Department of Computer Science

Advances in Randomly-Weighted Neural Networks and Temporal Gaussian Processes

Alexander Grigorievskiy

Aalto University

DOCTORAL DISSERTATIONS

Aalto University publication series
DOCTORAL DISSERTATIONS 144/2019

Advances in Randomly-Weighted Neural Networks and Temporal Gaussian Processes

Alexander Grigorievskiy

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, at the public examination held at the lecture hall T2, Konemiehentie 2, Espoo (Finland) on 20th of September 2019 at noon.

Aalto University
School of Science
Department of Computer Science
Probabilistic Machine Learning (PML) group

Supervising professor
Professor Aki Vehtari, Aalto University, Finland

Thesis advisor
Emeritus Professor Juha Karhunen, Aalto University, Finland

Preliminary examiners
Professor Jonathan Rougier, University of Bristol, United Kingdom
Professor Andrew Beng Jin Teoh, Yonsei University, South Korea

Opponent
Professor Tommi Kärkkäinen, University of Jyväskylä, Finland

Aalto University publication series
DOCTORAL DISSERTATIONS 144/2019

ISBN 978-952-60-8670-5 (printed)
ISBN 978-952-60-8671-2 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)
http://urn.fi/URN:ISBN:978-952-60-8671-2

Unigrafia Oy
Helsinki 2019

Finland

NORDIC SWAN ECOLABEL
Printed matter
4041-0619

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author
Alexander Grigorievskiy

Name of the doctoral dissertation
Advances in Randomly-Weighted Neural Networks and Temporal Gaussian Processes

Publisher School of Science

Unit Department of Computer Science

Series Aalto University publication series DOCTORAL DISSERTATIONS 144/2019

Field of research Computer and Information Science

Manuscript submitted 31 July 2019 **Date of the defence** 20 September 2019

Permission for public defence granted (date) 27 June 2019 **Language** English

☐ **Monograph** ☒ **Article dissertation** ☐ **Essay dissertation**

Abstract

This dissertation consists of three main parts. In the first part, the existing methods of machine learning are applied to the environmental and astronomical datasets. The problems addressed in this part are the prediction of phosphorus concentration in the Pyhäjärvi lake (Finland) and the analysis of the correlation of geomagnetic storms with solar activity. For the first problem, several different models are built and the final accuracy is improved by variable selection and making an optimal ensemble. The second problem is solved by considering a correlation coefficient and estimating its uncertainty by the bootstrap method.

The second part of the dissertation is devoted to studying randomly-weighted neural networks or in more narrow terminology Extreme Learning Machines (ELM). Vanilla ELM is trained by ordinary linear regression. As a consequence, ELM has reasonable accuracy but its training is much faster than the training of other neural networks. In this dissertation ELM for time series forecasting is investigated. It is shown that Optimally Pruned ELM (OP-ELM) algorithm in combination with a certain prediction strategy is better than a baseline model for time series data from different domains. Besides, the general regression algorithm (Inc)-OP-ELM is proposed which is significantly faster than the original OP-ELM but has the same performance.

Finally, in the third part of the dissertation, the probabilistic models for time series data are studied. Two types of probabilistic time series models are considered: linear state-space models and temporal Gaussian processes (GP). The connections between them are studied and new Gaussian process covariance functions are derived. These new covariance functions correspond to state-space models which are popular in the literature. Temporal Gaussian processes can be converted to state-space form as well. It is shown that this conversion allows expressing the inference in temporal GPs as operations with block-tridiagonal matrices. These matrix operations can be computed in linear time with respect to the number of samples or in sub-linear time if parallel algorithms are utilized. Algorithms developed in this dissertation can serve as a basis for more complex models like spatio-temporal models and models with non-Gaussian likelihoods.

Keywords Randowmly-weighted neural networs, Extreme Learning Machines, Gaussian Processes, Time series prediction, State-space models

ISBN (printed) 978-952-60-8670-5 **ISBN (pdf)** 978-952-60-8671-2

ISSN (printed) 1799-4934 **ISSN (pdf)** 1799-4942

Location of publisher Helsinki **Location of printing** Helsinki **Year** 2019

Pages 202 **urn** http://urn.fi/URN:ISBN:978-952-60-8671-2

Preface

The research in this dissertation has been conducted at the department of Computer Science of Aalto University mostly during years 2013-2017. Final revising and polishing has been done during years 2018-2019. I started studies in the group of my instructor Prof. Amaury Lendasse and Prof. Olli Simula. I am very grateful to them for noticing me, providing opportunity to pursue PhD degree and to start the research career.

The role of two other supervisors Prof. Juha Karhunen and Prof. Aki Vehtari is also invaluable for this thesis. They have been providing general guidance and helping with practical matters. The department's secretary Minna Kauppila was helping a lot with organizing trips and dealing with the bureaucracy. I would like to thank the Finnish state, society and Aalto University for organizing excellent opportunities for curious minds. The Nokia grant was a nice extra encouragement.

I am grateful to Prof. Simo Särkkä for the ideas for the last paper, interesting teaching and collaboration. Also, I would like to thank Prof. Arno Solin for explaining me tricky details of state-space models. The experience of visiting the lab of Prof. Neil Lawrence at Sheffield University and at Amazon UK was very exiting from scientific and personal angles. I have learned a lot from the members of his team: Andreas, Zhenwen, Xavier, Alan. Visits to United Kingdom became possible because of the funding from *EIT Digital*.

Collaboration with the group of Prof. Maarit Käpylä has been very valuable; Thank you! I would also like to express appreciation my opponent Prof. Tommi Kärkkäinen and preexaminers: Prof. Jonathan Rougier and Prof. Andrew Beng Jin Teoh. Their comments has helped to make the dissertation much better.

I am very appreciated to my peer students and colleagues: Giancarlo, Jonathan, Nigul, Yoan, Mark, Emil, Luiza, Dusan, David, Kyunghyun, Anton, Frederik, Matthias, Abdulmelik, Juho, Dima, Ruslan. I have learned a lot because of you and enjoyed communicating with you.

No long term commitment can be completed without support from the family. I am deeply grateful to my mother Vera for encouraging me during many years. My, already passed away, father Valeriy thought me several important lessons. Finally, my dear wife Alexandra has shared happy and difficult moments during this period, so this thesis is the result of her efforts and patience as well. My daughters Anna and Sveta has added some joy to the process.

Espoo, August 6, 2019,

Alexander Grigorevskiy

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Alexander Grigorevskiy, Anton Akusok, Marjo Tarvainen, Anne-Mari Ventelä, and Amaury Lendasse. Practical Estimation of Missing Phosphorus Values in Pyhäjärvi Lake Data. In *Workshop on New Challenges in Neural Computation 2013*, Saarbrücken (Germany), Machine Learning Reports, Volume 2, ISSN: 1865-3960, pages 8-16, September 2013.

II E. K. J. Kilpua, N. Olspert, A. Grigorievskiy, M. J. Käpylä, E. I. Tanskanen, H. Miyahara,R. Kataoka, J. Pelt, and Y. D. Liu. Statistical Study of Strong and Extreme Geomagnetic Disturbances and Solar Cycle Characteristics. *The Astrophysical Journal*, Volume 806, Number 2, pages 272, June 2015.

III Alexander Grigorievskiy, Yoan Miche, Anne-Mari Ventelä, Eric Sèverin, and Amaury Lendasse. Long-term time series prediction using OP-ELM. *Neural Networks*, Volume 51, pages 50-56 (`https://doi.org/10.1016/j.neunet.2013.12.002`) , March 2014.

IV Alexander Grigorievskiy, Yoan Miche, Maarit Mantere, and Amaury Lendasse. Singular Value Decomposition Update and Its Application to (Inc)-OP-ELM. *Neurocomputing*, Volume 174, Part A, pages 99-108 (`https://doi.org/10.1016/j.neucom.2015.03.107`), January 2016.

V Alexander Grigorievskiy, Maarit Mantere, Anton Akusok, Emil Eirola and Amaury Lendasse. Forecasting the Outbursts of the Photometry Light Curve of Star V363 Lyr. In *International work-conference on Time Series (ITISE-2014)*, Granada (Spain), Proceedings of ITISE-2014, Volume 2, pages 520-531, I.S.B.N: 978-84-15814-97-4, June 2014.

VI Alexander Grigorievskiy and Juha Karhunen. Gaussian Process Kernels for Popular State-Space Time Series Models. In *International Joint Conference on Neural Networks (IJCNN 2016)*, Vancouver (Canada), pages 3354-3363, July 2016.

VII Alexander Grigorievskiy, Neil Lawrence, and Simo Särkkä. Parallelizable Sparse Inverse Formulation Gaussian Processes (SpInGP). In *In Proceedings of the Workshop on Machine Learning for Signal Processing (MLSP2017)*, Tokyo (Japan), September 2017.

Author's Contribution

Publication I: "Practical Estimation of Missing Phosphorus Values in Pyhäjärvi Lake Data"

Grigorevskiy was responsible for paper writing, doing experiments and participated in methods selection. Lendasse was a supervisor of the work and participated in methods selection. Other co-authors did dataset preparation, red the final manuscript and contributed with biological interpretation.

Publication II: "Statistical Study of Strong and Extreme Geomagnetic Disturbances and Solar Cycle Characteristics"

Grigorevskiy proposed to estimate correlation coefficients and their uncertainty by applying bootstrap method. He also wrote the code and performed the corresponding experiment. Other authors did phase shift analysis experiments, wrote the paper, and prepared the dataset.

Publication III: "Long-term time series prediction using OP-ELM"

Lendasse proposed the general idea of the paper and selected the datasets. Grigorevskiy implemented the algorithms, ran experiments and wrote the paper. Other co-authors participated with some practical advices and red the final manuscript.

Publication IV: "Singular Value Decomposition Update and Its Application to (Inc)-OP-ELM"

Grigorevskiy was the author of idea, implemented the algorithms, designed and

ran experiments and wrote the paper. Other co-authors evaluated feasibility of the approach, helped with some practical advices and experiments design. They also red the final manuscript.

Publication V: "Forecasting the Outbursts of the Photometry Light Curve of Star V363 Lyr"

Lendasse proposed the general idea of the paper, helped with methods selection and experiments design. Grigorevskiy contributed with methods selection and adaptation as well as experiments design. He also was responsible for paper writing, implementing the algorithms and running experiments.

Publication VI: "Gaussian Process Kernels for Popular State-Space Time Series Models"

Grigorevskiy was responsible for most of the work in this paper. Karhunen evaluated the feasibility of main ideas and helped with revising the final manuscript.

Publication VII: "Parallelizable Sparse Inverse Formulation Gaussian Processes (SpInGP)"

Särkkä proposed the main idea of the paper, gave practical advices and revised the final manuscript. Lawrence gave practical advices and ideas, he also helped with algorithms design. Grigorevskiy designed the experiments, implemented the algorithms and wrote the paper.

Other Publications

The following publications are not included in this dissertation.

VIII Hannes Nickisch, Arno Solin, Alexander Grigorievskiy. State Space Gaussian Processes with Non-Gaussian Likelihood. In *International Conference on Machine Learning (ICML 2018)* , Stockholm (Sweden), July 2018.

IX Nigul Olspert, Juri Lehtinen, Maaarit Käpylä, Jaan Pelt, Alexander Grigorievskiy. Estimating activity cycles with probabilistic methods II. The Mount Wilson Ca H&K data. *Astronomy and Astrophysics*, Volume 619, pages 1-20, October 2018.

List of Abbreviations

AIC Akaike Information Criterion. 19

ARD Automatic Relevance Determination. 19, 22, 41

ARIMA Auto Regressive Integrated Moving Average. 53, 54, 65–67, 93

BF Bayes Factor. 19

BIC Bayesian Information Criteria. 19

BLR Bayesian Linear Regression. 71–73

BMA Bayesian Model Averaging. 25

BO Bayesian Optimization. 17, 18

BTD block-tridiagonal. 84–88

ELM Extreme Learning Machine. 2, 3, 35–43, 45, 46, 53, 65, 91, 92

EOF Empirical Orthogonal Functions. 11

EP Expectation Propagation. 29

ERM Empirical Risk Minimization. 8, 14, 15, 26, 27

GCV Generalized Cross Validation. 16

GP Gaussian Process. 33, 67, 69–71, 74–77, 79–82, 85–87, 89, 90, 92, 93

GP-LVM Gaussian Process Latent Variable Model. 33

GPR Gaussian Process Regression. 18, 66, 67, 69, 71–73, 75, 80, 81, 85, 89

K-NN K-Nearest Neighbour. 32

KF Kalman Filter. 68, 71, 81–83, 89, 90, 92

LARS Least Angle Regression. 43–45

LLLM Local Linear Trend Model. 72, 76, 77

LLM Local Level Model. 72, 75, 77, 81

LOO leave-one-out. 15, 39, 40, 43, 50–52

LS-SVM Least-Squares Support Vector Machine. 58, 59

LS-SVR Least-Squares Support Vector Regression. 22

LSTM Long Short-Term Memory. 92

MAE Mean Absolute Error. 62–64

MAP maximum a posteriori. 68

MCMC Markov Chain Monte Carlo. 29, 30

MDL Minimum Description Length. 19

ML Machine Learning. 5, 6, 27

MLP Multi-layer Perceptron. 35

MRSR Multi-response Sparse Regression. 22, 43, 44, 51

MSE Mean Squared Error. 23, 58

NN Neural Network. 54

OLS Ordinary Least Squares. 44

OP-ELM Optimally-Pruned ELM. 43, 44, 50–52, 56, 58–60, 62, 63

PCA Principal Component Analysis. 9, 10, 12, 20, 33

PRESS PREdicted residual Sum of Squares error. 40, 43

RBFN Radial Basis Functions Network. 36

RF Random Forests. 24, 62–64

RNN Recurrent Neural Network. 66

RR Ridge Regression. 22

RTS Rauch-Tung-Striebel. 68, 81–83

SS model state-space model. 70, 71

STS structural time series. 66–68, 70, 75, 77, 78, 82

SVD Singular Value Decomposition. 6, 9–11, 37, 40, 41, 45–51

SVM Support Vector Machine. 15

SVR Support Vector Regression. 17, 22, 26

TROP-ELM Tikhonov Regularized Optimally-Pruned ELM. 43, 44, 50, 51, 62, 63

TS Time series. 12

TSP Time Series Prediction. 53, 65

VC-dimension Vapnik-Chevonenkis dimension. 38, 39

VI Variational Inference. 29

1. Introduction

1.1 Motivation and scope

The last three decades can be characterized by the exponential growth of digital technologies and computers. During this time, the computational power of computer processors, hard drive capacity, RAM size, has increased by several orders of magnitude. Mobile phones these days are more powerful than personal computers back in the 1990s. Large technological corporations have build data centers where the data from social networks of millions of users is stored and processed.

Communication channels have also expanded tremendously. Nowadays, in many countries, watching a high-quality video on a personal smartphone is a usual thing. There are many more examples of such rapid expansion of digitalization in other domains. For instance, in the scientific domain, genomic or astronomical data require huge storage capacity and processing power. This digitalization trend demands intelligent understanding and utilization of accumulated information. The field of science which deals with such tasks is called *Machine Learning*. It also became the most successful approach to the *Artificial Intelligence* today.

Machine Learning is a field which emerged as an intersection of three other fields: statistics, computer science and optimization. Its main purpose is to make the past data useful for humans. For examples, it addresses the problem of summarizing the previous data and extracting useful knowledge, using the past data to predict the future, and making optimal decisions and planning. In this dissertation, machine learning is used to solve real world problems. Besides, certain machine learning models are extended and compared.

The field of machine learning is very broad. It solves many different tasks. There exist plenty of models even for a single machine learning task. Models

can be classified in several different ways. For instance, one way to classify machine learning models is inherited from statistics. There is a *frequentist* and *Bayesian* approach to machine learning. Furthermore, models can be divided into probabilistic and deterministic. These and other model characteristics are described in Chapter 2. There are pros and cons of different models and in a practical situation, the most suitable model must be chosen. In this dissertation, several types of models from different categories have been considered.

