MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Machine learning methods for the prediction of micromagnetic magnetization dynamics"

verfasst von / submitted by

Sebastian Alexander Schaffer, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna, 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 910
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Computational Science
Betreut von / Supervisor:	Univ.-Prof. Dr. Norbert Mauser
Mitbetreut von / Co-Supervisor:	Dipl.-Ing. Dr. Lukas Exl

Abstract

This master thesis studies a data-driven approach for the prediction of time-dependent PDEs. In particular, we are interested in model reduction methods and machine learning for fast response curve estimation modeled by the (parameter-dependent) Landau-Lifschitz-Gilbert (LLG) equation, the fundamental partial differential equation of motion in the field of micromagnetics. The work on the thesis consists of a theoretical conception of certain dimensionality reduction methods as well as nonlinear regression schemes for efficient machine learning of the solution trajectories of the LLG equation. Specifically, kernel methods will be studied with a focus on numerically stable and efficient implementation of Kernel Ridge Regression (KRR) and Kernel Principal Component Analysis (kPCA) with (novel) low-rank treatment of certain dense operators. Many algorithms map the data to another feature space. Pre-image computation is an essential element in the course of this work. Therefore, we discuss a supervised approach utilizing Kernel Ridge Regression.

With all those elements in mind, we derive an iterative solution to Kernel Dependency Estimation (KDE) and further develop an explicit multistep feature space integration scheme capable of learning the complicated magnetization dynamics in a reduced dimensional space. This algorithm offers fast prediction with similar accuracy as KDE. We discuss the data structure for this task and have a look at storage requirements for different approaches. For the implementation, we use the *numpy* and *scikit-learn* python modules, and simulations were partially computed on the *Vienna Scientific Cluster (VSC)*.

As a comparative analysis, a second part deals with neural network autoencoders for dimensionality reduction for the data sets with a focus on smooth latent space variable description with the help of regularization in contractive autoencoders. We find that the latent space description of the autoencoder is more powerful than the one offered by kernel principal component analysis. Further, a forward-looking objective function is used to train a neural network regression scheme to replace the kRR. Using *Keras* and *Tensorflow*, with automated differentiation we are able to perform optimization of very difficult objective functions.

In the case of kernel methods, the explicit feature space integration scheme offers an easy training process with explicit solutions to arising optimization problems. This becomes more difficult in the case of neural networks and requires advanced stochastic optimizers with adaptive moment estimation. We use the Adam optimization algorithm for this purpose. Using cross-validation, we perform a series of model validation and hyper-parameter estimation tasks and find that these methods offer a semi-supervised learning method with considerably lower computational cost than other approaches.

The thesis is thematically at the forefront of current research in computational physics/-mathematics and in the course of this thesis, two preprints were submitted to internationally accepted and peer-reviewed journals while, up until now, one has been accepted.

Acknowledgments

I acknowledge financial support by the Austrian Science Foundation (FWF) via the projects "ROAM" under grant No. P31140-N32 and the SFB "Complexity in PDEs" under grant No. F65., as well as the support from the Christian Doppler Laboratory Advanced Magnetic Sensing and Materials (financed by the Austrian Federal Ministry of Economy, 190 Family and Youth, the National Foundation for Research, Technology and Development). Further, I acknowledge the University of Vienna research platform MMM Mathematics - Magnetism - Materials, as well as the support of the Wolfgang Pauli Institute (WPI), which provided me a working place and everything else necessary for a professional working atmosphere, and also carries the cost for binding the thesis. The computations were partly achieved by using the Vienna Scientific Cluster (VSC) via the funded project No. 71140. The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

I gratefully acknowledge the support of my supervisor Univ.-Prof. Dr. Norbert Mauser. A special thanks goes to my co-supervisor Dipl.-Ing. Dr. Lukas Exl who was a great help with his insights and ideas. Without his contribution, this thesis could not cover as many aspects.

I also want to acknowledge the support of my parents Elisabeth Schaffer and Dr. Walter Schaffer, as well as the great support of my dear girlfriend Harriet Larcher de Brito Smith.

Contents

List of Figures

List of Tables

List of Algorithms

Authorship statement

This thesis consists partially of the following publications of the candidate that are freely available at the cited arXiv links. The version in the thesis includes extra work of the candidate that had to be omitted for the publication for the sake of brevity.

- S. Schaffer, N. J. Mauser, T. Schrefl, D. Suess, and L. Exl, "Machine learning methods for the prediction of micromagnetic magnetization dynamics", *IEEE Transactions on Magnetics* (accepted) (2021). arXiv: 2103.09079 [physics.comp-ph], cited as [40]

- L. Exl, N. J. Mauser, S. Schaffer, T. Schrefl, and D. Suess, "Prediction of magnetization dynamics in a reduced dimensional feature space setting utilizing a low-rank kernel Method", 2021. arXiv: 2008.05986 [physics.comp-ph], cited as [16]

The candidate gave the following substantial contributions to [40]:

> Contributed to software, validation, investigation, data curation, visualization, resources and writing of the original draft. In particular, implementation of the software and algorithms using *scikit-learn*, *Tensorflow* and *Keras*, algorithmic development and design, numerical experiments, validation and visualization of the results.

The candidate gave the following substantial contributions to [16]:

> Contributed to software, validation, investigation, data curation, visualization, resources and writing of the original draft. In particular, implementation of the software and algorithms using *scikit-learn*, algorithmic development and design, numerical experiments, validation and visualization of the results.

Chapter 1

Introduction and Problem Setting

Parts of this chapter were previously submitted for publication [16, 40] (author of this thesis is co-author) and here restated with the permission of the first and corresponding author.

1.1 Machine learning in Micromagnetism

Computational micromagnetism is an established discipline that is useful for simulations aiming at magnetic devices design in applications such as permanent magnets [19] or magnetic sensors [49]. E.g. permanent magnet research searches for novel superior magnetic materials with reduced critical rare earth content, and electronic circuit design and real time process control models quick and reliable sensor response. The theoretical foundation of micromagnetism is the continuum theory of micromagnetism, which treats magnetization processes on above atomistic length scales but without losing the capability to resolve magnetic domains, which are important for magnetic effects in the devices. The magnetization vector field is modelled as a continuous function inside a magnetic material in three dimensional space. Dynamics of the magnetization field in a magnetic material is governed by internal and external fields, such as exchange field, stray field, anisotropy field and Zeeman field, and mathematically described by the Landau-Lifschitz-Gilbert (LLG) equation, a time-dependent partial differential equation (PDE). The exchange field contribution is local but makes the PDE spatially second order, which yields stability requirements for explicit time-integration typically of CFL-type $\Delta t \leq C\,(\Delta x)^2$. On the other hand, the non-local stray field complicates implicit treatment. Also the exchange length (typically in the range of a few nanometers) is an upper bound for the discretization size, hence indirectly limits the size of time-steps in the presence of the mentioned time-step restrictions. The LLG equation is usually treated in finite difference or finite element discretization frameworks [33, 42], where the main computational burden is due to the magnetostatic Maxwell equations posed in whole space [1, 14].

More precisely, the mathematical description of magnetization dynamics in a magnetic body $\Omega \subset \mathbb{R}^3$ is through the *Landau-Lifschitz-Gilbert* (LLG) equation [9, 30]. In micromagnetism we consider the magnetization as a vector field $\boldsymbol{M}(x,t) = M_s \boldsymbol{m}(x,t)$, $|\boldsymbol{m}(x,t)| = 1$ depending on the position $x \in \Omega$ and the time $t \in \mathbb{R}$. The LLG equation is given in explicit form as

$$\frac{\partial \boldsymbol{M}}{\partial t} = -\frac{\gamma_0}{1+\alpha^2} \boldsymbol{M} \times \boldsymbol{H} - \frac{\alpha\, \gamma_0}{(1+\alpha^2) M_s} \boldsymbol{M} \times \big(\boldsymbol{M} \times \boldsymbol{H}\big), \tag{1.1}$$

where γ_0 is the gyromagnetic ratio, α the damping constant and $\boldsymbol{H}$ the effective field, which is the sum of nonlocal and local fields such as the stray field and the exchange field, respectively, and the external field $\boldsymbol{h} \in \mathbb{R}^3$ with length h. The stray field arises from the magnetostatic Maxwell equations, that is the whole space Poisson equation for the scalar potential u_d

$$\Delta u_d = \nabla \cdot \mathbf{M} \quad \text{in } \mathbb{R}^3, \tag{1.2}$$

with $\mathbf{H}_d = -\nabla u_d$. The exchange term is a continuous micro-model of Heisenberg exchange, that results in $\mathbf{H}_{ex} = \frac{2\,A}{\mu_0 M_s^2}\Delta\mathbf{M}$, where μ_0 is the vacuum permeability, M_s the saturation magnetization and A the exchange constant. Equation (1.1) is a time-dependent PDE in 3 spatial dimensions supplemented with an initial condition $\boldsymbol{M}(x, t = 0) = \boldsymbol{M}_0$ and (free) Neumann boundary conditions. For further details on micromagnetism the interested reader is referred to the literature [6, 2, 30]. Typically, equation (1.1) is numerically treated by a semi-discrete approach [48, 13, 12, 18], where spatial discretization by collocation using finite differences or finite elements leads to a rather large system of ordinary differential equations. Clearly, the evaluation of the right hand side of the system is very expensive mostly due to the stray field. Hence, effective methods are of high interest, especially in the case of successive solutions for different parameters such as an external field. On the other hand, applications need quick and effective solutions. A way to meet such demands for applications is offered by diverse reduced order models (ROMs) in micromagnetism. So far most ROMs were (multi)linear, e.g., tensor methods [15] and model reduction based on spectral decomposition [7] such as via a subset of the eigenbasis of the discretized self-adjoint effective field operator [11]. While these are keen ideas, they are clearly limited due to the inherent linearity of the reduced models. Recently, the authors introduced data-driven nonlinear model order reduction (nl-MOR) to effectively predict the magnetization LLG-dynamics subject to the external field based on simulated data [29, 17, 16]. Fast response to an external field can be obtained from such data-driven PDE machine learning (ML) models combined with unsupervised nonlinear model reduction. Another inspiration of the proposed machine learning scheme in [17, 16] was to construct a time-stepping predictor on the basis of a non-black-box nonlinear dimensionality reduction approach such as kernel principal component analysis (kPCA) [41] for the better understanding of the underlying approximations. In this connection, the key idea is to use a data set of simulated magnetization trajectories to learn a time-stepping model scheme that is capable of predicting the dynamics step by step for a new unseen external field without having to solve the LLG equation numerically, and hence with practically negligible computational effort. The challenging part is the combination of the learning process with reduced dimensionality of the feature space, which is initially proportional to the size of the discretization space used in the data generation, thus, several orders of magnitude too large for regression. A nonlinear kernel version of principal component analysis for the feature space dimensionality reduction was successfully established in [17, 16], where each time-step was learned on the basis of magnetization states represented via truncated kernel principal components. In the forthcoming presentation one novel extension of the idea in [17] will be the simultaneous learning of all steps via an entire dimension-reduced feature space integration scheme. Besides, the second improvement concerns the feasibility of the kernel learning scheme by the introduction of low-rank approximation to the kernel matrix, which allows the use of larger learning data. The reason for its importance is the fact that the learning process gets gradually infeasible as data size increases, as such, a common problem in data-driven methods but especially the case for kernel methods in machine learning [23]. Thus, while the original approach already leads to an exceptional reduction in feature dimension and fast learning owing to the nonlinear kernel, the novel approach performs training and predictions entirely in reduced coordinates and is capable of exploiting information from large training data sample sets owing to the low-rank kernel principal component analysis (low-rank kPCA). The low-rank kernel methods with applications to the prediction of magnetization dynamics was recently developed and submitted for publication as [16].

1.2 Supervised Learning

When considering a general supervised learning problem in machine learning, the goal is to establish a model which can reliably predict the variable $y \in \mathcal{Y}$, given an input $x \in \mathcal{X}$. To create a model, one first needs a set of training data $\mathcal{S} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y} | i = 1, \dots, m\}$. The samples are drawn from a data generating distribution ρ and are independently and identically distributed (i.i.d). Further a hypothesis class $\mathcal{H}$ and a loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is needed. The optimal hypothesis $f^* \in \mathcal{H}$ is given by the following optimization problem:

$$f^* = \arg\min_{f \in \mathcal{H}} \{\mathcal{L}_\rho(f)\}, \tag{1.3}$$

where $\mathcal{L}_\rho(f)$ is called the expected risk [45]

$$\mathcal{L}_\rho(f) := \mathop{\mathbb{E}}_{(x,y)\sim\rho} \{L(y, f(x))\}. \tag{1.4}$$

Using the Lebesgue integral, the expected risk can be expressed with

$$\mathcal{L}_\rho(f) = \int_{\mathcal{X}\times\mathcal{Y}} L(y, f(x)) d\rho, \tag{1.5}$$

where the loss function L is defined on the probability space $(\mathcal{X} \times \mathcal{Y}, \Sigma_{\mathcal{X}\times\mathcal{Y}}, \rho)$ and ρ is the unknown joint probability distribution.

When calculating the loss $L(y, f(x))$, the hypothesis f is applied to the input x in order to predict $\hat{y}$. With this notion, the loss function can be seen as a measure of similarity between the estimate $\hat{y} = f(x)$ and the true output y.

Hence, knowing the loss function gives knowledge of the similarity measure which is used in the output space $\mathcal{Y}$. In a classification setting this is often the zero-one loss and in regression the square loss. However, more complex outputs would also require a more complex loss function [52].

1.3 Data-driven solution of PDEs

Parameter-dependent PDEs are solved with conventional numerical methods on computational grids for varying parameter and the associated snapshots of the discrete solutions are collected in a training data set. Our approach to learn the solution manifold spanned by the solutions w.r.t. the varying parameter is to mimic time-stepping schemes in the training phase. For each time step Δt, a dependency map is learned between the solutions at times $\{t, t + \Delta t, \dots, t + (\nu - 1)\Delta t\}$ (input space) and at time $t + \nu\Delta t$ (output space), where $\nu \in \mathbb{N}$ defines the "multisteps" in a ν-step scheme. In more detail, the generic dependency estimation is here restated from [16]. We denote $\mathcal{X}$ as the *input set* and $\mathcal{Y}$ as the *output set*. A general *learning problem* is to estimate a map between inputs $x \in \mathcal{X}$ and outputs $y \in \mathcal{Y}$. The underlying mathematical task is that of estimating a map from an Hilbert space $\mathcal{V}$ by minimizing the risk functional

$$f^* \in \arg\min_{f \in \mathcal{V}} \mathcal{J}(f) := \int_{\mathcal{X}\times\mathcal{Y}} L(y, f(x))\, \mathrm{d}\rho(x, y),$$

on the measure space $\left(\mathcal{X} \times \mathcal{Y}, \Sigma_{\mathcal{X}\times\mathcal{Y}}, \rho\right)$ but with *unknown* joint probability distribution ρ. If we have available inputs $x \in \mathcal{X}$ and outputs $y \in \mathcal{Y}$ from a given training set $(x_1, y_1), (x_2, y_2), \dots,$
$(x_m, y_m) \in \mathcal{X} \times \mathcal{Y}$, we can try to empirically solve the problem in a model class or hypothesis class like e.g. $\mathcal{H} = \{f(.;\alpha) : \alpha \text{ feasible parameter}\}$. In [17] we defined L as the distance in output feature space using a radial basis function as kernel $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ on the output

set. This gives a RKHS $\mathcal{F}_\ell$ with associated map $\phi_\ell : \mathcal{Y} \to \mathcal{F}_\ell$ and $\ell(y, y') = \phi_\ell(y) \cdot \phi_\ell(y')$ and a loss expression $L(y, f(x)) = \|\phi_\ell(y) - \phi_\ell(f(x))\|^2_{\mathcal{F}_\ell}$, which can be expressed entirely through the kernel ℓ using the kernel trick. For the purpose of finding the minimizer f^*, only few kernel principal components of the representation of feature vectors are used and a ridge regression is used in [17]. Generally, the problem of estimating the map f can be decomposed in subtasks using the idea of *kernel dependency estimation (KDE)* [52], where f is the composition of three maps, i.e.,

$$f = \phi_\ell^{-1} \circ f_{\mathcal{F}} \circ \phi_k, \tag{1.6}$$

where $\phi_k : \mathcal{X} \to \mathcal{F}_k$ is the feature map for inputs associated with a kernel k, $f_{\mathcal{F}} : \mathcal{F}_k \to \mathcal{F}_\ell$ the map between input and output feature spaces and $\phi_\ell^{-1} : \mathcal{F}_\ell \to \mathcal{Y}$ an approximate inverse onto $\mathcal{Y}$ which is called the *pre-image map*. See Figure 1.1 for an illustration of the involved mappings. In this thesis we model the time-evolution of LLG dependent on external

$$\begin{tikzcd}
\mathcal{X} \arrow[r, "f"] \arrow[d, "\phi_k"'] & \mathcal{Y} \arrow[d, shift right, "\phi_\ell"'] \\
\mathcal{F}_k \arrow[r, "f_{\mathcal{F}}"'] & \mathcal{F}_\ell \arrow[u, shift right, "\phi_\ell^{-1}"']
\end{tikzcd}$$

Figure 1.1: Illustration of the mappings in (1.6).

field in a certain range. For instance, we will estimate a time-stepping map with (low-rank) kernel-ridge regression between truncated kPCA coordinate representations of magnetization configurations in the actually infinite dimensional feature space. Our learning approach works entirely with low-dimensional magnetization representations in feature space. All time steps are learned within the reduced dimensional setting and the pre-image is computed after the final time step only. We will use the computational low-rank approach of section 2.3.2 to further improve this method. Figure 1.2 illustrates the feature space integration scheme.

$$\begin{tikzcd}
\mathcal{X} \arrow[r] \arrow[d, "\phi_k"'] & \overset{f}{\cdots} \arrow[r] & \mathcal{X}' \\
\mathcal{F}_k \arrow[r] & \overset{f_{\mathcal{F}}}{\cdots} \arrow[r] & \mathcal{F}_k' \arrow[u, "\phi_k^{-1}"']
\end{tikzcd}$$

Figure 1.2: Illustration of the mappings involved in the feature space integration procedure.

Chapter 2

Kernel methods and nonlinear dimensionality reduction

Parts of this chapter were previously submitted for publication [16, 40] (author of this thesis is co-author) and here restated with the permission of the first and corresponding author.

2.1 Kernels in RKHS

Many algorithms auch as Support Vector Machines (SVMs) or Principal Component Analysis (PCA) are based on inner products measuring the distance between samples in the input space $\mathcal{X}$. A notion of distance can be encoded in a symmetric positive definite kernel function k.

Definition 2.1.1 (Positive definite kernel function [16])**.** *Let $\mathcal{X}$ be a nonempty set. A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a positive definite kernel on $\mathcal{X}$ if for all $m \in \mathbb{N}$ any choice of inputs $\mathbf{x} = \{x_1, \ldots, x_m\} \subseteq \mathcal{X}$ gives rise to a positive definite gram matrix $K[\mathbf{x}] \in \mathbb{R}^{m\times m}$ defined as $K_{ij} = k(x_i, x_j)$, $i, j = 1, \ldots, m$. To distinguish an involved second subset $\mathbf{y} = \{y_1, \ldots, y_\ell\} \subseteq \mathcal{X}$ we define the matrix $K[\mathbf{x}, \mathbf{y}] \in \mathbb{R}^{m\times \ell}$ via its entries $K_{ij} = k(x_i, y_j)$, $i = 1, \ldots, m$, $j = 1, \ldots, k$.*

A kernel therefore calculates an inner product in some Hilbert space $\mathcal{F}_k$ which can replace the inner product in the original space $\mathcal{X}$. This means there exists a map $\phi_k : \mathcal{X} \to \mathcal{F}_k$ such that $k(x, x') = (\phi_k(x)^T \phi_k(x'))$. The inner product space $\mathcal{F}_k$ is also known as the Reproducing Kernel Hilbert Space (RKHS) [3].

The replacement of the inner product by a positive definite kernel function is also known as the *kernel trick*. There are many different kernel functions. Two examples which give rise to a symmetric positive kernel matrix are the polynomial kernel $k(x, x') = (x^T x' + 1)^p$ with degree p and the Radial Basis Function (RBF) $k(x, x') = \exp(-\gamma\|x - x'\|^2)$. For the choice $\gamma = \frac{1}{2\sigma^2}$ the kernel is also known as the Gaussian kernel of variance σ^2.

2.1.1 Kernel methods

There are many methods available to detect linear relations in a given dataset. E.g. Linear Regression is used to model the relation between a set of independent explanatory variables and a dependent real valued output variable. Principal Component Analysis (PCA) [54] (see section 2.3) can be used for compression and denoising and is based on a linear transformation of the data. Logistic Regression [28], on the other hand, can be used for classification if the data is separable by a halfspaces. Those methods are well established and extensively studied in literature. Together they form a powerful set of methods for machine learning, but they all suffer from the problem that they cannot detect and utilize nonlinear patterns in the dataset. Kernel methods help to overcome this issue by mapping the data onto a Hilbert space in which the nonlinear dependencies can again be expressed in a linear fashion [46]. For example the dataset in Figure 2.1 is not separable in one dimension, but by using the feature map $\phi : \mathbb{R} \to \mathbb{R}^2, x \mapsto (x, x^2)$, the data becomes linearly separable.

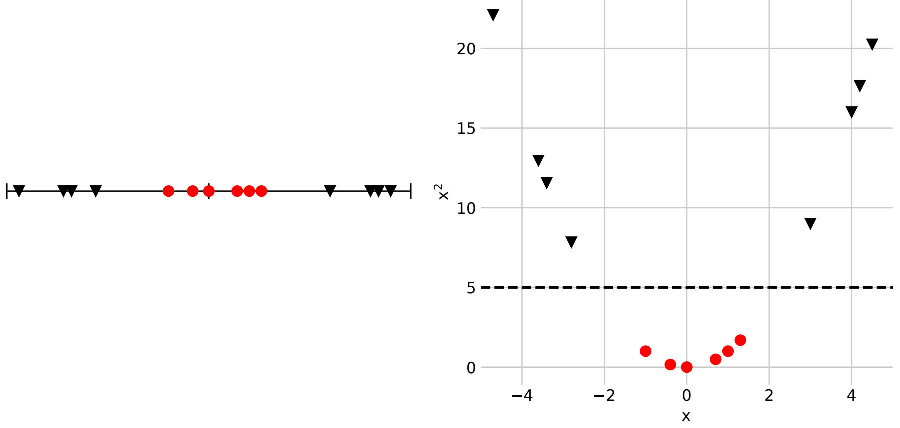


Figure 2.1: Feature space map $\phi : \mathbb{R} \to \mathbb{R}^2, x \mapsto (x, x^2)$

Using complicated feature mappings can result in very high-dimensional, or even infinite dimensional feature spaces. With the notion of kernels, we can still make use of those mappings, without ever explicitly evaluating them. Many algorithms depend on the evaluation of inner products in feature space. In practice, those inner products can be replaced by the specified kernel function. We call this step *kernelization*. It therefore is a natural way to extend linear models in order to recognize nonlinear patterns in the data. We will see this in more detail in 2.2, when we will kernelize Ridge Regression and also in section 2.3 for Principal Component Analysis.

2.1.2 Low-rank kernel approximation

For large sample size m we seek a low-rank approximation of the Gram matrix as a Nyström approximation of the kernel matrix arising from the training data split into $r \leq m$ randomly selected basis samples and the $m - r$ remaining samples [53].
In the case where the kernel matrix has rank $r \leq m$ we get an explicit form of the low-rank decomposition.

Lemma 2.1.1 (Low-rank approximation of the kernel matrix)**.** *Given a set of samples $\mathbf{x} = \{x_1, \dots, x_m\} \subseteq \mathcal{X}$ we assume to be able to pick a subset of $r \leq m$ samples $\mathbf{x}_r \subseteq \mathbf{x}$ such that $K[\mathbf{x_r}] \in \mathbb{R}^{r \times r}$ has full rank r. Let us further denote the set of remaining $m - r$ samples with $\mathbf{x}_{m-r}$ and assume the relabeled initial sample set $\mathbf{x} = \{\mathbf{x}_r, \mathbf{x}_{m-r}\}$ such that the associated kernel matrix gets block form*

$$K[\mathbf{x}] = \begin{pmatrix} K_{r,r} & K_{m-r,r}^T \\ K_{m-r,r} & K_{m-r,m-r} \end{pmatrix}, \tag{2.1}$$

where $K_{r,r} := K[\mathbf{x}_r] \in \mathbb{R}^{r \times r}$, $K_{m-r,r} := K[\mathbf{x}_{m-r}, \mathbf{x}_r] \in \mathbb{R}^{r \times (m-r)}$ and $K_{m-r,m-r} := K[\mathbf{x}_{m-r}] \in \mathbb{R}^{(m-r)\times(m-r)}$. Then there holds

$$K[\mathbf{x}] = \Phi_r \, \Phi_r^T, \tag{2.2}$$

with

$$\Phi_r := \Phi_r[\mathbf{x}] = \begin{pmatrix} K_{r,r}^{1/2} \\ K_{m-r,r} \, K_{r,r}^{-1/2} \end{pmatrix} = K_{m,r} \, K_{r,r}^{-1/2} \in \mathbb{R}^{m \times r}. \tag{2.3}$$

Proof. We first observe that

$$\Phi_r\,\Phi_r^T = \begin{pmatrix} K_{r,r} & K_{m-r,r}^T \\ K_{m-r,r} & K_{m-r,r}K_{r,r}^{-1}K_{m-r,r}^T \end{pmatrix}. \tag{2.4}$$

Since $K[\mathbf{x}]$ has rank r, we have the eigenvalue decomposition $K[\mathbf{x}] = U\Lambda U^T$ with $U \in \mathbb{R}^{m\times r}$ and the diagonal matrix $\Lambda \in \mathbb{R}^{r\times r}$ built from the r nonzero eigenvalue of $K[\mathbf{x}]$. Using the block notation $U = (U_r^T, U_{m-r}^T)^T$ we get

$$K[\mathbf{x}] = U\Lambda U^T = \begin{pmatrix} U_r\Lambda U_r^T & U_r\Lambda U_{m-r}^T \\ U_{m-r}\Lambda U_r^T & U_{m-r}\Lambda U_{m-r}^T \end{pmatrix}. \tag{2.5}$$

Note that $U_r^T U_r = I$ and hence

$$K_{m-r,r}K_{r,r}^{-1}K_{m-r,r}^T = (U_{m-r}\Lambda U_r^T)(U_r\Lambda^{-1}U_r^T)(U_r\Lambda U_{m-r}^T) = U_{m-r}\Lambda U_{m-r}^T = K_{m-r,m-r}, \tag{2.6}$$

which shows $K[\mathbf{x}] = \Phi_r\,\Phi_r^T$. Finally, the identity in Eqn. (2.3) simply follows from $K_{r,r}^{1/2} = K_{r,r}K_{r,r}^{-1/2}$. □

The computation of Φ_r needs $\mathcal{O}(mr + r^2)$ kernel evaluations. Each kernel evaluation takes an additional $\mathcal{O}(dim(\mathcal{X}))$ floating point operations. Evaluation of $K_{r,r}^{-1/2}$ has a cost of $\mathcal{O}(r^3)$ and matrix multiplication takes $\mathcal{O}(mr^2)$ operations. We see that the algorithm depends only linearly on the sample size m. This is a huge advantage to conventional kernel methods which depend on the calculation of the full kernel matrix and hence have a complexity of $\mathcal{O}(m^2)$. We further remark that for some kernels such as Gaussian RBF the above rank r assumption will only hold approximately for sufficiently large r, but the faster the eigenvalues of the Gramm matrix decrease, the smaller we can choose an approximate rank r.

From Lemma 2.1.1 Eqn. (2.3) we see that when mapping an individual data sample $y \in \mathcal{X}$ under $\phi : \mathcal{X} \to \mathbb{R}^r$ the corresponding feature vector is given as

$$\Phi_r(y) = \big(k(y,x_1),\ldots,k(y,x_r)\big)\,K_{r,r}^{-1/2}, \tag{2.7}$$

which holds true for $y \notin \mathbf{x}_r$ as it represents the respective row in $K_{m,r}\,K_{r,r}^{-1/2}$, but also for $y \in \mathbf{x}_r$ due to the identity $K_{r,r}^{1/2} = K_{r,r}K_{r,r}^{-1/2}$. We summarize this remark for later reference.

