods and machine learning*. Cambridge University Press, 2014.

[90] Ren Jieh Kuo, L.M Ho, and C.M Hu. Cluster analysis in industrial market segmentation through artificial neural network. *Computers & Industrial Engineering*, 42(2-4):391–399, 2002.

[91] Amy N. Langville and Carl D. Meyer. *Google's PageRank and beyond: The science of search engine rankings*. Princeton university press, 2011.

[92] Neil Lawrence. Probabilistic Non-linear Principal Component Analysis with Gaussian Process Latent Variable Models. *J. Mach. Learn. Res.*, 6:1783–1816, 2005. ISSN 1532-4435.

[93] Christina Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for svm protein classification. In *Biocomputing 2002*, pages 564–575. World Scientific, 2001.

[94] Roderick J. A. Little and Donald B. Rubin. *Statistical Analysis with Missing Data*. John Wiley & Sons, Inc., 1986. ISBN 0471802549.

[95] Xinwang Liu, Yong Dou, Jianping Yin, Lei Wang, and En Zhu. Multiple kernel k-means clustering with matrix-induced regularization. In *Proceedings of the thirtieth AAAI conference on artificial intelligence*, pages 1888–1894, 2016.

[96] Xinwang Liu, Xinzhong Zhu, Miaomiao Li, Lei Wang, En Zhu, Tongliang Liu, Marius Kloft, Dinggang Shen, Jianping Yin, and Wen

Gao. Multiple kernel k k-means with incomplete kernels. *IEEE transactions on pattern analysis and machine intelligence*, 42(5):1191–1204, 2019.

[97] Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb):419–444, 2002.

[98] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[99] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[100] Wolfgang Maass and Henry Markram. On the computational power of circuits of spiking neurons. *Journal of Computer and System Sciences*, 69(4):593 – 616, 2004.

[101] Donald MacDonald and Colin Fyfe. The kernel self-organising map. In *KES'2000. Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies. Proceedings (Cat. No. 00TH8516)*, volume 1, pages 317–320. IEEE, 2000.

[102] Marina Meila and Jianbo Shi. A Random Walks View of Spectral Segmentation. In *A random walks view of spectral segmentation.*, 2001.

[103] James Mercer. Functions of Positive and Negative Type, and their Connection with the Theory of Integral Equations. *Philosophical Transactions of The Royal Society*, A(209):415–446, 1909.

[104] Carl D. Meyer and Charles D. Wessell. Stochastic data clustering. *SIAM Journal on Matrix Analysis and Applications*, 33(4):1214–1236, 2012.

[105] Sebastian Mika, Gunnar Ratsch, Jason Weston, Bernhard Scholkopf, and Klaus-Robert Mullers. Fisher discriminant analysis with kernels. In *Neural networks for signal processing IX: Proceedings of the 1999*

IEEE signal processing society workshop (cat. no. 98th8468), pages 41–48. Ieee, 1999.

[106] Karl Øyvind Mikalsen, Filippo Maria Bianchi, Cristina Soguero-Ruiz, and Robert Jenssen. Time series cluster kernel for learning similarities between multivariate time series with missing data. *Pattern Recognition*, 76:569–581, 2018.

[107] Nina Mishra, Robert Schreiber, Isabelle Stanton, and Robert E. Tarjan. Clustering social networks. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 56–67. Springer, 2007.

[108] Stefano Monti, Pablo Tamayo, Jill Mesirov, and Todd Golub. Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data. *Machine learning*, 52(1-2):91–118, 2003.

[109] Takayasu Moriya, Holger R. Roth, Shota Nakamura, Hirohisa Oda, Kai Nagara, Masahiro Oda, and Kensaku Mori. Unsupervised segmentation of 3d medical images based on clustering and deep representation learning. In *Medical Imaging 2018: Biomedical Applications in Molecular, Structural, and Functional Imaging*, volume 10578, page 1057820. International Society for Optics and Photonics, 2018.

[110] Julie L. Morrison, Rainer Breitling, Desmond J. Higham, and David R. Gilbert. Generank: using search engine technology for the analysis of microarray experiments. *BMC bioinformatics*, 6(1):233, 2005.

[111] Michael C. Mozer. A focused back-propagation algorithm for temporal pattern recognition. *Complex systems*, 3(4):349–381, 1989.

[112] Sudipto Mukherjee, Himanshu Asnani, Eugene Lin, and Sreeram Kannan. Clustergan: Latent space clustering in generative adversarial networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4610–4617, 2019.

[113] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[114] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[115] Mariá Cristina Vasconcelos Nascimento, Franklina Maria Bragion De Toledo, and André C. Ponce Leon Ferreira Carvalho. Consensus clustering using spectral theory. In *International Conference on Neural Information Processing*, pages 461–468. Springer, 2008.

[116] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On Spectral Clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, pages 849–856, 2001.

[117] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[118] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[119] Therese D. Pigott. A review of methods for missing data. *Educational research and evaluation*, 7(4):353–383, 2001.

[120] Anthony J. Pinar, Joseph Rice, Lequn Hu, Derek T. Anderson, and Timothy C. Havens. Efficient multiple kernel classification using feature and decision level fusion. *IEEE Transactions on Fuzzy Systems*, 25(6):1403–1416, 2016.

[121] Jose C. Principe and Dongxin Xu. An introduction to information theoretic learning. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*, volume 3, pages 1783–1787 vol.3, 1999.