The dissertation consists of three main parts. In the first part, the existing machine learning methods are applied to real world environmental and astronomical data. The focus of the second part is the randomly-weighted random networks which are also called *Extreme Learning Machines* and the third part deals with temporal Gaussian processes and state-space models. Approaches and methods in this dissertation have been mostly developed with time series modeling and forecasting tasks in mind. However, some algorithms e.g. (Inc)-OP-ELM are applicable more broadly, since it is a general purpose regression algorithm.

Time series data is ubiquitous in science and engineering. Sometimes it is beneficial to take into account the time dimensionality of the data even though the problem can be solved without doing it. For example, recommender systems can take into account the exact time of previous purchases to recommend new items to buy [Wu et al., 2017]. Temporal models can be used as building blocks in more complex tasks e.g. in spatio-temporal models, system identification, reinforcement learning. Hence, the improvements in time series models proposed in this thesis can be used in other domains as well.

In publication I the problem of missing values estimation for Pyhäjärvi lake (Finland) data has been considered. This problem has been addressed by two different tasks: regression task and missing values imputation task. These approaches have been compared and the optimal ensemble has been built. In publication II, the correlations between geomagnetic storms strengths and solar cycle activities have been studied. The significance of the observed correlations has been estimated by the *bootstrap* method. It has been shown that moderate storms correlate positively with solar activity while super storms have no such correlation.

Publications III and IV are dealing with Extreme Learning Machine (ELM) models. Vanilla ELM provides good generalization accuracy with simple and fast optimization method - least squares algorithm. In publication III the application of ELM for time series forecasting has been studied. It has been concluded that using ELM with DirRec prediction strategy provides higher accuracy than

the baseline method while keeping the training complexity on almost the same level. In publication IV the approach to reduce the overfitting of ELM has been proposed. The idea is based on using leave-one-out cross-validation (LOO) and *Singular Value Decomposition* update algorithm to compute it in a fast way. Publication V develops an approach to predict the photometric outbursts of the variable star using ELMs.

Finally, in publications VI and VII temporal Gaussian processes (GPs) and structural time series models are investigated. These types of models are advantageous because of their probabilistic nature, flexibility, interpretability, applicability to unevenly sampled data and linear inference time with respect to the number of data points. In publication VI connection between popular structural time series models and Gaussian processes are studied. New GP kernels which correspond to the popular state-space time series models have been derived, and the approach to their derivation has been elucidated.

In publication VII the way to parallelize the inference for temporal Gaussian process models have been considered. By expressing a temporal GP in a stochastic differential equation (SDE) form the inference time can be made linear with respect to the number of data points. However, by exploiting the sparseness of the inverse covariance matrix the problem can be reformulated in terms of computational primitives involving operations with sparse matrices. If these primitives are solved by subroutines which take advantage of parallelization then total computational time can be sublinear.

1.2 Structure of the disseration

The remainder of the dissertation consists of three chapters. Chapter 2 presents an overview and highlights certain parts of machine learning which are relevant for understanding the methods in other chapters. It also includes the results of publications I and II. Chapter 3 introduces extreme learning machines and summarizes publications III, IV and V. Chapter 4 presents the connections between state-space models and Gaussian process and describes publications VI and VII. Finally, chapter 5 includes a brief discussion on possible future directions and links to other related research.

2. Machine Learning Background and Examples of Applications

2.1 Chapter scope

In this chapter we overview the key concepts of Machine Learning (ML) which are relevant to this dissertation. Machine Learning is a field of science which studies knowledge extraction from the observed data. Machine Learning is a broad field which can be divided and classified in various ways depending on the view angle. In this chapter, several such ways are considered. They are relevant as an introduction to the subsequent parts of the dissertation. Since the size of the text is rather small, it can be considered as a *short introduction to machine learning*. Good general and comprehensive introductions can be found in the following books: [Bishop, 2006], [Hastie et al., 2009], [Murphy, 2012], [Barber, 2012].

Machine Learning has been a rapidly developing field in the past several decades. During this time the transition from modest *perceptron* model [Rosenblatt, 1958] to the large deep neural networks which are widely used in industry [van den Oord et al., 2016] has happened. Applications of machine learning are encountered in many industries: robotics, finance, forecasting, manufacturing, natural language processing , and so forth. The practical work of machine learning specialists has also changed noticeably in the past several years. The large industrially supported software toolboxes and cloud services have appeared [Abadi et al., 2016], [Chen et al., 2015] which automate many routine tasks.

There are several reasons for the fast progress in the filed. The amount of available data has been growing exponentially, and the need to analyze this data has been increasing as well. This need encouraged larger research efforts. Moreover, the computer hardware has been improved significantly: the processors' speed and hard drives storage capacity has been rapidly advancing.

Considering the large and diverse amount of available datasets, it is not surprising that there exist also many tasks which are possible to do with this data. First, there is a division between *supervised* and *unsupervised* learning. These are considered in Section 2.2. Then, in the same section, follows the introduction to *missing values* problem and the description of *Singular Value Decomposition (SVD)* which is widely used method in machine learning. In particular, it can be applied for missing values problem and for unsupervised learning.

Model selection is one of the central topics in machine learning. It is addressed in Section 2.3. There several different methods of model selection are described. They are also related to the types of models in use. For example, for probabilistic models, the *marginal likelihood (or evidence)* optimization can be used, while for non-probabilistic models *cross-validation* or *bootstrap* procedure are usually exploited.

Theoretical foundations of machine learning lie in statistics, computer science, and optimization. In statistics there is a fundamental division between *Bayesian* and *frequentist* frameworks. This division spreads also to machine learning. In Section 2.4, these two approaches are described and compared. In this dissertation, both approaches have been successfully applied.

Finally, in Section 2.5 other ways to discriminate models are defined. For instance, there exist *generative* and *discriminative* models for classification. Models can also be divided in *parametric* and *non-parametric*. *Latent variable models* as a subclass of unsupervised models are also described.

2.2 Machine Learning tasks

2.2.1 Terminology and data modeling workflow

In this section, the basic terminology used in machine learning is explained and the data modeling workflow is briefly described. Machine Learning (ML) is a relatively new discipline, which is formed in the intersection of other disciplines, hence the terminology is not well settled yet. Moreover, in the field of Statistics there are two approaches to modeling: Bayesian and frequentist (Sec. 2.4), so their terminologies and conceptual differences are also transferred to Machine Learning.

The goal of this section is not to give an exhaustive overview of statistical modeling, but to give some high-level framework and intuitions which may be

relevant for the readers. The diagram presented on the Figure 2.1 is quite rough and simple, however it presents the main blocks of the modeling and relations between them. Hopefully, it allows to understand the terminology better. The deficiencies of this diagram for the frequentist statistical modeling are also mentioned below.

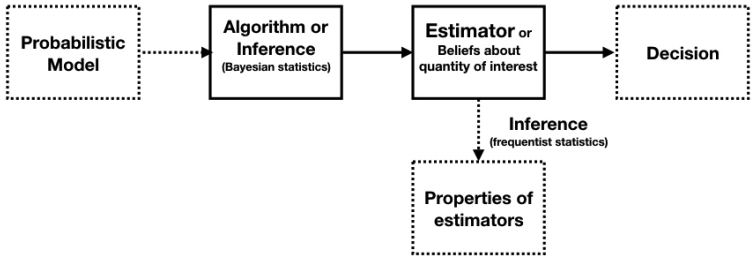


Figure 2.1. High-level diagram of the statistical modeling, inference and decision making.

The data modeling may start from the *probabilistic model*. This model describes the relationships between observed variables in the data and may also include unobserved *latent* variables. In the Bayesian approach to statistics the probabilistic model is a compulsory component, however, in frequentist approach, the probabilistic model is not a prerequisite and one can directly design algorithms for computing values of interest (estimators). If the probabilistic model is provided it may also help to build better algorithms to compute the desired quantities. If the probabilistic model of the data is not provided some assumptions (e.g about moments of data distribution) are still necessary to prove estimator's optimality regarding a desired property.

In the Bayesian statistics, the procedure of computing beliefs about quantities of interest is called *inference*. Inference typically means estimating the probabilistic properties of these values. In the Bayesian statistics, probabilistic properties are typically computed simultaneously with computing the values itself (during posterior computation Sec.2.4.3). However, in frequentist statistics computing the values (estimators) is done separately by some *algorithm*. The inference or calculation of the probabilistic properties of estimators is done separately and denoted by the most bottom rectangle.

In the Bayesian statistics, the inference is separated from the decision making. First, the inference is performed, and after that, the optimal actions can be computed provided the loss function is available (Sec. 2.4). For instance, an e-mail can be classified as being spam or not spam if the corresponding probabilities and a loss function are available. In the frequentist statistics, the decision theory is not clearly decoupled from the algorithms which compute the estimators. The frequentist decision theory provides a way to design algorithm:

Empirical Risk Minimization (ERM) framework (Sec. 2.3.2). So, the Decision block on the diagram on Figure 2.1 is not applicable to the frequentist statistical approach.

The description in this section is very brief. For example, there should be backward connections (or cycles) on the diagram because, in practice, the results obtained on one step of data analysis workflow often allow to improve the previous step. For the subsequent reading it beneficial to distinguish and understand the terms like *model*, *algorithm*, *inference*, *decision*.

2.2.2 Supervised learning

In the supervised learning scenario we have a dataset:

$$\begin{aligned} &D = \left\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right\}_{i=1}^{N}, \\ &\text{where:} \quad \mathbf{x}_i \in \mathbb{R}^D, \quad \text{and} \quad \mathbf{y}_i \in \mathbb{S}. \end{aligned} \tag{2.1}$$

It consist of input variables $\mathbf{x}^{(i)}$ and output variables $\mathbf{y}^{(i)}$. The task is to learn the dependency between inputs and outputs which predicts the corresponding output $\mathbf{y}$ for any new input observation $\mathbf{x}$. The output set $\mathbb{S}$ can be almost any set, but the two most typical cases in supervised learning are *classification* and *regression*. In the classification task the output variables are $\mathbf{y}^{(i)} \in \{C_1 \cdots C_H\}$. So, the set $\mathbb{S}$ is a finite set of distinct classes[1]. In the regression task $\mathbf{y}_i \in \mathbb{R}^H$. Hence, the output takes real, possibly multidimensional, values.

The measurements in the dataset D can be noisy, and the methods which learn the dependency may take this into account. One way to incorporate this noisiness into the mapping between inputs and outputs is to build a probabilistic model:

$$p(\mathbf{y}, \mathbf{x}|\boldsymbol{\theta}) = p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) p(\mathbf{x}|\boldsymbol{\theta})$$

Thus, the mapping can be expressed in terms of probabilities and $\boldsymbol{\theta}$ is the vector of parameters which defines a mapping and which is learned. Probabilistic models are mentioned in the Section 2.5.1.

2.2.3 Unsupervised learning

In the unsupervised learning the dataset consist only of one set of variables:

[1] Sometimes this set is called *categorical* values, which means that there is no order between values

$$D = \left\{\mathbf{x}^{(i)}\right\}_{i=1}^{N}, \quad \text{where:} \quad \mathbf{x}_i \in \mathbb{R}^D \tag{2.2}$$

The purpose of this task is to find some hidden structure or pattern in the data. For instance, in *clustering* the aim is to map each data point to some cluster C_k selected from predefined set of clusters $\{C_1 \cdots C_H\}$. Thus, the clustering problem is similar to the classification, except that clustering is completely unsupervised. The assumption is that the points in one cluster are close to each other in some sense. In the probabilistic clustering the values of interest are probabilities $p(C_k|\mathbf{x}_i)\ \forall k,i$.

In contrast to clustering where the data points are mapped to a finite set of clusters $\mathbb{R}^D \rightarrow \{C_1 \cdots C_H\}$, there is a different approach where the inputs are mapped to a *latent space*. The latent space is a real space of dimensionality L, so in *latent variable modeling* we look for a mapping: $(\mathbf{x}_i \in \mathbb{R}^D) \rightarrow (\mathbf{z}_i \in \mathbb{R}^L)$. In the probabilistic view the probabilities $p(\mathbf{z}_i|\mathbf{x}_i)$ are of interest. The detailed overview of latent variable modeling can be found in [Lee and Verleysen, 2007]. Some novel models and algorithms can be found e.g. in [Kingma and Welling, 2013], [Lawrence, 2012].

The most known latent variable model is Principal Component Analysis (PCA) [Jolliffe, 1986]. Usually, PCA is thought more like an algorithm, and several different optimization criteria lead to the same algorithm. For example there are: direction of largest data variance formulation, minimum reconstruction error formulation, and others [Bishop, 2006, Chap. 12]. One formulation is the latent variable model formulation:

$$\mathbf{x}^{(i)} = W\mathbf{z}^{(i)} + \boldsymbol{\mu} + \epsilon \tag{2.3}$$

It is assumed that each data sample $\mathbf{x}^{(i)} \in \mathbb{R}^D$ is the sum of some mean vector $\boldsymbol{\mu} \in \mathbb{R}^D$, noise $\epsilon \sim \mathcal{N}(0, \sigma^2\boldsymbol{I})$, and a linear transformation of a lower dimensional latent vector $\mathbf{z}^{(i)} \in \mathbb{R}^L,\ L < D$.

Singular value decomposition

The PCA algorithm solves the task of finding all the unknown parameters and latent variables in Eq. (2.3). The computationally efficient and stable way to compute PCA is via Singular Value Decomposition (SVD). The data matrix $\overset{(N\times D)}{\boldsymbol{X}}$ where samples are represented row-wise is decomposed as:

$$\overset{(N\times D)}{\boldsymbol{X}} = \overset{(N\times D)}{\boldsymbol{U}}\ \overset{(D\times D)}{\boldsymbol{S}}\ \overset{(D\times D)}{\boldsymbol{V}^\top} \tag{2.4}$$

Thus, the matrix is decomposed into three matrices, two of which: $\boldsymbol{U}$ and $\boldsymbol{V}$ are *orthogonal* and the matrix in the middle - $\boldsymbol{S}$ is diagonal with positive elements sorted in a decreasing order. There are many useful properties of SVD [Golub and Van Loan, 1996], in particular, the optimal low-rank reconstruction property. To be more precise, only the L first elements of the matrix $\boldsymbol{S}$ are left intact and the rest are nullified, rendering the decomposition an approximation. The approximation matrix $\boldsymbol{U}\boldsymbol{S}_L\boldsymbol{V}^\top$ has the rank L and has minimum reconstruction error with respect to Frobenius-norm among all the matrices of rank L. SVD is also useful in computing the pseudo-inverse and solving least-squares problems.

Algorithm 1: PCA through SVD

Input : 1) Dimensionality of latent space L
2) Dataset matrix $\overset{(N\times D)}{\boldsymbol{X}}$ where samples are rows

Output: Latent variable matrix $\overset{(N\times L)}{\boldsymbol{Z}}$, projection matrix $\boldsymbol{W}$.

1 $\bar{\boldsymbol{X}} = (\boldsymbol{I}_N - \boldsymbol{1}\boldsymbol{1}^\top)\boldsymbol{X}$; // Subtract the mean from the data. Equivalent to estimating $\boldsymbol{\mu}$ in 2.3 as data mean. Denote the resulting matrix as $\bar{\boldsymbol{X}}$

2 $\bar{\boldsymbol{X}} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$; // do SVD of $\bar{\boldsymbol{X}}$

3 $\bar{\boldsymbol{V}} = [\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_L]$; // Take only first L columns of matrix $\boldsymbol{V}$

4 $\boldsymbol{Z} = \bar{\boldsymbol{X}}\bar{\boldsymbol{V}}$; // Compute the latent variables. Consequently, the projection matrix $\boldsymbol{W}$ in the problem formulation becomes $\boldsymbol{W} = \bar{\boldsymbol{V}}^\top$

/* **Comment:** It can be shown that PCA through SVD is equivalent to a more standard formulation when latent space is obtained by projecting the datapoints on the empirical covariance matrix of the data */

The algorithm for solving PCA problem via the SVD is shown above. This solution is mathematically equivalent to a more standard PCA algorithm where the projection directions (columns of the matrix $\boldsymbol{W}$) are obtained as eigenvectors of the empirical covariance matrix of the data.

2.2.4 Missing values problem

There are other types of tasks which are encountered in machine learning besides supervised and unsupervised learning. In the famous NetFlix competition [Bennett et al., 2007] the task was to infer how would certain users grade movies which they have not seen before. This task can be represented as a matrix where columns correspond to users and rows to movies. Some values of the matrix are filled - the users explicitly evaluated the movies. There are many empty values which correspond to the movies not evaluated by users. This

problem is called *Missing Values Problem*.