Corollary 2.1.2. *Under the assumptions of Lemma 2.1.1 the matrix of feature vectors of data samples $\mathbf{y} = \{y_1,\ldots,y_\ell\}$ is given as*

$$\Phi_r[\mathbf{y}] = K[\mathbf{y},\mathbf{x}_r]\,K_{r,r}^{-1/2}, \tag{2.8}$$

with $K[\mathbf{y},\mathbf{x}_r] = \big(k(y_i,x_j)_{i,j}\big) \in \mathbb{R}^{\ell\times r}$.

Later we will use the low-rank approximation of the kernel matrix to enhance efficiency in (un)supervised learning via Kernel Principal Component Analysis (kPCA) and Kernel Ridge Regression (KRR).

2.2 Kernel Ridge Regression

2.2.1 Linear Regression

Possibly the most elementary algorithm that can be kernelized is ridge regression. The task is to find a linear function that models the dependencies between input variables $X \subseteq \mathcal{X}$ and output variables $Y \subseteq \mathcal{Y}$. Let X be a $m \times \tilde{N}$ real valued matrix, where the rows contain the training samples x_i, for $i = 1, \ldots, m$, having $\tilde{N}$ features. Respectively let Y be a $m \times \tilde{M}$ matrix, containing the output samples y_i with $\tilde{M}$ features. The classical way to do that is to minimize the quadratic cost,

$$\mathcal{L}_{LR}(W) = \frac{1}{2} \sum_{i=1}^{m} \|y_i - W^T x_i\|^2, \tag{2.9}$$

where W is a $\tilde{N} \times \tilde{M}$ matrix containing the weights [51]. Minimizing $\mathcal{L}$ leads to classical linear regression.

2.2.2 Ridge Regression

However, calculation of the linear regression coefficients can be difficult if the data tends to be non-orthogonal. This can cause large coefficient values and some might even have the wrong sign [22]. On the other hand, very high dimensional data often suffers from independent variables being highly correlated, which often leads to poor performance of Multiple Linear Regression. The usual way to handle this issue is regularization. A simple yet effective way to regularize, is to penalize the norm of W. This is sometimes called *weight-decay*. Adding the regularization term, the cost function becomes

$$\begin{aligned} \mathcal{L}_{RR}(W) &= \frac{1}{2} \sum_{i=1}^{m} \|y_i - W^T x_i\|^2 + \frac{1}{2}\alpha \|W^T\|_F^2 \\ &= \frac{1}{2}\|Y^T - W^T X^T\|_F^2 + \frac{1}{2}\alpha\|W^T\|_F^2. \end{aligned} \tag{2.10}$$

The regularization parameter $\alpha \in \mathbb{R}^+$ determines the strength of the regularization. Further, $\|A\|_F = \sqrt{\sum_{i=1}^{M} \sum_{j=1}^{N} a_{ij}^2}$ denotes the Frobenius norm.

Proposition 2.2.1. *The solution of the minimization problem*

$$\arg\min_{W} \mathcal{L}_{RR}(W)$$

is unique and given by

$$W^T = Y^T X (X^T X + I\alpha)^{-1} \quad \textit{primal solution} \tag{2.11}$$

$$W^T = Y^T (X X^T + I\alpha)^{-1} X \quad \textit{dual solution.} \tag{2.12}$$

The following lemma will be useful to proof proposition 2.2.1.

Lemma 2.2.2 (Push-through identity). *Let A be a $l \times k$ and B a $k \times l$ sized matrix, then the following identity holds*

$$A(BA + I_k\alpha)^{-1} = (AB + I_l\alpha)^{-1}A, \tag{2.13}$$

where I_k denotes the identity of size k.

Proof. Starting with

$$(AB + I_l\alpha)A = A(BA + I_k\alpha),$$

and multiplying with $(BA + I_k\alpha)^{-1}$ from the right and $(AB + I_l\alpha)^{-1}$ from the left results in

$$A(BA + I_k\alpha)^{-1} = (AB + I_l\alpha)^{-1}A.$$

□

The identity has its name, because the matrix A, when pushed into the inverse from the left, is pushed through to the right.

Proof of Prop. 2.2.1. By taking the matrix derivative of the loss function (2.10) and equating to zero gives

$$\begin{aligned}\nabla_W \mathcal{L}_{RR}(W) &= \frac{1}{2}\nabla_W\big(\operatorname{tr}(YY^T) - 2\operatorname{tr}(XWY^T) + \operatorname{tr}(XWW^TX^T) + \alpha\operatorname{tr}(WW^T)\big)\\ &= -Y^TX + W^TX^TX + \alpha W^T = 0\end{aligned}$$

Solving for W results in the primal solution

$$W^T = Y^TX(X^TX + I\alpha)^{-1}$$

To derive the dual solution, one can apply Lemma 2.2.2 and get

$$W^T = Y^T(XX^T + I\alpha)^{-1}X.$$

□

To predict the value of a new point x, the point is projected onto the space spanned by the rows of W^T. Out hypothesis f therefore becomes

$$f(x) = W^Tx = Y(X^TX + I\alpha)^{-1}X^Tx. \tag{2.14}$$

2.2.3 Kernelization

To apply the kernel trick, we use the feature map $\phi : \mathcal{X} \to \mathcal{F}_k$, $x \mapsto \phi(x)$ to transform the features to a RKHS. This gives us the following cost function:

$$\mathcal{L}_{KRR}(W) = \frac{1}{2}\sum_{i=1}^{m} \|y_i - W^T\phi(x_i)\|^2 + \frac{1}{2}\alpha\|W\|_F^2. \tag{2.15}$$

The problem arises when we actually want to solve this optimization problem, since we cannot explicitly calculate $\phi(x)$. Nevertheless, the solution of W is given by

$$W^T = Y^T\Phi[\mathbf{x}](\Phi[\mathbf{x}]^T\Phi[x] + I\alpha)^{-1} \quad \text{primal solution} \tag{2.16}$$

$$W^T = Y^T(\Phi[\mathbf{x}]\Phi[\mathbf{x}]^T + I\alpha)^{-1}\Phi[\mathbf{x}] \quad \text{dual solution,} \tag{2.17}$$

where $\Phi[\mathbf{x}]$ denotes the matrix of size $m \times dim(\mathcal{F}_k)$ which holds the samples mapped to the feature space

$$\Phi[\mathbf{x}] = [\phi(x_1)|\dots|\phi(x_m)]^T. \tag{2.18}$$

The inner products of the RKHS can now be replaced by the kernel function (Def. 2.1.1), e.g. $K[\mathbf{x}] = \Phi[\mathbf{x}]\Phi[\mathbf{x}]^T$. By applying the kernel trick to the hypothesis (2.14), the equation changes to

$$f_{KRR}(x) = W^T\phi(x) = Y^T(K[\mathbf{x}] + I\alpha)^{-1}K[\mathbf{x}, x]. \tag{2.19}$$

Therefore, when using KRR, it is only necessary to calculate the kernel function, but not to explicitly evaluate the map $\phi(x)$.

The hypothesis f_{KRR} can also be seen as a weighted sum of *centered* kernels:

$$f_{KRR}(x) = \sum_{i=1}^{m} \omega_i k(x_i, x), \tag{2.20}$$

where ω_i is the i_{th} column of $Y^T(K[\mathbf{x}] + I\alpha)^{-1}$. As an illustration, we show Figure 2.2. A KRR with a RBF kernel $k(x, y) = e^{-\gamma\|x-y\|^2}$ is performed on a dataset with labels -1 and 1 and with two different value for the kernel parameter γ. We can see that for a larger value of gamma, the centered kernel functions are more isolated, whereas for smaller values they merge together and create a sharper decisions boundary.

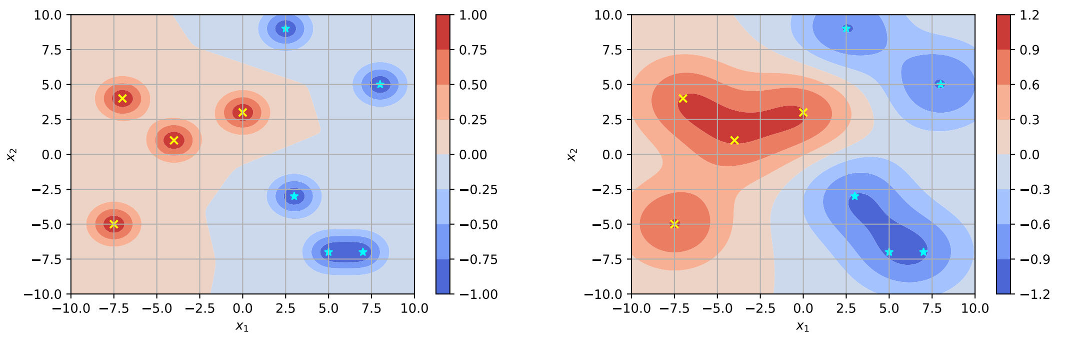


Figure 2.2: Example of Kernel Ridge Regression with a RBF kernel $k(x, y) = e^{-\gamma\|x-y\|^2}$ on a dataset with features x_1, x_2 and labels -1 and 1.

2.2.4 Low-rank KRR and the primal solution

By utilizing a low-rank approach or rank r to approximate the Gram matrix $K[\mathbf{x}] \approx \Phi_r\Phi_r^T$, it is possible to use the primal solution (2.16) in order to solve the hypothesis

$$f_{KRR}(x) \approx Y^T(\Phi_r\Phi_r^T + I\alpha)^{-1}\Phi_r\Phi_r[x]^T = Y^T\Phi_r(\Phi_r^T\Phi_r + I\alpha)^{-1}\Phi_r[x]^T. \qquad (2.21)$$

The low-rank mapping $\Phi_r[x] \in \mathbb{R}^{1\times r}$ of the new data sample x is calculated by using Corollary 2.1.2. The approximate Gram matrix $\Phi_r\Phi_r^T$ is still of size $m \times m$, but only has a rank of r. By using the primal solution, we have to calculate the covariance matrix $\Sigma = \Phi_r^T\Phi_r$, which is only of size $r \times r$ instead. This is equivalent to first apply the Nyström approximation (2.3) to the dataset, following a linear Ridge regression (2.14).

2.2.5 Low-rank eigenvalue regularization

Since the map Φ is equivalent to a change of the feature space, the new feature space might be very high (possibly infinite) dimensional, depending on the kernel k. Since the number of training samples m is much smaller than the number of features, the problem becomes underdetermined. We therefore need the regularization parameter α in order to shift the small eigenvalues of the Gram matrix away from zero, such that $K[\mathbf{x}] + I\alpha$ becomes well conditioned. Instead of shifting the eigenvalues, another way to regularize could be done by performing an eigenvalue decomposition of the Gram matrix $K[\mathbf{x}] = U\Lambda U^T$ and then neglecting small eigenvalues and their corresponding eigenvectors up to a predetermined threshold ϵ. The problem would change to

$$f_\epsilon(x) = W_\epsilon^T\phi(x) = Y(U\Lambda_\epsilon U^T)^{-1}K[\mathbf{x}, x], \qquad (2.22)$$

where Λ_ϵ denotes the diagonal matrix with all eigenvalues $< \epsilon$ set to zero. On first glance this might not seem very useful for kernel matrices of size $m \times m$ if m is large, since it adds the extra burden of eigenvalue decomposition. For large training sets, this can become infeasible. However, if we use a low-rank approximation as described above, we could think of regularizing the covariance matrix $\Sigma = \Phi_r^T \Phi_r$ by this technique instead

$$f_\epsilon(x) = Y\Phi_r(V\Lambda_\epsilon V^T)^{-1}\phi(x), \tag{2.23}$$

with $\Phi_r^T\Phi_r = V\Lambda V^T$ being the eigenvalue decomposition of the covariance matrix.

There are two regularizations happening here. The first one occurs when we choose the rank r and the second one is given by the parameter ϵ. If we choose the rank appropriately, we could omit the second regularization. But since we usually don't know the rank in the first place, we can achieve the desired regularization with the second regularization parameter ϵ. One could calculate the rank by first setting ϵ and further investigate the number of eigenvalues which had been cut off. If this number is very large, one can likely choose a smaller rank and hence reduce the computational effort.

For this task, it is useful to set the parameter ϵ relative to the approximation error of the covariance matrix. This way, choosing a higher rank r does not change the result. Using the *Eckart-Young theorem* (see Appendix A.2) the error is given by $\sqrt{\sum_{i:\lambda_i<\epsilon}|\lambda_i|^2}$, where λ_i denotes the diagonal entries of Λ. The relative error is therefore given by

$$\epsilon_{rel} = \frac{\sqrt{\sum_{i:\lambda_i<\epsilon}|\lambda_i|^2}}{\sqrt{\sum_{i=1}^r|\lambda_i|^2}}. \tag{2.24}$$

We can calculate the relative error for all eigenvalues and remove those which are smaller than the chosen threshold.

2.3 Principal Component Analysis

Principal Component Analysis is an unsupervised learning method which aims to find the subspace of largest variance. The number of dimensions can be chosen beforehand. This is useful if the directions of small variance do not contain interesting information and can be neglected. The method can therefore be used to filter noise from a signal. Often, however, the directions of small variance contain valuable information. In this case Independent Component Analysis (ICA) can be used instead [50].

PCA projects a N dimensional dataset orthogonally onto a d-dimensional subspace. Figure 2.3 illustrates how a two-dimensional dataset is projected onto a line.

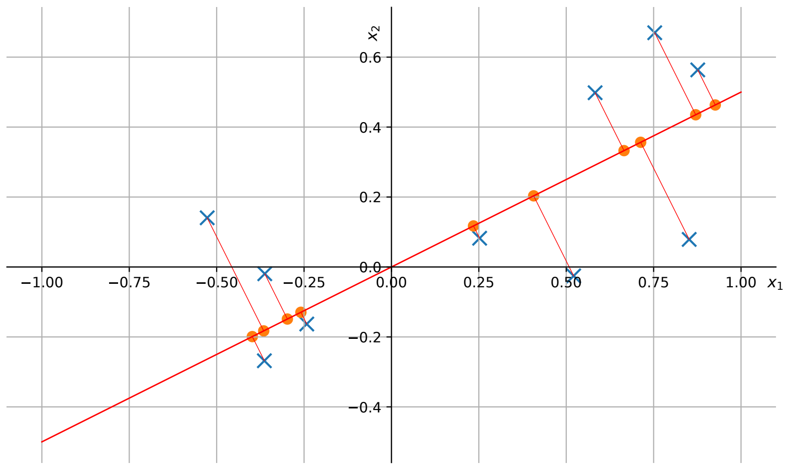


Figure 2.3: Orthogonal projection of a two-dimensional dataset onto a one dimensional subspace.

Let $x \in \mathbb{R}^N$ be a N dimensional vector and $U_d = [v^{(1)}, \ldots, v^{(d)}] \in \mathbb{R}^{N \times d}$ a set of orthonormal vectors. To project x onto the d-dimensional subspace which is spanned by the components of U_d, we can calculate the inner products $v^{(i)T}x,\ i = 1, \ldots, d$. This can be written in matrix form

$$z = U_d^T x, \tag{2.25}$$

where z denotes the d-dimensional *encoded* vector of x. The reconstruction $\hat{x}$ of x is given by

$$\hat{x} = U_d z = U_d U_d^T x. \tag{2.26}$$

Theorem 2.3.1 (PCA). *Let $X = [x_1, \ldots, x_m]^T \in \mathbb{R}^{m \times N}$ be the matrix of m training samples with zero mean. Suppose we want to find an orthogonal set of d linear basis vectors $U_d = [v^{(1)}, \ldots, v^{(d)}] \in \mathbb{R}^{N \times d}$, such that the average* reconstruction error

$$\mathcal{L}_{PCA}(U_d) = \sum_{i=1}^{m} \|x_i - \hat{x}_i\|^2 \tag{2.27}$$

is minimized. The optimal solution is obtained by choosing U_d equal to the d eigenvectors with largest eigenvalues of the empirical covariance matrix $\hat{\Sigma} = \frac{1}{m}\sum_{i=1}^{m} x_i x_i^T = \frac{1}{m}X^T X$. Furthermore, the optimal low-dimensional encoding of the data is given by $Z = U_d^T X$, which is an orthogonal projection of the data onto the column space spanned by the eigenvectors. The matrix $U_d U_d^T$ is an orthogonal projection matrix.

Proof [34]. Let $z_i = U_d^T x_i$ be the low dimensional representation of x_i under the transformation U_d. We can express the average reconstruction error as follows

$$\begin{aligned} \mathcal{L}_{PCA}(U_d) &= \frac{1}{m}\sum_{i=1}^{m} \|x_i - U_d U_d^T x_i\|^2 \\ &= \frac{1}{m}\sum_{i=1}^{m}(x_i^T x_i - 2x_i^T U_d U_d^T x_i + 2x_i^T (U_d U_d^T)(U_d U_d^T) x_i). \end{aligned} \tag{2.28}$$

Since the matrix U_d is orthonormal, it follows that $(U_d U_d^T)(U_d U_d^T) = U_d U_d^T$. Therefore,

$$\begin{aligned} \mathcal{L}_{PCA}(U_d) &= \frac{1}{m}\sum_{i=1}^{m}(x_i^T x_i - x_i^T U_d U_d^T x_i) \\ &= \frac{1}{m}\sum_{i=1}^{m} x_i^T x_i - \frac{1}{m}\sum_{i=1}^{m} z_i^T z_i. \end{aligned} \tag{2.29}$$

The first term is constant and if we denote the j^{th} component of the low-dimensional representation z_i with $z_i^{(j)}$, we can write the objective as

$$\mathcal{L}_{PCA}(U_d) = \text{const} - \frac{1}{m}\sum_{j=1}^{d}\sum_{i=1}^{m} z_i^{(j)^2}. \tag{2.30}$$

Let $\tilde{z}^{(j)} \in \mathbb{R}^m$ denote the j^{th} component of all the low-dimensional vectors $z_i,\ i = 1, \ldots, m$. Then, assuming that the data is centered in feature space, the empirical variance of the j^{th} projected coordinate vector is given by

$$\mathrm{var}\left\{\tilde{z}^{(j)}\right\} = \mathbb{E}\left\{\tilde{z}^{(j)^2}\right\} - (\mathbb{E}\left\{\tilde{z}^{(j)}\right\})^2 = \mathbb{E}\left\{\tilde{z}^{(j)^2}\right\} = \frac{1}{m}\sum_{i=1}^{m}(z_i^{(j)^2}). \tag{2.31}$$

Therefore, minimizing the reconstruction error is equivalent to maximizing the empirical variance of the low d-dimensional representation. This is why it is said that PCA finds the directions of maximal variance.

The variance of the j^{th} component can also be written as

$$\frac{1}{m}\sum_{i=1}^{m} {z_i^{(j)}}^2 = \frac{1}{m}\sum_{i=1}^{m} {v^{(j)}}^T x_i x_i^T v^{(j)} = {v^{(j)}}^T \hat{\Sigma} v^{(j)}, \tag{2.32}$$

It would be trivial to maximize the variance of the projection by letting $\|v^{(j)}\| \to \infty$, so the additional constrain $\|v^{(j)}\| = 1$ is needed. By using Lagrange Multiplier calculus, the objective, which we need to maximize now, is given by

$$\hat{\mathcal{L}}_{PCA}(v^{(j)}) = {v^{(j)}}^T \hat{\Sigma} v^{(j)} + \lambda_j ({v^{(j)}}^T v^{(j)} - 1) \tag{2.33}$$

where λ_j is the Lagrange multiplier. By setting the derivative to zero we get the following *eigenvalue problem*,

$$\begin{aligned} \frac{\partial}{\partial v^{(j)}} \hat{\mathcal{L}}_{PCA}(v^{(j)}) &= 2\hat{\Sigma} v^{(j)} - 2\lambda_j v^{(j)} = 0 \\ \hat{\Sigma} v^{(j)} &= \lambda_j v^{(j)} \end{aligned} \tag{2.34}$$

Hence, the direction that maximizes the variance of the j^{th} component of the low-dimensional representation is an eigenvector of the covariance matrix. The matrix $\hat{\Sigma}$ is SPD, therefore, left multiplying by $v^{(j)}$ shows that

$${v^{(j)}}^T \hat{\Sigma} v^{(j)} = \lambda_j > 0, \quad \text{for all } j = 1, \ldots, N. \tag{2.35}$$

In order to maximize the variance, we must therefore pick those eigenvectors which correspond to the d largest eigenvalues. □

The algorithm requires that the data has zero mean. We can simply center the data by subtracting the mean

$$\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i. \tag{2.36}$$

Having found the eigenvectors U_d of the covariance matrix, we can now project a new sample $x' \in \mathbb{R}^N$ onto the space spanned by the columns of U_d and hence get a low level representation of the new data point

$$z' = U_d^T (x' - \bar{x}). \tag{2.37}$$

2.3.1 Kernel Principal Component Analysis

In case there are more dimensions than data samples, such that some dimensions remain unoccupied, it is not hard to see that the eigenvectors that span the projection space must lie in the subspace spanned by the sample vectors, i.e.,

$$\lambda_j v^{(j)} = \hat{\Sigma} v^{(j)} = \frac{1}{m}\sum_{i=1}^{m} x_i x_i^T v^{(j)} = \frac{1}{m}\sum_{i=1}^{m} (x_i^T v^{(j)}) x_i. \tag{2.38}$$

For $\lambda_j \neq 0$ we get for the eigenvector the following linear combination of data points

$$v^{(j)} = \sum_{i=1}^{m} \frac{x_i^T v^{(j)}}{m\lambda_j} x_i. \tag{2.39}$$

We now derive a kernelized version of the above procedure. For that purpose, we define the covariance matrix of the data points in feature space $\mathcal{F}_k$.

$$\bar{\Sigma} = \frac{1}{m}\sum_{i=1}^{m} \phi(x_i)\phi(x_i)^T. \tag{2.40}$$

. The solution of PCA of this transformed system is given by the eigenvectors $v^{(j)}$ of

$$\bar{\Sigma} v^{(j)} = \lambda_j v^{(j)}. \tag{2.41}$$

Since the dimension $dim(\mathcal{F}_k)$ might not be finite, in general, problem (2.41) can not be computed directly. However, from equation (2.39) we see that the solution is given by a linear combination of the transformed data points [41]

$$v^{(j)} = \sum_{i=1}^{m} \hat{\alpha}_i^{(j)} \phi(x_i). \tag{2.42}$$

Therefore, instead of solving for $v^{(j)} \in dim(\mathcal{F}_k)$, we can also solve for the *dual* coefficients $\hat{\alpha}^{(j)} \in \mathbb{R}^m$.

Theorem 2.3.2 (kPCA). *Let $\mathbf{x} = \{x_1, \dots, x_m\} \subseteq \mathcal{X}$ be a dataset in $\mathcal{X}$. Further, let $\Phi : \mathcal{X} \to \mathcal{F}_k$ be a feature space mapping to some Hilbert space (possibly of infinite dimension) and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, a kernel function as defined in 2.1.1. The kernel PCA generates the kernel principal axes $v^{(j)} = \frac{1}{\sqrt{\lambda_j}} \sum_{i=1}^{m} \alpha_i^{(j)} \phi(x_i)$, $j = 1, 2, \dots, m$, where the coefficient vectors $\alpha^{(j)} \in \mathbb{R}^m$, $j = 1, \dots, m$, result from the eigenvalue problem*

$$\bar{K}\alpha^{(j)} = \lambda_j \alpha^{(j)}, \tag{2.43}$$

with the centered Gram matrix $\bar{K} = K[\mathbf{x}] - \mathbf{1}_m K[\mathbf{x}] - K[\mathbf{x}]\mathbf{1}_m + \mathbf{1}_m K[\mathbf{x}]\mathbf{1}_m \in \mathbb{R}^{m\times m}$ and $(\mathbf{1}_m)_{ij} = 1/m$. The eigenvalue problem (2.43) is solved for nonzero eigenvalues. The j^{th} kernel principal component of a data point $x \in \mathcal{X}$ can be extracted by the projection

$$p^{(j)}(x) = \phi(x)^T v^{(j)} = \frac{1}{\sqrt{\lambda_j}} \sum_{i=1}^{m} \alpha_i^{(j)} k(x_i, x). \tag{2.44}$$

Proof [41]. Since the eigenvectors can be written as a linear combination of the data instances, this implies that instead of the eigenvalue equation (2.34), one can also solve for $\alpha^{(j)}$ by considering the m projected equations $\phi(x_i)^T\bar{\Sigma}v^{(j)} = \hat{\lambda}_j \phi(x_i)^T v^{(j)}$ for $i = 1, \dots, m$ as follows:

$$\begin{aligned}
\phi(x_i)^T\bar{\Sigma}v^{(j)} &= \hat{\lambda}_j \phi(x_i)^T v^{(j)} \\
\phi(x_i)^T \left[\frac{1}{m}\sum_{l=1}^{m}\phi(x_l)\phi(x_l)^T\right] \sum_{k=1}^{m} \alpha_k^{(j)}\phi(x_k) &= \hat{\lambda}_j \phi(x_i)^T \sum_{l=1}^{m} \alpha_l^{(j)} \phi(x_l) \\
\frac{1}{m}\sum_{l=1}^{m}\sum_{k=1}^{m} \alpha_k^{(j)} (\phi(x_i)^T\phi(x_l))(\phi(x_l)^T\phi(x_k)) &= \hat{\lambda}_j \sum_{l=1}^{m} \alpha_l^{(j)} (\phi(x_i)^T\phi(x_l))
\end{aligned} \tag{2.45}$$

Starting with equation (2.45) and assuming the data is centered in feature space, we apply the kernel trick be replacing the inner products with the kernel function, i.e. $\phi(x_i)^T\phi(x_l) = k(x_i, x_l)$.

$$\frac{1}{m}\sum_{l=1}^{m}\sum_{k=1}^{m} \alpha_k^{(j)} k(x_i, x_l) k(x_l, x_k) = \hat{\lambda}_j \sum_{l=1}^{m} \alpha_l^{(j)} k(x_i, x_l). \tag{2.46}$$

Using matrix notation we can express the m equations as,

$$\bar{K}[\mathbf{x}]^2\alpha^{(j)} = m\hat{\lambda}_j\bar{K}[\mathbf{x}]\alpha^{(j)}, \tag{2.47}$$

and simplifying further, using the non-singularity of the kernel matrix,

$$\bar{K}[\mathbf{x}]\alpha^{(j)} = \lambda_j\alpha^{(j)} \text{ with } \lambda_j = m\hat{\lambda}_j. \tag{2.48}$$

The definition of PCA requires that the data is centered. For now we assume that the data is centered in feature space. Later we will see how to center the data using only the kernel matrix.

Equation (2.48) gives an eigenvalue equation for $\alpha^{(j)}$ for $j = 1, \dots, m$ which in turn completely determines the eigenvectors $v^{(j)}$ by (2.42). Let $\hat{\alpha}^{(j)}$ be the j^{th} eigenvector of (2.48) such that $v^{(j)}$ is normalized. By using equation (2.42), the norm $\|\hat{\alpha}^{(j)}\| = 1/\sqrt{\lambda_j}$, which follows from the calculation:

$$\begin{aligned} {v^{(j)}}^T v^{(j)} &= \sum_{i=1}^{m}\sum_{k=1}^{m}\hat{\alpha}_i^{(j)}\hat{\alpha}_k^{(j)}k(x_i, x_k) \\ &= {\hat{\alpha}^{(j)}}^T K[\mathbf{x}]\hat{\alpha}^{(j)} \\ &= \lambda_j {\hat{\alpha}^{(j)}}^T\hat{\alpha}^{(j)} = \lambda_j\|\hat{\alpha}^{(j)}\|^2 = 1 \end{aligned} \tag{2.49}$$

For $\|\alpha^{(j)}\| = 1$, it is necessary to scale the eigenvectors such that $\hat{\alpha}^{(j)} = \frac{1}{\sqrt{\lambda_j}}\alpha^{(j)}$.

□

We define the d-dimensional projection operator P_d which minimizes the reconstruction error in the feature space $\mathcal{F}_k$, i.e.,

$$P_d = \arg\min_P \mathcal{L}_{kPCA}(P) := \sum_{i=1}^{m}\|\phi(x_i) - P\phi(x_i)\|^2_{\mathcal{F}_k}. \tag{2.50}$$

This is analogue to the reconstruction error of linear PCA (2.27).