[122] Yunchen Pu, Zhe Gan, Ricardo Henao, Xin Yuan, Chunyuan Li, Andrew Stevens, and Lawrence Carin. Variational autoencoder for deep

learning of images, labels and captions. In *Advances in neural information processing systems*, pages 2352–2360, 2016.

[123] Carl Edward Rasmussen. *Gaussian Processes in Machine Learning*, pages 63–71. Springer Berlin Heidelberg, 2004.

[124] Donald B Rubin. *Multiple imputation for nonresponse in surveys*, volume 81. John Wiley & Sons, 2004.

[125] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *nature*, 323 (6088):533–536, 1986.

[126] Eder Santana, Matthew Emigh, and Jose C. Principe. Information theoretic-learning auto-encoder. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3296–3301. IEEE, 2016.

[127] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[128] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International conference on artificial neural networks*, pages 583–588. Springer, 1997.

[129] Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer, 2001.

[130] Matthias Scholz and Ricardo Vigário. Nonlinear PCA: a new hierarchical approach. In *ESANN*, pages 439–444, 2002.

[131] Shokri Z. Selim and Mohamed A. Ismail. K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(1):81–87, 1984.

[132] Uri Shaham, Kelly Stanton, Henry Li, Boaz Nadler, Ronen Basri, and Yuval Kluger. Spectralnet: Spectral clustering using deep neural networks. *arXiv preprint arXiv:1801.01587*, 2018.

[133] John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis.* Cambridge university press, 2004.

[134] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, August 2000. ISSN 0162-8828.

[135] Tao Shi, Mikhail Belkin, Bin Yu, et al. Data spectroscopy: Eigenspaces of convolution operators and clustering. *The Annals of Statistics*, 37 (6B):3960–3984, 2009.

[136] Bernard W. Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[137] Paul Smolensky. Parallel distributed processing: Volume 1: Foundations, de rumelhart, jl mcclelland, eds, 1986.

[138] Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*, 2015.

[139] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[140] Alexander Strehl and Joydeep Ghosh. Cluster Ensembles — a Knowledge Reuse Framework for Combining Multiple Partitions. *The Journal of Machine Learning Research*, 3:583–617, 2002.

[141] Takashi Takahashi and Takio Kurita. Robust de-noising by kernel pca. In *International Conference on Artificial Neural Networks*, pages 739–744. Springer, 2002.

[142] Devis Tuia and Gustavo Camps-Valls. Semisupervised remote sensing image classification with cluster kernels. *IEEE Geoscience and Remote Sensing Letters*, 6(2):224–228, 2009.

[143] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71):13, 2009.

[144] Sandro Vega-Pons and José Ruiz-Shulcloper. A survey of clustering ensemble algorithms. *International Journal of Pattern Recognition and Artificial Intelligence*, 25(03):337–372, 2011.

[145] David Verstraeten and Benjamin Schrauwen. On the quantification of dynamics in reservoir computing. In *International Conference on Artificial Neural Networks*, pages 985–994. Springer, 2009.

[146] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[147] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

[148] S.V.N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

[149] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007. ISSN 0960-3174.

[150] Fei Wang and Jimeng Sun. Survey on distance metric learning and dimensionality reduction in data mining. *Data mining and knowledge discovery*, 29(2):534–564, 2015.

[151] Chris Watkins. Dynamic alignment kernels. *Advances in neural information processing systems*, pages 39–50, 1999.

[152] Paul J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356, 1988.

[153] Sławomir T. Wierzchoń and Mieczysław Kłopotek. *Modern algorithms of cluster analysis*. Springer, 2018.

[154] C. F. Jeff Wu. On the convergence properties of the em algorithm. *The Annals of Statistics*, 11(1):95–103, 1983.

[155] Jing Wu and Mohamed R. Mahfouz. Robust x-ray image segmentation by spectral clustering and active shape model. *Journal of Medical Imaging*, 3(3):034005, 2016.

[156] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487, 2016.

[157] Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

[158] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3861–3870. JMLR. org, 2017.

[159] M-H Yang, Narendra Ahuja, and David Kriegman. Face recognition using kernel eigenfaces. In *Proceedings 2000 International Conference on Image Processing (Cat. No. 00CH37101)*, volume 1, pages 37–40. IEEE, 2000.

[160] Shuo Yang, Kai Shu, Suhang Wang, Renjie Gu, Fan Wu, and Huan Liu. Unsupervised fake news detection on social media: A generative approach. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5644–5651, 2019.

[161] Dit-Yan Yeung and Hong Chang. A kernel approach for semisupervised metric learning. *IEEE Transactions on Neural Networks*, 18(1): 141–149, 2007.

[162] Xiangrong Zhang, Licheng Jiao, Fang Liu, Liefeng Bo, and Maoguo Gong. Spectral clustering ensemble applied to sar image segmentation. *IEEE Transactions on Geoscience and Remote Sensing*, 46(7):2126–2136, 2008.

[163] Yan-Qing Zhang and Jagath Chandana Rajapakse. *Machine learning in bioinformatics*, volume 4. Wiley Online Library, 2009.

[164] Dengyong Zhou, Jason Weston, Arthur Gretton, Olivier Bousquet, and Bernhard Schölkopf. Ranking on data manifolds. In *Advances in neural information processing systems*, pages 169–176, 2004.

[165] Zhi-Hua Zhou and Ji Feng. Deep forest. *National Science Review*, 6 (1):74–86, 2019.

[166] Jerry Zhu, Jaz Kandola, Zoubin Ghahramani, and John D. Lafferty. Nonparametric transforms of graph kernels for semi-supervised learning. In *Advances in neural information processing systems*, pages 1641–1648, 2005.