Table 2.1. Scheme of a supervised learning problem represented as a missing value problem.

$\boldsymbol{X}$	y

Also, supervised learning problem with real inputs and outputs can be written as a matrix completion problem by concatenating inputs and outputs in one matrix as shown in Table 2.1. The missing value problem may seem like some intermediate task between supervised and unsupervised tasks. On one hand, we need to estimate missing values which is more similar to the supervised task. On the other hand, if we devise an algorithm which is able to perform latent variable model with missing data, then projecting back from the latent space we can fill the missing data.

The latter approach has a wide adoption in the field of climate research where the missing values problem has been addressed by the Empirical Orthogonal Functions (EOF) method. This simple method is based on SVD. It has been used in Publication I and is presented in Algorithm 2.

Algorithm 2: Empirical Orthogonal Functions

Input : 1) Incomplete matrix $\boldsymbol{X} \in \mathbb{R}^{M,N}$

2) $K \in [1..D]$ - hyperparameter which controls the level of denoising. Larger values correspond to larger denoising.

Output: Complete matrix $\boldsymbol{X}$

1 Make initial imputation $\boldsymbol{X}^0$, for example, impute column means;
2 i = 0 (iteration number);
3 **repeat**
4 Perform SVD: $\boldsymbol{X}^i = \boldsymbol{U}^i \boldsymbol{D}^i (\boldsymbol{V}^i)^T$ to obtain $\boldsymbol{U}^i, \boldsymbol{D}^i$ and $(\boldsymbol{V}^i)^T$;
5 Nullify K smallest singular values of $\boldsymbol{D}^i$. Denote this modified matrix as $\boldsymbol{D}_0^i$;
6 Do inverse transformation: $\boldsymbol{X}_0^i = \boldsymbol{U}^i \boldsymbol{D}_0^i (\boldsymbol{V}^i)^T$;
7 Restore known values exactly: $known(\boldsymbol{X}_0^i) = known(\boldsymbol{X}^0)$; `// known denotes the values which are not missing in the original matrix`
8 i = i + 1 (iteration number);
9 **until** *Convergence*;

A comprehensive description of the missing values problem and its solutions are given in [Little and Rubin, 2002]. For instance, the approach which is used

for the probabilistic models is based on the EM-algorithm. The missing values are treated as extra latent variables and marginalized out during the E-step.

2.2.5 Time series prediction

Another common machine learning task is Time series (TS) prediction and modeling. Time series is a sequential data where the order of samples plays an important role. This type of data is frequently encountered in the real world. For instance, economic data, weather data, video data are the examples where the sequential modeling is essential. TS models are considered in detail in Chapter 3

$$\begin{aligned} \mathbf{z}_k &= f(\mathbf{z}_{k-1}) + \boldsymbol{\nu}; \qquad \boldsymbol{\nu} - \text{state noise} \\ \mathbf{y}_k &= g(\mathbf{z}_k) + \boldsymbol{\varepsilon}; \qquad \boldsymbol{\varepsilon} - \text{measurement noise} \end{aligned} \tag{2.5}$$

Quite general type of time series models is demonstrated in Eq. (2.5). It consists of two type of variables: $\mathbf{z}_k$ and $\mathbf{y}_k$. The first component models the latent/non-observed process. The second component models the observations of the time series and is linked with the first component. There are also noise terms which allow taking into account the uncertainty. Uncertainty is important for time series prediction because it facilitates the decision making based on future observations.

2.3 Model evaluation, selection and combination

Regardless of the task solved by a machine learning method there arises the problem of *model selection* and/or *model combination*. For example, in PCA, described previously in Sec. 2.2.3, one needs to choose the number of latent components. As another example, consider the supervised learning problem. We can select several different models for predicting the output given an input: $\mathbf{y}^\star = M_1(\mathbf{x}^\star)$, $\mathbf{y}^\star = M_2(\mathbf{x}^\star)$,.... Instead of selecting the best model, we can take as the output an average of the outputs of all the models. This is an example of *model combination* or *model averaging*. The basics of the model selection are described and the model combination is only briefly mentioned, because it has only been used in one publication in this dissertation.

The need for using model selection and/or averaging is often dictated by the various modeling goals. One of such goals is the *interpretsability* of the model. In this case, the human expert is supposed to understand the decisions produced by the model. Usually, interpretability implies that the structure of the model is simple and that size of the model is small in terms of the number

of parameters. *Variable selection* is a type of model selection which selects only a small number of input variables used in the model and ignores all the rest. Performance considerations can also motivate variable selection. Alternatively, if the focus is limited to only predictive accuracy, we would rather consider the model combination. Its predictive power is better than the predictive powers of individual models [Bishop, 2006, Ch. 14].

Model selection is also directly related to *model complexity*. The best performing model has an optimal complexity. If an overly complex model is used in a certain task, *overfitting* occurs. In contrast, *underfitting* occurs if the model is overly small. For example, by increasing the number of layers in a neural network, the number of parameters grows as well as the model complexity. Thus, it is important to optimally choose the model complexity.

For a given machine learning problem the choice of the class of methods to be used is stipulated by many criteria, among which are the researcher's familiarity with a particular method class, ease of optimization, training and prediction speed and ease of uncertainty estimation. For instance, given a dataset for supervised learning, possible method classes include (deep) neural networks, decision trees, Gaussian processes, and others. Once the method class is selected, the particular method usually depends on several *hyperparameters*. For Gaussian processes, the hyperparameters are the kernel type and parameters of the kernel. For a neural network, those are learning rate, the number of hidden layers, their types, and the number of neurons of each layer. Usually, a model implies or assumes a certain learning algorithm to train model parameters, but the hyperparameters selection is usually the task of a researcher.

Model class selection and model hyperparameters selection are introduced as two distinct steps, however, they both are usually referred as *model selection*. The reason is that the computational approaches for performing these two steps are the same. They are considered further in this section.

The two main criteria which are exploited in hyperparameters selection (or model selection) are *model predictive performance* and *marginal likelihood (evidence)* [Bishop, 2006, Ch. 3]. Each of these two criteria can not be applicable to all machine learning tasks and models. The model predictive performance criteria is applicable to the supervised machine learning tasks. In this task, the goal is to optimize the model predictions for the previously unseen data. The synonym term for model predictive performance is *estimation of generalization error* (next section). Generalization error criteria can be exploited in two ways. Firstly, the model error computed on the training data can be penalized in a certain way to take into account possible *overfitting* on the training data. The

second way is to evaluate the model predictions on samples artificially excluded from the training set. This technique is called cross-validation and is discussed in Section 2.3.4

For the *probabilistic models* the second criteria - *evidence* for hyperparameters selection is applicable. It is discussed further in the section 2.3.7.

2.3.1 Generalization error

In machine learning we are interested not only in how our model performs on the training data but also, more importantly, in its performance on previously unseen data. Consider, for example a supervised learning problem for which we have an available training dataset $D = \left\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right\}_{i=1}^{N}$. The goal is to build a model which predicts a new output $\mathbf{y}^{\star}$ given the new input point $\mathbf{x}^{\star}$. Imagine that a model is built, which predicts an output $\mathbf{y}_{\star} = M(\boldsymbol{x}^{\star})$[2]. To be able to evaluate the "quality" of prediction, a *loss function* $L(\mathbf{y}_{true}, \mathbf{y}_{\star})$ has to be defined. It receives two arguments: the true output from the unknown data generation model, and the output predicted by the model. The loss function computes the discrepancy between the true and predicted outputs.

The goal of the supervised learning is to minimize the *generalization error*. The generalization error is defined as:

$$E_{gen} = E_{p(\mathbf{x},\mathbf{y})}[L(\mathbf{y}, \mathbf{y}^{\star})] = E_{p(\mathbf{x},\mathbf{y})}[L(\mathbf{y}, M(\mathbf{x}))]. \tag{2.6}$$

This is the average of the loss function across the true data generative distribution $p(\mathbf{x},\mathbf{y})$. In practise, the true data generative distribution is not known. If we insert instead the empirical distribution $p_{emp}(\mathbf{x},\mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N}\delta[\mathbf{x} = \mathbf{x}^{(i)}, \mathbf{y} = \mathbf{y}^{(i)}]$ (which is the training data) in this formula, we obtain the training error, not generalization error:

$$E_{train} = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{y}^{(i)}, M(\mathbf{x}^{(i)}))]. \tag{2.7}$$

2.3.2 Empirical risk minimization

Actually, the equation (2.7) can be used to train the model with respect to the parameters of M. This approach is called Empirical Risk Minimization (ERM), and empirical risk is exactly $R_{emp}(D, M) = E_{train}$. Typically, the empirical risk is regularized with some measure of the complexity of the model M, so the

[2]The output can also be a probability distribution.

function to optimize becomes:

$$R_{emp}(D,M) + \lambda C(M) \tag{2.8}$$

The Support Vector Machine (SVM) model in (2.21) is an example of ERM framework. Empirical Risk is monotonically connected with the generalization error, but is not equal to it [Murphy, 2012, p. 206]. Therefore, it can also be used as a criteria for model selection, but can not be used to measure the uncertainty of predictions. Finally, if the loss in (2.7) is the negative log-likelihood, then complexity penalty (2.8) can be interpreted as log prior, and the Empirical Risk Minimization is equivalent to maximum (MAP) estimate.

2.3.3 Covariance penalties

For the sake of completeness it is worth to mention another class of methods for generalization error estimation. This class is called *covariance penalties* [Efron and Hastie, 2016, Ch. 12]. The expressions for generalization error estimation are similar to ERM objective, but they estimate the true generalization error. This class include *Mallows' C_p* estimate and Stein's Unbiased Risk Estimator (SURE).

2.3.4 Cross-Validation

To obtain the estimation of the generalization error, we need to insert the empirical distribution of the data, which was not used to train the model, into the Eq. (2.6). This technique to approximate the generalization error is called *cross-validation*.

In practice, the available dataset D is divided into K parts $D_1, D_2, \cdots D_K$. Each of this parts in turn is excluded from the training data set, and used only for estimation of the generalization error $E_{gen}^{cv_k}$. To obtain the final estimation all these errors are averaged:

$$E_{gen}^{empirical} = \frac{1}{K} \sum_{k=1}^{K} E_{gen}^{cv_k}. \tag{2.9}$$

This expression is used to estimate the generation error achieved by the model M. If there is a need to evaluate several models, then the generalization errors, corresponding to different models are compared and the model with minimal error is selected.

In the ultimate case the number of dataset partitions can be equal to the number of data points $K = N$. This is called leave-one-out (LOO) cross-validation.

Leave-one-out cross validation has a computational advantages in linear models, where it is called PRESS (Prediction Sum of Squares) statistic or Generalized Cross Validation (GCV) [Allen, 1974], [Golub et al., 1979].

Note, that this estimation of generalization error is used only for model selection, and should not be used as the final estimation of generalization error achieved by the method. Since the validation data is taken into account during the model selection, the error estimate in Eq. (2.9) is lower then it would have been on completely unseen data. Hence, to estimate the true generalization error of the machine learning algorithm the *test data* is taken apart from the available data. This test data is not used in the cross-validation process but used only for the evaluation of generalization error on the empirical distribution of test data.

2.3.5 Bootstrap procedure

There is an alternative method to estimate the generalization error. It is called *bootstrap* [Hastie et al., 2009, p. 249]. Its performance is almost the same as the performance of cross-validation [Hastie et al., 2009, p. 254], but the usage is more difficult. Bootstrap is the method to estimate any statistic (e.g. confidence interval) in a frequentist statistical approach (see Sec. 2.4.1). So, it can also be used to estimate the generalization error.

The first step in the bootstrap method is the generation of *bootstrap samples*. The bootstrap sample is a sample of the same size as the length of the data. This sample is generated with replacement from the data. After many bootstrap samples are generated, the generalization error is computed similarly to the cross-validation by comparing true and predicted values. Because of the need to produce an unbiased estimate of generalization error, the actual formulas are more difficult and can be found in [Hastie et al., 2009, p. 249]. This unbiased procedure is called *Bootstrap .632+*.

2.3.6 Hyperparameters search methods

The previous section introduced the criterion which is, in fact, the approximation of the generalization error. This criterion should be optimized in order to find good values of hyperparameters. There are several methods which are applied for hyperparameter optimization in practice.

The simplest algorithm which has has been actively used for hyperparameters optimization is *grid search*. In the grid search a range of values is defined for each individual hyperparameter. Usually, these values are selected to be

equidistant. Hence, the grid is defined in the space of all hyperparameters. Then the estimation of generalization error is done at every grid point and the optimal value with the smallest generalization error is selected. Grid search can be very time consuming if the hyperparameter space is high dimensional. In this case, some alternative strategies should be used.

There are many other strategies for hyperparameter optimization. In particular, the simple heuristic strategy of [Cherkassky and Ma, 2004] can be applied for kernel methods. In this work three hyperparameters of Support Vector Regression (SVR) are estimated on the basis of relatively simple statistical arguments and no optimization is needed.

In Publication I, the *pattern search* of [Momma and Bennett, 2002] has been used. This is a different method to avoid the heavy computations of grid search. Pattern search is a local search method in the space of hyperparameters. It performs a coordinate-wise descent on a grid . The grid is defined similarly as in grid search method. All hyperparameters are considered at each iteration and the one is selected which minimizes the objective function (generalization error) the most. If there are no hyperparameters which minimize the function then the grid step is divided by two and the process continues. Since it is a local minimization approach, the initial values of hyperparameters has to be appropriately chosen. Multiple restarts can be done as well. The method of [Cherkassky and Ma, 2004] is suitable for this initialization as well as initialization on a rough grid.

In the Deep Learning literature, the gradient methods to tune hyperparameters similarly to the model parameters has been proposed [Luketina et al., 2015]. The gradients of hyperparameters are computed on a separate validation set and alternate with the regular parameters update.

There is also a *random search* of hyperparameters proposed in [Bergstra and Bengio, 2012]. It addresses the inefficiency of grid search in high dimensional spaces. If the objective function is not changing significantly along some dimensions then the grid search is wasting computational resources to evaluate all grid points in this dimension. Random search generates random combinations of hyperparameters and for each point in hyperparameter space values in each dimension vary. Hence, the exploratory power of the random search is higher than the one of the grid search.

Recently, the Bayesian Optimization (BO) has become a popular method of hyperparameter optimization. Recent review of BO can be found in [Shahriari et al., 2016] and references therein. Bayesian optimization is a general method of optimizing complex non-convex functions $f(\mathbf{x})$, $\mathbf{x} \in \mathbb{X}$. It can be applied not

only to hyperparameters optimization but to many other optimization problems in science and engineering.

The distinctive property of BO from other global optimization algorithms is that BO is more suitable for optimizing functions which are costly to evaluate. The reason for this superiority is that in Bayesian optimization there is a probabilistic model of the optimization function. This model takes into account the uncertainty of underlying function and hence naturally balances the exportation/exploitation trade-off. There are several existing Bayesian optimization software packages e.g. [authors, 2016], [Bergstra et al., 2013], [Li et al., 2017]. In spite of that, BO is still an area of active research. The recent developments of BO algorithms related to hyperparameter optimization are [Klein et al., 2017], [Klein et al., 2016].

2.3.7 Marginal likelihood (evidence) optimization

Instead of doing a cross-validation, for the probabilistic models it is possible to find hyperparameters by optimizing *marginal likelihood* (or *evidence*). This procedure is called Maximum Likelihood Type-II. More details on probabilistic models are given in Section 2.5.1. As an example of probabilistic model for regression consider Gaussian Process Regression (GPR). Detailed introduction to GPR is presented in Chapter 4. In GPR the outputs $\mathbf{y}$ are modeled as noisy versions of random function outputs, and the prior of this random function is a Gaussain process prior:

$$y_i = f(\mathbf{x}_i) + \epsilon_i, \quad \epsilon_i \sim \mathbb{N}(0, \sigma_n^2) \tag{2.10}$$

$$f(\cdot) \sim \mathbb{GP}(0, K(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta})) \tag{2.11}$$

.

Often the covariance function is taken to be exponentiated quadratic:

$$K(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta}) = \sigma^2 \exp\left\{ -\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2} \right\} \tag{2.12}$$

.