Lemma 2.3.3. *The projection image $P_d\phi(x)$ is spanned by the kernel principal axes $v^{(j)}$, $j = 1 \dots, d$ and P_d is an orthogonal projection.*

Proof. For a projection $P_dP_d\phi(x) = P_d\phi(x)$ must hold. The operation can be written as a linear combination in its orthonormal basis

$$P_d\phi(x) = \sum_{k=1}^{d}({v^{(k)}}^T\phi(x))v^{(k)}. \tag{2.51}$$

Therefore, we have

$$\begin{aligned} P_d(P_d\phi(x)) &= \sum_{k=1}^{d}{v^{(k)}}^T\Big(\sum_{l=1}^{d}({v^{(l)}}^T\phi(x))v^{(l)}\Big)v^{(k)} \\ &= \sum_{k=1}^{d}\sum_{l=1}^{d}\Big(({v^{(l)}}^T\phi(x))({v^{(k)}}^T v^{(l)})\Big)v^{(k)} \\ &= \sum_{k=1}^{d}\Big(({v^{(k)}}^T\phi(x))\Big)v^{(k)} = P_d\phi(x). \end{aligned} \tag{2.52}$$

Further, a projection is orthogonal if and only if it is self-adjoint $P_d\phi(x)^T\phi(y) = \phi(x)^T P_d\phi(y)$:

$$\begin{aligned} P_d\phi(x)^T\phi(y) &= \Big(\sum_{k=1}^{d}(v^{(k)^T}\phi(x))v^{(k)^T}\Big)\phi(y) \\ &= \sum_{k=1}^{d}(v^{(k)^T}\phi(x))(v^{(k)^T}\phi(y)) \\ &= \phi(x)^T\Big(\sum_{k=1}^{d}(v^{(k)^T}\phi(y))v^{(k)}\Big) = \phi(x)^T P_d\phi(y). \end{aligned} \tag{2.53}$$

Since we have proven that P_d is an orthogonal projection, it remains to proof that it minimizes the reconstruction error. This is analogue to the proof of Theorem 2.3.1. □

The individual components of a new data point x are given by equation (2.44). This defines the d-dimensional low level representation of P_d.

Corollary 2.3.4. *Given a dataset $\mathbf{x} = \{x_1, \ldots, x_m\} \subseteq \mathcal{X}$ and let the matrix L denote the $m \times d$ matrix with the entries $L_{ij} = \frac{1}{\sqrt{\lambda_j}}\alpha_i^{(j)}$, the optimal d-dimensional encoding of the data is given by*

$$p(x) = L^T K[\mathbf{x}, x]. \tag{2.54}$$

□

We define the matrix $p[\mathbf{x}] \in \mathbb{R}^{m\times d} = [p(x_1), \ldots, p(x_m)]^T$ which contains the kernel principal features in its columns.

Note that we still assume that the data is centered in feature space. In practice, the kernel matrix of the new data instance needs to be centered as well, which is described in the next section, see equation (2.58).

Centering the data in feature space

In the kernelized version, the data is mapped onto a new feature space by the map $\phi : \mathcal{X} \to \mathcal{F}_k$. Even if the data might be centered in the original space $\mathcal{X}$, it is not guaranteed that the data is still centered in the new feature space $\mathcal{F}_k$. It is very difficult, and in many cases impossible to explicitly center the data in feature space. But, since the algorithm only depends on the kernel matrix, one can ask the question of how the centered kernel matrix would have to look.

Lemma 2.3.5. *For the feature space mapping $\phi : \mathcal{X} \to \mathcal{F}_k, x \mapsto \phi(x)$ and a dataset $\mathbf{x} = \{x_1, \ldots, x_m\} \subseteq \mathcal{X}$, let $\Phi[\mathbf{x}] = [\phi(x_1), \ldots, \phi(x_m)]$ be the dataset of size $m \times dim(\mathcal{F}_k)$ which contains the transformed data points in its rows. The kernel matrix is given by $K[\mathbf{x}] = \Phi[\mathbf{x}]\Phi[\mathbf{x}]^T$. Then the centered version of the kernel matrix is given by*

$$\bar{K}[\mathbf{x}] = K[\mathbf{x}] - \mathbf{1}_m K[\mathbf{x}] - K[\mathbf{x}]\mathbf{1}_m + \mathbf{1}_m K[\mathbf{x}]\mathbf{1}_m, \tag{2.55}$$

where $\mathbf{1}_m$ denotes the $m \times m$ matrix with all entries equal to $1/m$.

Proof [43]. A centered version of the data $\Phi[\mathbf{x}]$ is given by

$$\bar{\Phi}[\mathbf{x}] = \Phi[\mathbf{x}] - \mathbf{1}_m\Phi[\mathbf{x}], \tag{2.56}$$

The centered kernel matrix $\bar{K}[\mathbf{x}]$ is therefore given by

$$\begin{aligned} \bar{K}[\mathbf{x}] &= (\Phi[\mathbf{x}] - \mathbf{1}_m\Phi[\mathbf{x}])(\Phi[\mathbf{x}] - \mathbf{1}_m\Phi[\mathbf{x}])^T \\ &= \Phi[\mathbf{x}]\Phi[\mathbf{x}]^T - \mathbf{1}_m\Phi[\mathbf{x}]\Phi[\mathbf{x}]^T - \Phi[\mathbf{x}]\Phi[\mathbf{x}]^T\mathbf{1}_m + \mathbf{1}_m\Phi[\mathbf{x}]\Phi[\mathbf{x}]^T\mathbf{1}_m \\ &= K[\mathbf{x}] - \mathbf{1}_m K[\mathbf{x}] - K[\mathbf{x}]\mathbf{1}_m + \mathbf{1}_m K[\mathbf{x}]\mathbf{1}_m. \end{aligned} \tag{2.57}$$

□

The *rectangular* kernel matrix related to a set of new data points $\mathbf{y} \subseteq \mathcal{X}$ can be centered by

$$\begin{aligned}\bar{K}[\mathbf{x},\mathbf{y}] &= (\Phi[\mathbf{x}] - \mathbf{1}_m\Phi[\mathbf{x}])(\Phi[\mathbf{y}] - \mathbf{1}_m\Phi[\mathbf{x}])^T \\ &= K[\mathbf{x},\mathbf{y}] - \mathbf{1}_m K[\mathbf{x},\mathbf{y}] - K[\mathbf{x}]\mathbf{1}_m + \mathbf{1}_m K[\mathbf{x}]\mathbf{1}_m. \end{aligned} \tag{2.58}$$

2.3.2 Low-rank kernel principal component analysis

In the course of the kPCA algorithm one has to solve d eigenvalue problems of the form $\bar{K}\alpha^{(j)} = m\lambda_j\alpha^{(j)}$, $j = 1,\dots,d$, which now take the particular form

$$\bar{\Phi}_r\bar{\Phi}_r^T\alpha^{(j)} = \lambda_j\alpha^{(j)},\ j = 1,\dots,d, \tag{2.59}$$

with $\bar{\Phi}_r = \Phi_r - \mathbf{1}_m\Phi_r$ and $\Phi_r = \Phi_r[\mathbf{x}] = K_{m,r}K_{r,r}^{-1/2} \in \mathbb{R}^{m\times r}$ with $K[\mathbf{x}] \approx \Phi_r[\mathbf{x}]\Phi_r[\mathbf{x}]^T$ being the low-rank approximation of the kernel matrix. An eigenvalue problem of the form (2.59) can be efficiently solved for nonzero eigenvalues.

Lemma 2.3.6 (Low-rank eigenvalue problem)**.** *The eigenpairs (v,λ) with $\lambda \neq 0$ of $\Phi_r\Phi_r^T \in \mathbb{C}^{m\times m}$ with $\Phi_r \in \mathbb{C}^{m\times r}$ are given by $(\Phi_r w,\lambda)$ with $\Phi_r^T\Phi_r w = \lambda w$. Particularly, there holds for $\|w\|_2 = 1$ that $\|v\|_2 = \lambda^{1/2}$, i.e., $v = \lambda^{-1/2}\,\Phi_r w$ has unit length.*

Proof. Suppose $\Phi_r^T\Phi_r w = \lambda w$ with $\lambda \neq 0$. Then we have $\Phi_r\Phi_r^T(\Phi_r w) = \lambda(\Phi_r w)$ with $\Phi_r w \neq 0$, since otherwise multiplication with Φ_r^T yields $\Phi_r^T\Phi_r w = 0$ and thus, $\lambda = 0$, contradicting the assumption $\lambda \neq 0$ in the first place. Hence, $(\Phi_r w,\lambda)$ is an eigenpair of $\Phi_r\Phi_r^T$. Moreover, $\|\Phi_r w\|_2^2 = w^T(\Phi_r^T\Phi_r w) = \lambda w^T w = \lambda$. □

The remarkable consequence of Lemma 2.3.6 is a significant reduction in complexity when solving the eigenvalue problems in the kPCA with low-rank kernel matrix approximation in the case $r \ll m$. Specifically, the computational complexity is reduced from $\mathcal{O}(m^2)$ to $\mathcal{O}(r^2)$ for each of the d eigenvalue problems. We now have the tools to define a low-rank version of the kPCA.

Definition 2.3.1 (Low-rank kPCA)**.** *Given inputs $\mathbf{x} = \{x_1,\dots,x_m\} \subseteq \mathcal{X}$, a kernel $k : \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ and a low-rank approximation of $\bar{K}[\mathbf{x}] = K[\mathbf{x}] - \mathbf{1}_m K[\mathbf{x}] - K[\mathbf{x}]\mathbf{1}_m + \mathbf{1}_m K[\mathbf{x}]\mathbf{1}_m \in \mathbb{R}^{m\times m}$ by $\bar{\Phi}_r\bar{\Phi}_r^T = (\Phi_r - \mathbf{1}_m\Phi_r)(\Phi_r - \mathbf{1}_m\Phi_r)^T$ from a choice of a subset $\mathbf{x}_r \subseteq \mathbf{x}$ according to Lemma 2.1.1. The low-rank version of the kernel PCA generates $r \le m$ kernel principal axes $v^{(j)} = \frac{1}{\sqrt{\lambda_j}}\sum_{i=1}^m \alpha_i^{(j)}\bar{\Phi}_r(x_i)$, $j = 1,2,\dots,r$, where the coefficient vectors $\alpha^{(j)} \in \mathbb{R}^m$ result from the eigenvalue problem*

$$\bar{\Phi}_r\bar{\Phi}_r^T\alpha^{(j)} = \lambda_j\alpha^{(j)}, \tag{2.60}$$

which is solved for nonzero eigenvalues using Lemma 2.3.6. We choose $d \le r$ kernel principal components, where the j^{th} component of a data point $x \in \mathcal{X}$ can be extracted by the projection

$$p_j(x) = \bar{\Phi}_r(x)^T v^{(j)} = \frac{1}{\sqrt{\lambda_j}}\sum_{i=1}^m \alpha_i^{(j)}\bar{\Phi}_r(x_i)\bar{\Phi}_r(x)^T. \tag{2.61}$$

Given data points $\mathbf{y} = \{y_1,\dots,y_\ell\} \subseteq \mathcal{X}$ their j^{th} kernel principal components are collectively calculated by

$$(p_j(y_1),\dots,p_j(y_\ell)) = \left(\tfrac{1}{\sqrt{\lambda_j}}\alpha^{(j)^T}\bar{\Phi}_r[\mathbf{x}]\right)\bar{\Phi}_r[\mathbf{y}]^T = L^{(j)}\bar{\Phi}_r[\mathbf{y}]^T, \quad j = 1,\dots,d, \tag{2.62}$$

where $\bar{\Phi}_r[\,\cdot\,]$ stands for $\Phi_r[\,\cdot\,]$ centered w.r.t. the training data. Moreover we defined the vectors $L^{(j)} = \frac{1}{\sqrt{\lambda_j}}\alpha^{(j)^T}\bar{\Phi}_r[\mathbf{x}_r] \in \mathbb{R}^r$ for $j = 1,\dots,d$. According to Corollary 2.1.2 the projections (2.62) are exact under the assumption of Lemma 2.1.1 and $K_{r,r}$ full rank r.

Note that ${L^{(j)}}^T = \bar{\Phi}_r^T \alpha^{(j)} / \sqrt{\lambda_j}$ can be directly extracted from the algorithm of the low-rank eigenvalue problem (2.60) owing to w in Lemma 2.3.6 and the fact that

$$(\bar{\Phi}_r^T \bar{\Phi}_r)(\bar{\Phi}_r^T \alpha^{(j)}) = \lambda_j (\bar{\Phi}_r^T \alpha^{(j)}). \tag{2.63}$$

Hence, the low-rank kPCA needs to store a matrix of unit eigenvectors $L = [{L^{(1)}}^T | \cdots | {L^{(d)}}^T] \in \mathbb{R}^{r \times d}$. Only $K[\mathbf{y}, \mathbf{x}_r] \in \mathbb{R}^{\ell \times r}$ is newly computed for projections onto the kernel principal axes in the course of the computation of $\bar{\Phi}_r[\mathbf{y}]$ [16].

2.4 Pre-image problem of kPCA

We cannot assume that linear PCA will detect all patterns in a given data set, but if we utilize suitable nonlinear features, we can extract more information. The kPCA as a nonlinear feature extractor is a powerful tool suited to find interesting nonlinear structures. Unfortunately, identifying the pre-image is generally an ill-posed problem. This is an outcome of the higher dimensionality of the feature space compared to the input space. As a consequence, most elements in the feature space might not have a unique pre-image. If the kernel is an invertible function, for example the polynomial kernel with odd degree and the sigmoid kernel, the pre-image can be computed exactly [24]. For many kernels though, the results provided by kernel PCA live in some very high dimensional feature space and, thus, need not have pre-images in the input space. We still can try to find approximate pre-images using the squared distance in the feature space [32].

Lemma 2.4.1. *Let $P_d\phi(x)$ denote the d-dimensional (low-rank) kernel principal projection derived from a dataset $\mathbf{x} = \{x_1, \dots, x_m\} \subseteq \mathcal{X}$ of the data instance $x \notin \mathbf{x}$. The approximate pre-image z is given by*

$$z = \operatorname*{arg\,min}_{\hat{x} \in \mathcal{X}} \|\phi(\hat{x}) - P_d\phi(x)\|^2. \tag{2.64}$$

By expressing inner products by kernels, the minimization problem can be written as

$$\begin{aligned} z &= \operatorname*{arg\,min}_{\hat{x} \in \mathcal{X}} \|\phi(\hat{x}) - P_d\phi(x)\|^2 \\ &= \operatorname*{arg\,min}_{\hat{x} \in \mathcal{X}} \left(\phi(\hat{x})^T \phi(\hat{x}) - 2\phi(\hat{x})^T P_d\phi(x)\right) \\ &= \operatorname*{arg\,min}_{\hat{x} \in \mathcal{X}} \left(\phi(\hat{x})^T \phi(\hat{x}) - 2P_d\phi(\hat{x})^T P_d\phi(x)\right) \\ &= \operatorname*{arg\,min}_{\hat{x} \in \mathcal{X}} \left(k(\hat{x}, \hat{x}) - 2K[\hat{x}, \mathbf{x}]^T L L^T K[\mathbf{x}, x]\right). \end{aligned} \tag{2.65}$$

There are different methods to solve this problem. For the RBF kernel one could use an iterative approach, similar to (4.10) [32] or gradient descent techniques [24]. Using supervised learning [4], we can establish a model to train a pre-image map $\Gamma : \mathcal{F}_k \to \mathcal{X}$ that approximates

$$x \approx z = \Gamma P_d \phi(x), \tag{2.66}$$

e.g. by KRR using the training set $\mathbf{x}$ and its kPCA projections $p[\mathbf{x}] \in \mathbb{R}^{m \times d}$. We define the KRR problem for determining the linear map W^T, which, together with a kernel ℓ, represents approximately $\Gamma : \mathcal{F}_k \to \mathcal{X}$ in the form

$$\min_W \frac{1}{2} \sum_{i=1}^m \|x_i - W^T \phi_\ell(p(x_i))\|^2 + \frac{1}{2}\alpha \|W\|^2, \tag{2.67}$$

with the regularization parameter $\alpha > 0$. The dual solution is given by equation (2.17)

$$W^T = X^T (K_\ell[p[\mathbf{x}]] + I\alpha)^{-1} \Phi_\ell[p(\mathbf{x})], \tag{2.68}$$

where $X = [x_1, \ldots, x_m]^T \in \mathbb{R}^{m \times dim(\mathcal{X})}$ and $K_{ell}[p[\mathbf{x}]] = \Phi_\ell[p[\mathbf{x}]]\Phi_\ell[p[\mathbf{x}]]^T$. The principal component encoding p is given by Corollary 2.3.4. The pre-image of x is then given as

$$z = W^T \phi_\ell(p(x)) = X^T(K_\ell[p[\mathbf{x}]] + I\alpha)^{-1} K_\ell[p(\mathbf{x}), p(x)]. \tag{2.69}$$

This is also the implementation which is used by *scikit-learn* and works well for our purposes. In theory one can choose a different kernel ℓ than the one used by kPCA. But we achieved good results by setting them both equal.

Low-rank KRR and pre-image computation

As described in section 2.2.4 we can further improve on our algorithm by using a low-rank version of KRR. In the course of the later prediction of micromagnetic time-evolution we will also use this low-rank version of KRR to estimate the time-stepping maps in feature space (see Chapter 4) [16, 40].

In the first step, we compute a low-rank factorization $K_\ell[p[\mathbf{x}]] \approx \hat{\Phi}_r[p[\mathbf{x}]]\hat{\Phi}_r^T[p[\mathbf{x}]]$ with rank r, with $\hat{\Phi}_r[p[\mathbf{x}]] \in \mathbb{R}^{m \times r}$, $m \geq r$, by Lemma 2.1.1. Hence, the pre-image computation becomes

$$z = B\hat{\Phi}_r^T[p(x)], \; B = X^T\hat{\Phi}_r[p[\mathbf{x}]](\hat{\Phi}_r^T[p[\mathbf{x}]]\hat{\Phi}_r[p[\mathbf{x}]] + I\alpha)^{-1} \tag{2.70}$$

with the low-rank mapping $\hat{\Phi}_r[p(x)] \in \mathbb{R}^{1 \times r}$ of the new data sample x calculated by Corollary 2.1.2.

2.4.1 Numerical validation of the low-rank kPCA

We summarize the low-rank kPCA and pre-image procedure in algorithm 1. This generates the unit norm eigenvectors $L^{(j)}$, $j = 1, \ldots, d$ for the prediction of new data according to (2.62) as well as the operator for the pre-image map B in (2.70).

Algorithm 1 Low-rank kPCA and pre-image

```
Data: Training data $\mathbf{x} = (\mathbf{x}_r, \mathbf{x}_{m-r}) \subseteq \mathcal{X}$, kernel $k(.,.)$ and $\ell(.,.)$, $d \leq r$.
Result: Eigenvector matrix $L = [L^{(1)}|\cdots|L^{(d)}] \in \mathbb{R}^{r \times d}$, truncated kernel PC's $p[\mathbf{x}]$, Operator for pre-image map $B$ in (2.70).
Low-rank kPCA:
    • Calculate $\Phi_r$ in (2.3).
    • Solve low-rank eigenvalue problem (2.60) for $d \leq r$ eigenvectors of unit length.
Low-rank pre-image map:
    • Calculate $\hat{\Phi}_r[p[\mathbf{x}]]$ using (2.3) and kernel $\ell$.
    • Calculate $B$ in (2.70).
```

We validate the low-rank version of kPCA and the pre-image solution via a test example from the scikit learn documentation [36], which uses both $m = 1000$ training data and test data drawn from concentric circles with noise, see Figure 2.4. A RBF kernel with $\gamma = 4$ is used and 3 kernel principal components are extracted. Two noise levels $\varepsilon = 0.02$ and 0.07 are used, where in both cases fast (exponential) convergence for increasing rank r can be observed, see Figure 2.5 which shows the mean squared error of the pre-images of the predictions compared with the original data with varying rank r used in the low-rank kPCA.

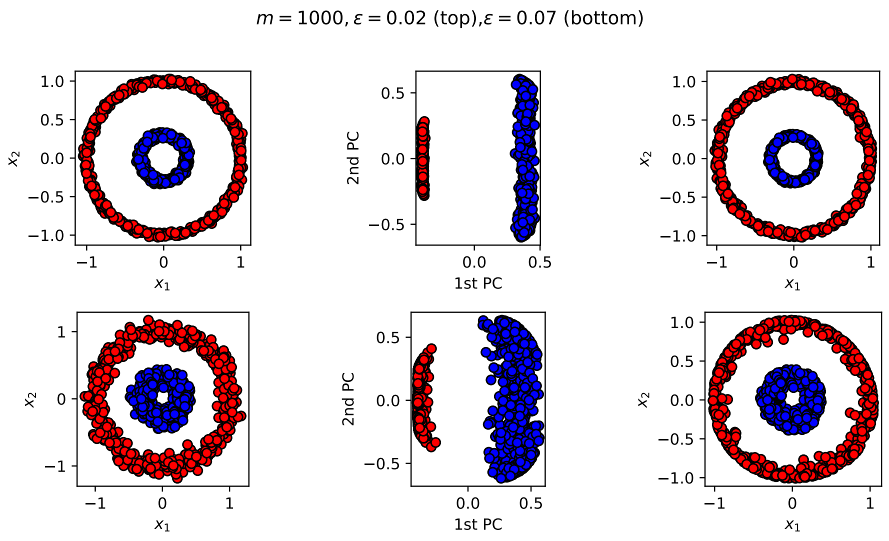


Figure 2.4: The original data set (left column), the kPCA transformed samples (middle column) and the pre-images (right column). Noise level $\varepsilon = 0.02$ (top) and $\varepsilon = 0.07$ (bottom). Number of training samples $m = 1000$. Rank $r = 120$ is used.

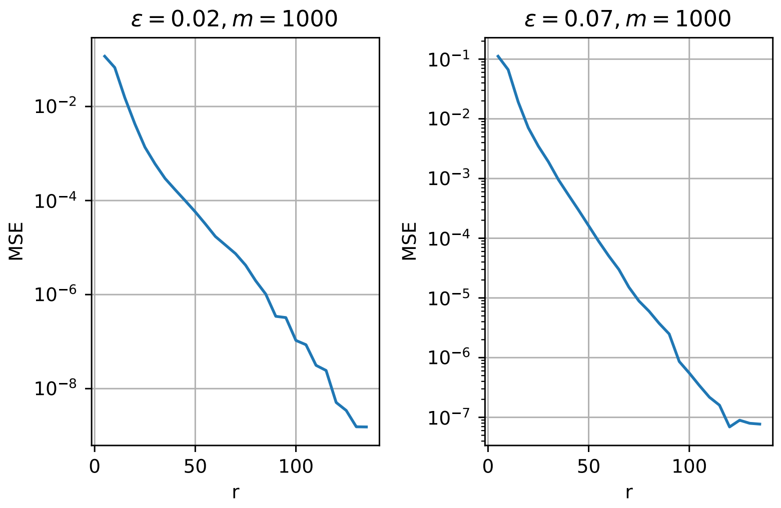
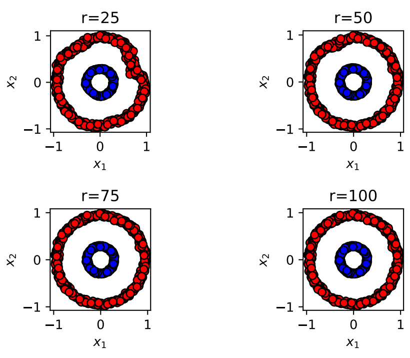


Figure 2.5: Mean squared reconstruction error for increasing rank r in the case of noise level $\varepsilon = 0.02$ and $\varepsilon = 0.07$.

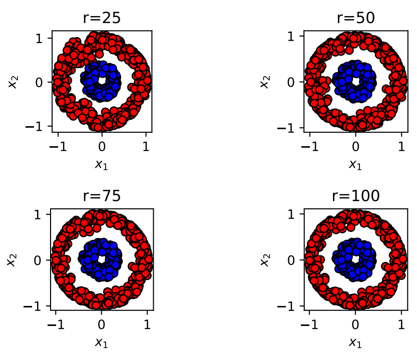
m	r	training time	prediction time	mse
8000	100	0.075 (2.813)	0.053 (1.696)	0.132
4000	100	0.043 (0.791)	0.028 (0.440)	0.132
2000	100	0.037 (0.246)	0.014 (0.153)	0.092
1000	100	0.010 (0.069)	0.004 (0.047)	0.081
500	100	0.018 (0.030)	0.003 (0.013)	0.041
1000	800	0.307	0.056	1.14e-18
1000	400	0.119	0.027	1.10e-18
1000	200	0.039	0.021	5.76e-16
1000	100	0.010	0.004	0.081
1000	50	0.013	0.002	0.055

Table 2.1: Cpu times in seconds for training and prediction of low-rank kPCA for varying samples m and rank r. Numbers in brackets refer to computing times of the respective dense kPCA version. The last column shows the mean squared error of the computed three kernel principal components compared to the results obtained from the conventional kPCA [16].

Figures 2.6a and 2.6b show the pre-images for increasing rank in the two noise cases, respectively.



(a) $\epsilon = 0.02$



(b) $\epsilon = 0.07$

Figure 2.6: Pre-images for increasing rank r.

Chapter 3

Artificial Neural Networks and Autoencoders

Parts of this chapter were previously submitted for publication [16, 40] (author of this thesis is co-author) and here restated with the permission of the first and corresponding author.

The brain connects a vast number of neurons in a complex network, which allows it to carry out highly complex computations. An Artificial Neural Network (ANN) aims to mimic such networks, in order to create a powerful set of nonlinear hypotheses. The paradigm was introduced in the mid-20th century and in recent years, with an increase in computational power and improved software, the field grew rapidly. Today, artificial neural networks outperform many other supervised learning methods in various fields, such as natural language processing or image recognition. Also in recent years, ANN arouse interest as a hypothesis class to solve PDEs in a supervised or even unsupervised manner. In the course of this chapter we will look at the basic setup of a neural network and how to train it for solving a particular problem setting. Further we will investigate autoencoders for unsupervised feature extraction, which will be an important model for learning magnetization dynamics [45].

3.1 Feedforward Neural Networks

A general neural network can be described by a graph $G = (V, E)$ and a weight function over the edges $w : E \to \mathbb{R}$. Each node then corresponds to a (nonlinear) scalar activation function and each edge links the output of this activation to the input of another node. The inputs to a neuron are combined by calculating the weighted sum with weights w.

If G is an directed acyclic graph, information only flows into one direction. If further the nodes are grouped into layers $V = \dot{\bigcup}_{t=0}^{T} V_t$, such that every edge in E connects some node in V_{t-1} to some node in V_t, for some $t = 0, \dots, T$, we call this a Feedforward Neural Network (FFNN) or multilayer perceptron (MLP). We refer to T as the number of layers, or depth of the network. Figure 3.1 shows such a network with a single scalar output. The last neuron in each layer $t = 0, \dots, T-1$ is the "constant" neuron, which always outputs 1. This is similar to the bias term from linear regression. The network has an input and an output layer and the mid layers are called *hidden layers*. The information flows from one layer to the next and each layer extracts and processes features in order to produce the final output $\hat{y}$ [45].

Linear algebra description of a multilayer perceptron

We want to establish a nonlinear model in order to learn a mapping from the input space $\mathcal{X}$ to the output space $\mathcal{Y}$. A FFNN $f_{\hat{W}} : \mathcal{X} \to \mathcal{Y}, f_{\hat{W}}(x) \mapsto \hat{y}$ with T layers can be expressed in a recursive way

$$\begin{aligned} a^{(0)} &= \sigma^{(0)}(x) \\ a^{(t)} &= \sigma^{(t)}(W^{(t)}a^{(t-1)} + b^{(t)}) \quad \text{for } t = 1, \dots, T \end{aligned} \tag{3.1}$$

The weight matrices $W^t \in \mathbb{R}^{N_t \times N_{t-1}}$ and the bias terms $b^{(t)} \in \mathbb{R}^{N_t}$, $t = 1, \dots, T$ hold the adjustable parameters. Each layer has an activation function $\sigma^{(t)} : \mathbb{R}^{N_t} \to \mathbb{R}^{N_t}$, $t = 0, \dots, T$, where N_t is the layer size excluding the constant neuron. Note that an activation might as well be the identity function I (linear activation) and an intercept term $b^{(t)}$ might as well be set to 0. The final output is then given by $\hat{y} = \sigma^{(T)}(a^{(T)})$. For a network of four layers $T = 3$ (i.e two hidden layer) this results in

$$\begin{aligned} a^{(0)} &= \sigma^{(0)}(x) \\ a^{(1)} &= \sigma^{(1)}(W^{(1)}a^{(0)} + b^{(1)}) \\ a^{(2)} &= \sigma^{(2)}(W^{(2)}a^{(1)} + b^{(2)}) \\ \hat{y} = a^{(3)} &= \sigma^{(3)}(W^{(3)}a^{(2)} + b^{(3)}). \end{aligned} \tag{3.2}$$

Or by nesting it, we can write it as

$$\hat{y} = a^{(3)} = \sigma^{(3)}(W^{(3)}\sigma^{(2)}(W^{(2)}\sigma^{(1)}(W^{(1)}\sigma^{(0)}(x) + b^{(1)}) + b^{(2)}) + b^{(3)}). \tag{3.3}$$

This is what we call the *forward propagation algorithm*.

Let $\mathcal{W}$ be the set of all possible combinations of weight matrices and intercept terms, and $\hat{W} = \{(W^{(t)}, b^{(t)}) : t = 1, \dots, T\} \in \mathcal{W}$ be an instance of this set. Further, let $f_{\hat{W}} : \mathcal{X} \to \mathcal{Y}$ be a neural network with weights $\hat{W}$, we can then define the hypothesis class for a FFNN $\mathcal{H}_{FFNN} = \{f_{\hat{W}} : \hat{W} \in \mathcal{W}\}$ from which we can pick one which minimizes a chosen objective. Since the hypothesis class is highly nonlinear, the objective function may contain many local minima. We can still perform e.g. Stochastic Gradient Decent (SGD) in hope of finding a good hypothesis, which is often the case. In the next section we will discuss the algorithm which enables us to use gradient decent techniques.

3.2 Back-propagation

For a neural network $f_{\hat{W}}$, the empirical risk is given by the average of the loss function L

$$\mathcal{L}_{NN} = \frac{1}{m}\sum_{k=1}^{m} L\left(f_{\hat{W}}(x_k, y_k)\right). \tag{3.4}$$

We want to find out how sensitive the risk is w.r.t a specific weight, i.e what is the derivative of $\mathcal{L}_{NN}$ with respect to the weight $w_{ij}^{(t)}$, which connects the j^{th} neuron in layer $t-1$ to the i^{th} neuron in layer t, i.e.,

$$\frac{\partial \mathcal{L}_{NN}}{\partial w_{ij}^{(t)}} = \frac{1}{m}\sum_{k=1}^{m} \frac{\partial L(f_{\hat{W}}(x_k), y_k)}{\partial w_{ij}^{(t)}}. \tag{3.5}$$

As before, we denote the output of the affine transformation with $z^{(t)} = W^{(t)}a^{(t-1)} + b^{(t)}$ and the activation with $a^{(t)} = \sigma^{(t)}(z^{(t)})$. For a training sample (x, y), we then take the chain rule to derive

$$\frac{\partial L(a^{(T)}, y)}{\partial w_{ij}^{(t)}} = \frac{\partial L(a^{(T)}(x), y)}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial z^{(t)}} \frac{\partial z^{(t)}}{\partial w_{ij}^{(t)}}. \tag{3.6}$$

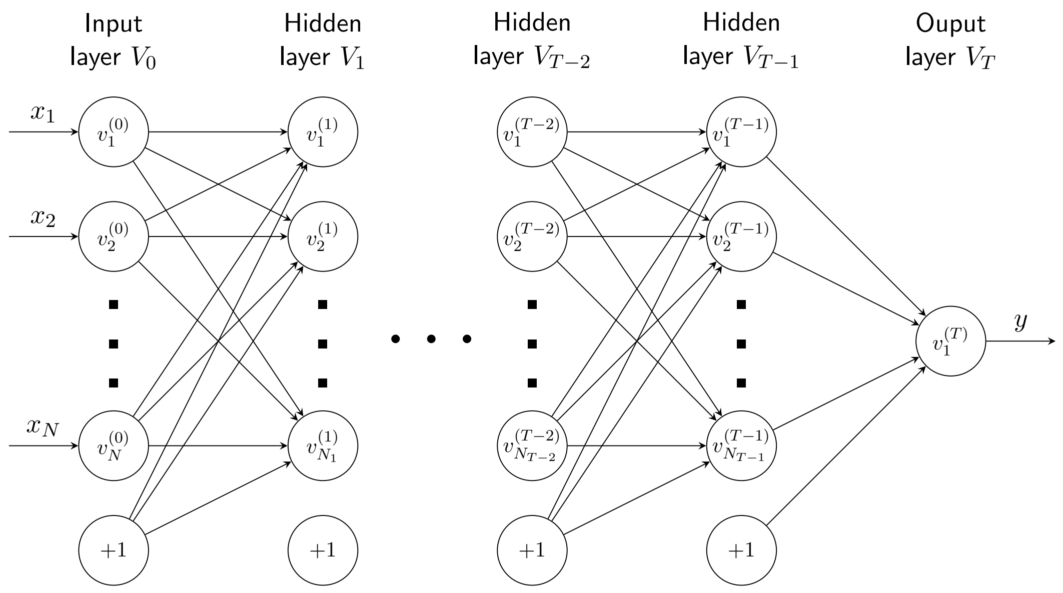


Figure 3.1: A multilayer perceptron which consists of $T+1$ layers. The network has an input size of N and only one output. Each edge represents a weight. The information flows through the network from left to right and is processed along the way. Each layer extracts features which are used to generate the predicted output. Each layer t has $|V_t|$ nodes.

Note here that for layer t with N_t neurons, the first term is the $1 \times N_t$ Jacobian matrix of the loss, the second term is a $N_t \times N_t$ diagonal matrix, with its i^{th} diagonal entry being the derivative of the activation function $(\sigma_i^{(t)}(z_i^{(t)}))'$ and the last term is just the zero vector of size N_t with its i^{th} entry set to $z_j^{(t)}$. When we take the derivative w.r.t a weight $b_i^{(t)}$ in the intercept term $b^{(t)}$, not much changes. The only difference is that the last term contains 1 as its i^{th} entry.

We can then calculate the derivatives in a backward fashion. First we start by calculating

$$\delta^{(T)} = \frac{\partial L(a^{(T)}, y)}{\partial a^{(T)}} \frac{\partial a^{(T)}}{\partial z^{(T)}}. \tag{3.7}$$

For a quadratic loss term $L_2(x, y) = \frac{1}{2}\|x-y\|^2$, the derivative is given by $\frac{\partial L_2(a^{(T)},y)}{\partial a^{(T)}} = (a^{(T)} - y)^T$. We continue going backwards through the network, by calculating

$$\delta^{(t)} = \delta^{(t+1)} W^{(t+1)} \frac{\partial a^{(t)}}{\partial z^{(t)}}. \tag{3.8}$$

The partial derivatives of layer w.r.t the weights in $w_{ij}^{(t)}$, $i = 1, \dots, N_t$ and $j = 1, \dots, N_{t-1}$ and $b_i^{(t)}$, $i = 1, \dots, N_t$ of the current layer t are easy to obtain using equation (3.6)

$$\frac{\partial L(a^{(T)}, y)}{\partial w_{ij}^{(t)}} = \delta^{(t)} \frac{\partial z^{(t)}}{\partial w_{ij}^{(t)}}, \qquad \frac{\partial L(a^{(T)}, y)}{\partial b_i^{(t)}} = \delta^{(t)} \frac{\partial z^{(t)}}{\partial b_i^{(t)}}. \tag{3.9}$$

To train a neural network, for each sample we make a forward pass through the network using equation (3.1), and then calculate the gradient vector using back-propagation (equation (3.8) and (3.9)). The weighted sum of all gradient vectors gives the full gradient for the risk function (3.5). We can then use SGD or other variants of gradient decent such as Adam [27] and pass the gradient vector to the optimizer.

3.3 Autoencoders

An autoencoder is an unsupervised neural network which tries to copy its input to its output. It is often used for dimensionality reduction or feature discovery or extraction. Given a dataset $\mathbf{x} = \{x_1, \ldots, x_m\} \subseteq \mathcal{X}$, an autoencoder can learn nonlinear relations in the dataset and compress the data to only the most descriptive features which are needed for reconstruction. Formally, an instance $x \in \mathcal{X}$ is mapped to a latent space $\mathcal{F}_\mathcal{X}$ with an encoder function $E : \mathcal{X} \to \mathcal{F}_\mathcal{X},\ h = E(x)$. A decoder part $D : \mathcal{F}_\mathcal{X} \to \mathcal{X}, \hat{x} = D(h)$ is trained to find a good reconstruction of the original data instance x measured with the loss function L. This could be the *mean squared error* for real valued input or the *cross-entropy* loss for binary values. The objective of the neural network is to minimize the average *reconstruction error*

$$\mathcal{L}_{AE}(E, D) = \frac{1}{m} \sum_{i=1}^{m} L(D(E(x_i)), x_i). \tag{3.10}$$

Both the encoder and the decoder are typically modeled with some kind of neural network. This could be a simple Feedforward Neural Network but often a Convolutional Neural Network [35] is chosen. The design of a simple autoencoder is shown in Figure 3.2. More

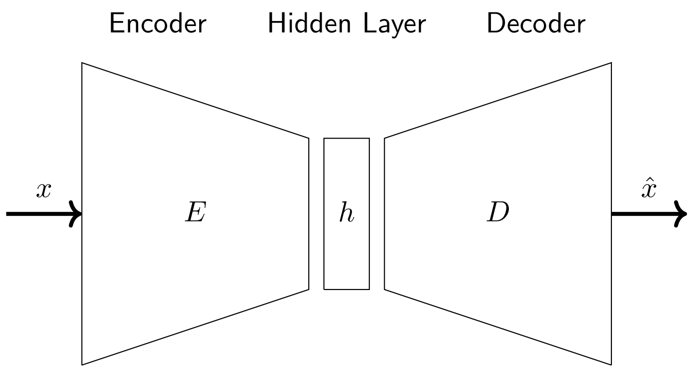


Figure 3.2: Architecture of a simple feedforward auto-encoder with a hidden layer h, an encoder E and a decoder D.

modern versions of autoencoders have generalized the idea beyond deterministic functions to stochastic mappings. However, in this chapter we will not focus on these concepts. A comprehensive overview of different autoencoder models can be found in [21].

If it happens that the autoencoder just learns the identity function $D \circ E = I$, it is not very useful in the first place. In particular, if $\mathcal{X} = \mathbb{R}^N$ and $\mathcal{F}_\mathcal{X} = \mathbb{R}^d$, imagine the hidden layer has the same dimension of the input $N = d$. The encoder can just copy its input to the hidden layer and the decoder can copy the information from the hidden layer to the output. One way to prevent the autoencoder from learning the identity is to design it in a way that $d < N$. This is what we call an undercomplete autoencoder. Respectively, if $d > N$, we say the autoencoder is overcomplete. Having an undercomplete autoencoder might still not be enough to ensure that the network just learns $D(E(x)) = x$ for each training sample x, e.g. if some features are duplicates. In theory, even if the hidden layer has just one neuron, having an encoder and decoder with enough memory to remember each training sample, the autoencoder will still do a perfect job on the training data. However, the model will have a large generalization error and hence would overfit. This scenario does not occur in practice, but illustrates how the autoencoder can fail to learn anything interesting from the data. In many cases a simple undercomplete autoencoder might still do a good job. By having a *bottleneck* ($d < N$), the amount of information which can flow through the system is limited. Therefore, minimizing the objective will cause the model to capture the most relevant aspects of the data which can be used for reconstruction [21]. A good auto-encoder model should balance the following:

- sensitivity with respect to the input for accurate reconstruction,
- insensitivity to the input to discourage memorization.

This trade-off should force the model to learn only those variations in the data which are required [25].

3.3.1 Linear Autoencoders

In this section we will see how linear autoencoders are related to linear Principal Component Analysis. If we were to choose a linear function for both, the encoder $h = E(x) = W_E x$ and the decoder $\hat{x} = D(h) = W_D x$, where $W_E \in \mathbb{R}^{d \times N}$ and $W_D \in \mathbb{R}^{N \times d}$, we would see a similar latent space as we get with PCA.

Assuming we want to minimize the mean squared error $L(x, x') = \|x - x'\|_2^2$, then the objective is given by

$$\begin{aligned} \mathcal{L}_{AE}(E, D) &= \frac{1}{m} \sum_{i=1}^{m} \|x_i - W_D W_E x_i\|_2^2 \\ &= \frac{1}{m} \|X^T - W_D W_E X^T\|_F^2, \end{aligned} \tag{3.11}$$

where $X \in \mathbb{R}^{m \times N}$. Let $X^T = USV^T$ be a singular value decomposition, with the unitary matrices $U \in \mathbb{R}^{N \times N}$ and $V \in \mathbb{R}^{m \times m}$, and the rectangular diagonal matrix $S \in \mathbb{R}^{N \times m}$ with non-negative entries. We know from the Eckart-Young Theorem (see Appendix A.2) that the best rank-d approximation of X^T is given by $U\bar{S}_d V^T$, where $\bar{S}_d \in \mathbb{R}^{N \times m}$ contains only the d largest singular values of S, while the others are set to zero. Let $S_d \in \mathbb{R}^{d \times d}$ be the diagonal matrix with only the largest singular values. Further, $U_d \in \mathbb{R}^{N \times d}$ and $V_d \in \mathbb{R}^{d \times m}$ are the matrices which only contain the left and right-singular vectors associated with the singular values in S_d. There holds that $U\bar{S}_d V^T = U_d S_d V_d^T$. If the data is centered, it can be shown that the optimal linear autoencoder is equivalent to linear PCA.

Lemma 3.3.1. *Given the data tensor $X \in \mathbb{R}^{m \times N}$, the singular value decomposition $X^T = USV^T$ and the orthonormal basis matrix U_d with only those columns in U corresponding to the d largest singular values. For given rank d, with $0 < d \leq N$, setting $W_E = U_d^T$ and $W_D = U_d$ minimizes equation (3.11).*

Proof. We know from the Eckart-Young Theorem A.2 that the best rank-d matrix approximation to X^T is given by $U\bar{S}_d V^T$. Given the particular form of the autoencoder and a prescribed d with $0 < d \leq N$, we find that $W_D W_E X^T = U\bar{S}_d V^T$. Now, if we choose $W_D W_E = U_d U_d^T$, we find

$$W_D W_E X^T = U_d U_d^T U \bar{S}_d V^T. \tag{3.12}$$

Assuming that the singular values and the associated singular vectors are arranged in descending order, we get $U_d^T U = [I, 0] = I_{d \times N} \in \mathbb{R}^{d \times N}$. It follows that $I_{d \times N} \bar{S}_d V^T = S_d V_d^T$ and therefore,

$$W_D W_E X^T = U_d S_d V_d^T. \tag{3.13}$$

The left hand side in (3.13) is exactly what we got from the Eckart-Young Theorem in the first place, i.e., $U\bar{S}_d V^T = U_d S_d V_d^T$. It follows that $W_D W_E = U_d U_d^T$ is an optimal solution to (3.11). $\square$

Therefore, if we subtract the mean upfront, the linear autoencoder is equivalent to PCA and would span the same subspace. Given that the data is normally distributed, we find that the encoding is optimal. Otherwise a nonlinear encoder/decoder may learn a more powerful representation of the data.

3.4 Regularized Autoencoders

Sensitivity to the input data is handled by the average reconstruction error. One way to reduce model complexity is to design the autoencoder in a undercomplete way. Sometimes, if this restriction is too limiting, e.g. in case we want to design an overcomplete autoencoder with $d > N$, we can introduce a regularization term in our objective, i.e.,

$$\mathcal{L}(E, D) = \sum_{i=1}^{m} L(E, D, x_i) + \text{regularizer}. \tag{3.14}$$

Sparse auto-encoders [21] for example penalize the activations in the hidden layers

$$\mathcal{L}(E, D) = \sum_{i=1}^{m} L(E, D, x_i) + \Omega(h_i), \tag{3.15}$$

thus non-descriptive latent features are discouraged.

3.4.1 Contractive Auto-Encoder

Often latent features are required to be *smooth*. Rifai et al. [39] suggest a regularization term which restricts the size of the Jacobian matrix in order to keep gradients small, i.e.,

$$\Omega(h) = \lambda \left\| \frac{\partial E(x)}{\partial x} \right\|_F^2. \tag{3.16}$$

Penalizing the Jacobian trains the autoencoder to resist pertubations of its input. Neighboring input points are mapped to a smaller neighborhood of output points, such that the contraction is only locally. A perturbation of an input x is therefore mapped closely to $E(x)$. This regularization will cause most of the derivatives in $\frac{\partial E(x)}{\partial x}$ to be tiny. The final goal is to learn the data manifold structure, so the directions with large Jacobian, that rapidly change h, are directions that are likely to approximate the tangent planes of the manifold.

The name contractive arises from the way that the Contractive Auto-Encoder (CAE) wraps space. Specifically, because the CAE is trained to resist perturbations of its input, it is encouraged to map a neighborhood of input points to a smaller neighborhood of output points. The contraction is only locally. This means that a perturbation of x is mapped close to $E(x)$, but two very different points x and x' are mapped to $E(x)$ and $E(x')$ which are farther apart.

One way to think about the Jacobian matrix $\frac{\partial E(x)}{\partial x}$ at a point x is as a linear approximaton to the nonlinear encoding $E(x)$. A linear operator A is said to be contractive if the norm of Ax remains less than or equal to 1 for all unit-length vectors x. The unit sphere completely encapsulates the image of the unit sphere.

Computation of the Jacobian is easy for a single hidden layer, but becomes increasingly harder for deeper autoencoders. The original approach was to train a series of single layer contractive autoencoders, each one reconstructing the previous autoencoder's hidden layer. This does not give the same result as optimizing the original objective, but because each layer is locally contractive, the composite autoencoder is contractive as well. Here, we suggest another approach, by approximating the Jacobian matrix and finding an upper bound to the norm. This way, we can overcome the complicated original design, and train a suitable autoencoder for our purposes in an economic way.

Given an undercomplete autoencoder which uses a simple FFNN for the encoder part, we can find an upper bound to the Jacobian norm

$$\lambda \left\| \frac{\partial E(x)}{\partial x} \right\|_F = \lambda \left\| \frac{\partial E(x)}{\partial z^{(1)}} W^{(1)} \right\|_F \leq \lambda \left\| \frac{\partial E(x)}{\partial z^{(1)}} \right\|_F \left\| W^{(1)} \right\|_F. \tag{3.17}$$

The term $W^{(1)}$ denotes the weights of the first layer and $z^{(1)} = W^{(1)}x + b^{(1)}$. This approximation is very useful if $dim(z_1) \ll dim(x)$.

In this thesis, we use the mean square error plus a contractive term which yields the objective

$$\mathcal{L}_{AE}(E, D) = \frac{1}{m} \sum_{i=1}^{m} \left(\|x_i - D(E(x_i))\|_2^2 + \lambda \left\| \frac{\partial E(x)}{\partial x_i} \right\|_F^2 \right), \tag{3.18}$$

or respectively

$$\mathcal{L}_{AE}(E, D) = \frac{1}{m} \sum_{i=1}^{m} \left(\|x_i - D(E(x_i))\|_2^2 + \lambda \left\| \frac{\partial E(x_i)}{\partial z^{(1)}} \right\|_F \left\| W^{(1)} \right\|_F \right), \tag{3.19}$$

An issue, which apparently can occur in practice, is that the contraction penalty can yield useless results if we do not impose some sort of scale on the decoder. This is likely to happen in the overcomplete case, since the mapping to a higher dimension makes overfitting more likely. One way to restrict the decoder is a coupling of the weight matrices, such that the weight matrices of D are the transpose of E. In this work we do not consider the overcomplete case and the reduced latent space of $E(x)$ serves as a regularization for our decoder D.

There are many interesting connections to other types of autoencoders (e.g. Denoising Auto-Encoder, Sparse Auto-Encoder), which are not further discussed here. The relationship with weight decay should be mentioned however. Since the Jacobian of h is given by the weight matrix in the case of a linear encoder, the regularization term corresponds to a L^2-weight decay. Keeping the weights small is the only way to have a contraction in the linear case. But with a nonlinear activation, contraction and robustness can also be achieved by driving the hidden units to their saturated regime [21, 39].

Chapter 4

Learning Feature Space Maps

Parts of this chapter were previously submitted for publication [16, 40] (author of this thesis is co-author) and here restated with the permission of the first and corresponding author.

4.1 Kernel Dependency Estimation

As it is possible to think about the kernel k as a similarity measure in the input space, likewise the loss function can be seen as a similarity measure in the output space and can therefore also be encoded using a kernel $\ell(y, y') = \phi_\ell(y)^T \phi_\ell(y')$, where $\phi_\ell : \mathcal{Y} \to \mathcal{F}_\ell$. This map makes it possible to consider a large class of nonlinear loss functions [52].

In order to learn a map $f : \mathcal{X} \to \mathcal{Y}$, $y = f(x)$, one would need to minimize the expected risk (equation (1.3)) given a set of training data and a hypothesis class like e.g. $\mathcal{H} = \{f(\,.\,;\alpha) : \alpha \text{ feasible parameter}\}$. The idea behind Kernel Dependency Estimation (KDE) is to minimize a risk functional, which uses the feature space $\mathcal{F}_k$ induced by the kernel k and a loss function measured in the output space $\mathcal{F}_\ell$ induced by the kernel ℓ. Therefore the feature space mapping $f_\mathcal{F} : \mathcal{F}_k \to \mathcal{F}_\ell$ and a pre-image map ϕ_ℓ^{-1} is needed. This is illustrated in Figure 4.1. The map f can be decomposed into subtasks, i.e.,

$$f = \phi_\ell^{-1} \circ f_\mathcal{F} \circ \phi_k. \tag{4.1}$$

In the original KDE algorithm of Weston et al. [52], Φ_ℓ is decomposed into d orthogonal directions using Kernel Principal Component Analysis (kPCA). A mapping from $\mathcal{F}_k$ to each direction can then be learned independently by using a standard kernel regression method, e.g SVM regression or Kernel Ridge Regression (KRR). In the final step, the pre-image problem (see section 2.4) must be solved to obtain an estimate in the original space $\mathcal{Y}$.

Here, we present an approach as discussed by Cortes et al. [10]. It leads to a neat algorithm which does not rely on Kernel Principal Component Analysis. For that purpose,

$$\begin{tikzcd}
\mathcal{X} \arrow[r, "f"] \arrow[d, "\phi_k"'] & \mathcal{Y} \arrow[d, shift right, "\phi_\ell"'] \\
\mathcal{F}_k \arrow[r, "f_\mathcal{F}"'] & \mathcal{F}_\ell \arrow[u, shift right, "\phi_\ell^{-1}"']
\end{tikzcd}$$

Figure 4.1: Illustration of the mappings of Kernel Dependency Estimation.

we can express the map f in terms of a more general pre-image problem which relies on kernels k and ℓ,

$$f(x) = \arg\min_{y \in \mathcal{Y}} \|\phi_\ell(y)) - f_\mathcal{F}(\phi_k(x))\|^2. \tag{4.2}$$

Let $f_\mathcal{F}$ be the set of all linear functions from $\mathcal{F}_k$ to $\mathcal{F}_\ell$ and $f_\mathcal{F}(\phi_k(x)) = W^T\phi_k(x)$, where W is a linear map. If the problem is modeled with a Kernel Ridge Regression, the optimization problem is given by

$$\arg\min_{W} \sum_{i=1}^{m} \|\phi_\ell(y_i) - W^T\phi_k(x_i)\|^2 + \alpha\|W\|_F^2. \tag{4.3}$$

From equation (4.3), we see that KDE is equivalent to Kernel Ridge Regression with the labels y_i in feature space $\mathcal{F}_\ell$. See equation (2.15) for comparison. Analogue to equation (2.16), the dual solution of (4.3) is given by

$$W^T = \Phi_\ell[\mathbf{y}]^T(K_k[\mathbf{x}] + \alpha I)^{-1}\Phi_k[\mathbf{x}], \tag{4.4}$$

where $\Phi_k[\mathbf{x}]$ defines the matrix of size $m \times dim(\mathcal{F}_k)$ of input data mapped to the feature space $\mathcal{F}_k$ and $\Phi_\ell[\mathbf{y}]$ the transformed output data in $\mathcal{F}_\ell$, i.e.,

$$\Phi_k[\mathbf{x}] = [\phi_k(x_1)| \dots |\phi_k(x_m)]^T, \quad \Phi_\ell[\mathbf{y}] = [\phi_\ell(y_1)| \dots |\phi_\ell(y_m)]^T, \tag{4.5}$$

and the kernel matrix $K_k[\mathbf{x}] = \Phi_k[\mathbf{x}]\Phi_k[\mathbf{x}]^T$. By using this result and plugging it into the pre-image problem (4.2) we can derive the following optimization problem

$$\begin{aligned} f(x) &= \arg\min_{y \in \mathcal{Y}} \|W^T\phi_k(x) - \phi_\ell(y)\|^2 \\ &= \arg\min_{y \in \mathcal{Y}}(\phi_\ell(y)^T\phi_\ell(y) - 2\phi_\ell(y)^T W^T\phi_k(x)) \\ &= \arg\min_{y \in \mathcal{Y}}(\phi_\ell(y)^T\phi_\ell(y) - 2\phi_\ell(y)^T\phi_\ell[\mathbf{y}]^T(K_k[\mathbf{x}] + \alpha I)^{-1}\phi_k[\mathbf{x}]\phi_k(x)) \\ &= \arg\min_{y \in \mathcal{Y}}(\ell(y, y) - 2K_\ell[y, \mathbf{y}](K_k[\mathbf{x}] + \alpha I)^{-1}K_k[\mathbf{x}, x]). \end{aligned} \tag{4.6}$$

Let $\gamma = (K_k[\mathbf{x}] + \alpha I)^{-1}K_k[\mathbf{x}, x] \in \mathbb{R}^m$, then for the kernel ℓ being a radial symmetric kernel such as for the RBF kernel with $\ell(y, y) = \text{const}$, the optimization problem becomes

$$f(x) = \arg\min_{y \in \mathcal{Y}}(-2\sum_{i=1}^{m} \ell(y, y_i)\gamma_i). \tag{4.7}$$

For the RBF kernel, we can find an implicit solution to (4.7) by taking the gradient respectively y

$$\nabla_y(-2\sum_{i=1}^{m} \ell(y, y_i)\gamma_i) = \sum_{i=1}^{m} \ell(y, y_i)(y - y_i)\gamma_i = 0. \tag{4.8}$$

Rearranging (4.8) to y gives

$$y = \frac{\sum_{i=1}^{m} \ell(y, y_i)\gamma_i y_i}{\sum_{i=1}^{m} \ell(y, y_i)\gamma_i}. \tag{4.9}$$

The solution of equation (4.9) is well defined if the denominator $\sum_{i=1}^{m} \ell(y, y_i)\gamma_i \neq 0$. Since the RBF kernel ℓ is smooth, we can conclude that there exists a neighborhood of the extremum of the objective function in which the denominator is not zero. This can be shown in the case that W is an orthogonal projection matrix, which was done by Mika et al. [32] for pre-image computation of kPCA. Therefore the solution can be incorporated into an iterative scheme [24, 32]. In case W is not an orthogonal projection, we lack a rigorous proof in general. However, we incorporated the solution into an iterative fixed point scheme (equation (4.10)) nonetheless, which we use for pre-image computation for KDE as follows:

$$y^{(t+1)} = \frac{\sum_{i=1}^{m} \ell(y^{(t)}, y_i)\gamma_i y_i}{\sum_{i=1}^{m} \ell(y^{(t)}, y_i)\gamma_i} \tag{4.10}$$

We have not encountered any problems in our numerical examples so far. In fact, the iteration scheme converged extremely fast in only a few iterations.

When training such an estimator, the inverse $(K[\mathbf{x}] + \alpha I)^{-1}$ can be calculated upfront. The iteration scheme, though, must be solved for every new sample, which makes the prediction very time consuming. As starting value for the iteration one can use the initial training samples.

4.2 Feature Space Integration for solving PDEs

The numerical treatment of PDEs is often difficult, since conventional methods often impose constraints on either space or time discretization. The resulting numerical methods often lead to very large linear systems which require special solvers and a big computational load. Even advanced methods do not scale to growing real world problem settings, since small-scale effects often affect large-scale behavior. Using traditional numerical solvers, for a time-dependent PDE $u_t = Lu(x,t)$ we would precompute a discretization of the differential operator L and the derivative u_t to solve the PDE. This fixed discretization imposes a limitation to our numerical method. In a supervised learning setting, we could establish a model to find such a good discretization for us [5], and hence, weaken this restriction to make more efficient numerical solvers.

Further, since the dimension of an inertial solution manifold for a nonlinear PDE is finite and the dynamics of a time-dependent system can be reduced to a finite dimensional ODE [20], we may want to embed our high dimensional solution of a dynamic system into a lower dimensional space such that this manifold is preserved. This would allow us to find an easier description of the solution to gain a computational advantage over traditional solvers.