The covariance matrix $K(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta})$ in this expression depends on kernel parameters which are hyperparameters in our definition. Another hyperparameter in GPR is the noise variance σ_n^2. This model is a probabilistic non-parametric model (see Sec. 2.5.4). The function f serves as a latent function here and is not observed explicitly. For the observed data it serves as a link between an input and output data. Therefore, the probabilistic model is the following:

$p(\mathbf{y},\mathbf{f}|X) = p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|X,\boldsymbol{\theta})$. If we integrate out the unobserved latent function values $\mathbf{f}$ we obtain:

$$p(\mathbf{y}|X,\boldsymbol{\theta}) = \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|X,\boldsymbol{\theta})d\mathbf{f} \tag{2.13}$$

.

The expression $p(\mathbf{y}|X,\boldsymbol{\theta})$[3] depends only on input, output data and hyperparameters. Hence it can be optimized with respect to hyperparameters. This procedure is called *Evidence framework* or *Maximum Likelihood Type-II* [Murphy, 2012, Ch. 13]. In contrast to the cross-validation procedure, the evidence can be optimized by the gradient methods (e.g. conjugate gradients) and there is no need to explore the whole space. The evidence is regularly not a convex function, so the multiple restarts are often done in order to find a better minimum.

In the same framework, it is also possible to do a variable selection. Instead of a single parameter ℓ of the covariance function we should take a set $\ell_1 \cdots \ell_K$ each of which affects only one dimension of the input. If after an optimization, one parameter becomes very large, then it means that the corresponding input dimension is switched off. This method is called Automatic Relevance Determination (ARD).

There is approximation to Marginal Likelihood which is called Bayesian Information Criteria (BIC) [Schwarz, 1978]. It can be used for the cases when marginal likelihood is difficult to compute but the log-likelihood can be computed:

$$BIC = \log p(D|\boldsymbol{\theta}) - \frac{dof\{\boldsymbol{\theta}\}}{2}\log N \tag{2.14}$$

.

Here the dof denotes *degrees of freedom* of the model parameters. BIC has a form of penalized likelihood where an extra complexity is penalized by the second term. There exist other model selection criteria which also take the form of penalized likelihood, however, they are not approximations to the marginal likelihood. Such examples are Akaike Information Criterion (AIC) [Akaike, 1973], Minimum Description Length (MDL) [Rissanen, 1985] and the ones mentioned in the Sec. 2.3.3.

If there are only two models, then the ratio between their marginal likelihoods can be calculated. This ratio is called Bayes Factor (BF) and can be viewed as an analog of the p-value in frequentist statistics. BF allows choosing one model based on this ratio [Murphy, 2012, Ch. 5]

[3] actually, it is also conditioned on the model structure, but we ignore it for brevity

2.3.8 Variable selection

Variable selection is also a type of model selection. There are several motivations to do a variable selection. Variable selection can make a model more robust towards noise. This is achieved because noisy or redundant variables are discarded from the model. The second motivation is the increase of model interpretability. Often an end user wants to see only several of the most important variables which the value of interest depends on. Another incentive to do a variable selection is the computational gain. Many models in machine learning have a superlinear e.g. polynomial dependency in the number of variables, so decreasing the number of variables may provide significant computational advantages. Thus, the variable selection is a very common model selection procedure in machine learning.

Variable selection is a subclass of dimensionality reduction methods. Besides variable selection, there is also *variable extraction*, where new variables are built by transforming the original variables. The most prominent example of variable extraction is PCA (Section 2.2.3). Here we describe only variable selection methods with the focus on those used in the dissertation.

Variable selection methods can be divided into several method classes [Guyon and Elisseeff, 2003]. For instance, there are supervised and unsupervised variable selection. Another way to divide methods is onto *filter*, *wrapper* and *embedded* methods. Here, the supervised class is considered and basic description of a filter, wrapper and embedded methods is provided. Filter and wrapper approaches have been used in Publication I.

The filter approach to variable selection is completely independent of the predictive model. Filter methods optimize some external criterion and select the subset of variables which correspond to the optimum. The diagram of the approach is shown on the Figure 2.2a. At first, all the variables are transferred into the filter method, then selected variables are conveyed into the predictive model. Thus, the variable selection is completely independent of the predictive model.

In Publication I the particular filter method *Delta Test* [Eirola et al., 2014], [Guillén et al., 2008] has been applied. Delta test is subclass of a more general method called *Gamma Test*. Both of these methods are non-parametric methods of the noise variance estimation. The main assumption is that the output variable is a continuous function of the input variables:

$$y = f(\mathbf{x}) + \epsilon.$$

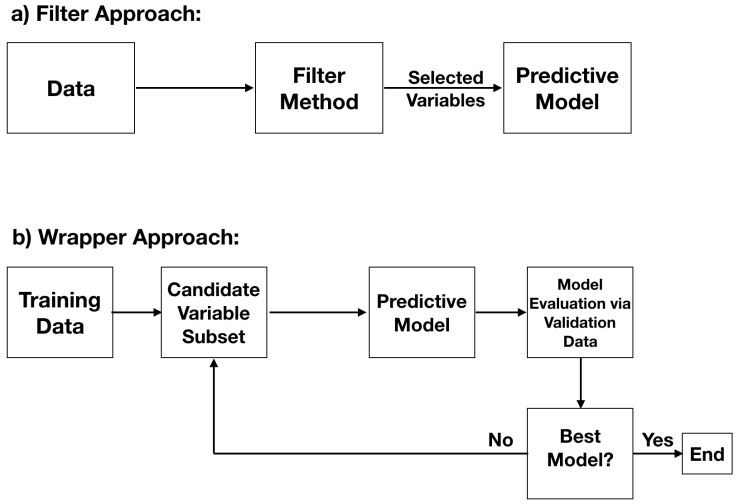


Figure 2.2. Approaches for variable selection. a) Filter approach b) Wrapper approach

In the equation above ϵ is the random noise term with the variance σ^2. The Delta test estimates the noise variance using the expression:

$$\sigma^2 = \frac{1}{2N}\sum_{i=1}^{N}(y_i - y_{NN(i)})^2 \tag{2.15}$$

The notation $NN(i)$ means the *nearest neighbour* of the point i. The nearest neighbours are estimated in the input domain as: $NN(i) = \arg\min_j \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|^2$. A subset of variables of a data point $\mathbf{x}$ is denoted as $\tilde{\mathbf{x}}$. So, the nearest neighbors can change depending on which variables are used in their estimation. The variable subset which provides the smallest estimation of the noise variance is chosen as an output of the Delta test. The reason is that this variable subset is capable to connect inputs and outputs in an optimal way. In this case, the explanatory power of the regression function is maximally enhanced and, hence, the noise variance becomes the smallest. More details about the Delta test method for variable selection can be found in [Eirola et al., 2014].

The diagram of the *wrapper* approach is depicted in the Figure 2.2b. In the wrapper approach, there is no separate variable selection model. The original predictive model is wrapped with an external variable selection loop. There are two extra components in this loop. One component proposes new candidate variable subset and the second component evaluates this subset by fitting the predictive model on the training data and evaluating its performance on the validation data.

Yet another class of variable selection methods is the *embedded* class. In this class, the predictive model itself has a variable selection capability. An example

of this class is Automatic Relevance Determination (ARD) which has been mentioned in Section 2.3.7. The evidence of a probabilistic model is minimized with respect to scaling coefficients of each variable. If a coefficient corresponding to a variable tends to zero then the variable is eliminated. Another embedded method is called Multi-response Sparse Regression (MRSR) which is introduced later is Section 3.2.3. MRSR is a variables ranking method developed for a linear regression model. Its selection criterion is the correlation of a variable with the current model residual [Similä and Tikka, 2005].

Usually, wrapper approaches are more accurate for a particular predictive model because variable selection is done jointly with model training. However, the computational cost of a wrapper approach is usually high since the model has to be retrained for many variable subsets. Filter approaches are computationally lighter, but they are also less accurate. Hence, in Publication I the hierarchical variable selection is applied. First, more coarse but faster Delta test has been used, then more computationally heavy wrapper approach. The outcome of the Delta test is three sets of variables:

1. Relevant variables
2. Not relevant variables
3. Variables which are investigated further

.

In particular, the selected variables after the Delta test are:

No. of samples	Relevant variables	Variables to be investigated further
227	"Flow S11", "Temperature", "Integrated Flow S11"	"Smoothed Flow S11", "Rains", "Sin Week", "Cos Week", "Rain int 1", "Rain int 4"

.

Table 2.2. Variable selection via Delta test for regression datasets in Publication I. .

After this first phase, the variable selection has been done by the wrapper approach using 3 regression models: Least-Squares Support Vector Regression (LS-SVR), Ridge Regression (RR), Support Vector Regression (SVR). Selected variables may be different for different regression models. The results are shown in the Table 2.3.

Best model	Relevant variables	$NMSE \pm (std)$
LS-SVR	"Flow S11", "Temperature", "Integrated Flow S11", "Smoothed flow S11", "Sin Week", "Cos Week"	$0.530 \pm (0.312)$
Ridge Regression	"Flow S11", "Temperature", "Integrated Flow S11", "Sin Week", "Int. Rain 2", "Int. Rain 5"	$0.675 \pm (0.394)$
SVM	"Flow S11", "Temperature", "Integrated Flow S11", "Smoothed flow S11", "Sin Week", "Cos Week"	$0.570 \pm (0.359)$

Table 2.3. Relevant variables and best models for the regression dataset in Publication I..

2.3.9 Model combination

If the goal is to maximize the model's predictive performance then the better way is not to select the best model but to combine several models in an appropriate way. One theoretical consideration which helps elucidate the advantage of model combination is the *bias-variance decomposition*. Under the assumption that the data has been generated from the model $y = f(\mathbf{x}) + \epsilon$, and having observed the dataset $D = \left\{\mathbf{x}^{(i)}, y^{(i)}\right\}_{i=1}^{N}$ the model $\hat{f}(\cdot)$ is build to predict the output of any new observation $y^{\star}$. Then the mean-squared error of prediction at new data point $\mathbf{x}^{\star}$ is:

$$\begin{aligned} &\mathbb{E}[(y^{\star} - \hat{f}(\mathbf{x}^{\star})|\mathbf{x}^{\star})^2] = \\ &\underbrace{\sigma_{\epsilon}^2}_{Noise} + \underbrace{(\mathbb{E}[\hat{f}(\mathbf{x}^{\star})] - f(\mathbf{x}^{\star}))^2}_{Bias} + \underbrace{\mathbb{E}[(\mathbb{E}[\hat{f}(\mathbf{x}^{\star})] - \hat{f}(\mathbf{x}^{\star}))^2]}_{Variance}. \end{aligned} \tag{2.16}$$

The expectations here should be understood as expectations over the training data where each point comes from $(\mathbf{x}^{(i)}, y^{(i)}) \sim p(\mathbf{x}, y)$ and over the conditional distribution of a test data point $p(y^{\star}|\mathbf{x}^{\star},)$.

The Mean Squared Error (MSE) is the sum of three terms where the first one is the irreducible noise term. The bias term measures the difference between the mean of prediction and the true regression function. The variance term measures variability of a model under the different training data which come from the true data distribution. If in the above formulation another loss function is selected instead of MSE, the algebra becomes not so convenient, but the

principle remains [Domingos, 2000].

The idea of a model combination is to reduce the variance term in the bias-variance decomposition. Consider, for example, the predictive model which is an average of M uncorrelated models: $\hat{f}(\mathbf{x}^\star) = \frac{1}{M}\sum_{m=1}^{M}\hat{f}_m(\mathbf{x}^\star)$. The variance of this average model[4]:

$$
\begin{aligned}
&E\left[\left(\frac{1}{M}\sum_{m=1}^{M}\hat{f}_m - \frac{1}{M}\sum_{m=1}^{M}E[\hat{f}_m]\right)^2\right] = \frac{1}{M^2}E\left[\left(\sum_{m=1}^{M}(\hat{f}_m - E[\hat{f}_m])\right)^2\right] = \\
&= \frac{1}{M^2}\sum_{m=1}^{M}E\left[\left(\hat{f}_m - E[\hat{f}_m]\right)^2\right] + \underbrace{\frac{1}{M^2}\sum_{m\neq l}E\left[(\hat{f}_m - E[\hat{f}_m])(\hat{f}_l - E[\hat{f}_l])\right]}_{=0} = \\
&= \frac{1}{M}\left(\frac{1}{M}\sum_{m=1}^{M}E\left[\left(\hat{f}_m - E[\hat{f}_m]\right)^2\right]\right).
\end{aligned}
\tag{2.17}
$$

The term in the middle line equals zero because of the uncorrelatedness of separate models $\hat{f}_m$. Hence, it is seen that the variance of the model averaging reduces the average variance by the factor $\frac{1}{M}$. This example demonstrates the benefits of using a model combination.

One technique to make the uncorrelated models is called *bagging* [Hastie et al., 2009]. In bagging the same model is trained on the different bootstrap[5] data samples. Then the output of the prediction is taken as a mean of the trained models. The models used in bagging should have a small bias and as a consequence a large variance [Hastie et al., 2009, p. 587]. Hence, decision trees are mostly used with bagging. In practice, it is hard to guaranty uncorrelatedness, hence in Random Forests (RF) [Breiman, 2001] extra source of randomness is introduced. Besides random subsets of data, random subsets of features are selected to train different models.

One of the most powerful methods of model combination (or ensemble) is *boosting*. The output of the algorithm is the weighted sum of the base models:

$$
y(\mathbf{x}^\star) = \sum_{m=1}^{M} w_m \hat{f}^m(\mathbf{x}^\star). \tag{2.18}
$$

The output of the boosted prediction is the weighted sum of the outputs of the base learners. Actually, this linear combination is the common form for all the ensemble methods. For instance, in bagging all the weights are assumed to be the same. The weights in boosting are trained sequentially one-by-one. More details can be found e.g. in [Hastie et al., 2009, p. 337]

[4] ignoring the dependency of $\mathbf{x}^\star$ in the notation

[5] see Section 2.3.4 for bootstrap sample definition

The weights of the ensemble can be found using the predictive performance of each model on the unseen data (validation data). This approach is called *stacking*. Details are given in [Hastie et al., 2009, p. 290]. Finally, for the probabilistic models, the Bayesian Model Averaging (BMA) can be used. The marginal likelihood for each model can be calculated as in Sec.2.3.7. These evidences can be used to weight the probabilities of the outputs of each separate model.

Ensemble methods and model combinations are extensive topics. Introduction to these topics can be found in machine learning textbooks.

2.4 Frequentists and Bayesian machine learning

In this section we compare and contrast the *frequentist* (Sec. 2.4.1) and *Bayesian* (Sec. 2.4.3) methodologies to statistical inference. Both approaches have been used by the author and are present in the papers in this dissertation. The difference appear in basic assumptions, approaches to inference, decision making. The Bayesian approach is becoming prevalent these days, although still methods and terminology of frequentist statistics are used for many problems. Section 2.2.1 also contain some useful definitions and the diagram.

Probabilistic model of the data may serve as a starting point for both statistical methodologies. However, for the Bayesian methodology this is an innate component while for the frequentist statistics this is an optional component. It is possible to design algorithms and perform inference without probabilistic model using the methods of frequentist statistics (see also section 2.5.1). When the probabilistic model is defined it is possible to apply methods of Bayesian as well as frequentist statistics.

Let's illustrate the statements above with the simple probabilistic supervised models example. The linear regression model is:

$$y_i = f(\mathbf{x}_i|\boldsymbol{\theta}) + \epsilon, \quad \text{where } \epsilon \sim \mathbb{N}(0, \sigma^2). \tag{2.19}$$

So, each observation is a parametric transformation of the observed variable $\mathbf{x}_i$ plus a Gaussian noise. Using this model, the probability of each observed pair is $p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) = \mathbb{N}(f(\mathbf{x}_i|\boldsymbol{\theta}) - y_i, \sigma^2)$. Therefore, the log of the probability density ($\ell\ell(\boldsymbol{\theta}|D)$ - likelihood function) of the whole data is:

$$\ell\ell(\boldsymbol{\theta}, \sigma^2|D) = \log \prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \boldsymbol{\theta}). \tag{2.20}$$

The likelihood function serves as a connection between the frequentist and Bayesian statistics. Having the likelihood function it is possible to compute

estimators of the model parameters (one of them is the maximum likelihood estimator), compute predictions at a new point $\mathbf{x}^\star$, and compute uncertainties of those. Alternatively, it is possible to use a Bayesian approach: assign a prior over parameters, compute posterior of the parameters, and posterior predictive distribution.