In particular, given a time series data of the solution of a PDE, $\{x_0, \ldots, x_s\} \subseteq \mathcal{X}$ and a corresponding additional *tag* $a \in \mathcal{A}$, we want to train a model f which uses ν previous consecutive timesteps in order to predict the next data instance in the time evolution

$$f : \mathcal{X}^\nu \times \mathcal{A} \to \mathcal{X},\ f(x_{i-\nu}, \ldots, x_{i-1};\ a) = x_i \quad \text{for} \quad i = \nu, \ldots, s. \tag{4.11}$$

We could use KDE to first map the data $x_{i-1}, \ldots, x_{i-\nu}$ to some feature space $\mathcal{F}_k$ and learn a map to the feature space $\mathcal{F}_\ell$. Given the embedding in $\mathcal{F}_\ell$, we can then learn the pre-image map to find the data instance x_i in the original feature space $\mathcal{X}$. The involved mappings are shown in Figure 4.2. For instance, we can use equation (4.10) and use an iterative approach to predict the next timestep. This by itself is a powerful model which can gain impressive results, as shown in Chapter 6. We still face the problem that pre-image computation may be very costly. Therefore, if we were to train explicit maps ϕ_k, $f_\mathcal{F}$ and ϕ_ℓ^{-1}, we can vastly improve computational complexity by omitting pre-image evaluation. Let $\mathcal{F}_k = \bigotimes_{j=1}^{\nu} \mathcal{F}_\ell$, we can learn the map $\phi_\ell : \mathcal{X} \to \mathcal{F}_\ell, \phi_\ell(x_i) = z_i$ for $i = 0, \ldots, s$ and the corresponding pre-image map ϕ_ℓ^{-1} and set

$$\phi_k([x_{i-\nu}, \ldots, x_{i-1}])^T = [\phi_\ell(x_{i-\nu})^T, \ldots, \phi_\ell(x_{i-1})^T] \quad \text{for } i = \nu, \ldots, s. \tag{4.12}$$

Since we designed $\mathcal{F}_\ell$ to be a subspace of $\mathcal{F}_k$, we can replace elements from the output space $\mathcal{F}_\ell$ back to the input space $\mathcal{F}_k$, averting pre-image computation. This allows us to perform integration entirely in feature space. Since elements in $\mathcal{F}_\ell$ belong to a subspace of $\mathcal{F}_k$, we

$$\begin{tikzcd}
\mathcal{X}^{\nu} \arrow[r, "f(\,\cdot\,;a)"] \arrow[d, "\phi_k"'] & \mathcal{X} \arrow[d, shift right, "\phi_\ell"'] \\
\mathcal{F}_k \arrow[r, "f_{\mathcal{F}}(\,\cdot\,;a)"'] & \mathcal{F}_\ell \arrow[u, shift right, "\phi_\ell^{-1}"']
\end{tikzcd}$$

Figure 4.2: Illustration of the mappings of explicit feature space integration.

Algorithm 2 Explicit feature space integration

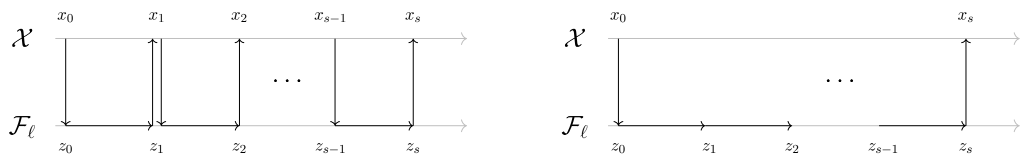
Input: Initial data $x_1, \ldots, x_\nu$, *tag* a, low-dimensional embedding ϕ_ℓ, pre-image map ϕ_ℓ^{-1} and predictor model $f_{\mathcal{F}}$.
Result: Final timestep x_s.

Computation:

1. Calculate the low-dimensional embedding $z_i = \phi_\ell(x_i)$ for $i = 0, \ldots, \nu - 1$,
2. Predict $z_i = f_{\mathcal{F}}\left(z_{i-\nu}, \ldots, z_{i-1}; a\right)$ for $i = \nu, \ldots, s$,
3. Calculate pre-image $x_s \approx \phi_\ell^{-1}(z_s)$.

can place elements from the output space $\mathcal{F}_\ell$ back into the input space $\mathcal{F}_k$, averting pre-image computation. Algorithm 2 describes the computational multistep scheme to perform explicit feature space integration:

Learning a mapping ϕ_ℓ from a high dimensional feature space $\mathcal{X}$ to an even higher dimensional space $\mathcal{F}_\ell$, might not seem to be useful to improve computational complexity in the first place. If we truncate $\mathcal{F}_\ell$ to only those dimensions which contain the most important information, we can compress the initial data and hence have a more efficient algorithm. Figure 4.3 illustrates feature space integration for $\nu = 1$ for KDE and our explicit integration approach listed in Algorithm 2. We can see that we can completely avoid the expensive pre-image computation and make the algorithm more efficient.



(a) Implicit feature space integration.

(b) Explicit feature space integration.

Figure 4.3: Illustration of explicit feature space integration compared to implicit feature space integration using KDE for $\nu = 1$.

To perform explicit feature space integration, we need to learn a nonlinear embedding ϕ_ℓ of the feature space $\mathcal{X}$ which captures the most important data relevant to predict the next timestep. There are several candidate models for this. In section 2.3.1 we learned about Kernel Principal Component Analysis for nonlinear feature space reduction, which captures those directions of the data with the highest variance. We also learned in section 2.3.2 how to make the kPCA algorithm computationally more effective, by using a low-rank approach. And in section 2.4 we discussed different approaches to find the pre-image map of kPCA. The feature space map $f_{\mathcal{F}}$ can be modeled by different methods, s.a. SVM regression or KRR. In chapter 3 we looked at Neural Networks and how we can capture the most relevant features using Autoencoders. We can model the map ϕ_ℓ using the encoder part naturally get the pre-image map ϕ_ℓ^{-1} given by the decoder part. The map $f_{\mathcal{F}}$ can be modeled by a FFNN as well. In practice, it turned out to be a difficult task to model $f_{\mathcal{F}}$ with a simple FFNN and required complicated regularization in order to prevent overfitting. This is described in more detail in chapter 5.

Chapter 5

Prediction of magnetization dynamics

Parts of this chapter were previously submitted for publication [16, 40] (author of this thesis is co-author) and here restated with the permission of the first and corresponding author.

Equation (1.1) describes the magnetization dynamics of a magnetic body $\Omega \subset \mathbb{R}^3$. As stated before, evaluation of the right hand side of the system is very expensive and even with modern numerical solvers, temporal and spatial discretization is very small. Hence, effective numerical solvers are of high interest and with the development of machine learning, data driven solvers became a new field of research in hope of efficient computation of even large scale problems which would otherwise be intractable.

Solutions to the LLG equation for a specific problem, with N discretization points, can be highly complex. The data might still embed a manifold which entirely described the magnetization dynamics. Knowledge of this structure can therefore help to understand the system. We propose a algorithmic framework which uses prior information regarding the solution in order to learn a feature space which allows an easier description of the dynamics. This yields a predictor model without any need for field evaluations after a data generation and training phase has been established as a pre-computation, making the computation of new solutions fast and computationally efficient.

5.1 Data structure for the time stepping learning method

In the pioneering work Kovacs et al. [29] the standard problem 4 was chosen as a suitable benchmark. The test problem is well established and usually considered as a kind of "must pass" criterion for any new micromagnetic method for integrating the LLG equations of motion. Essentially, the problem is split into two external field cases applied to a magnetic thin film equilibrium s-state. Both fields initialize a nonlinear and non-trivial de-magnetization dynamical behavior. Predictions via the neural network model from [29] exhibited high accuracy in the first field case. However, in the second field case predictions yielded larger deviations from the ground truth states. Several additional features had been incorporated into the learning phase, such as extra scaling of the z-component, a magnetization length preserving loss term and a rather complicated usage of eight previously obtained magnetization snap-shots.

Following [17] we generate data associated with the NIST μMAG Standard problem $\#4$ [31, 37]. The geometry is a magnetic thin film of size $500 \times 125 \times 3$ nm with material parameters of permalloy: $A = 1.3 \times 10^{-11}$ J/m, $M_s = 8.0 \times 10^5$ A/m, $\alpha = 0.02$. The initial state is an equilibrium s-state, obtained after applying and slowly reducing a saturating field along the diagonal direction $[1, 1, 1]$ to zero. Then two scenarios of different external fields

are studied: field 1 of magnitude 25mT is applied with an angle of $170°$ c.c.w. from the positive x axis, field 2 of magnitude 36mT is applied with an angle of $190°$ c.c.w. from the positive x axis. For data generation we use a spatial discretization of $100 \times 25 \times 1$ and apply finite differences [33] to obtain a system of ODEs that is then solved with a projected Runge-Kutta method of second order with constant step size of 40fs.
We denote the number of discretization cells with N. For the purpose of collecting training data samples we use numerically obtained approximations for $n \in \mathbb{N}$ different external field values. Following the splitting of training data of Exl et al. [17], the external field is either in the range of the *field* 1

$$\mathbf{H}_{ext,1} : \|\mathbf{H}_{ext,1}\| =: h \in [20, 30]\text{mT}, \, \arg \mathbf{H}_{ext,1} =: \varphi \in [160°, 180°] \tag{5.1}$$

or in the range of the *field* 2

$$\mathbf{H}_{ext,2} : \|\mathbf{H}_{ext,2}\| =: h \in [30, 40]\text{mT}, \, \arg \mathbf{H}_{ext,2} =: \varphi \in [180°, 200°]. \tag{5.2}$$

For $s = 100$ time steps we assemble the data into a 3-tensor $\mathcal{D}$, defined slice-wise by

$$\mathcal{D} \in \mathbb{R}^{(s+1)\times n \times 3N} : \mathcal{D}(i, :, :) = [\mathbf{m}_x(t_i)|\mathbf{m}_y(t_i)|\mathbf{m}_z(t_i)] \in \mathbb{R}^{n\times 3N}, \, i = 0, \ldots, s, \tag{5.3}$$

where $\mathbf{m}_q(t_i) \in \mathbb{R}^{n\times N}$, $q = x, y, z$ denotes the magnetization component grid vector at time t_i for each of the n field values. Each sample is tagged with a external field at time t_i with h and φ components. The external fields are grouped in the matrix $\mathbf{h}(t_i) \in \mathbb{R}^{n\times 2}$. Figure 5.1 shows the data tensor $\mathcal{D}$ with the corresponding field values.

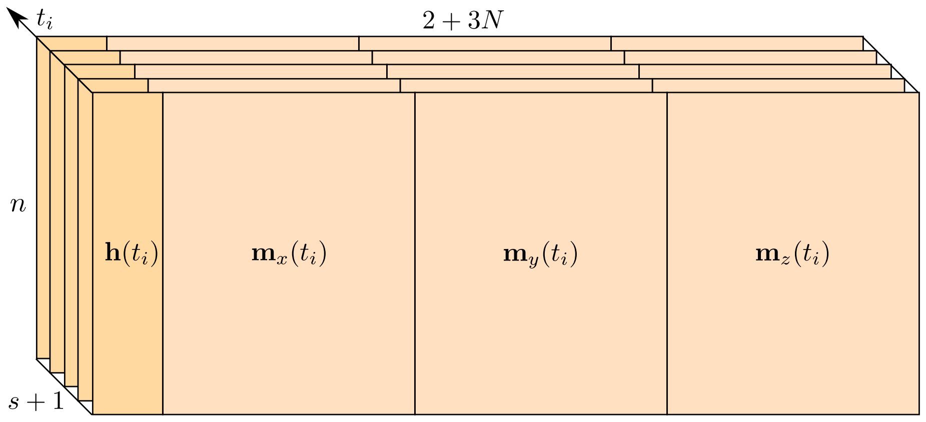


Figure 5.1: Data tensor $\mathcal{D}$ together with the external field tags $\mathbf{h}(t_i)$.

The data generation was performed using the Vienna Scientific Cluster (VSC). We used the Python machine learning package scikit learn [36] which we extended by the low-rank kPCA (ℓ-kPCA) variant with pre-image computation and low-rank kRR (ℓ-KRR) introduced in Chapter 2. We used those to establish the models, which we describe in the following section.

5.2 Implicit Feature Space integration using KDE

Suppose we want to develop a ν-step scheme, the KDE algorithm trains a map from the input space $\mathcal{X} = \mathbb{R}^{2+\nu N}$, which contains ν previous magnetization states and the field tags, to the output space $\mathcal{Y} = \mathbb{R}^N$. The input is implicitly mapped to a space $\mathcal{F}_k$ with associated kernel k and the output is mapped to $\mathcal{F}_\ell$ with kernel ℓ. Given the last ν timesteps, the algorithm produces the next timestep based on the iterative formula which is given by equation (4.10). Using the tensor notation from Chapter 5, the input and output to the model is

$$\text{input:} \quad \{\mathbf{h}(t_{i-1}), \mathcal{D}(i-\nu, :, :), \ldots, \mathcal{D}(i-1, :, :)\}, \quad \text{output:} \quad \{\mathcal{D}(i, :, :)\}, \tag{5.4}$$

with $i = \nu, \dots, s$.

Since this method does not perform an explicit change into the feature space, it increases the computational cost dramatically. Together with the iterative scheme, the method does not scale to larger problem sets very easily.

5.3 Explicit Feature Space integration

Given the data tensor $\mathcal{D}$ obtained from a pre-computation step, our model uses $\mathcal{D}$ as input data in order to first train a feature space mapping such as kPCA or an autoencoder as well as its inverse. The goal is to drastically compress the data size while keeping the most descriptive information. This is followed by a regression scheme which entirely performs in feature space. In this work we use a Kernel Ridge Regression and a Feedforward Neural Network for this task. The model is therefore a composite of two models.

5.3.1 Feature Space mapping and data compression

Prior to the training of the regression model, the data tensor $\mathcal{D}$ is compressed to a size $d < 3N$, yielding a truncated data tensor $\mathcal{D}_{\mathcal{F}} \in \mathbb{R}^{(s+1)\times n\times d}$. In theory many dimensionality reduction methods are possible. We focus mainly on kPCA and autoencoders, since the pre-image computation is incorporated and both are established models which can perform well on a variety of different tasks. Further, in the case of kPCA, a low-rank version can be used to further improve computational efficiency. Figure 5.2 shows the truncated tensor together with the magnetic field components.

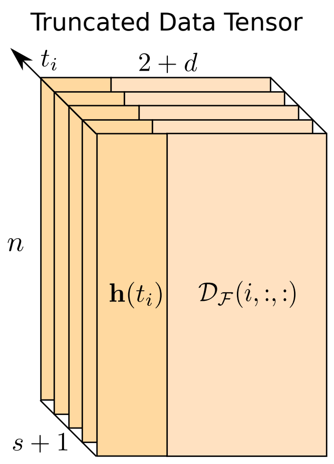


Figure 5.2: Illustration of the truncated data tensor $\mathcal{D}_{\mathcal{F}}$ and the field values $\mathbf{h}$.

In Section 2.3.1 we covered the kPCA algorithm together with its low-rank version. Note, even though we have a compression of the data, what we actually do is to map the data onto a higher dimensional feature space. We only take the d most descriptive principal components of this high dimensional space, leaving us with a net compression of our data.

Section 3.3 on the other hand describes autoencoders and their use cases. In our model, the encoder and the decoder take the form of a FFNN with a simple structure. The network has $3N$ input and output units, some hidden layers, and a bottleneck layer with only d units. The exact architecture can be seen in Chapter 6. Training of the autoencoder is straightforward, using gradient decent techniques. Further, in Section 3.4 we described further improvements to the autoencoder using additional regularization terms, such as *sparse autoencoders* and *contractive autoencoders*. We presumed that those additional regularization terms would give rise to a smoother feature space, enabling easier and more accurate feature space integration.

5.3.2 Feature Space integration model

As described in Section 4.2, we train a predictor model which uses ν consecutive previous timesteps. Time stepping maps are now learned by taking reduced dimensional input and output data tensor to fit a kRR model. In its simplest form one can use a one step scheme mapping from $t \to t+\Delta t$ by taking input data defined via the slices $\mathcal{D}_{\mathcal{F}}(i,:,:)$, $i = 0,\dots,s-1$, and output data defined by $\mathcal{D}_{\mathcal{F}}(i,:,:)$, $i = 1,\dots,s$, which corresponds to data shifted by one time step Δt. However, inspired by [29], we found enhanced stability by introducing time stepping with multi-steps, e.g., choosing ν steps in a scheme $\{t, t+\Delta t, \dots, t+(\nu-1)\Delta t\} \to t+\nu\Delta t$. For that purpose we choose a time stepping number $\nu(< s) \in \mathbb{N}$ and train the model with the following input and output:

$$\text{input:} \quad \{\mathbf{h}(t_{i-1}), \mathcal{D}_{\mathcal{F}}(i-\nu,:,:), \dots, \mathcal{D}_{\mathcal{F}}(i-1,:,:)\}, \quad \text{output:} \quad \{\mathcal{D}_{\mathcal{F}}(i,:,:)\}, \tag{5.5}$$

with $i = \nu, \dots, s$.

FFNN Feature Space Integration

There are several neural network models which could be used to predict time series with a neural network. We choose a simple Feedforward Neural Network for this task. The network has $\nu\, d + 2$ input units for the truncated feature space magnetization components and the magnetic field parameters, and d output units. Further, we take a rather complex architecture for this task, with 5 hidden layers. Table 6.7 shows the full architecture for both field cases. Our experiments show that a FFNN alone is not a good fit for this task, since a simple time-stepping scheme would easily overfit, and the prediction results are of poor quality. Therefore, we use a forward looking objective, as applied in [26, 29], to gain accuracy.

Let $c_{i,j}$ be the low dimensional embedding of $\mathcal{D}(i,j,:)$ for time i and sample j. We want to train the NN-predictor model with weights $\boldsymbol{w}$

$$\mathcal{N} : \mathbb{R}^{2+\nu d} \to \mathbb{R}^{d}, \mathcal{N}(c_{i-1,j}, \dots, c_{i-\nu,j}; \boldsymbol{w}, h_j(t_{i-1})) \mapsto c_{i,j}. \tag{5.6}$$

To this purpose, we define the tensor $\hat{\mathcal{D}}^{(t)}_{\mathcal{F}}(i,:,:) \in \mathbb{R}^{n\times d}$ to be a prediction of $\mathcal{D}_{\mathcal{F}}(i,:,:) = [c_{i,1}, \dots, c_{i,n}]^T \in \mathbb{R}^{n\times d}$, which used t previous consecutive predictions. The rather complicated objective is then given by

$$\min_{\boldsymbol{w}} \sum_{t=0}^{f_t} \sum_{i=\nu+t}^{s} \|\mathcal{D}_{\mathcal{F}}(i,:,:) - \hat{\mathcal{D}}^{(t)}_{\mathcal{F}}(i,:,:)\|_F^2. \tag{5.7}$$

The forward looking parameter f_t regulates how well the model predicts a time step based on previous predictions.

Example: To better understand this complex objective, we will go through a step-by-step example with $f_t = 3$ and $\nu = 3$. Let $\hat{c}^{(t)}_{i,j}$ be a prediction of $c_{i,j}$ for step i and sample j, which uses t previous predictions. First, we start of with $t = 0$. The tensor $\hat{\mathcal{D}}^{(0)}_{\mathcal{F}}$ uses no previous predictions. This means, for a sample j we calculate the predictions $\hat{c}^{(0)}_{i,j}$ for $i = 3, \dots, s$ using (5.6), i.e.,

$$\mathcal{N}(c_{0,j}, c_{1,j}, c_{2,j}; \boldsymbol{w}, h_j(t_2)) = \hat{c}^{(0)}_{3,j}, \dots, \mathcal{N}(c_{s-3,j}, c_{s-2,j}, c_{s-1,j}; \boldsymbol{w}, h_j(t_{s-1})) = \hat{c}^{(0)}_{s,j}. \tag{5.8}$$

The predictions can be grouped into the tensor $\hat{\mathcal{D}}^{(0)}_{\mathcal{F}}(i,j,:) = \hat{c}^{(0)}_{i,j}$ for $j = 1, \dots, n$.

Next, we set $t = 1$ and use the previous predictions in $\hat{\mathcal{D}}_{\mathcal{F}}^{(0)}$ to calculate $\hat{\mathcal{D}}_{\mathcal{F}}^{(1)}$. Therefore, we replace the last timestep to calculate $\hat{c}_{i,j}^{(1)}$ for $i = 4, \ldots, s$

$$\mathcal{N}(c_{1,j}, c_{2,j}, \hat{c}_{3,j}^{(0)}; \boldsymbol{w}, h_j(t_3)) = \hat{c}_{4,j}^{(1)}, \ldots, \mathcal{N}(c_{s-3,j}, c_{s-2,j}, \hat{c}_{s-1,j}^{(0)}; \boldsymbol{w}, h_j(t_{s-1})) = \hat{c}_{s,j}^{(1)}. \quad (5.9)$$

Note that we had to drop the first step $c_{0,j}$, because there is no previous prediction available for $c_{2,j}$. As before, we set $\hat{\mathcal{D}}_{\mathcal{F}}^{(1)}(i,j,:) = \hat{c}_{i,j}^{(1)}$ for $j = 1, \ldots, n$.

We continue with $t = 2$ and calculate $\hat{c}_{i,j}^{(2)}$ for $i = 5, \ldots, s$ the same way. We now put both prediction $\hat{c}_{i,j}^{(0)}$ and $\hat{c}_{i,j}^{(1)}$ into the model, in order to predict $\hat{c}_{i,j}^{(2)}$ for $i = 5, \ldots, s$

$$\mathcal{N}(c_{2,j}, \hat{c}_{3,j}^{(0)}, \hat{c}_{4,j}^{(1)}; \boldsymbol{w}, h_j(t_4)) = \hat{c}_{5,j}^{(2)}, \ldots, \mathcal{N}(c_{s-3,j}, \hat{c}_{s-2,j}^{(0)}, \hat{c}_{s-1,j}^{(1)}; \boldsymbol{w}, h_j(t_{s-1})) = \hat{c}_{s,j}^{(2)}. \quad (5.10)$$

Then we set $\hat{\mathcal{D}}_{\mathcal{F}}^{(2)}(i,j,:) = \hat{c}_{i,j}^{(2)}$.

Finally, we use all three previous predictions to predict $\hat{c}_{i,j}^{(3)}$ for $i = 6, \ldots, s$

$$\mathcal{N}(\hat{c}_{3,j}^{(0)}, \hat{c}_{4,j}^{(1)}, \hat{c}_{5,j}^{(2)}; \boldsymbol{w}, h_j(t_5)) = \hat{c}_{6,j}^{(3)}, \ldots, \mathcal{N}(\hat{c}_{s-3,j}^{(0)}, \hat{c}_{s-2,j}^{(1)}, \hat{c}_{s-1,j}^{(2)}; \boldsymbol{w}, h_j(t_{s-1})) = \hat{c}_{s,j}^{(3)} \quad (5.11)$$

and set $\hat{\mathcal{D}}_{\mathcal{F}}^{(3)}(i,j,:) = \hat{c}_{i,j}^{(3)}$.

Now that we calculated $\hat{\mathcal{D}}_{\mathcal{F}}^{(0)}$, $\hat{\mathcal{D}}_{\mathcal{F}}^{(1)}$, $\hat{\mathcal{D}}_{\mathcal{F}}^{(2)}$ and $\hat{\mathcal{D}}_{\mathcal{F}}^{(3)}$, we can plug them into the objective (5.7) and evaluate the gradient respectively $\mathbf{w}$ using Automatic Differentiation tools.

Chapter 6

Results

Parts of this chapter were previously submitted for publication [16, 40] (author of this thesis is co-author) and here restated with the permission of the first and corresponding author.

6.1 KDE

As stated in Chapter 5, KDE is a powerful model for predicting magnetization dynamics. It can convince with a high prediction accuracy, while also being able to resolve small scale dynamic effects. On the other hand, it suffers from expensive pre-image computation, making prediction slow. Nonetheless, we can use KDE as a benchmark model, to compare performance and speed of other methods like the explicit feature space integration scheme.

Figure 6.2 to 6.1 show the result of the KDE algorithm for the prediction of the time course of the magnetization state. The algorithm performs very well when it comes to prediction accuracy. The drawback of this method is the computational cost. This can be seen in Table 6.1. Also, since the magnetization components are not mapped to a lower dimensional space, using a multi time-stepping scheme leads to tremendous storage requirements. The method converges very fast. For a tolerance of 1×10^{-5} it only takes 3 iterations to arrive at a solution.

We used the RBF kernel for the input and output space. The kernel parameter γ was set to the default value, which corresponds to one over the number of features.

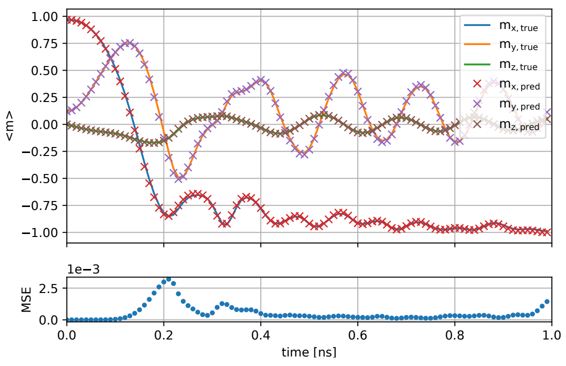
n	pred. error [-]	fit time [s]	pred. time [s]
10	0.0691	0.35	9.94
50	0.0102	5.07	55.44
100	0.0054	21.66	122.35
200	0.0055	116.19	343.11

(a) Field 1 case

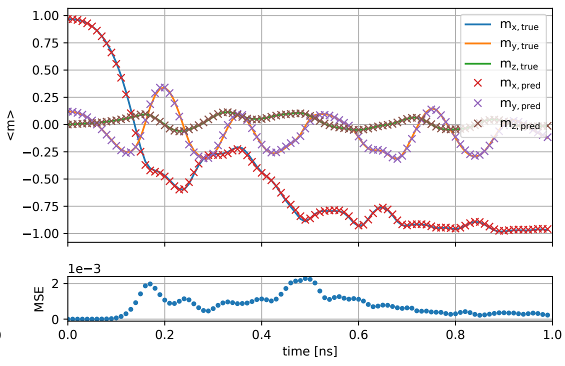
n	pred. error [-]	fit time [s]	pred. time [s]
10	0.1418	0.48	16.98
50	0.0655	6.67	83.52
100	0.0785	29.99	198.10
200	0.0535	159.75	447.85

(b) Field 2 case

Table 6.1: Results of a 5-fold cross validation of the KDE algorithm, using a different number of samples n. The prediction error relates only to the last timestep. The parameters were chosen as follows: kernel parameter $\gamma_k = \frac{1}{3N+6}$, $\gamma_\ell = \frac{1}{3N}$, time-stepping $\nu = 3$ and regularization $\alpha = 1 \times 10^{-6}$.

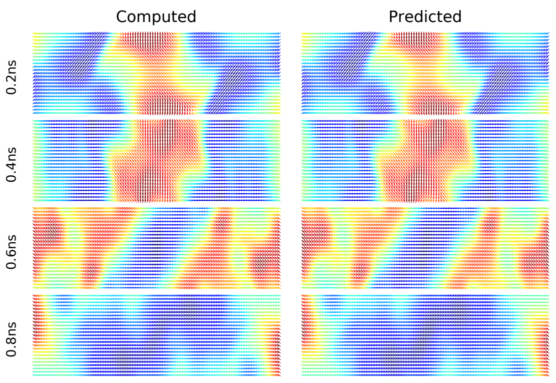


(a) Field 1 case

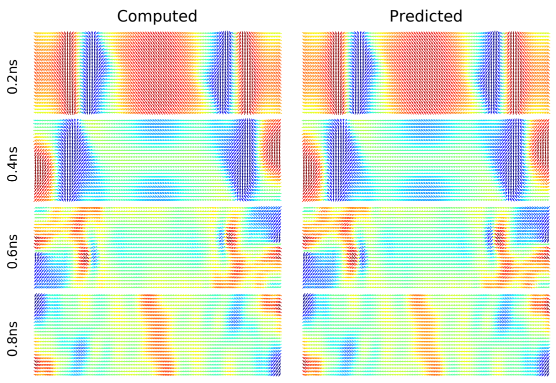


(b) Field 2 case

Figure 6.1: Predictions versus computed results for mean magnetization, using the iterative KDE algorithm (4.10). We used $m = 100$ samples for the prediction and the parameters were chosen as follows: kernel parameter $\gamma_k = \frac{1}{3N+6}$, $\gamma_\ell = \frac{1}{3N}$, time-stepping $\nu = 3$ and regularization $\alpha = 1 \times 10^{-6}$.



(a) Field 1 case



(b) Field 2 case

Figure 6.2: Snap shots of computed (Comp) and predicted magnetization states, using the iterative KDE algorithm (4.10). We used $n = 100$ samples for the prediction and the parameters were chosen as follows: kernel parameter $\gamma_k = \frac{1}{3N+6}$, $\gamma_\ell = \frac{1}{3N}$, time-stepping $\nu = 3$ and regularization $\alpha = 1 \times 10^{-6}$.

6.2 kPCA and KRR

In this section we compare the explicit feature space integration model, which we established in Section 4.2 and 5.3, with implicit feature space integration using KDE. We use kPCA as feature space mapping and KRR for feature space integration. Further, we also use KRR for the pre-image map, as stated in Section 2.4. For all models we use the RBF kernel.

Tabel 6.2 shows, how the combined model performs with different dataset sizes. Training the model scales at least quadratic, due to the evaluation of the kernel matrix. We also note that prediction time is considerably lower than with KDE, since we do not need pre-image computation for intermediate steps.

One issue is the determination of the kernel for feature space integration. We chose a RBF kernel with the kernel parameter $\gamma_{KRR} = 1$, since it worked for our purposes. Nonetheless, for real world application, we would need a proper evaluation for the kernel. We note here that a Gaussian Process model would be a better fit for this task, since it is able find a good kernel. The training process would become computationally more intense though.

We can see that prediction accuracy is comparable with the implicit feature space integration (Table 6.1), but we also see from Figure 6.4 that the algorithm does not preserve as many details (Figure 6.2). Still, we can observe that the model indeed learns the magnetization dynamics and even has a better score for the second field case.

n	pred. error [-]	fit time [s]	pred. time [s]
10	0.0089	0.43	0.06
50	0.0062	12.16	0.26
100	0.0078	68.52	0.62
200	0.0056	414.53	2.04

(a) Field 1 case with $\nu = 3$ and $d = 20$

n	pred. error [-]	fit time [s]	pred. time [s]
10	0.0401	0.43	0.08
50	0.0274	12.25	0.40
100	0.0415	67.75	1.01
200	0.0225	409.60	2.92

(b) Field 2 case with $\nu = 5$ and $d = 40$

Table 6.2: Prediction results of a 5-fold cross validation of the explicit feature space integration using kPCA as reducer and KRR as integrator. For a different number of samples n we show fit time, prediction time and MSE. The prediction error relates only to the last timestep. We used a RBF kernel for the kPCA, the integrator and the pre-image KRR map. The parameters were chosen as follows: kernel parameter $\gamma_{KPCA} = \frac{1}{3N}$ and $\gamma_{KRR} = 1$, integrator regularization parameter $\alpha_{KRR} = 0.01$, pre-image regularization parameter $\alpha_{KPCA} = 0.001$ and $n = 100$ samples.

In Table 6.3 we used a varying number of principal components for feature space integration. We note that prediction time increases for more principal components, but the model has better performance. Since we need to solve the eigenvalue problem d times, the kPCA algorithm should scale linearly with increasing d, but this also depends on the actual implementation.

In Figure 6.3 and 6.4, we show the computed mean magnetization and the predicted one for various values of principal components. We see that more principal components are able to capture more information, and increase the overall accuracy.

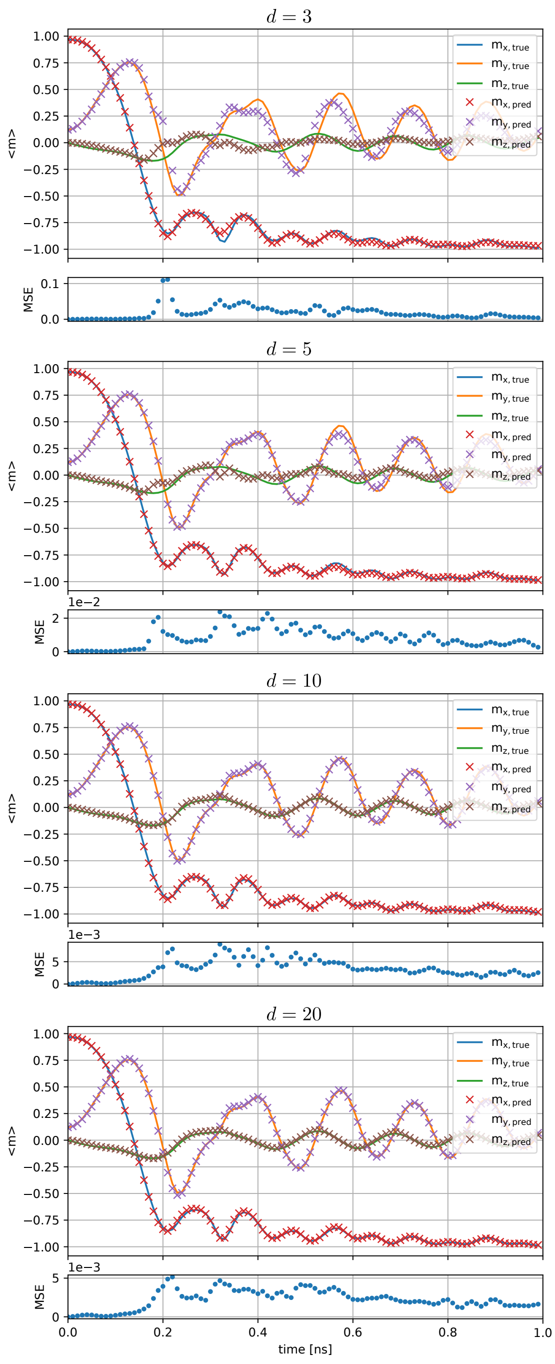
d	pred. error [-]	fit time [s]	pred. time [s]
3	0.0146	29.80	0.54
5	0.0115	29.74	0.52
10	0.0073	64.01	0.54
20	0.0067	63.87	0.62

(a) Field 1 case with $\nu = 3$

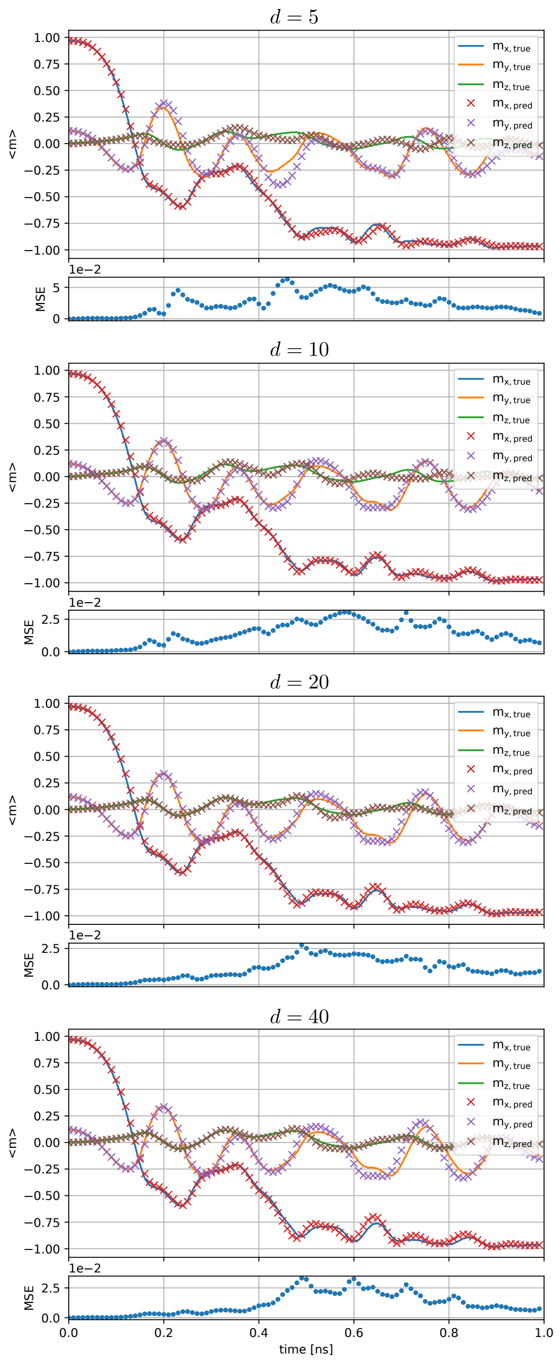
d	pred. error [-]	fit time [s]	pred. time [s]
5	0.0574	29.79	0.60
10	0.0410	63.60	0.61
20	0.0352	63.26	0.73
40	0.0375	63.95	0.96

(b) Field 2 case with $\nu = 5$

Table 6.3: Prediction results of a 5-fold cross validation of the explicit feature space integration using kPCA as reducer and KRR as integrator. For a different number of principal components d we show fit time, prediction time and MSE. The prediction error relates only to the last timestep. We used a RBF kernel for the kPCA, the integrator and the pre-image KRR map. The parameters were chosen as follows: kernel parameter $\gamma_{KPCA} = \frac{1}{3N}$ and $\gamma_{KRR} = 1$, integrator regularization parameter $\alpha_{KRR} = 0.01$, pre-image regularization parameter $\alpha_{KPCA} = 0.001$ and $n = 100$ samples.

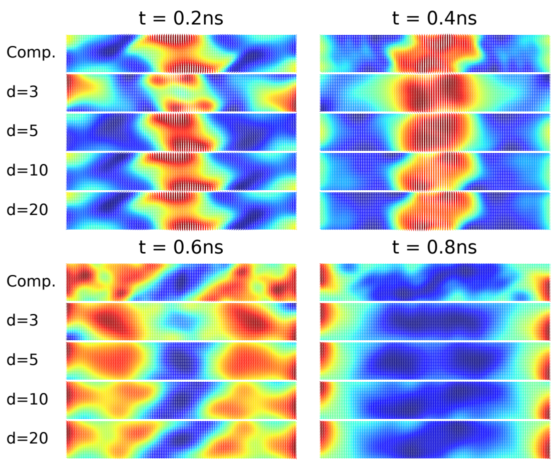


(a) Field 1 case with $\nu = 3$

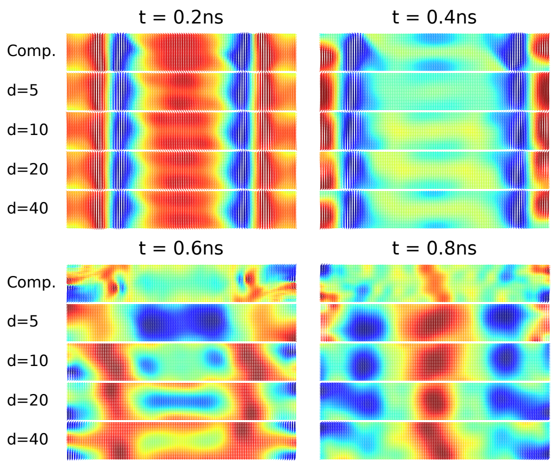


(b) Field 2 case with $\nu = 5$

Figure 6.3: Prediction using kPCA as reducer and KRR as integrator. We show the mean magnetization for different values of principal components d. We used a RBF kernel for the kPCA, the integrator and the pre-image KRR map. The parameters were chosen as follows: kernel parameter $\gamma_{KPCA} = \frac{1}{3N}$ and $\gamma_{KRR} = 1$, integrator regularization parameter $\alpha_{KRR} = 0.01$, pre-image regularization parameter $\alpha_{KPCA} = 0.001$ and $n = 100$ samples.



(a) Field 1 case with $\nu = 3$



(b) Field 2 case with $\nu = 5$

Figure 6.4: Magnetization snapshots for different values of principal components d corresponding to Figure 6.3. We used a RBF kernel for the kPCA, the integrator and the pre-image KRR map. We used kPCA as reducer and KRR as integrator. We used a RBF kernel for the kPCA, the integrator and the pre-image KRR map. The parameters were chosen as follows: kernel parameter $\gamma_{KPCA} = \frac{1}{3N}$ and $\gamma_{KRR} = 1$, integrator regularization parameter $\alpha_{KRR} = 0.01$, pre-image regularization parameter $\alpha_{KPCA} = 0.001$ and $n = 100$ samples.

6.3 ℓ-kPCA and ℓ-KRR

Selection of basis vectors for the low-rank procedure (compare with $\mathbf{x}_r$ in section 2.3.2) is accomplished by choosing r field values and collecting the corresponding discrete magnetization trajectories for all $s+1$ time points for each chosen field value. This results in a reduced sample size of $(s+1)r \leq (s+1)n = m$.

In the course of learning the time-stepping via low-rank kPCA the data tensor is used with reduced dimensionality. Note that we have $d \leq r(s+1) \leq n(s+1) = m$. We denote the reduced dimensional data tensor resulting from the low-rank kPCA approach with $\mathcal{D}_{\mathcal{F}} \in \mathbb{R}^{(s+1)\times n\times d}$. Figure 5.2 shows the compressed (resp. truncated) data tensor $\mathcal{D}_{\mathcal{F}}$ with the corresponding magnetic field values, where the large grid size $3N$ is reduced to d and the field is appended, compare with the original data tensor from Figure 5.1. From the truncated tensor, we take r horizontal slices as basis vectors. Additionally we illustrate in Figure 6.5 the tensor required in storage to project new data onto the kernel principal components, compared to the tensor which is required for the low rank version.

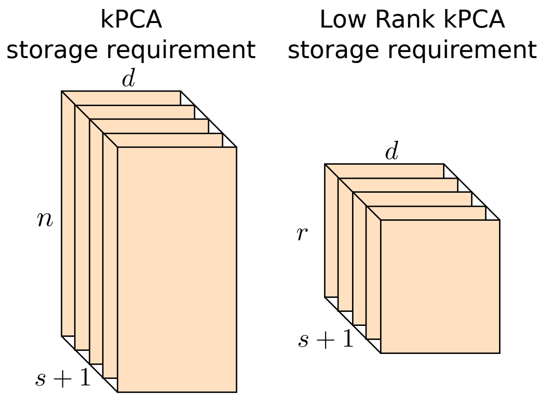


Figure 6.5: Illustration of the storage requirements in the low-rank kPCA and the low-rank kRR for the time stepping model (right), compared with full-rank kPCA (left). Storage of the compressed tensor is only $\mathcal{O}((s+1)dr)$.

6.3.1 Cross-validation of the hyper-parameters

First we focus on the important validation of model- and method-specific hyper-parameters such as the kernel defining $\gamma > 0$, the time stepping number $\nu \in \mathbb{N}$, the number of kernel principal components $d \in \mathbb{N}$ and study the dependence on the rank $r \in \mathbb{N}$.

Figure 6.6 shows the kernel approximation error decay for increasing r. We see that the field 1 case is easier to approximate and the field 2 decay is rather slow. Therefore, for some of the coming experiments, we use a value $r = 20$ for the first field and $r = 40$ for the second one, such that approximation error is at most around 10^{-9}.

We determine the hyper-parameters γ, ν and d via grid search. For that purpose we measure the mean error norms in the magnetization between the prediction and the simulation of 1ns for the standard problem in both ranges of field 1 and 2. This shows that a (default) value of $\gamma = 1/N$ is quite optimal. Furthermore, the regularization parameters in the kRR were chosen to be between 0.001 and 0.01. Figure 6.7 shows for varying d and ν the mean error norms in the magnetization between the predictions and simulations of 1ns for the standard problem in the range of field 1 and 2 (compare with (5.1) and (5.2)), respectively, obtained from a 10-fold cross-validation with random split strategy and 10% test size. Here we used a rather large rank $r = 40$.

We show in Table 6.4, how the algorithm scales with increasing dataset size. The fit time is considerably lower than the original kernel version of kPCA and KRR. But we also

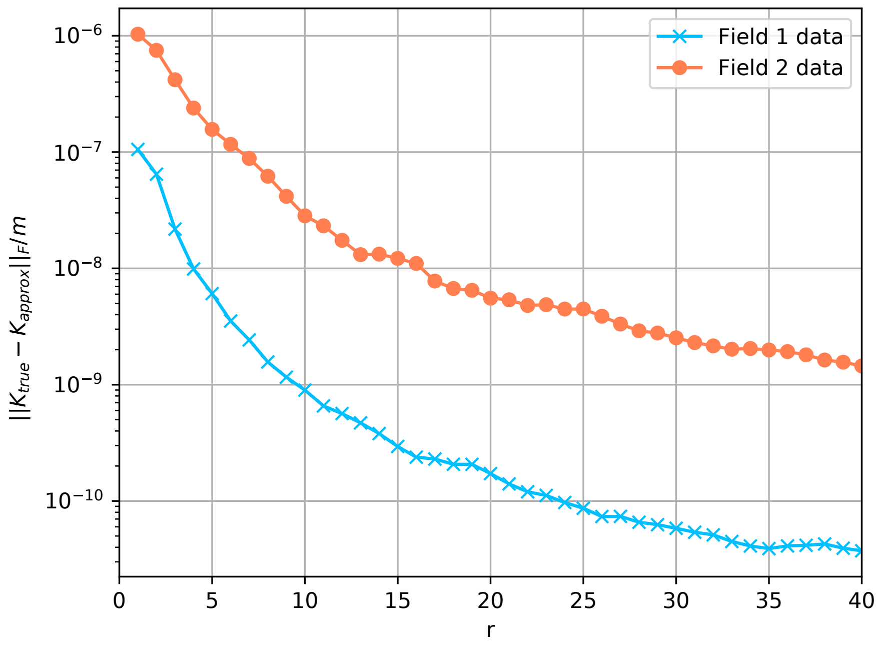


Figure 6.6: Low-rank kernel matrix approximation for increasing rank r for the data sets corresponding to field 1 and 2, respectively. We used a RBF kernel with $\gamma = \frac{1}{3N}$ and $n = 300$ samples.

see that the prediction time actually increases. This is due to the low-rank approximation, since the truncated data tensor of size $n \times \nu d + 2$ is inflated again to a size of $n \times (s+1)r$. This seems like a big drawback at first, but we also need to consider the computability of the kernel matrix. For very large datasets, the kernel matrix would not fit into memory, but the low-rank approximation might still do. This tradeoff between ease of computation and speed depends on the kernel approximation error decay as shown in Figure 6.6. If the decay is slow, we might want to consider another kernel, or another method for feature space integration. In Table 6.5, where we show the result of a 5-fold cross validation for the parameter r, this issue is even more apparent. We used r vectors for the kPCA and select the same truncated basis vectors, to perform feature space integration. It is not clear, if the kernel matrix of low-rank KRR actually need as many basis vectors for a good approximation.

n	pred. error [-]	fit time [s]	pred. time [s]
10	0.0107	1.23	0.08
50	0.0062	24.80	0.72
100	0.0050	34.56	0.88
200	0.0038	52.76	1.22

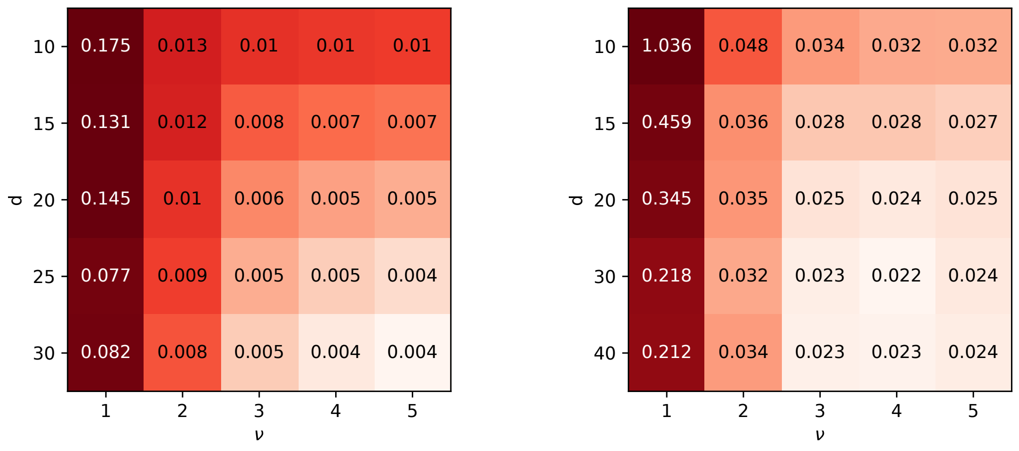
(a) Field 1 case with $\nu = 3$, $d = 20$ and $r = 30$

n	pred. error [-]	fit time [s]	pred. time [s]
10	0.0559	1.20	0.09
50	0.0493	49.59	1.10
100	0.0331	103.53	2.36
200	0.0242	138.96	3.16

(b) Field 2 case with $\nu = 5$, $d = 40$ and $r = 50$

Table 6.4: Prediction results of a 5-fold cross validation of the explicit feature space integration using ℓ-kPCA as reducer and ℓ-KRR as integrator. For a different number of samples n we show fit time, prediction time and MSE. The prediction error relates only to the last timestep. We used a RBF kernel for the kPCA, the integrator and the pre-image KRR map. The parameters were chosen as follows: kernel parameter $\gamma_{KPCA} = \frac{1}{3N}$ and $\gamma_{KRR} = 1$, integrator regularization parameter $\alpha_{KRR} = 0.01$, pre-image regularization parameter $\alpha_{KPCA} = 0.001$.

In Figure 6.8 we show the learning curve. We note that, due to the explicit solution of kPCA and KRR, we have only a small variance in the model performance and therefore a



(a) Field 1 case with $r = 20$ (b) Field 2 case with $r = 40$

Figure 6.7: Cross validation (5-fold) table for varying number of kernel principal components d and step parameter ν. The tables show the mean error norm in the magnetization for the prediction with ℓ-kPCA as reducer and ℓ-KRR as integrator over all timesteps. The parameters were chosen as follows: kernel parameter $\gamma_{KPCA} = \frac{1}{3N}$ and $\gamma_{KRR} = 1$, integrator regularization parameter $\alpha_{KRR} = 0.01$, pre-image regularization parameter $\alpha_{KPCA} = 0.001$ and $n = 300$ samples.

more predictable model. Also, we see that the training error decreases, which is unusual. We assume this has to do with regularization. For less training samples, we have a smaller kernel matrix and the regularization will have a stronger effect on the smaller matrix, resulting in a stiffer model.

r	pred. error [-]	fit time [s]	pred. time [s]
5	0.0270	16.30	0.23
10	0.0047	31.65	0.67
50	0.0031	322.82	7.50
100	0.0031	1269.50	31.44

(a) Field 1 case with $\nu = 3$ and $d = 20$

r	pred. error [-]	fit time [s]	pred. time [s]
5	0.1937	17.13	0.28
10	0.2287	31.24	0.71
50	0.0274	325.36	7.61
100	0.0173	1301.31	34.96

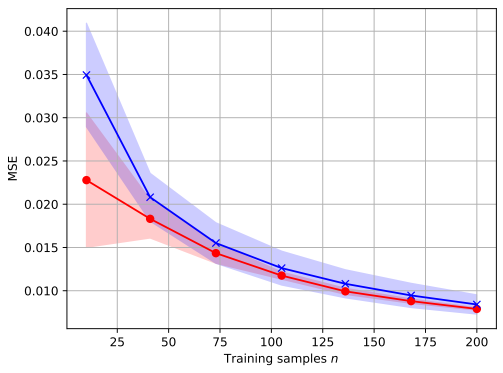
(b) Field 2 case with $\nu = 5$ and $d = 40$

Table 6.5: Prediction results of a 5-fold cross validation of the explicit feature space integration using ℓ-kPCA as reducer and ℓ-KRR as integrator. For a varying number of basis vectors r we show fit time, prediction time and prediction error. The prediction error relates only to the last timestep. We used a RBF kernel for the kPCA, the integrator and the pre-image KRR map. The parameters were chosen as follows: kernel parameter $\gamma_{KPCA} = \frac{1}{3N}$ and $\gamma_{KRR} = 1$, integrator regularization parameter $\alpha_{KRR} = 0.01$, pre-image regularization parameter $\alpha_{KPCA} = 0.001$ and $n = 200$ samples.

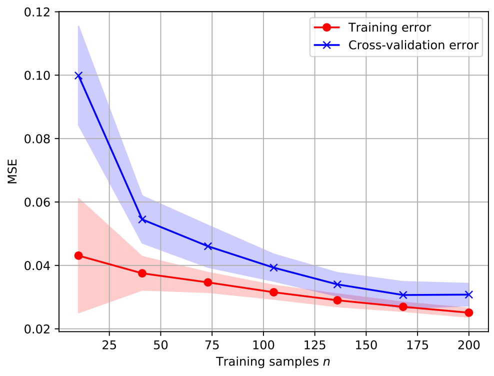
6.3.2 Prediction results

Figure 6.9 shows the mean magnetization for a varying number of basis vectors. We can see that a better approximation of the kernel matrix offers better prediction results. Figure 6.10 shows that even with a few basis vectors we already have a good estimate for the first few steps. For field 1, only after the magnetization component m_x completely reverses and the dynamics becomes more complex, more basis vectors are needed for a good estimate.

For completeness, we show also the low-rank approximation with a varying number of principal components in Figure 6.11 and 6.12 (compare with Figure 6.3 and 6.4). We can observe that the prediction error is very similar to the full-rank version.

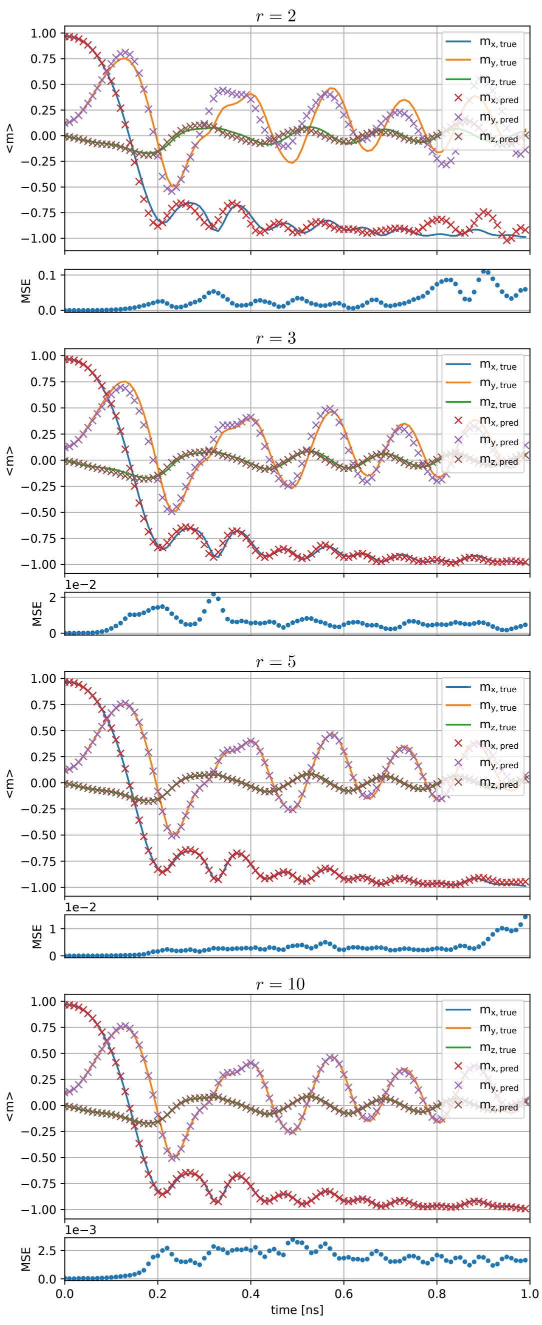


(a) Field 1 case with $\nu = 3$, $d = 20$ and $r = 30$



(b) Field 2 case with $\nu = 5$, $d = 40$ and $r = 50$

Figure 6.8: Learning curves based on a 10-fold cross-validation with a testset size of 20% for the explicit feature space integration using ℓ-kPCA as reducer and ℓ-KRR as integrator. The prediction error is averaged over all timesteps. Further, we show the standard deviation of training and test error represented by the filled area. We used a RBF kernel for the kPCA, the integrator and the pre-image KRR map. The parameters were chosen as follows: kernel parameter $\gamma_{KPCA} = \frac{1}{3N}$ and $\gamma_{KRR} = 1$, integrator regularization parameter $\alpha_{KRR} = 0.01$, pre-image regularization parameter $\alpha_{KPCA} = 0.001$.