The algorithms of the frequentist framework may be designed without a probabilistic model. One such approach is the Empirical Risk Minimization (Sec. 2.3.2). This approach allows direct optimization of the decision function. So, the decision theory is not separated form the algorithm, unlike Bayesian statistics (Sec. 2.4.3). The example, of a non-probabilistic method for regression is Support Vector Regression (SVR). The objective function of SVR has the form:

$$J_{SVR} = \sum_{i=1}^{N} L_\epsilon(y_i|\mathbf{x}_i,\boldsymbol{\theta}) + \lambda\|\boldsymbol{\theta}\|^2 \tag{2.21}$$

$$L_\epsilon(y_i|\mathbf{x}_i,\boldsymbol{\theta}) = \begin{cases} 0, & \text{if } |y_i - \boldsymbol{\theta}^\top \mathbf{x}_i - \boldsymbol{\theta}_0| \le \epsilon \\ |y_i - \boldsymbol{\theta}^\top \mathbf{x}_i - \boldsymbol{\theta}_0| - \epsilon, & \text{otherwise} \end{cases} \tag{2.22}$$

Even though the form of the objective function J_{SVR} has similar form to $\ell\ell(\boldsymbol{\theta},\sigma^2)$ in (2.20), the loss function $L_\epsilon(y_i|\mathbf{x}_i,\boldsymbol{\theta})$ can not be interpreted as a logarithm of any probability distribution of y_i [Murphy, 2012, p. 505].

Frequentist and Bayesian statistical inference differ in assumptions and in treatment of uncertainty of the parameters. Sticking to the example above, the assumption of frequentist statistics is that there are fixed but unknown parameters $\boldsymbol{\theta}^\star$ and that the observed data D is a sample from the probability distribution $p(y_i|\mathbf{x}_i,\boldsymbol{\theta}^\star)$. Actually, here is one substantial difficulty of the frequentist statistics. Both the parameters are unknown and the dataset is only a sample from the implied data distribution with these unknown parameters. The uncertainty in the parameters comes from the possible variability of the datasets we may observe.

In Bayesian statistics the assumptions are different. It is assumed that the parameters are random variables which have a prior distribution $p(\boldsymbol{\theta})$ but the observed dataset is fixed. Using the data, prior uncertainty is reduced into the posterior which is represented by the posterior probability. The posterior probability of the parameters is computed in accordance with the Bayes formula:

$$p(\boldsymbol{\theta}|D) = \frac{p(D|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(D)} \tag{2.23}$$

Hence, in the Bayesian framework the dataset is considered to be fixed and the initial variability in model parameters which is expressed as the prior

uncertainty is transformed into posterior probability.

2.4.1 Frequentist inference

In frequentist statistics (also called classical statistics) an *estimator* is computed by an *algorithm* [Efron and Hastie, 2016, Chap. 1]. The *inference* is the process of estimating the uncertainty (or distributional properties) of an estimator. Hence, in contrast to Bayesian statistics, the inference and the computation of estimator are different processes. If the probabilistic model is available then the maximum likelihood procedure is one algorithm to compute estimators. This algorithm assigns to the estimators (parameters) the values which maximize the likelihood function.

If we are interested in doing optimal decisions the loss function must be provided. The loss function encodes the losses of incorrect decisions. Having the loss function this decision problem can be solved in the Empirical Risk Minimization framework. Empirical risk is the loss function between the parametrized decision function and the true output from the dataset. Then, empirical risk is minimized with respect to the parameters of the decision function. So, essentially, the Empirical Risk Minimization (ERM) (Sec. 2.3.2) is another algorithm which computes the parameters of the decision function. Hence, in the frequentist statistics the decision theory is coupled with the algorithms for computing estimators.

It is worth to say that in classical statistics the properties on estimators are studied extensively. For example, these properties are unbiasness, variance, consistency, optimality in a certain class of data models etc. [Murphy, 2012, Chap. 6]. In Machine Learning these properties are used in many subproblems, so it is worth to be aware of those.

Inference is the computation of the uncertainty of an estimator. In frequentist statistics it is done separately from the computation of estimators. There two main ways to compute estimator uncertainty: analytical and computational. Sometimes the data model is simple enough like in the linear regression model. In this case, it is possible to compute analytically the distribution of the estimator of parameters. Analytic approximations to the uncertainty of estimators can also be obtained with large sample approximations to maximum likelihood [Murphy, 2012, Chap. 6]. It is also possible to propagate the uncertainty of parameters to the uncertainty of the new predicted value $y^\star$ at new regressor value $\mathbf{x}^\star$.

On the contrary, if an estimator or probabilistic model are relatively complex, or if the quantity of interest does not have an analytical expression, then the universal way to measure the uncertainty of any quantity is to perform *boot-*

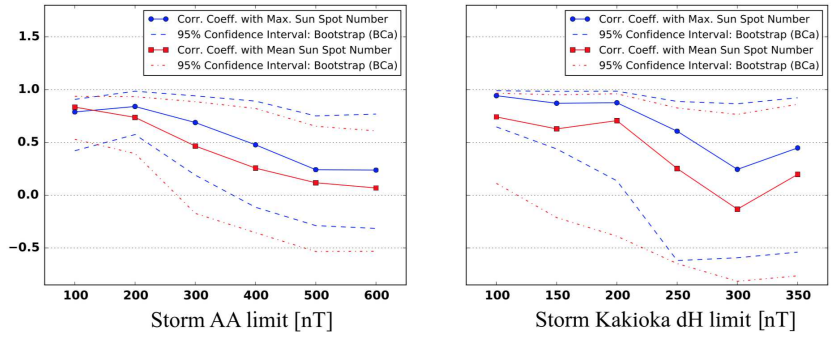


Figure 2.3. Estimation of Pearson correlation coefficient between solar phase strength and geomagnetic storm strength for different thresholds of storm strength. See details in Publication II.

straping [Efron and Tibshirani, 1993]. The bootstrapping (Section 2.3.5) is a process which models the data generation assumptions of frequentist statistics. Its essence is the generation of multiple datasets $D_1, D_2 \cdots D_B$ and the whole inference procedure is repeated on each of those. The different values of the quantity of interest obtained from the different bootstrap datasets represent the uncertainty of this quantity.

2.4.2 Example of frequentist uncertainty estimation

On Figure 2.3 the example of frequentist uncertainty estimation is shown. The example is taken from Publication II. In this paper, the dependency between the strength of geomagnetic storms and solar phases has been studied. All the storms have been divided into subsets with a certain maximum value. Then the Pearson correlation coefficient has been computed between the strengths of geomagnetic storms in a subset and the strength of a solar cycle.

The geomagnetic activity can be measured in several different ways. In this paper two sources of data has been used: the AA index[6] and Kakioka laboratory data[7]. Results for these two sources are plotted on the left and right subfigures respectively. The solar strength is measured by the international sunspot number (ISSN) from the Belgian Royal Observatory[8].

In this work, the estimation of the uncertainty (or confidence intervals) of the correlation coefficient is provided. Without uncertainty estimation speculations about the possible correlations are not justified enough. The confidence intervals have been estimated by the bootstrap BCa method [Efron and Tibshirani, 1993,

[6] ftp://ftp.ngdc.noaa.gov/STP/GEOMAGNETIC_DATA/AASTAR/

[7] http://www.kakioka-jma.go.jp/metadata/geomagnetic/geomag_kak

[8] http://www.sidc.be/silso/datafiles

p. 187]. It has been concluded that the there is a positive correlation between moderately strong geomagnetic storms and the solar phase. For extremely strong storms the correlation is absent.

2.4.3 Bayesian inference

In the Bayesian framework to statistics, the observed data is considered fixed, but the parameters ($\boldsymbol{\theta}$ is our example) must have a prior distribution which represent initial uncertainty about them. There are many ways to choose a prior distribution which depend on a particular situation and sometimes on mathematical convenience [Murphy, 2012, p. 165]. There is a subjective decision in choosing a prior. This is the main critique of the Bayesian statistics. It can be considered as both a strength and weakness depending on situation. Priors allow consistent treatment of uncertainty propagation. In practice, it is worth to check how the conclusions of the statistical analysis change when a prior distribution changes.

After assigning a prior distribution, the posterior can be computed using *Bayes Formula* (2.23). In the numerator of the right-hand side, there is a function of $\boldsymbol{\theta}$ which defines the shape of the distribution and in the denominator, there is a normalizing constant. Applying Bayes rule is the *inference* in Bayesian statistics, because Bayes rule computes probability of posterior. Therefore, inference and algorithm for computing estimators is the same in Bayesian framework. This is in contrast with frequentist approach where the algorithm and inference are different procedures.

If the likelihood and the prior are in relatively simple and conjugate form then the normalizing constant and, hence, the posterior can be computed analytically. However, in the majority of cases, the normalization constant can not be computed analytically and numerical or approximation methods are required.

If the dimensionality of $\boldsymbol{\theta}$ is small, regular numerical integration methods can be used. If it is not the case, the Markov Chain Monte Carlo (MCMC) methods are able to generate directly the samples from the posterior distribution e.g. [Neal, 2012], [Hoffman and Gelman, 2014]. Besides, there exists two popular methods to compute an approximation to the posterior distribution: Variational Inference (VI) and Expectation Propagation (EP). In variational inference, the posterior is usually approximated by some simpler distribution whose parameters are found by numerical optimization. As the result, the numerical integration is substituted by the numerical optimization [Jordan et al., 1999]. The EP algorithm assumes that the posterior is a multiplication of simpler distributions (e.g. Gaussians) and is able to update these factors one-by-one

to approximate the posterior [Minka, 2001]. By utilizing these approximation methods the normalizing constant in (2.23) can be approximated. If there are hyperparameters the model depends on, then the normalizing constant also depend on them (e.g. $p(D|\boldsymbol{\alpha})$). Using the above approximation, hyperparameters can be optimized which correspond to the evidence framework in Section 2.3.7. Alternatively, they may be treated as regular parameters (assign a prior and perform Bayesian inference) but on the higher level of hierarchy.

After the posterior distribution of parameters $p(\boldsymbol{\theta}|D)$ has been found the prediction at a new point $\mathbf{x}^\star$ is done according to the expression below. It is a *posterior predictive distribution*:

$$p(y^\star|\mathbf{x}^\star) = \int p(y^\star|\mathbf{x}^\star,\boldsymbol{\theta})p(\boldsymbol{\theta}|D)d\boldsymbol{\theta}. \tag{2.24}$$

So, the parameters are integrated out. The integration above is often intractable, however, if approximation methods are used then integration becomes analytically tractable. In the general case, samples from the posterior obtained during the Markov Chain Monte Carlo run can be used to perform Monte Carlo integration.

The uncertainty estimation in Bayesian statistics is done automatically because of the built-in uncertainty propagation capabilities of the framework. No separate procedure is required for uncertainty estimation, however the inference in the general case i. e. MCMC computation can be very computationally demanding.

After the distributions of parameters and/or hidden variables are computed, the role of decision theory is to convert these distribution into actions. For instance, we might need to classify an e-mail as spam or not spam, given corresponding probabilities. Like in the frequentist approach, the loss function L is required to do that. Possible actions are denoted as a. Then, the *posterior expected loss* can be computed as:

$$\rho(a|\mathbf{x}^\star) = \mathbb{E}_{p(y|\mathbf{x}^\star)}[L(a,y)] = \int_y L(a,y)p(y^\star|\mathbf{x}^\star)dy. \tag{2.25}$$

Then, the decision function becomes: $\delta(\mathbf{x}^\star) = \arg\max_a \rho(a|\mathbf{x}^\star)$. Hence, making decisions in the Bayesian framework is completely separated from the inference, where $p(y^\star|\mathbf{x}^\star)$ is computed. It also means that the inference may be done only once, but decision making can be recomputed for different loss functions. This is an advantage over the frequentist methodology.

2.4.4 Summary of frequentists and Bayesian approaches to statistics

Compare and contrast the Frequentist and Bayesian frameworks		
Category	**Frequentist inference**	**Bayesian inference**
Assumptions	Data is a sample from a distribution. Parameters of this distribution are fixed but unknown.	Observed data is fixed and it reduces the uncertainty of the parameters.
Probabilistic model	Not compulsory, but is useful (e.g. for maximum likelihood algorithm).	Compulsory.
Prior	No prior needed.	Prior of parameters is required. It can be considered as both a strength and a weakness depending on situation.
Inference	Estimators of parameters by some algorithm. *Inference* is done separately by analytical or numerical methods.	Application of Bayes rule. No separation between an *algorithm* and *inference*.
Decisions	Decision function is embedded into the estimator.	Decisions are separated from inference.
Statistical questions e.g. $\mathbb{E}[y]-?$, $p(y > 40)-?$	Need to solve a separate problem for each question.	These questions are answered simultaneously using the posterior of y.
Data comes in batches	Hard to deal with the data arriving in portions.	Easy. Posterior in one step becomes a prior on the next step.

2.5 Model types

In the previous sections the focus was on the machine learning problems and inference methods. In this section more details about the models are given. Machine learning models are divided into several intersecting classes with different properties. Below is the short summary of various model classes which have been used on this dissertation.

2.5.1 Probabilistic vs non-probabilistic modeling

The probabilistic model models directly the probabilities of the data, in particular, it models the likelihood function as was described in Section 2.4. There, the linear regression is shown as an example of a probabilistic model. This model explicitly shows the presence of noise in the model which defines the likelihood function. Sometimes, in the regression setting, the model formulation is not probabilistic but inference procedure reveals its probabilistic essence. For instance, if neural network minimizes the mean-squared error on the data it implicitly assumes that the output variable is generated from the Gaussian distribution where mean is modeled as a neural network and the variance is fixed but unknown.

An example of non-probabilistic model is the K-Nearest Neighbour (K-NN) model for a classification. Another example is a model for binary clustering where the output $y_n \in 0,1$ is modeled directly from $\mathbf{x}_n$: $y_n = f(\mathbf{x}_n|\boldsymbol{\theta})$. In contrast logistic regression is a probabilistic binary classifier which models $p(y_n|\mathbf{x}_n,\boldsymbol{\theta})$.

Probabilistic models are currently more frequently used in machine learning because of several advantages. A probabilistic output is often required to make optimal decisions. Probabilistic description of data is very flexible, it can include latent variables, hierarchical structure and deal with many types of random variables e.g. continuous and discrete. Also, there are standard well-developed inference and learning methods like sum-product algorithm, junction tree algorithm, maximum livelihood approach, EM algorithm, variational inference [Barber, 2012]. In summary, the probabilistic approach is valued for consistency, flexibility and powerful inference and learning methods. In this dissertation, all the described models are assumed to be probabilistic unless the opposite is stated.

2.5.2 Generative vs discriminative models

In supervised learning the probabilistic data modeling can be done in *discriminative* and *generative* way. The joint distribution of dependent and independent variables can be written is two ways: $p(y,\mathbf{x}) = p(y|\mathbf{x})p(\mathbf{x}) = p(\mathbf{x}|y)p(\mathbf{x})$. The first way of modeling is called discriminative, because the predictive distribution $p(y|\mathbf{x})$ is modeled directly. In the generative model the distribution $p(\mathbf{x}|y)$ is being modeled and the predictive distribution is obtained by applying the Bayes Formula (2.23). Actually, these two approaches can be merged [Bishop and Lasserre, 2007], but it is not considered here.

If we are interested only in classification, then the discriminative model seems

more feasible since it directly gives what is required. Is also considered to be more accurate, but not always [Ng and Jordan, 2002]. On the other hand, the generative approach is more flexible. It provides a straightforward way to address semi-supervised learning and missing data problem [Bishop, 2006].

2.5.3 Latent variable models

Latent variable models are models which include non-observed random variables. Moreover, usually, the model is organized so that each latent variable correspond to each data sample. This correspondence between each latent variable and each data sample make latent variables distinct from the hyperparameters in the model of data distribution.