(a) Field 1 case with $\nu = 3$ and $d = 20$

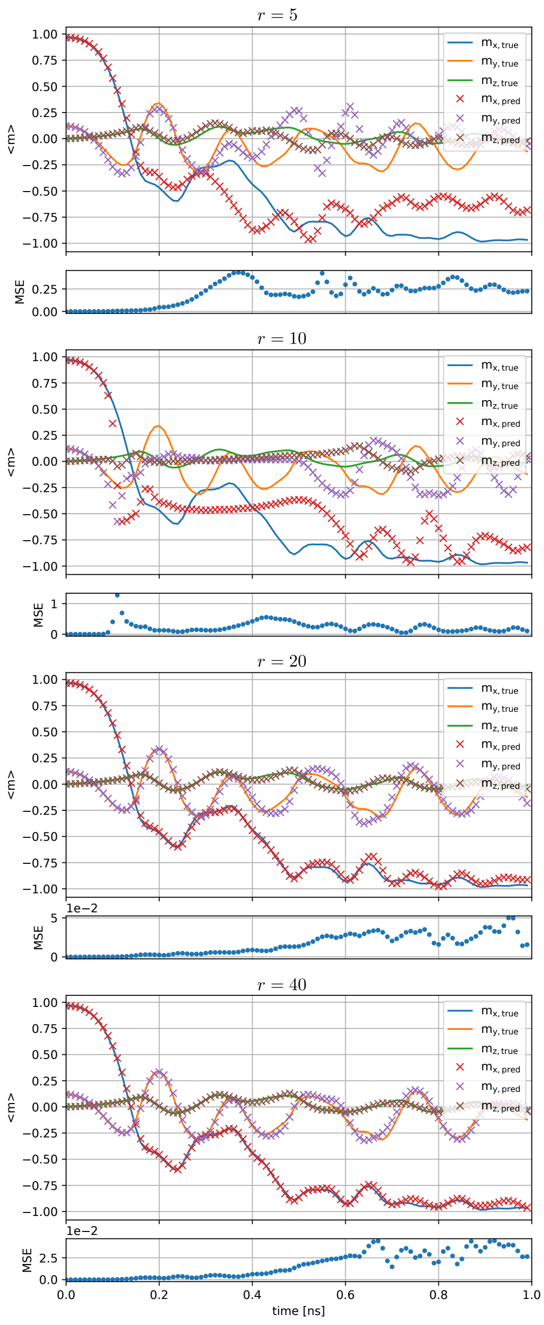
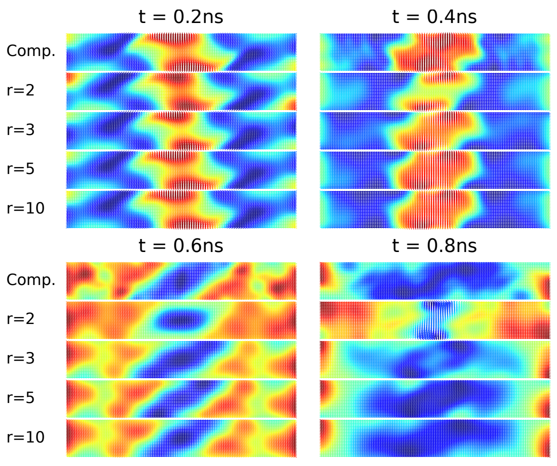
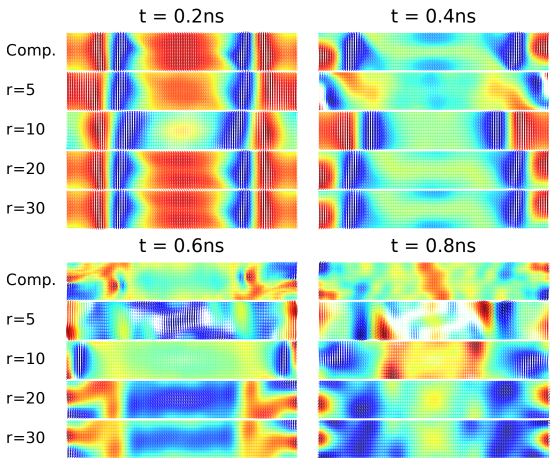

(b) Field 2 case with $\nu = 5$ and $d = 40$

Figure 6.9: Prediction using ℓ-kPCA as reducer and ℓ-KRR as integrator. We show the mean magnetization for a varying number of basis vectors r. We used a RBF kernel for the ℓ-kPCA, the integrator and the pre-image ℓ-KRR map. The parameters were chosen as follows: kernel parameter $\gamma_{KPCA} = \frac{1}{3N}$ and $\gamma_{KRR} = 1$, integrator regularization parameter $\alpha_{KRR} = 0.01$, pre-image regularization parameter $\alpha_{KPCA} = 0.001$ and $n = 200$ samples.

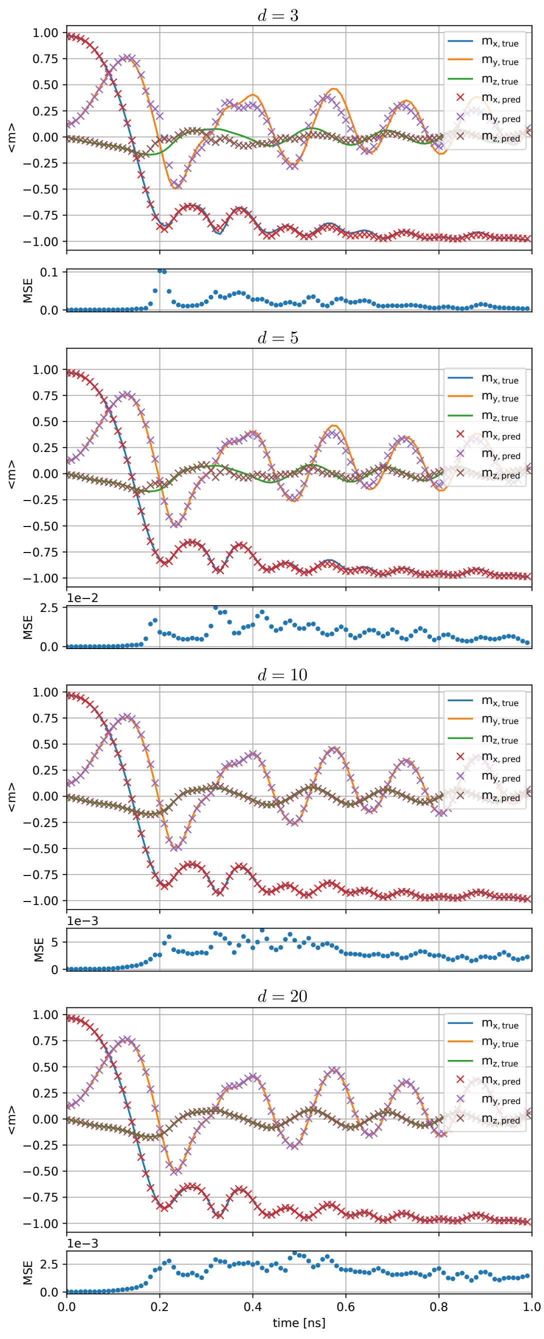


(a) Field 1 case with $\nu = 3$ and $d = 20$

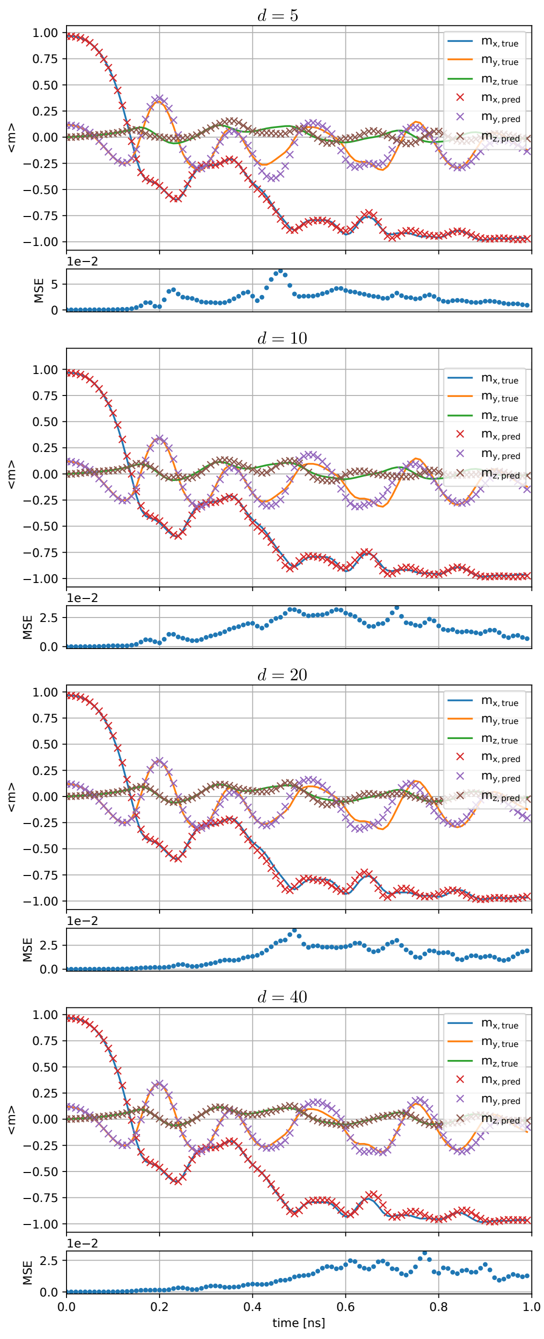


(b) Field 2 case with $\nu = 5$ and $d = 40$

Figure 6.10: Magnetization snapshots for a varying number of basis vectors r corresponding to Figure 6.9. We used a RBF kernel for the kPCA, the integrator and the pre-image KRR map. We used ℓ-kPCA as reducer and ℓ-KRR as integrator. We used a RBF kernel for the kPCA, the integrator and the pre-image KRR map. The parameters were chosen as follows: kernel parameter $\gamma_{KPCA} = \frac{1}{3N}$ and $\gamma_{KRR} = 1$, integrator regularization parameter $\alpha_{KRR} = 0.01$, pre-image regularization parameter $\alpha_{KPCA} = 0.001$ and $n = 200$ samples.

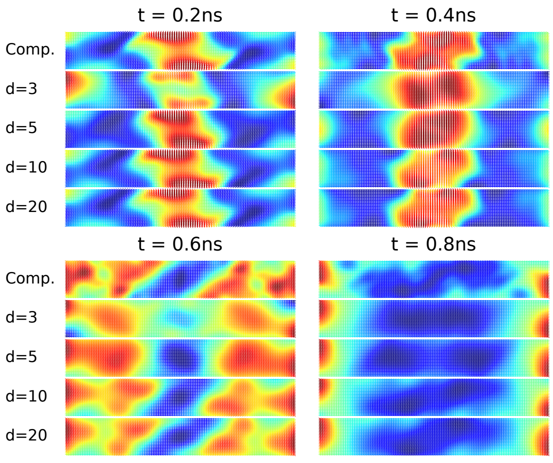


(a) Field 1 case with $\nu = 3$ and $r = 30$

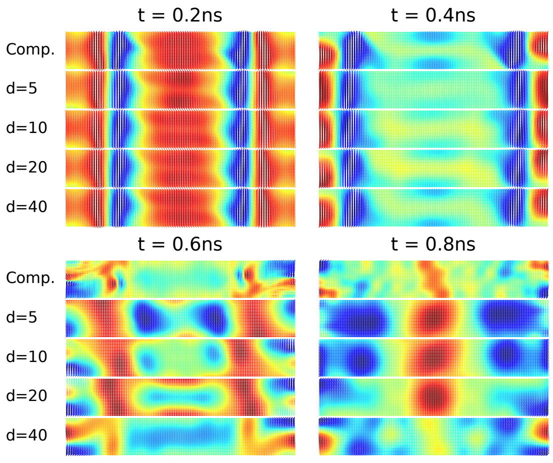


(b) Field 2 case with $\nu = 5$ and $r = 50$

Figure 6.11: Prediction using ℓ-kPCA as reducer and ℓ-KRR as integrator. We show the mean magnetization for a varying number of principal components d. We used a RBF kernel for the ℓ-kPCA, the integrator and the pre-image ℓ-KRR map. The parameters were chosen as follows: kernel parameter $\gamma_{KPCA} = \frac{1}{3N}$ and $\gamma_{KRR} = 1$, integrator regularization parameter $\alpha_{KRR} = 0.01$, pre-image regularization parameter $\alpha_{KPCA} = 0.001$ and $n = 200$ samples.



(a) Field 1 case with $\nu = 3$ and $r = 30$



(b) Field 2 case with $\nu = 5$ and $r = 50$

Figure 6.12: Magnetization snapshots for a varying number of principal components d corresponding to Figure 6.11. We used a RBF kernel for the kPCA, the integrator and the pre-image KRR map. We used ℓ-kPCA as reducer and ℓ-KRR as integrator. We used a RBF kernel for the kPCA, the integrator and the pre-image KRR map. The parameters were chosen as follows: kernel parameter $\gamma_{KPCA} = \frac{1}{3N}$ and $\gamma_{KRR} = 1$, integrator regularization parameter $\alpha_{KRR} = 0.01$, pre-image regularization parameter $\alpha_{KPCA} = 0.001$ and $n = 200$ samples.

6.4 Autoencoder and FFNN for Feature Space integration

There are many ways to design an autoencoder for this task and optimizing the architecture would be a special topic by its own. We choose a simple feed forward architecture. If we want to reduce the dimension from one layer of size l to the next layer of size k, we need $lk + k$ parameters for a dense layer. This can become problematic for a big input dimension, since it requires a big reduction from the first layer to the second one, in order to reduce the model complexity. Other models such as a Convolutional Autoencoder have less parameters, but have worse reconstruction properties. If we want to apply additional regularization terms, such as a contractive term (3.10), having a strong reduction in the first layer gives the additional advantage that we can use objective 3.17 for faster training. Figure 6.13 illustrates this architecture and Table 6.6 shows the detailed implementation.

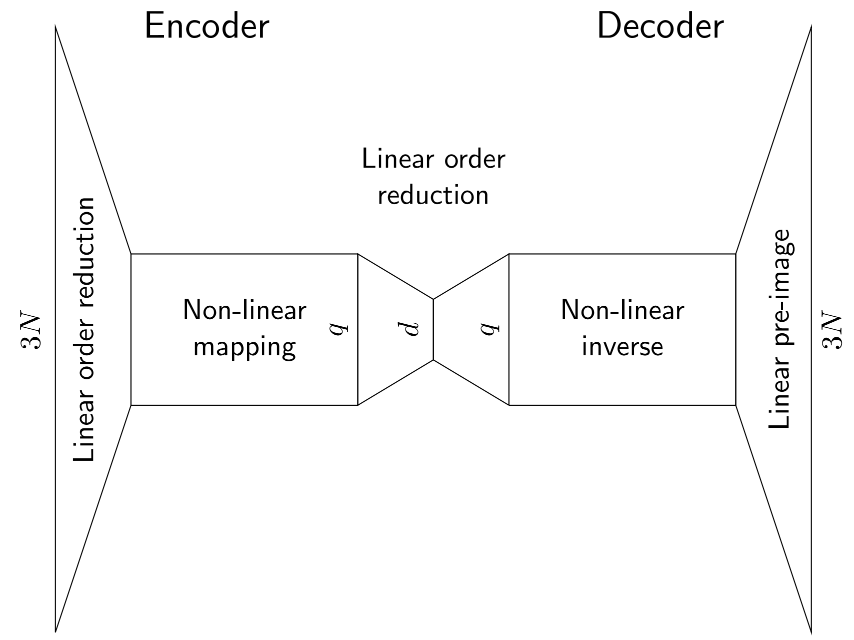


Figure 6.13: Autoencoder model for the prediction of magnetization dynamics. Input and output size is $3N$. The encoder maps the input to the feature space with latent-dimension d. The linear mapping of the first layer reduces the dimension to a number $q > d$. This is followed by a nonlinear mapping which consists of some dense layers of size q with an *ELU* activation. The final order reduction is done in a linear manner again. The decoder is a mirrored version of the encoder.

6.4.1 Cross-validation of the hyper-parameters

We perform a similar cross validation as in Section 6.3 over the time stepping parameter ν and the latent dimension of the autoencoder d. From Figure 6.14 we can see that we get a similar prediction accuracy as with the low-rank version (compare Figure 6.7), but with a smaller feature space. We only performed a 2-fold cross validation because of the computational cost of optimization.

Further, we performed a cross validation over the forward looking parameter f_t. Table 6.8 shows the results of this cross validation. The prediction accuracy does not improve if $f_t > 10$, but also the results are not significantly worse. However, fit time increases since the objective function becomes more difficult to optimize.

If Figure 6.15, we show the learning curves of the autoencoder and FFNN. If we compare the learning curve with Figure 6.8, we see immediately that the variance is much higher. This

Layer	Activation	Output shape
Input	-	$3N = 7500$
Dense	elu	400
Dense	elu	400
Dropout	-	400
Dense	linear	d
Dense	elu	400
Dropout	-	400
Dense	elu	400
Dense	linear	7500

Table 6.6: Autoencoder architecture (layer type, activation function and output shape).

Layer	Activation	Output shape
Input	-	$2 + \nu d$
Dense	elu	400
Dense	elu	300
Dense	elu	300
Dense	elu	200
Dense	elu	100
Dense	linear	d

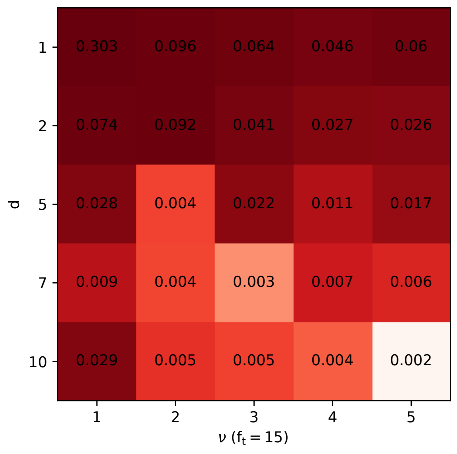
Table 6.7: Architecture of the feed-forward neural network used for latent space integration of the autoencoder. For regularization we apply a dropout layer after each nonlinear activation.

means, when training a model, we cannot assume that it is optimal. This is a disadvantage, since we are interested in a predictable model. One could use Quasi-Newton methods, such as *Limited-memory BFGS* to obtain a more accurate solution to the optimization problem, but this is more difficult to perform on large datasets. We also note that the training error decreases for the field 1 case. Given that a larger training set will result in a longer training phase, we can assume that the solution is not yet optimal for less samples. If it were optimal, the training error would have to increase, since the model complexity is not influenced by the number of training samples as it is for the ℓ-kPCA/ℓ-KRR model.

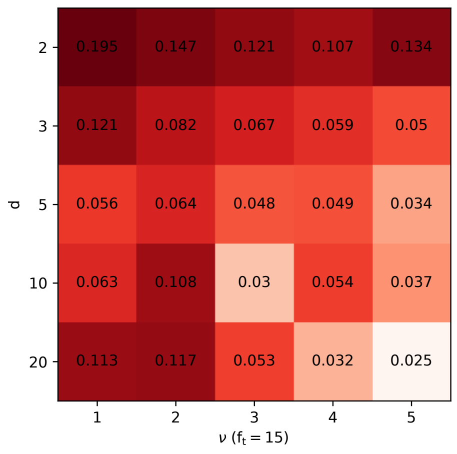
So far, all experiments did not include a contractive term. Table 6.9 and 6.10 show a 2-fold cross validation over the contraction term λ for the objective function (3.18) and (3.19). We see that the contractive term, in our particular case, has a negative effect on the prediction results. Depending on the dataset, the term might be beneficial as well. Further, we see a small decrease in the fit time when we switch to the objective (3.19). Since the feature space integration objective is more complicated, the benefit is only small. Note that we average the mean square error, which requires pre-image computation over all timesteps. This increases the prediction time.

6.4.2 Prediction results

We show the prediction results for a varying forward looking parameter in Figure 6.16 and the corresponding magnetization snapshots in Figure 6.17. We can see from these that the neural networks can store more details than the low-rank approach with less parameters. On



(a) Field 1 case with $\nu = 3$



(b) Field 2 case with $\nu = 5$

Figure 6.14: Cross validation (2-fold) table for varying number of latent space dimensions d and step parameter ν. The tables show the mean error norm in the magnetization for the prediction with the autoencoder from Table 6.6 as reducer and a FFNN (Table 6.7) as integrator over all timesteps. For optimization we used the Adam optimizer. The parameters were chosen as follows: sample size $n = 200$, autoencoder learning rate $l_{AE} = 0.0002$ and FFNN learning rate $l_{FFNN} = 0.0002$.

f_t	pred. error [-]	fit time [s]	pred. time [s]
1	0.1079	285.32	3.20
5	0.0027	366.87	2.87
10	0.0036	492.57	2.89
15	0.0038	611.72	2.87

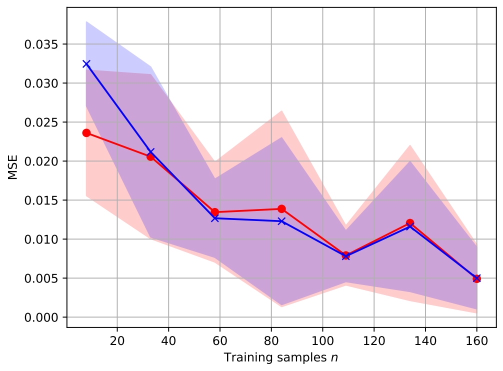
(a) Field 1 case with $\nu = 3$, $d = 10$

f_t	pred. error [-]	fit time [s]	pred. time [s]
1	0.4578	306.75	3.11
5	0.0487	392.30	2.69
10	0.0253	538.24	2.70
15	0.0358	677.65	2.65

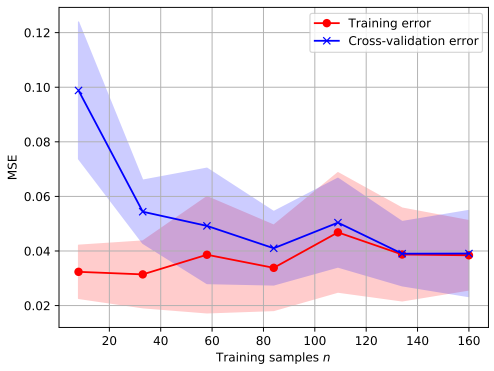
(b) Field 2 case with $\nu = 5$, $d = 20$

Table 6.8: Prediction results of a 2-fold cross validation of the explicit feature space integration using an autoencoder (Table 6.6) as reducer and FFNN as integrator (Table 6.7). For the forward looking parameter f_t, we show fit time, prediction time and MSE. The prediction error relates only to the last timestep. We used the Adam optimizer with the autoencoder learning rate $l_{AE} = 0.0002$ and FFNN learning rate $l_{FFNN} = 0.0002$ and sample size $n = 200$.

the downside, optimization is way more difficult. Further, the number of parameters is not dependent on the training set size.



(a) Field 1 case with $\nu = 3$ and $d = 10$



(b) Field 2 case with $\nu = 5$ and $d = 20$

Figure 6.15: Learning curve based on a 10-fold cross-validation with a testset size of 20% for the autoencoder from Table 6.6 as reducer and a FFNN (Table 6.7) as integrator. The prediction error is averaged over all timesteps. Further, we show the standard deviation of training and test error represented by the filled area. For optimization we used the Adam optimizer. The parameters were chosen as follows: sample size $n = 200$, autoencoder learning rate $l_{AE} = 0.0002$, FFNN learning rate $l_{FFNN} = 0.0002$ and $f_t = 15$.

λ	pred.error [-]	fittime [s]	pred.time [s]
0	0.0087	647.93	6.10
1E-6	0.0070	763.36	6.11
1E-4	0.0054	766.33	5.94
1E-2	0.0112	764.93	5.94
1E-0	0.0181	764.93	6.21

(a) Field 1 case with $\nu = 3$ and $d = 10$

λ	pred.error [-]	fittime [s]	pred.time [s]
0	0.0277	619.27	6.20
1E-6	0.0463	866.89	6.00
1E-4	0.0291	860.21	6.14
1E-2	0.0672	859.41	6.07
1E-0	0.0623	875.66	6.09

(b) Field 2 case with $\nu = 5$ and $d = 20$

Table 6.9: Prediction results of a 2-fold cross validation of the explicit feature space integration using an autoencoder (Table 6.6) as reducer and FFNN as integrator (Table 6.7). Equation (3.18) was used as objective function. For the contraction parameter λ, we show fit time, prediction time and MSE. The prediction error is averaged over all timesteps. We used the Adam optimizer with the autoencoder learning rate $l_{AE} = 0.0002$ and FFNN learning rate $l_{FFNN} = 0.0002$, a sample size $n = 200$ and $f_t = 15$.