A well-known example of a latent variable model is the Gaussian mixture model. It is a model of unsupervised learning, and latent variables there correspond to components of the mixture where each sample is generated from. Other examples of latent variable models in unsupervised learning are PCA and Gaussian Process Latent Variable Model (GP-LVM) [Lawrence, 2003]. An example of the supervised latent variable model is Gaussian Process (GP) (see Chapter 4) where the latent variables are the values of the latent function.

The maximum likelihood inference or MAP inference can not be applied to the latent variable models directly. The reason is that likelihood function involves a marginalization of unobserved latent variables which is often impossible to compute in practice. The standard algorithm to do inference and learning is EM-algorithm. Other approximate inference algorithms like variational inference are also applicable.

2.5.4 Parametric vs non-parametric models

Probabilistic statistical models can be also divided into parametric and non-parametric ones. Even though this division is slightly ambiguous, the main characteristics are described here. In a parametric model, there is a fixed number of parameters which need to be learned by some inference approach. Typically, the data is modeled as a statistical model and the parameters of this models are to be found. So, the complexity of the model is fixed since it depends on the number of parameters. Hence, the model complexity is not adjusted to the amount of available data. Examples of parametric models are the logistic regression, multilayer perception and many other deep learning models.

A non-parametric model does not imply that there no parameters, but rather, that the number of parameters grows with the size of the training data. As

a result, the model can automatically adjust its complexity to the amount of available training data. Typical examples of the non-parametric models are K-nearest neighbour classifier, *decision trees* and *kernel* methods. Gaussian process models are Bayesian non-parametric models.

Even though non-parametric methods have the advantage of automatically adjusting the model complexity, in practice, parametric methods are also frequently used. For instance, the best results in image classification have been obtained by parametric models. The reason is that the final performance of the method depends not only on the model, but also on the approximations which are made, and on the learning algorithm.

3. Extreme Learning Machines and Time series Prediction

3.1 Extreme Learning Machine

Extreme Learning Machine (ELM) is a single-layer feed-forward neural network which is depicted in Figure 3.1. The hidden layer weights are randomly generated and fixed. The error function of ELM is linear in the output weights and can be computed by solving least squares problem [Huang et al., 2006]. In contrast, the standard neural networks such as a Multi-layer Perceptron (MLP) are trained by back-propagation algorithm [Rumelhart et al., 1988]. Back-propagation is a slow stochastic gradient type learning algorithm. Since there is no back-propagation in ELM training, the training time is very short in comparison to standard neural network training. It makes the ELM very suitable for some applications. Additionally, the accuracy of ELM is comparable with the modern deep neural networks [McDonnell et al., 2015], at least for some problems.

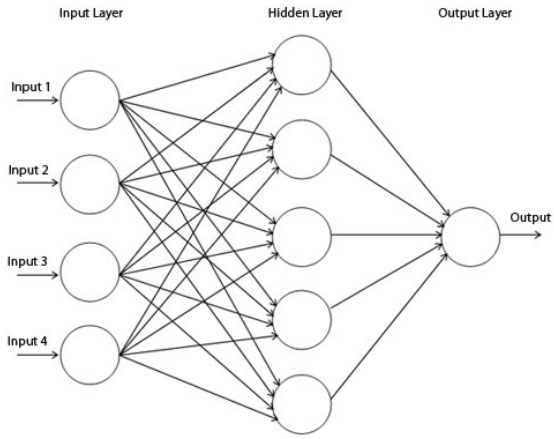


Figure 3.1. Typical scheme of a single-layer feed-forward neural network

3.1.1 Basic algorithm

The output of an ELM is computed using the formula:

$$f(\mathbf{x}) = \sum_{i=1}^{H} \beta_i h_i(\mathbf{x}) = \mathbf{h}(\mathbf{x})^\top \boldsymbol{\beta} \quad . \tag{3.1}$$

The output $f(\mathbf{x})$ is the linear combination of the basis functions $h_i(\mathbf{x}), i = \{1,...,H\}$. There are several types of basis functions which can be applied in practice; they are presented in Table 3.1. The basis functions where the input variable is first changed in a linear way $\mathbf{a}^\top \mathbf{x} + b$ may be interpreted as a hidden layer of a neural network. Hence, the ELM is the one layer neural network, as shown in Figure 3.1. The basis functions where the input is transformed as $\|\mathbf{x} - \mathbf{a}\|$ can be interpreted as a Radial Basis Functions Network (RBFN). Therefore, the basis functions are often called neurons.

Table 3.1. Commonly used basis functions for ELM [Huang et al., 2015]

Sigmoid function	$h(\mathbf{x};\mathbf{a},b) = \frac{1}{1+\exp\{\mathbf{a}^\top \mathbf{x}+b\}}$
Hyperbolic tangent function	$h(\mathbf{x};\mathbf{a},b) = \frac{1-\exp\{\mathbf{a}^\top \mathbf{x}+b\}}{1+\exp\{\mathbf{a}^\top \mathbf{x}+b\}}$
Gaussian function	$h(\mathbf{x};\mathbf{a},b) = \exp\{-b\|\mathbf{x}-\mathbf{a}\|\}$
Multiquadric function	$h(\mathbf{x};\mathbf{a},b) = (\|\mathbf{x}-\mathbf{a}\| + b^2)^{1/2}$
Hard limit function	$h(\mathbf{x};\mathbf{a},b) = \begin{cases} 1, & \text{if } \mathbf{a}^\top \mathbf{x} + b \le 0. \\ 0, & \text{otherwise.} \end{cases}$
Fourier basis	$h(\mathbf{x};\mathbf{a},b) = \cos(\mathbf{a}^\top \mathbf{x} + b)$

In ELM the weights of each neuron $(\mathbf{a}_i, b_i)$ are randomly generated according to some continuous probability distribution. The common choices are Gaussian distribution $\mathbb{N}(0,1)$, or uniform $[-1,1]$. Later on, they are fixed and not optimized. This leads to the high computational efficiency of the ELM.

Suppose that the training data consist of N training pairs of inputs and outputs $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$ where $\mathbf{x}_i \in \mathbb{R}^D$ and $\mathbf{y}_i \in \mathbb{R}^Q$. After the generation of neuron weights and biases, output of the hidden layer for all the N data points can be represented in one matrix:

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}^\top(\mathbf{x}_1) \\ \mathbf{h}^\top(\mathbf{x}_2) \\ \vdots \\ \mathbf{h}^\top(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} h_1(\mathbf{x}_1) & h_2(\mathbf{x}_1) & \cdots & h_H(\mathbf{x}_1) \\ h_1(\mathbf{x}_2) & h_2(\mathbf{x}_1) & \cdots & h_H(\mathbf{x}_1) \\ \vdots & \vdots & \ddots & \vdots \\ h_1(\mathbf{x}_N) & h_2(\mathbf{x}_N) & \cdots & h_H(\mathbf{x}_N) \end{bmatrix}. \tag{3.2}$$

The output data can also be represented as a matrix:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^\top \\ \mathbf{y}_2^\top \\ \vdots \\ \mathbf{y}_N^\top \end{bmatrix} = \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1Q} \\ y_{21} & y_{22} & \cdots & y_{1Q} \\ \vdots & \vdots & \ddots & \vdots \\ y_{N1} & y_{N2} & \cdots & y_{NQ} \end{bmatrix}. \tag{3.3}$$

The cost function which is optimized by ELM is shown below. It is a least-squares problem:

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^{H \times Q}} \|\mathbf{H}\boldsymbol{\beta} - \boldsymbol{Y}\|^2 \tag{3.4}$$

Often it is beneficial to add Tikhonov regularization to the cost function, so the optimization problem becomes:

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^{Hs \times Q}} \|\mathbf{H}\boldsymbol{\beta} - \boldsymbol{Y}\|^2 + \lambda \|\boldsymbol{\beta}\|^2 \tag{3.5}$$

The solution can be expressed analytically. Under the assumption that the number of samples is not smaller than the number of neurons $N \geq H$, the solution is:

$$\boldsymbol{\beta}^\star = \boldsymbol{H}^\dagger \mathbf{Y} = (\mathbf{H}^\top \mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{H}^\top \mathbf{Y} \tag{3.6}$$

In practice, this solution can be computed by standard methods of solving least-squares problems: Cholesky decomposition of the normal equation matrix, QR-decomposition, SVD based methods, iterative methods and so forth.

The full ELM algorithm is given below:

Algorithm 3: Basic ELM

Input : 1) Training set $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^D$ and $\mathbf{y}_i \in \mathbb{R}^Q$

2) Continuous probability distribution for sampling neuron weights from

3) Activation function

4) Number of neurons H

Output: 1) Weights $\boldsymbol{\beta}$ which solve ELM minimization in Eq. (3.4)

2) Random weights $\mathbf{a}_i$ and b_i

1 Randomly assign input weights and biases $\mathbf{a}_i$ and b_i using the given probability distribution ;

2 Calculate the hidden layer matrix $\mathbf{H}$ (3.2) ;

3 Calculate the output weights $\boldsymbol{\beta} = \boldsymbol{H}^\dagger \mathbf{Y}$

It is worth noting that ELM has only one hyperparameter H. The choice of

the sampling distribution and activation function are related to the design of the architecture.

3.1.2 Theoretical foundations

Below is the list of theorems which justify the ELM approach. The main advantage of this approach is that it allows to avoid the gradient-based optimization of the hidden weights of a single layer neural network. The interpolation capability of ELM is guaranteed by the following theorem [Huang et al., 2006]:

Theorem 3.1. *Given any small positive value* $\epsilon > 0$ *any activation function which is infinitely differentiable in any interval and* N *arbitrary distinct samples* $\{\mathbf{x}_i, \mathbf{y}_i\} \in \mathbb{R}^d \times \mathbb{R}^m$*, there exist* $H < N$ *such that for any* $\{\mathbf{w}_i, b_i\}$ *randomly generated from* $\mathbb{R}^d \times \mathbb{R}$*, according to any continuous probability distribution, with probability one* $\|\mathbf{H}\beta - T\| \leq \epsilon$*. Furthermore, if* $H = N$*, then with probability one* $\|\mathbf{H}\beta - T\| = 0$*.*

The theorem states that any training set can be approximated with arbitrary precision if the number of hidden neurons is large enough. If the number of neurons is equal to the number of training samples then the approximation error is zero. However, this theorem considers only the approximation of the training set, it says nothing about the generalization properties of ELM.

The universal approximation theorem addresses the generalisation ability of a neural network. This theorem [Cybenko, 1989] shows that a single hidden layer neural network is able to approximate any continuous function on the compact subset of $\mathbb{R}^n$ with arbitrary precision; a similar theorem exists for ELM [Huang and Chen, 2008]:

Theorem 3.2. *Given any non constant piecewise continuous function* $G : \mathbb{R}^d \to \mathbb{R}$*, if span*$\{G(\mathbf{x}|\mathbf{a}, b) : \text{parameters } (\mathbf{a}, b) \in \mathbb{R}^d \times \mathbb{R}\}$ *is dense in* L^2*, then for any continuous target function* f *and function sequence* $\{f_H(\mathbf{x}) = \sum_{i=1}^{H} \beta_i G(\mathbf{x}|\mathbf{a}_i, b_i)\}$ *randomly generated from any continuous sampling distribution,* $\lim_{H\to\infty} \|f - f_H\| = 0$ *holds with probability one if the output weights* β_i *are determined by ordinary least squares to minimize* $\|f(\mathbf{x}) - \sum_{i=1}^{H} \beta_i G(\mathbf{x}|\mathbf{a}_i, b_i)\|$*.*

This theorem proves that any continuous function f can be approximated by ELM with arbitrary precision, given enough hidden units.

Finally, Vapnik-Chevonenkis dimension (VC-dimension) of an extreme learning machine is given by the following theorem [Liu et al., 2012]:

Theorem 3.3. *The VC-dimension of ELM with* H *hidden nodes which are infinitely differentiable in any interval is equal to* H *with probability 1.*

The VC-dimension characterizes the complexity of the model. The model with the lower complexity has better generalization capability. The theorem states that the VC-dimension of ELM is relatively small. For example, the VC-dimension of SVM with RBF kernel is infinite.

3.1.3 Historical reference

The idea of randomly initializing hidden layers and solving the output layer with the pseudo-inverse has been investigated by several researchers before, e.g. [Schmidt et al., 1992], [Braake et al., 1997]. However, these attempts have not been widely adopted.

The first ELM publication [Huang et al., 2006] had shown the universal approximation capability of ELM, and has also demonstrated the competitive results of this simple algorithm. Later, the idea of random weights has extended to a broader framework of methods based on randomization e.g. representation learning, unsupervised learning.

Similar models from the kernel-methods point of view have been proposed in [Rahimi and Recht, 2008] and further extended e.g. in Yang et al. [2015]. In these models, the kernel is approximated by an outer product of random Fourier features. By applying the *kernel trick* in the opposite direction this approach is very similar to ELM with cosine activation function.

3.1.4 Leave-one-out cross-validation

In linear systems, computing leave-one-out (LOO) cross-validation (see Sec. 2.3.4) with the same computational complexity as solving a least squares problem is an important property.

Having a least squares problem $\min_{\boldsymbol{\beta}\in\mathbb{R}^{H\times Q}} \|\mathbf{H}\boldsymbol{\beta}-\mathbf{T}\|^2$ with multivariate output matrix $\mathbf{T}$, the LOO mean squared error is a sum of squared errors between predicted outputs and true outputs:

$$E_{loo}^2 = \frac{1}{N}\sum_{i=1}^{N} \|(\mathbf{t}_i^{pred})^\top - \mathbf{t}_i^\top\|^2 = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{H}_{-i}.\boldsymbol{\beta}_{-i} - \mathbf{t}_i^\top\|_2^2 \quad , \tag{3.7}$$

where $\mathbf{H}_{-i}$ is a matrix $\mathbf{H}$ with row i removed, $\mathbf{t}_i^\top$ is the row i of the true output matrix $\boldsymbol{T}$ and N is the number of rows in $\mathbf{H}$. $\boldsymbol{\beta}_{-i}$ is the solution of least-squares problem with matrix $\mathbf{H}_{-i}$.

The naive approach to compute this error is to solve the linear system for each row of the matrix $\mathbf{H}$ removed. The computational complexity of this naive approach is $N \times \mathcal{O}(NH^2) = \mathcal{O}(N^2H^2)$ (assuming that $N > H$) where solving one least-squares problem is $\mathcal{O}(NH^2)$.

PREdicted residual Sum of Squares error (PRESS) statistic is an efficient formula to compute the LOO error:

$$E_{loo}^2 = \|\mathbf{D}(\mathbf{T} - \mathbf{H}(\mathbf{H}^\top\mathbf{H})^{-1}\mathbf{H}^\top\mathbf{T})\|_2^2 \quad , \tag{3.8}$$

where $\mathbf{D}$ is the diagonal matrix, with diagonal elements $d_{ii} = \frac{1}{1-(\mathbf{H}(\mathbf{H}^\top\mathbf{H})^{-1}\mathbf{H}^\top)_{ii}}$ (index ii denotes diagonal elements). The computational complexity of the PRESS statistic formula is $\mathcal{O}(H^3) + \mathcal{O}(NH)$ which scales only linearly with N. For large N this computational saving may be significant.

Interestingly, for the Tikhonov regularized least squares problem (3.5) the LOO error is almost the same, only the inversion $(\mathbf{H}^\top\mathbf{H})^{-1}$ is changed to $(\mathbf{H}^\top\mathbf{H} + \lambda\mathbf{I})^{-1}$

Leave-one-out cross-validation uses almost all the data points on one iteration to build a learning algorithm. Hence, the bias of LOO is smaller than the bias of other cross-validation methods. However, for the same reason the variance of LOO can be larger, especially if the learning algorithm has discontinuities. Hence, if the computational advantages, described in this section are not important, the cross-validation with 5 or 10 folds are typically more preferable [Arlot and Celisse, 2010], [Hastie et al., 2009, Section 7.10].

3.1.5 The role of singular value decomposition (SVD)

There are several approaches to solving least-squares problems: (3.4) or (3.5). For instance, solving the QR-decomposition of the matrix $\mathbf{H}$, and changing the right-hand side accordingly [Golub and Van Loan, 1996, p. 236]. Another approach is to multiply both sides of (3.4) by $\mathbf{H}^\top$. This is the projection of both sides of the equation to the column space of $\mathbf{H}$. The disadvantage of this approach is that the conditioning of the modified system is increased twice.

The Singular Value Decomposition (SVD) approach has a special advantage for solving the ELM optimization problem. It allows rapid computation of the optimal output weights for different values of regularization hyperparameter λ. If the SVD decomposition of $\mathbf{H}$ is computed: $\mathbf{H} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$, then then optimal weights are:

$$\boldsymbol{\beta}^\star = (\mathbf{H}^\top\mathbf{H} + \lambda\mathbf{I})\mathbf{H}^\top\mathbf{T} = \mathbf{V}\left(\frac{D}{D^2 + \lambda\mathbf{I}}\right)^{-1}\mathbf{U}^\top\mathbf{T} \tag{3.9}$$

The matrix which is to be inverted is the diagonal matrix, so its inversion is trivial. Hence, if the SVD of the matrix $\mathbf{H}$ is computed once, then the solution of the minimization problem (3.5) is fast to compute for any regularization parameter λ.

A similar result holds for the E_{loo} in (3.8). There are no expensive inversions

required for different regularization parameter values once the SVD has been computed. Moreover, it can be shown that $E_{loo}(\lambda)$ is a convex function, so it is easy to optimize e.g. using the method of [Nelder and Mead, 1965].

3.2 Improving the basic extreme learning machine

Even though the basic ELM has a universal approximation capability, the required number of neurons to achieve a small training and test error can be very large. One related practical problem is the variability of the ELM predictions because of the stochasticity of the hidden space weights. The fewer neurons there are the more variable the output is. This behavior is also observed in the perceptron type neural networks trained using back-propagation algorithm where the result of the optimization depends on the starting values of the parameters.

One way to deal with the variability of the output is to combine several ELM models into an ensemble. This would turn the negative effect of prediction variability into a positive effect of diversity in an ensemble e.g. [van Heeswijk et al., 2011]. However, this might not always be the appropriate option. Sometimes there is a need to choose a single better model.

Non-relevant or correlated data features also deteriorate the performance of basic ELM because they increase the dimensionality, hence increasing the total number of parameters without adding any new information.

There are several approaches to deal with the aforementioned problems.

3.2.1 Penalization approaches

First of all, it is quite natural to apply L_2-regularization to ELM as shown in Eq. (3.5). L_2-regularization typically reduces the generalization error and prevents overfitting. However, this requires extra computations to choose the regularization constant λ. This is only a one-dimensional optimization problem so in practice it can be done reasonably fast. Also, L_2-regularization can be embedded in other methods as we will see later.

L_1-regularization or Automatic Relevance Determination (ARD) [Luo et al., 2014] are other regularization approaches that select relevant neurons and thus reduce the model size.

Another term for the penalization approaches is *shrinkage*. The reason is that the values of the estimated parameters ($\boldsymbol{\beta}$ for ELM) are lower in comparison to the case when no penalization is used. The primal reason of using penalization

is the regularization of learning algorithms which is introducing extra bias but removing greatly the variance of estimated parameters. Another reason is when the parameters of the algorithms are the important on their own because they are interpretable. In this case, shrinkage becomes a crucial component [Efron and Hastie, 2016, Ch. 7]. In ELM models the parameters $\boldsymbol{\beta}$ have no meaning on their own, hence penalization approaches are optional.

3.2.2 Incremental approaches

The incremental approach to build a hidden layer of ELM is described in this section. The pruning approach is considered in the next section. They both optimize network architecture during training. The general algorithm for such network structure optimization methods is presented in Algorithm 4:

Algorithm 4: Template algorithm for adaptive ELM (incremental or pruning approaches)

Input : 1) Same as in basic ELM (Alg. 3)
2) **Maximal** number of neurons $H_{\max}$

Output: 1) Selected number of neurons H
2) The optimal ELM linear weights $\boldsymbol{\beta}$
3) Selected neurons' weights

Start from a small number (incremental approach) or from $H_{\max}$ neurons (pruning approach) ;
(pruning approach) compute the neurons evaluation order ;
repeat
    Generate candidate neurons (incremental approach) or consider next neuron to prune (pruning approach);
    Evaluate the criteria for inclusion or pruning ;
until *some exit criterion (e.g. related to the training error or validation error) is satisfied*;

In the incremental approach to ELM [Huang, 2006], the network starts with a small number of neurons and neurons are gradually added. On each iteration, a new neuron is randomly generated and its output weight is optimized to minimize the training error.

The idea of convex-incremental extreme learning machine [Huang and Chen, 2007] is very similar, except the output weights of the existing neurons are also changed in an appropriate way assuming that the network output f_k on iteration k is: $f_k = (1-\alpha)f_{k-1} + \alpha g_k$. The output at iteration k is the convex combination of the previous output f_{k-1} and the output of the generated neuron g_k. Optimal

selection of α at each iteration can increase the speed of convergence, to the desired level of training error. The modification of the existing weights is only multiplication by the value α. So, it does not require heavy computations.

Finally, error-minimized ELM (EM-ELM) [Feng et al., 2009] allows the addition of more than one neuron at each iteration. The authors show that the output weights can be sequentially updated so that they represent the solution of a least squares problem at each iteration.

All algorithms in this subsection minimize the training error, so these methods are prone to overfitting the data. Therefore, in order to prevent overfitting, the minimum training error should be controlled e.g. using a separate validation set.

3.2.3 Pruning approaches

In pruning approaches a large number of neurons are initially generated. During training, they are pruned according to some criteria. The widely used pruning approaches for ELM are Optimally-Pruned ELM (OP-ELM) [Miche et al., 2010] and its extension Tikhonov Regularized Optimally-Pruned ELM (TROP-ELM) [Miche et al., 2010].

The OP-ELM algorithm consists of three phases (see Algorithm 4). First, the maximum amount of neurons are generated, similar to the basic ELM algorithm. In the second phase, the neurons are ranked by Multi-response Sparse Regression (MRSR) [Similä and Tikka, 2005]. After ranking, the least relevant neurons are pruned one-by-one. The stopping criteria is the leave-one-out (LOO) (Sec. 3.1.4) cross-validation computed by the PRESS statistic. When the validation error starts to increase the pruning is stopped (Algorithm 4).

In contrast to incremental approaches, the pruning approach uses LOO as a stopping criterion, not the training error. Hence, there is no need to have a separate validation set. Below is a short description of MRSR.

Multiresponse Sparse Regression (MRSR)

MRSR has been proposed by [Similä and Tikka, 2005]. It is a multivariate output extension of Least Angle Regression (LARS) developed by [Efron et al., 2004]. LARS is a variable selection algorithm for a linear regression model. Under a small modification LARS algorithm computes the LASSO [Tibshirani, 1994] regression solution for all values of the regularization parameter. For several years after its development is has been the fastest algorithm to compute the LASSO solution, until the introduction of the coordinate descent algorithm [Friedman et al., 2007].

In case the LARS or MRSR, algorithms are run until completion (number of iterations equal the number of variables). The solution of linear regression is equal to the Ordinary Least Squares (OLS) solution, with the addition of the ranking of variables (regressors) in the order in which they enter the model. If the algorithm is not run until the end, then the solution is not equal to the OLS solution on the included variables. In OP-ELM (or TROP-ELM) only the ranking of the neurons is used, the output weights are computed as OLS on the selected neurons. The outline of the MRSR algorithm is given in Alg. 5. Further details

of the LARS algorithm can be found in [Hastie et al., 2009, p. 73].

Algorithm 5: Multiresponse sparse regression (MRSR)

Input : 1) $H_{\max}$ - number of variables (regressors) to select

2) $\mathbf{H}^{N\times H}$ - regressors matrix, $\mathbf{Y}^{N\times Q}$ - output matrix

Output: 1. Approximations $\mathbf{T}$ to the outputs $\mathbf{Y}$ which are computed as:

$\mathbf{T} = \mathbf{H}\boldsymbol{\beta}$

2) Solution $\boldsymbol{\beta}$, where the number of non-zero rows is $H_{\max}$.

3) Ranking of variables (regressors).

Normalize the columns of the regressor matrix $\mathbf{H}$ to have zero mean and unit norm ;

k = 0 (iteration number). Set initial output approximation $\mathbf{T}^0 = 0$;

Compute $c_j^0 = \|(\mathbf{Y} - \mathbf{T}^0)\boldsymbol{h}_j\|$. These measure correlations of current residuals with each regressor. $\boldsymbol{h}_j$ is a column of matrix H;

Compute the most correlated regressors, include them into an active set: $c_{\max}^0 = \max_j c_j^0$, $\mathscr{A} = \{j | c_j^0 = c_{\max}^0\}$;

repeat

  $k = k + 1$;

  Compute the OLS output approximation using active set: $\mathbf{T}^k = \mathbf{H}_{\mathscr{A}}\boldsymbol{\beta}^k$ where $\boldsymbol{\beta}^k = (\mathbf{H}_{\mathscr{A}}^\top \mathbf{H}_{\mathscr{A}})^{-1}\mathbf{H}_{\mathscr{A}}^\top \mathbf{Y}$;

  The value γ^k is increased until another variable enters the active set (exact equations skipped). Order in which variables enter into the active set is the **variable ranking**;

  Update output approximation: $\mathbf{T}^k = \mathbf{T}^{k-1} + \gamma^k(\mathbf{T}^k - \mathbf{T}^{k-1})$;

  Update the solution, similarly as output approximation: $\boldsymbol{\beta}^k = \boldsymbol{\beta}^{k-1} + \gamma^k(\boldsymbol{\beta}^k - \boldsymbol{\beta}^{k-1})$;

  if k == $H_{\max}$ **then**

    exit ;

  Update active set $\mathscr{A}$;

until *k<H-1*;

3.2.4 Singular Value decomposition update

The advantage of solving ELM optimization problem by SVD of the matrix $\overset{(N\times H)}{\mathbf{H}}$ has been considered in Sec. 3.1.5. For the incremental or pruning ELMs it would be too costly to compute SVD on the full matrix each time a neuron is added. Therefore, it would be highly desirable to compute the SVD update after a new column is added to the matrix $\mathbf{H}$. The practical implementation of an SVD

Table 3.2. Calculation of intermediate values for SVD update

$\mathbf{m} = \boldsymbol{U}^\top \mathbf{h}$
$\mu = \|(I - \boldsymbol{U}\boldsymbol{U}^\top)\mathbf{h}\|$
$\mathbf{u}_{n+1} = (I - \boldsymbol{U}\boldsymbol{U}^\top)\mathbf{h}/\mu$

update algorithm has been considered in publication IV. Here we consider core ideas of an SVD update. The notation is slightly different than in publication IV.

Besides application to ELM, the SVD update algorithm is of much broader benefit; SVD is a very common computational tool used in many systems and algorithms. Hence, speeding it up by utilizing a previously computed SVD of a small matrix is an important contribution.

Suppose the SVD of a matrix $\mathbf{H}$ is known $\mathbf{H} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$. When a new column is added to the matrix $\mathbf{H}$, we can write:

$$\begin{bmatrix}\mathbf{H} & \mathbf{h}\end{bmatrix} = \begin{bmatrix}\mathbf{U} & \mathbf{u}_{n+1}\end{bmatrix} \begin{bmatrix}\mathbf{D} & \mathbf{m} \\ \mathbf{0}^\top & \mu\end{bmatrix} \begin{bmatrix}\mathbf{V}^\top & \mathbf{0} \\ \mathbf{0}^\top & 1\end{bmatrix}, \tag{3.10}$$

where $\mathbf{u}_{n+1}$ is the new (n+1)-th column of matrix $\mathbf{U}$ orthogonal to all other columns. Matrix $\mathbf{D}$ is extended by the vector $\mathbf{m}$ and the scalar μ, and right singular vector matrix $\mathbf{V}^\top$ is extended by one on the diagonal. We are able to calculate all three new values $\mathbf{u}_{n+1}$, $\mathbf{m}$, μ from the column $\mathbf{h}$ and the known SVD components $\mathbf{U}$, $\mathbf{D}$, $\mathbf{V}^\top$. If we perform the multiplication in Equation (3.10) we obtain:

$$\mathbf{h} = \mathbf{U}\mathbf{m} + \mu\mathbf{u}_{n+1}. \tag{3.11}$$

Multiplying by $\mathbf{U}^\top$ and using orthogonality of columns of $\mathbf{U}$, we get:

$$\mathbf{U}^\top\mathbf{h} = \mathbf{U}^\top\mathbf{U}\mathbf{m} + \mathbf{U}^\top\mathbf{u}_{n+1}\mu = \mathbf{m}. \tag{3.12}$$

Vector $\mathbf{u}_{n+1}$ can be calculated as the projection of $\mathbf{h}$ on the subspace orthogonal to columns of $\mathbf{U}$, so $\mathbf{u}_{n+1} = \mathbf{h} - \mathbf{U}\mathbf{U}^\top\mathbf{h} = (I - \mathbf{U}\mathbf{U}^\top)\mathbf{h}$. This follows from the fact that the column subspace of $[\mathbf{H},\mathbf{h}]$ must be the same as the column subspace of $[\mathbf{U},\mathbf{u}_{n+1}]$ and that $\mathbf{u}_{n+1}$ must be orthogonal to all other columns of $\mathbf{U}$. In addition, we need to normalize the vector $\mathbf{u}_{n+1}$ so that it has unit norm. We leave constant the multiplication $\mu\mathbf{u}_{n+1}$, assign $\mu = \|\mathbf{u}_{n+1}\|$ and normalize as $\mathbf{u}_{n+1} = \mathbf{u}_{n+1}/\mu$. If the new column belongs to the same subspace as existing columns then μ is close to zero and the equations change but not principally. Therefore, the unknown components on the right hand side of Equation (3.10) can be calculated according to the Table 3.2.

After the decomposition in Equation (3.10) is done, we can proceed further and compute the SVD of the matrix $\begin{bmatrix} \boldsymbol{D} & \mathbf{m} \\ \mathbf{0}^\top & \mu \end{bmatrix} = \boldsymbol{U}_1 \boldsymbol{D}_1 \boldsymbol{V}_1^\top$. When this is completed the final SVD of $[\boldsymbol{H}, \mathbf{h}]$ is:

$$\begin{bmatrix} \boldsymbol{H} & \mathbf{h} \end{bmatrix} = (\boldsymbol{U}\boldsymbol{U}_1)\boldsymbol{D}_1(\boldsymbol{V}\boldsymbol{V}_1)^\top. \tag{3.13}$$

Therefore the issue of computing the SVD of the extended matrix $[\boldsymbol{H}, \mathbf{h}]$ is reduced to the computation of the SVD of matrix $\boldsymbol{B} = \begin{bmatrix} \boldsymbol{D} & \mathbf{m} \\ \mathbf{0}^\top & \mu \end{bmatrix}$. This is a diagonal matrix augmented by one column $\begin{bmatrix} \mathbf{m} \\ \mu \end{bmatrix}$. If we multiply the matrix $\boldsymbol{B}$ by its transpose, we derive:

$$\boldsymbol{B}\boldsymbol{B}^\top = \begin{bmatrix} \boldsymbol{D} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} + \begin{bmatrix} \mathbf{m}^\top & \mu \end{bmatrix} \begin{bmatrix} \mathbf{m} \\ \mu \end{bmatrix} = \boldsymbol{U}_1 \boldsymbol{D}_1^2 \boldsymbol{U}_1^\top. \tag{3.14}$$

Since $\boldsymbol{D}$ is a diagonal matrix, we see that matrix $\boldsymbol{B}\boldsymbol{B}^\top$ is a symmetric matrix plus a rank one matrix. Also, we see that by finding eigenvalues and eigenvectors of $\boldsymbol{B}\boldsymbol{B}^\top$, we also obtain $\boldsymbol{U}_1$ and $\boldsymbol{D}_1$. Matrix $\boldsymbol{V}_1$ can also be calculated by the simple transformation of the singular vectors in $\boldsymbol{U}_1$. In [Golub and Van Loan, 1996, p. 442], Theorem 8.5.3 demonstrates how to efficiently find eigenvalues and eigenvectors of the matrix $\boldsymbol{B}\boldsymbol{B}^\top$. Further details are provided in publication IV.