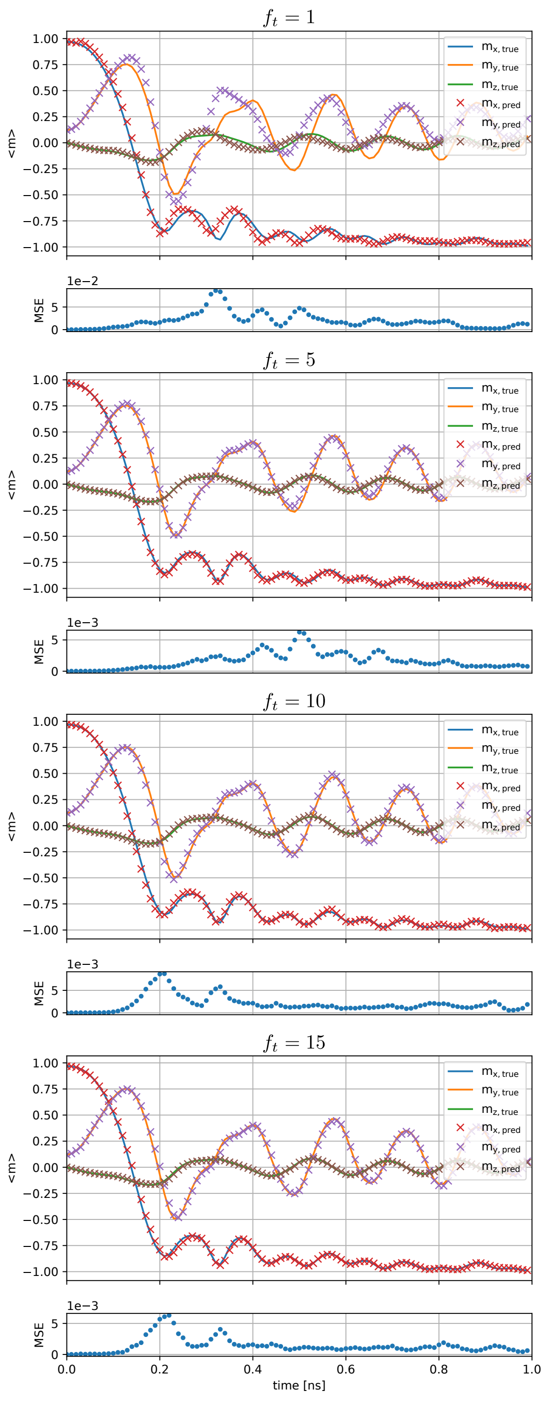
λ	pred.error [-]	fittime [s]	pred.time [s]
0	0.0033	640.85	6.07
1E-6	0.0088	668.67	6.42
1E-4	0.0113	646.44	6.09
1E-2	0.0264	661.66	5.94
1E-0	0.0458	646.98	6.10

(a) Field 1 case with $\nu = 3$ and $d = 10$

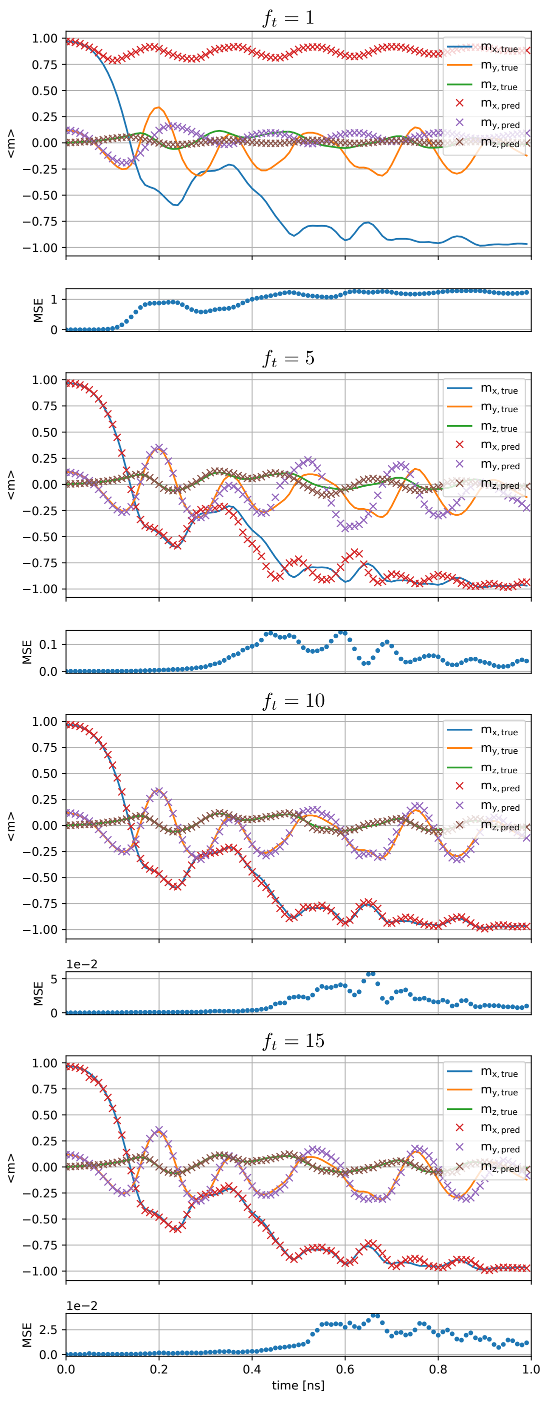
λ	pred.error [-]	fittime [s]	pred.time [s]
0	0.0322	626.60	6.22
1E-6	0.0266	638.89	6.15
1E-4	0.0501	633.60	6.10
1E-2	0.1164	633.64	5.95
1E-0	0.0971	643.88	5.94

(b) Field 2 case with $\nu = 5$ and $d = 20$

Table 6.10: Prediction results of a 2-fold cross validation of the explicit feature space integration using an autoencoder (Table 6.6) as reducer and FFNN as integrator (Table 6.7). Equation (3.19) was used as objective function. For the contraction parameter λ, we show fit time, prediction time and MSE. The prediction error is averaged over all timesteps. We used the Adam optimizer with the autoencoder learning rate $l_{AE} = 0.0002$ and FFNN learning rate $l_{FFNN} = 0.0002$, a sample size $n = 200$ and $f_t = 15$

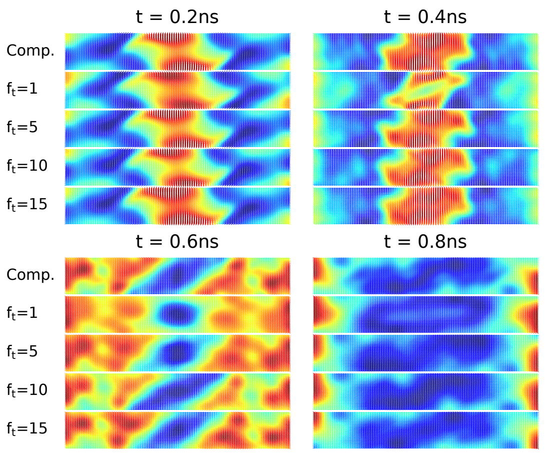


(a) Field 1 case with $\nu = 3$ and $d = 10$

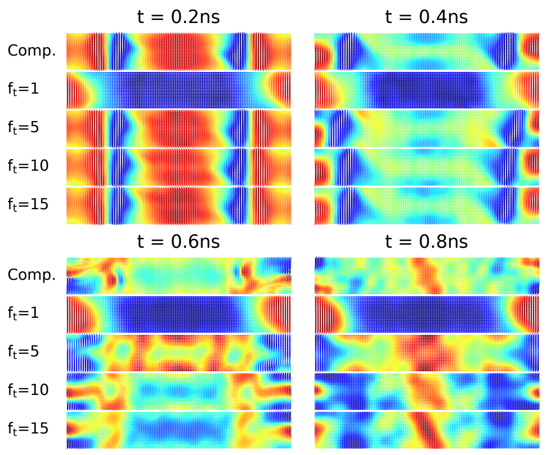


(b) Field 2 case with $\nu = 5$ and $d = 20$

Figure 6.16: Prediction using the autoencoder from Table 6.6 as reducer and a FFNN (Table 6.7) as integrator. We show the mean magnetization for a varying forward looking parameter f_t. The parameters were chosen as follows: sample size $n = 200$, autoencoder learning rate $l_{AE} = 0.0002$ and FFNN learning rate $l_{FFNN} = 0.0002$.



(a) Field 1 case with $\nu = 3$ and $d = 10$



(b) Field 2 case with $\nu = 5$ and $d = 20$

Figure 6.17: Magnetization snapshots for a varying forward looking parameter f_t corresponding to Figure 6.16 using the autoencoder from Table 6.6 as reducer and a FFNN (Table 6.7) as integrator. The parameters were chosen as follows: sample size $n = 200$, autoencoder learning rate $l_{AE} = 0.0002$ and FFNN learning rate $l_{FFNN} = 0.0002$.

Chapter 7

Discussion and Conclusion

In this thesis we went over the basics of kernel methods such as Kernel Ridge Regression and Kernel Principal Component Analysis. We then learned how to use low-rank approximations to improve performance. An essential, but often underrated part is the pre-image computation. We learned how to use supervised learning in order to train a pre-image map from a feature space to the original space. In Chapter 3 we had a look on Feedforward Neural Network and learned how to replace kPCA with an autoencoder. We derived the back-propagation algorithm and explained the tie between PCA and linear autoencoders.

The main inspiration of this thesis comes from the idea of Kernel Dependency Estimation (KDE), which we explained in Section 4.1. We went further and developed an explicit version of this algorithm, which is capable of faster prediction with comparable accuracy. Section 4.2 and 5 explain how to put those components together, in order to train a model which is capable of predicting the solution of a PDE. We developed the main algorithm, which is used for the prediction of magnetization dynamics (Algorithm 2) and did essential research and experiments for the NIST μMAG Standard problem $\#4$ [31]. We used three essentially different approaches in order to solve the LLG equation (1.1) and for validation we used two datasets corresponding to different scenarios. The first approach was the prediction with the pure implicit version of KDE. This algorithm shows an incredible prediction accuracy, but suffers from slow prediction time. We then used Algorithm 2 together with kPCA and KRR in order to perform explicit feature space integration (entirely in feature space). We saw that this method is able to learn the overall magnetization dynamics and has comparable errors as KDE. The novelty which we present is the replacement of the kernel methods with the corresponding low-rank versions. This is especially efficient, if only a few basis vectors are needed for a good approximation (see Figure 6.6) and if the original feature space is large. On the contrary, if the eigenvalues of the kernel matrix decay only slowly, we need many basis vectors and will therefore slow down the algorithm. A good choice of the kernel function is therefore essential. We examined the effect of the rank on the prediction results. In the last approach, we replaced the kPCA with an autoencoder and the KRR with a FFNN and saw that this version can reduce model complexity even further. On the downside, it suffers from a complicated optimization problem.

There are still many unsolved issues with out explicit integration scheme. One could argue that the kernel methods only work that well because all magnetization vectors are normalized. We were thinking of replacing the RBF kernel with a data-dependent version [56]. Further, we could utilize tensor decomposition and compression techniques [47] to further improve performance for larger datasets. Or, for very large datasets, there is a need for an incremental version which could be used in an online setting.

Deep neural networks supposedly solve all those issues above, but also bring their own. One main issue is the architecture choice. There are countless ways on how to design an autoencoder or a FFNN and it is very hard to determine a good choice. Further, autoencoders often lack smoothness of the latent space, which can become troublesome. In our case the problem setting turned out to work well with an autoencoder, but this does not mean that it always works. Therefore, we explained how to use further regularization techniques, such as a sparse or contractive term.

Feature space integration is based on a multistep scheme. This may offer good prediction results for our problem setting, but for other problems it would require the calculation or prediction of the first timesteps. Therefore, we would need another model for this task, or replace the feature space integration scheme by something more powerful. One such model could be a Neural Ordinary Differential Equation [8], but would require further research.

Physics-informed neural networks (PINNs) [38] offer an alternative approach to train models which are capable to solve PDEs. This can result in a model which is not limited in some kind of discretionary, but works on the full domain. Further, this method does potentially not even require training data, but can learn the solution from the weak formulation of the PDE. For the LLG equation, we face the problem that the stray field is not limited to a single point, but needs to be evaluated on the full domain. This can become hard to implement for PINNs.

For ever growing problem settings, traditional solvers often reach their limit. It is only a matter of time until we see the first solvers based on machine learning used on a commercial level. This thesis offers an introduction to this rather complicated task, but still misses many aspects. There are many different models which were not covered and as we know from the *no free lunch theorem* [55], there is no single model to cover all problem settings which arise in the real world.

Appendix A

A.1 Representer Theorem

One of the crucial properties of kernels is that even if the input domain $\mathcal{X}$ is only a (finite) set, we can think of the pair $(\mathcal{X}, k)$ as a (subset of a) Hilbert space. This is attractive from a mathematical point of view, since many data structures can be studied in Hilbert spaces. This raises the practical problem that for many kernels, the Hilbert space is known to be infinite-dimensional. However, we do not normally want to solve an optimization problem in an infinite-dimensional space. A large class of optimization problems with RKHS regularizers, have solutions that can be expressed as kernel expansions in terms of the training data [44].

Theorem A.1.1 (Representer Theorem [45])**.** *Given a mapping Φ from $\mathcal{X}$ to some Hilbert space $\mathcal{F}_{\mathcal{X}}$ with the inner product $\langle \cdot, \cdot \rangle$ and the optimization problem*

$$\min_{w \in \mathcal{F}_{\mathcal{X}}} \left(f\left(\langle w, \Phi(x_1) \rangle, \ldots, \langle w, \Phi(x_m) \rangle \right) + g\left(\|w\| \right) \right). \tag{A.1}$$

The norm $\|\cdot\|$ denotes the norm in the Hilbert space $\mathcal{F}_{\mathcal{X}}$.
Then, there exists a vector $\alpha \in \mathbb{R}^m$ such that $w = \sum_{i=1}^m \alpha_i \Phi(x_i)$ is an optimal solution of equation (A.1).

Proof. Let w^* be an optimal solution of equation (A.1). Because w^* is an element of a Hilbert space, we can rewrite it as an unique orthogonal sum

$$w^* = \sum_{i_1}^{m} \alpha_i \Phi(x_i) + u \tag{A.2}$$

where $\langle u, \Phi(x_i) \rangle = 0$ for all i. Set $w = w^* - u$. We can see that $\|w^*\|^2 = \|w\|^2 + \|u\|^2$ and therefore $\|w\| \leq \|w^*\|$. Since g is non-decreasing we obtain that $g(\|w\|) \leq g(\|w^*\|)$. Further, for all i we have that

$$\langle w, \Phi(x_i) \rangle = \langle w^* - u, \Phi(x_i) \rangle = \langle w^*, \Phi(x_i) \rangle, \tag{A.3}$$

and therefore

$$f\left(\langle w, \Phi(x_1) \rangle, \ldots, \langle w, \Phi(x_m) \rangle \right) = f\left(\langle w^*, \Phi(x_1) \rangle, \ldots, \langle w^*, \Phi(x_m) \rangle \right). \tag{A.4}$$

We have shown that the objective of equation (A.1) at w cannot be larger than the objective at w^* and therefore w is also an optimal solution. Because $w = \sum_{i=1}^m \alpha_i \Phi(x_i)$, this concludes the proof [45]. $\square$

The representer theorem says that, when we are searching for an optimal solution in $\mathcal{H}$ we can instead optimize (A.1) with respect to the coefficients $\alpha \in \mathbb{R}^m$ as follows. Applying

w to the objective and using the kernel trick yields the following

$$
\begin{aligned}
&\min_{\alpha\in\mathbb{R}^m}\left(f\left(\langle\sum_{i=1}^m \alpha_i\Phi(x_i),\Phi(x_1)\rangle,\dots,\langle\sum_{i=1}^m \alpha_i\Phi(x_i),\Phi(x_m)\rangle\right)\right.\\
&\qquad\left.+g\left(\sqrt{\langle\sum_{i=1}^m \alpha_i\Phi(x_i),\sum_{i=1}^m \alpha_i\Phi(x_i)\rangle}\right)\right)\\
\iff&\min_{\alpha\in\mathbb{R}^m}\left(f\left(\sum_{i=1}^m \alpha_i\langle\Phi(x_i),\Phi(x_1)\rangle,\dots,\sum_{j=1}^m \alpha_j\langle\Phi(x_j),\Phi(x_m)\rangle\right)\right.\\
&\qquad\left.+g\left(\sqrt{\sum_{i,j=1}^m \alpha_i\alpha_j\langle\Phi(x_i),\Phi(x_j)\rangle}\right)\right)\\
\iff&\min_{\alpha\in\mathbb{R}^m}\left(f\left(\sum_{i=1}^m \alpha_i k(x_i,x_1),\dots,\sum_{j=1}^m \alpha_j k(x_i,x_m)\right)+g\left(\sqrt{\sum_{i,j=1}^m \alpha_i\alpha_j k(x_i,x_j)}\right)\right)
\end{aligned}
\tag{A.5}
$$

A.2 Low-rank matrices and Eckart-Young theorem.

Finding a matrix approximation $\tilde{A}$ to a rank-p matrix A with rank $r \leq p$ can be found by the SVD. There holds:

Theorem A.2.1 (Eckart-Young)**.** *Given the matrix A with rank p and a SVD $A = USV^*$, the optimization problem*

$$\min_{\tilde{A}} \|A-\tilde{A}\|_F \quad \textit{with } rank(\tilde{A}) = r, \tag{A.6}$$

with $\|A\| = \sqrt{\sum_{i,j}|a_{i,j}|^2}$ being the Frobenius Norm, turns out to have the solution

$$A_r = U\bar{S}_r V^*. \tag{A.7}$$

The matrix $\bar{S}_r$ is a modified version of S that only contains the largest r singular values, while all the other ones are replaced by zero. For the error there holds

$$\|A_r - A\|_F = \sqrt{\sum_{j=r+1}^p \sigma_h^2}. \tag{A.8}$$

A.3 Deutsche Zusammenfassung

Diese Masterarbeit behandelt einen datengetriebenen („data-driven") Ansatz zum Lösen von zeitabhängigen PDEs. Im Speziellen werden Methoden zur Modellreduktion und maschinellen Lernen für die schnelle Abschätzung der Magnetisierungsdynamik in Abhängigkeit zum äußeren Feld entwickelt und angewandt, welche durch die Landau-Lifschitz-Gilbert (LLG) Gleichung, die fundamentale partielle Bewegungs-Differentialgleichung im Feld des Mikromagnetismus, modelliert wird. Die Arbeit beschreibt die Theorie von bestimmten Modellreduktionsmethoden, sowie nicht lineare Regressionsmodelle, zum Zweck des effizienten maschinellen Lernens der Lösungstrajektorien der LLG-Gleichung als parameterabhängige PDE. Ein Schwerpunkt lieg bei Kernel-Methoden, mit dem Fokus auf numerisch stabile und effiziente Implementierung von Kernel Ridge-Regression (kRR) und der Kernel-Hauptkomponentenanalyse (kPCA) mit einem modernen Niedrig-Rang-Approximationsverfahren zur Behandlung von bestimmten dichten Operatoren. Viele dieser Verfahren berechnen die Abbildung der Daten in einem anderen Funktionsraum, daher ist die Urbild-Berechnung essenziell. Im Zuge dieser Arbeit wird ein „supervised learning" Prozess vorgestellt, der wiederum auf Kernel Ridge-Regression beruht.

Nach Darlegung dieser Grundlagen, wird eine iterative Lösung des Kernel-Dependency-Estimation Algorithmus (KDE) hergeleitet und, basierend darauf, ein explizites Mehrschritt-Verfahren entwickelt welches ausschließlich im abgeleiteten Funktionsraum operiert und die komplizierte Magnetisierungsdynamik in einem reduzierten Raum erlernt. Dieser Algorithmus ermöglicht eine schnelle Vorhersage der Lösung mit einer ähnlichen Genauigkeit wie KDE. Weiters wird die Datenstruktur dieser Methode diskutiert, sowie die nötige Speicherkapazität verglichen. Die Implementierung erfolgte mit dem *numpy* und *scikit-lern* Python Modulen, und Simulationen wurden teilweise auf dem *Vienna Scientific Cluster (VSC)* gerechnet.

In einem zweiten Teil der Arbeit wird die Hauptkomponentenanalyse mit einem neuronalen Autoencoder zur Dimensionsreduktion des Datensatzes verglichen. Wir legen den Fokus auf eine glatte Beschreibung des Funktionsraumes mithilfe eines regularisierenden Terms eines kontraktiven Autoencoders. Es stellt sich heraus, dass die Funktionsraumbeschreibung des Autoencodes besser ist als die der kPCA und mithilfe einer zukunftsgerichteten Zielfunktion kann ein neuronales Regressions-Netzwerk trainiert, und damit die KRR ersetzt werden. Mithilfe des *Keras* und *Tensorflow* Moduls, und mittels automatischen Differenzierens, ist es möglich diese sehr komplizierten Optimierungsprobleme zu lösen.

Im Falle der Kernel Methoden bietet das explizite Funktionsraumintegrationsschema einen einfachen Trainingsprozess mit expliziter Lösung des aufkommenden Optimierungsproblems. Dies ist schwieriger im Falle des neuronalen Netzwerkes und erfordert fortschrittliche stochastische Optimieralgorithmen mit adaptiver Momentabschätzung. Im Speziellen wird der Adam Optimieralgorithmus verwendet. Mithilfe von Cross-Validation wird eine Serie von Modellvalidierungs- und Hyperparameterschätzungsaufgaben durchgeführt und es stellt sich heraus, dass diese Methoden einen teils überwachten Trainingsprozess bieten, der darüber hinaus auch einen beachtlich kleineren Rechenaufwand erfordert als andere Lösungsmethoden.

Diese Arbeit ist thematisch an der Spitze der derzeitigen Forschung im Bereich von *Computational Physics*, und es wurden im Zuge dieser Arbeit zwei Preprints zu international anerkannten und peer-reviewed Journalen eingereicht, wovon bis dato bereits einer zur Publikation akzeptiert wurde.

Bibliography

[1] C. Abert, L. Exl, G. Selke, A. Drews, and T. Schrefl. “Numerical methods for the stray-field calculation: A comparison of recently developed algorithms”. In: *Journal of Magnetism and Magnetic Materials* 326 (2013), pp. 176–185.

[2] A. Aharoni. *Introduction to the Theory of Ferromagnetism*. Vol. 109. Clarendon Press, 2000.

[3] N. Aronszajn. “Theory of reproducing kernels”. In: *Transactions of the American Mathematical Society* 68 (1950), pp. 337–404. DOI: 10.1090/S0002-9947-1950-0051437-7.

[4] G. Bakir, J. Weston, B. Schölkopf, S. Thrun, and L. Saul. “Learning to Find Pre-Images”. In: *Advances in Neural Information Processing Systems, 449-456 (2004)* (Mar. 2004).

[5] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner. “Learning data-driven discretizations for partial differential equations”. In: *Proceedings of the National Academy of Sciences* 116.31 (2019), pp. 15344–15349.

[6] W. F. Brown. *Micromagnetics*. 18. Interscience Publishers, 1963.

[7] F. Bruckner, M. d’Aquino, C. Serpico, C. Abert, C. Vogler, and D. Suess. “Large scale finite-element simulation of micromagnetic thermal noise”. In: *Journal of Magnetism and Magnetic Materials* 475 (2019), pp. 408–414.

[8] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. “Neural ordinary differential equations”. In: *arXiv preprint arXiv:1806.07366* (2018).

[9] M. Coey and S. Parkin. *Handbook of Magnetism and Magnetic Materials*. Springer International Publishing, 2021. ISBN: 978-3-030-63210-6.

[10] C. Cortes, M. Mohri, and J. Weston. “A general regression framework for learning string-to-string mappings”. In: *Predicting Structured Data* (Jan. 2007).

[11] M d’Aquino, C Serpico, G Bertotti, T Schrefl, and I. Mayergoyz. “Spectral micromagnetic analysis of switching processes”. In: *Journal of Applied Physics* 105.7 (2009). https://doi.org/10.1063/1.3074227, p. 07D540.

[12] M. d’Aquino, C. Serpico, and G. Miano. “Geometrical integration of Landau–Lifshitz–Gilbert equation based on the mid-point rule”. In: *Journal of Computational Physics* 209.2 (2005), pp. 730–753.

[13] M. J. Donahue and D. G. Porter. “Oommf user’s guide, version 1.0, interagency report nistir 6376”. In: *National Institute of Standards and Technology* (1999).

[14] L. Exl. “A magnetostatic energy formula arising from the L2-orthogonal decomposition of the stray field”. In: *Journal of Mathematical Analysis and Applications* 467.1 (2018), pp. 230–237.

[15] L. Exl. “Tensor grid methods for micromagnetic simulations”. In: http://repositum.tuwien.ac.at/urn:nbn:at:at-ubtuw:1-73100. Vienna UT (thesis), 2014c.

[16] L. Exl, N. J. Mauser, S. Schaffer, T. Schrefl, and D. Suess. *Prediction of magnetization dynamics in a reduced dimensional feature space setting utilizing a low-rank kernel method*. 2021. arXiv: 2008.05986 [physics.comp-ph].

[17] L. Exl, N. J. Mauser, T. Schrefl, and D. Suess. "Learning time-stepping by nonlinear dimensionality reduction to predict magnetization dynamics". In: *Communications in Nonlinear Science and Numerical Simulation* 84 (2020), p. 105205. ISSN: 1007-5704. DOI: https://doi.org/10.1016/j.cnsns.2020.105205.

[18] L. Exl, N. J. Mauser, T. Schrefl, and D. Suess. "The extrapolated explicit midpoint scheme for variable order and step size controlled integration of the Landau-Lifschitz-Gilbert equation". In: *Journal of Computational Physics* (2017a). https://doi.org/10.1016/j.jcp.2017.06.005.

[19] J. Fischbacher, A. Kovacs, M. Gusenbauer, H. Oezelt, L. Exl, S. Bance, and T. Schrefl. "Micromagnetics of rare-earth efficient permanent magnets". In: *Journal of Physics D: Applied Physics* 51.19 (2018), p. 193002.

[20] C. Foias, G. R. Sell, and R. Temam. "Inertial manifolds for nonlinear evolutionary equations". In: *Journal of differential equations* 73.2 (1988), pp. 309–353.

[21] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[22] A. E. Hoerl and R. W. Kennard. "Ridge regression: applications to nonorthogonal problems". In: *Technometrics* 12.1 (1970), pp. 69–82.

[23] T. Hofmann, B. Schölkopf, and A. J. Smola. "Kernel methods in machine learning". In: *The annals of statistics* (2008), pp. 1171–1220.

[24] P. Honeine and C. Richard. "Preimage Problem in Kernel-Based Machine Learning". In: *Signal Processing Magazine, IEEE* 28 (Apr. 2011), pp. 77 –88. DOI: 10.1109/MSP.2010.939747.

[25] J. Jordan. *Introduction to autoencoders*. https://www.jeremyjordan.me/autoencoders/. May 2018.

[26] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler. "Deep Fluids: A Generative Network for Parameterized Fluid Simulations". In: *Computer Graphics Forum* 38.2 (2019), pp. 59–70. DOI: https://doi.org/10.1111/cgf.13619. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13619.

[27] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[28] D. G. Kleinbaum, K Dietz, M Gail, M. Klein, and M. Klein. *Logistic regression*. Springer, 2002.

[29] A. Kovacs, J. Fischbacher, H. Oezelt, M. Gusenbauer, L. Exl, F. Bruckner, D. Suess, and T. Schrefl. "Learning magnetization dynamics". In: *Journal of Magnetism and Magnetic Materials* 491 (2019), p. 165548. ISSN: 0304-8853. DOI: https://doi.org/10.1016/j.jmmm.2019.165548.

[30] H. Kronmueller. *General Micromagnetic Theory*. John Wiley & Sons, Ltd, 2007. ISBN: 9780470022184. DOI: 10.1002/9780470022184.hmm201.

[31] R. McMichael. *μMAG micromagnetic modeling activity group*. accessed 09-June-2021. URL: https://www.ctcms.nist.gov/~rdm/mumag.org.html.

[32] S. Mika, B. Schölkopf, A. Smola, K.-R. Müller, M. Scholz, and G. Rätsch. "Kernel PCA and De-Noising in Feature Spaces". In: 11 (Jan. 1999).

[33] J. E. Miltat and M. J. Donahue. “Numerical micromagnetics: Finite difference methods”. In: *Handbook of magnetism and advanced magnetic materials* (2007).

[34] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020.

[35] K. O’Shea and R. Nash. “An introduction to convolutional neural networks”. In: *arXiv preprint arXiv:1511.08458* (2015).

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[37] D. Porter and M. Donahue. “Standard Problems in Micromagnetics”. In: *Compendium On Electromagnetic Analysis-From Electrostatics To Photonics: Fundamentals And Applications For Physicists And Engineers (In 5 Volumes)* (2020), p. 285.

[38] M. Raissi, P. Perdikaris, and G. E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[39] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. “Contractive Auto-Encoders: Explicit Invariance During Feature Extraction”. In: Jan. 2011.

[40] S. Schaffer, N. J. Mauser, T. Schrefl, D. Suess, and L. Exl. “Machine learning methods for the prediction of micromagnetic magnetization dynamics”. In: *IEEE Transactions on Magnetics (accepted)* (2021). arXiv: 2103.09079 [physics.comp-ph].

[41] B. Schölkopf, A. Smola, and K.-R. Müller. “Kernel principal component analysis”. In: *International conference on artificial neural networks* (1997), pp. 583–588.

[42] T. Schrefl, G. Hrkac, S. Bance, D. Suess, O. Ertl, and J. Fidler. “Numerical Methods in Micromagnetics (Finite Element Method)”. In: *Handbook of Magnetism and Advanced Magnetic Materials*. https://doi.org/10.1002/9780470022184.hmm203. John Wiley & Sons, Ltd, 2007.

[43] B. Schölkopf, A. Smola, and K. Müller. “Nonlinear Component Analysis as a Kernel Eigenvalue Problem”. In: *Neural Computation* 10.5 (1998), pp. 1299–1319. DOI: 10.1162/089976698300017467.

[44] B. Schölkopf, R. Herbrich, A. Smola, and R. Williamson. “A Generalized Representer Theorem”. In: *Computational Learning Theory* 42 (June 2000).

[45] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. DOI: 10.1017/CBO9781107298019.

[46] J. Shawe-Taylor, N. Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[47] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos. “Tensor decomposition for signal processing and machine learning”. In: *IEEE Transactions on Signal Processing* 65.13 (2017), pp. 3551–3582.

[48] D. Suess, V. Tsiantos, T. Schrefl, J. Fidler, W. Scholz, H. Forster, R. Dittrich, and J. Miles. “Time resolved micromagnetics using a preconditioned time integration method”. In: *Journal of Magnetism and Magnetic Materials* 248.2 (2002), pp. 298–311.

[49] D. Suess, A. Bachleitner-Hofmann, A. Satz, H. Weitensfelder, C. Vogler, F. Bruckner, C. Abert, K. Prügl, J. Zimmer, C. Huber, et al. "Topologically protected vortex structures for low-noise magnetic sensors with high linear range". In: *Nature Electronics* 1.6 (2018), p. 362.

[50] M. Welling. *Kernel Principal Components Analysis*. URL: https://www.ics.uci.edu/~welling/classnotes/papers_class/Kernel-PCA.pdf.

[51] M. Welling. *Kernel ridge Regression*. URL: https://www.ics.uci.edu/~welling/classnotes/papers_class/Kernel-Ridge.pdf.

[52] J. Weston, O. Chapelle, A. Elisseeff, B. Schölkopf, and V. Vapnik. "Kernel Dependency Estimation." In: (Jan. 2002), pp. 873–880.

[53] C. K. Williams and M. Seeger. "Using the Nyström method to speed up kernel machines". In: *Advances in neural information processing systems*. 2001, pp. 682–688.

[54] S. Wold, K. Esbensen, and P. Geladi. "Principal component analysis". In: *Chemometrics and Intelligent Laboratory Systems* 2.1 (1987). Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists, pp. 37–52. ISSN: 0169-7439. DOI: https://doi.org/10.1016/0169-7439(87)80084-9.

[55] D. H. Wolpert. "The lack of a priori distinctions between learning algorithms". In: *Neural computation* 8.7 (1996), pp. 1341–1390.

[56] H. Xiong, M. Swamy, and M. O. Ahmad. "Optimizing the kernel in the empirical feature space". In: *IEEE transactions on neural networks* 16.2 (2005), pp. 460–474.