Computational complexity of the SVD update

Computation of the intermediate variables presented in Table 3.2 takes $2NH$ FLOPS ignoring the low-order terms, provided computations are organized properly.

Finding the SVD of a square matrix $\overset{(H\times H)}{\boldsymbol{B}}$ costs $\mathcal{O}(H^2)$ where the constant factor is impossible to compute because the method is iterative. Finally, computing the whole decomposition according to Equation (3.13) requires $NH^2 + H^3$ operations. Comparing with the computational complexity of the regular SVD algorithm, the leading term scales similarly as $\mathcal{O}(NH^2)$, but the constant factor is lower for the SVD update. Experiments in the next section confirm this analysis.

Sequential SVD update

There are situations when the new columns (or rows) are provided sequentially (one-by-one) and we need to update the SVD many times. For this setting, the authors in [Brand, 2006] suggest a scheme that keeps SVD decomposed into five

matrices as in Equation (3.13).

There are two reasons for doing this. The first reason is valid when we are interested in the r-rank SVD approximation to the matrix, not in the complete SVD. In this case, keeping five matrices and updating them provides some computational advantage.

The second reason applies when we do not need the SVD after each new column (or row) arrival but need the SVD after every k-th ($k > 1$) column arrival. In this case, the expensive outer matrix multiplication is done every k-th update, while with every new column arrival smaller intermediate matrices are updated and stored.

Experimental demonstration of SVD update.
The SVD update algorithm with sequential updating, and the basic re-orthogonalization have been implemented in Python language using Numpy (version 1.8.2) and Scipy (version 0.14) packages [Jones et al., 2001]. The SVD update algorithm is implemented purely in Python using Scipy, except the part which computes roots of the secular equation used in the computation of SVD of $\boldsymbol{B}$ in (3.14).

The regular SVD is computed by the function *svd* from the Scipy package. Intrinsically, the *svd* function calls the subroutine *DGESDD* from LAPACK which applies the divide-and-conquer method [Golub and Van Loan, 1996, p. 445] to compute the SVD. LAPACK dates back to the 90s, is widely used and is a highly optimized library for numerical linear algebra [Anderson et al., 1990]. Originally it was written in Fortran and smooth interfaces to the C language exist. Other numerical packages like Scipy, Matlab or R typically just call functions from LAPACK. Therefore, the comparison with LAPACK's SVD is unfavorable for our algorithm because Python code is known to be noticeably slower than compiled Fortran or C code. Moreover, LAPACK has been optimized over two decades by many experts while our algorithm is only a prototypical Python implementation.

In the first experiment, we demonstrate the general feasibility of the SVD updating. We compute the regular SVD of a random matrix with ten thousand rows and a varying number of columns. Then we append a randomly generated column to this matrix. Our goal is to compute the SVD of the augmented matrix. The first option is to compute the SVD of the full matrix. The second option is to utilize the SVD update method. The computational times of these two approaches are shown on the Figure 3.2. From the figure, we can see that the SVD update is quite attractive in comparison to pure SVD. This follows from observing the computational time and scaling factor of both algorithms.

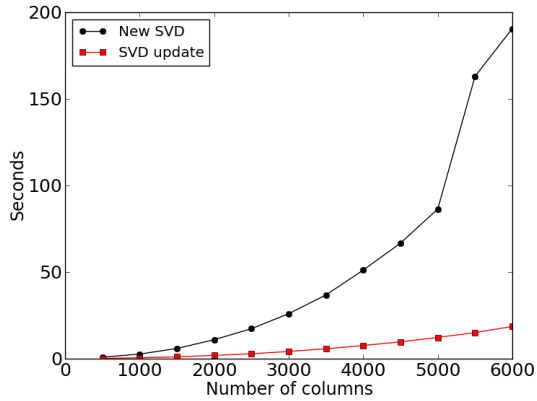


Figure 3.2. Computational time of *SVD update* and *new SVD* for random matrices with 10000 rows and various number of columns.

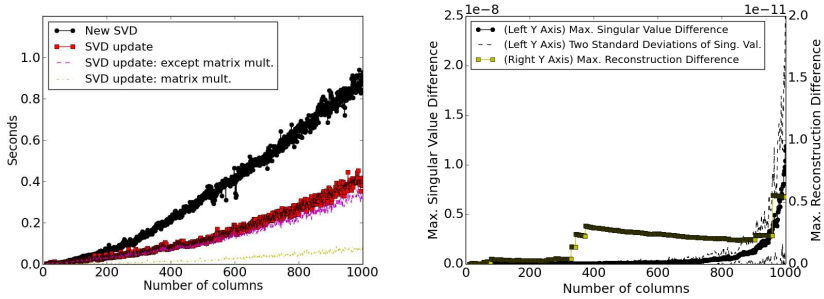
In the next experiment, we investigate the sequential SVD update algorithm described above and report its accuracy. This experiment is also initialized with a random matrix with ten-thousand rows, and columns are appended. However, in this experiment, we initialize with ten columns and incrementally append a new random column on each iteration until there are one thousand columns. When the number of columns is less than the number of rows, it is called a *thin* matrix. In the experiment with a *thin* matrix the next column is taken with probability 5% from the current column subspace and in 95% cases is generated randomly. When the column is randomly generated, with very high probability it has a component orthogonal to the current column subspace. Actually, these two cases differ slightly in implementation and experiments are designed to test both possibilities.

The results of these experiments are shown on Figures 3.3: Computational time on the left and accuracy on the right. Two measures of accuracy are reported: the maximum absolute relative singular value difference and the normalized norm difference of the true and reconstructed matrices. The true singular values are computed by the standard SVD. So, for each number of columns in the matrix (on every step) the following quantities are given as a measure of inaccuracy:

$$\Delta_1^{err} = max\left[\frac{|\sigma[i] - \sigma^{true}[i]|}{\sigma^{true}[i]}\right]. \tag{3.15}$$

$$\Delta_2^{err} = \frac{\|A^{reconstructed}\|_2 - \|A^{true}\|_2}{\sigma_{max}^{true}}. \tag{3.16}$$

The left figure showing computational time and, additionally, shows two constituent components of the SVD update: in yellow the final matrix multiplication



(a) Computational time of sequential *SVD update*, its components and *new SVD*. There are two components of *SVD update*: final matrix multiplication and all the rest.

(b) Maximum absolute relative singular value differences and normalized matrix norm difference for *SVD update*.

Figure 3.3. Sequential *SVD update* compared to the *new SVD* calculation for a *thin* matrix. Number of matrix rows is 1000.

in (3.13), in magenta the rest of the computations. Matrix multiplication scales as NH^2 which is in accordance with experimental results.

From these figures, we see that the computational time of the *thin* matrix SVD update grows slower than the time of the regular SVD.

The maximum absolute relative singular value difference is shown in Figure 3.3(b): the left Y-axis. It grows steadily and after 990 column updates reaches the value 1.1×10^{-8}. This error is very reasonable for application to OP-ELM, which we are going to describe further. We have also estimated the inaccuracy of these singular value errors (Equation (3.15)) by performing the experiment ten times and measuring the standard deviation of the singular values inaccuracy. The stochasticity appears because of the randomly generated matrices in this experiment. Dashed lines on the figure correspond to the mean (relative singular values difference), plus and minus two standard deviations. The normalized norm difference is depicted in Figure 3.3(b): the right Y-axis. It also does not grow high and does not exceed 1×10^{-11}. Further experimental details are provided in publication IV.

3.2.5 (Inc)-OP-ELM

Based on the SVD update algorithm described above, in the same publication, IV, the improvements to OP-ELM and TROP-ELM have been developed. The only difference between OP-ELM and TROP-ELM is that in TROP-ELM Tikhonov regularization is used in the calculation of the LOO in order to select the optimal number of neurons, and to solve the network's output weights. We present our

method for OP-ELM as its extension to TROP-ELM is straightforward. The OP-ELM algorithm is presented in Alg. 6.

Algorithm 6: OP-ELM

Input : 1) Training set $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$

2) **Maximal** number of neurons $H_{\max}$

Output: 1) Selected number of neurons H

2) Neuron weights of the network

3) The optimal weights $\boldsymbol{\beta}$

Generate $H_{\max}$ neurons as in standard ELM (Alg. 3);

Compute the ranking of neurons using MRSR (Alg. 5);

repeat

    Remove the less important neurons;

until *LOO starts to grow*;

Two modifications are proposed for the OP-ELM computation:

1. Avoid running MRSR until completion; MRSR selects individual variables (neurons) sequentially according to their LOO significance. Stop the algorithm when the LOO starts increasing.

2. Insert SVD update into the computations in MRSR and LOO . When a new neuron is added, the new SVD is computed using the previously computed SVD .

We call this new method of training OP-ELM - *Incremental* OP-ELM (or *(Inc)-OP-ELM*). This is an incremental algorithm for growing the network. The ideas and motivation are the same as for OP-ELM.

The original OP-ELM and (Inc)-OP-ELM have been compared experimentally in publication IV. For that purpose, we have taken 5 regression datasets from the UCI repository Dheeru and Karra Taniskidou [2017] with a large amount of training data. The summary of the datasets is presented in Table 3.3. Each dataset has only one output (response) variable.

Both OP-ELM and (Inc)-OP-ELM are completely implemented in Python using Numpy and Scipy. Experiments have been conducted on the same personal computer with Intel® Xeon(R) CPU E31230 @ 3.20GHz (8 cores, 4 floating point cores) processor and 8 gigabytes of RAM. The summary of the results is given in Table 3.4.

Accuracy is measured as Normalized Mean Square Error (NMSE) that is Mean Squared Error divided by the variance of the output variable. As we see from Table 3.4, accuracy is the same for OP-ELM and (Inc)-OP-ELM. The numbers in

Table 3.3. Characteristics of regression datasets used for the comparison of Inc-(OP)-ELM and OP-ELM (From Publication IV)

	Delta Elevators	Delta Ailerons	Banks	CASP	California Housing
Number of Variables	6	5	8	9	8
Training Set size	6344	4752	2999	30487	13760
Test Set size	3173	2377	1500	15243	6880

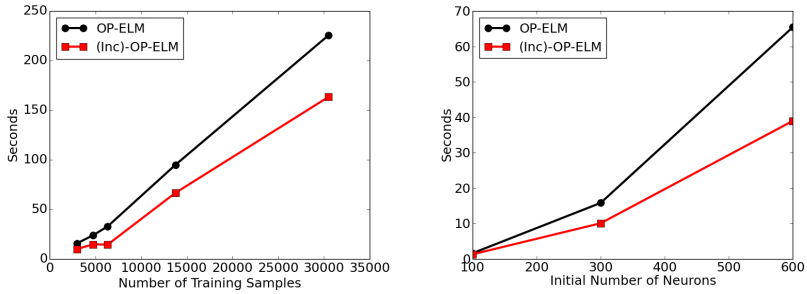
Table 3.4. Summary of computational times and accuracy of (Inc)-OP-ELM and OP-ELM (From Publication IV)

	Models	Datasets				
		Elevators	Ailerons	Banks	CASP	California Housing
Comp. time	(Inc)-OP-ELM	14.33 ± 2.90	14.23 ± 3.38	10.07 ± 1.58	163.33 ± 3.62	66.67 ± 5.55
	OP-ELM	32.47 ± 0.93	23.41 ± 1.1	15.43 ± 0.62	225.46 ± 1.42	94.94 ± 2.61
NMSE	(Inc)-OP-ELM	0.36 ± 0.01	0.30 ± 0.02	0.05 ± 0.001	0.56 ± 0.01	0.25 ± 0.006
	OP-ELM	0.36 ± 0.01	0.30 ± 0.02	0.05 ± 0.002	0.56 ± 0.01	0.25 ± 0.006

the table are rounded to two digits after the comma while not rounded NMSE may differ slightly in one or another model's favor. The running time of training depends on several factors, the most important of which is the number of training samples, the initial number of neurons and the step of LOO evaluation. These parameters have been set equally for both OP-ELM and (Inc)-OP-ELM for a fair comparison. In particular, a step of LOO is evaluated after every five added neurons. The initial number of neurons is 300.

Regarding the computational time, we see that (Inc)-OP-ELM is noticeably faster than OP-ELM. Computational time with the same data is shown in Figure 3.4(a). We see that (Inc)-OP-ELM scales better with the growing number of training samples. The training time for the dataset with 6344 samples is even less than for the dataset with 4752 samples. This can be explained by the smaller number of neurons selected as relevant for the larger dataset. Note, that in OP-ELM model all neurons must be ranked first.

In Figure 3.4(b) scaling of both algorithms with respect to the initial number of neurons is given. This result is for the Banks dataset. Again, here we see the computational advantage of (Inc)-OP-ELM over OP-ELM.



(a) Scaling with respect to the number of training samples

(b) Scaling with respect to initial number of neurons (Banks dataset)

Figure 3.4. Scaling of (Inc)-OP-ELM and OP-ELM (From Publication IV)

3.3 Time series prediction with ELM

Time series modeling and prediction is one of the oldest topics in statistics. The very first statisticians dealt with time-dependent data. For example, Beveridge wheat price (years 1500 to 1869) or Wolfer's sunspot number (years 1610-1960) [Yaglom, 1987] are examples of very early time series. Nowadays, time series analysis and forecasting are ubiquitous in many fields of science and engineering. Econometricians, physicists, statisticians, biologists, climatologists etc. encounter time-dependent data in their daily work. Time series is also considered in the next chapter, however here we consider ELM applied to time series prediction. ELM has been applied to time series prediction in publications III and V.

3.3.1 General overviews of time series prediction

Since time series prediction arises so frequently in applications, a large number of methods has been developed for this task. A relatively recent overview of various methods and future directions is given in [Gooijer and Hyndman, 2006]. Historically, statistical linear methods dominated in Time Series Prediction (TSP). In particular, Auto Regressive Integrated Moving Average (ARIMA) based modeling became widely adopted after the remarkable book [Box and Jenkins, 2008]. The complete methodology for a model selection, parameter optimization, and prediction was introduced there and it is still widely used. ARIMA models time series (or it's differences) as a linear combination of previous values of time series and previous values of noise (often called innovations). However, real time series come from many different sources and have very different properties. So it is obvious that there is no single best approach for time series modeling.

Not surprisingly, other methods which may outperform classical methods, have emerged.

Neural Network (NN) methods have attracted significant attention for time series prediction problems [Crone et al., 2011]. Neural Network (NN)s are general nonlinear regression techniques which can be applied to time series. They are able to relax some assumptions made by classical methods, e.g. model linearity and Gaussian distribution of noise. In contrast to ARIMA(p,n,q) models where the fine tuning of model hyperparameters - (p,n,q) is required for obtaining good forecasts, neural networks avoid this complication. Therefore, the way the forecasting process is done may be changed. Instead of many hours of work of a statistician (quite often with a domain knowledge) trying to select the right model and adjust hyperparameters, a modeler without a domain knowledge is able to apply NN and obtain competitive results. There are situations where intensive human involvement or large computational time is not affordable. Neither we nor other authors claim that neural networks are generally a better method than classical statistical methods, but they and other computational intelligence methods have shown their viability [Crone et al., 2011].

In time series prediction one can distinguish one-step-ahead prediction and long-term prediction. As is clear from these names, in one-step-ahead prediction interest constitutes only the estimation of the next single value, while in the long-term prediction estimations of multiple future values are of interest. Quite often researchers address these problems separately [McElroy and Wildi, 2013], [Ben Taieb et al., 2012] since an accumulation of errors and increasing uncertainties [Sorjamaa et al., 2007] make long-term prediction an inherently more difficult problem. In this dissertation, we mostly consider the long-term time series prediction.

There exist three universal strategies for long-term time series prediction: Recursive, Direct and DirRec strategies. Recently, another strategy [Bontempi and Taieb, 2011] was introduced but it has not been studied in this dissertation. A detailed description of prediction strategies is given in the next section. Strategies differ in how we estimate future values using the past values. There is no definite indication of superiority of one strategy over the others, as has been shown in [Ben Taieb et al., 2012],[Sorjamaa et al., 2007].

Earlier works have shown that variable selection is needed to improve the accuracy of long-term predictions. For instance, it has been shown [Sorjamaa et al., 2007] that DirRec strategy with the forward-backward variable selection and K-Nearest Neighbours (K-NN) model is beneficial in terms of accuracy. Some models can deteriorate in performance with the inclusion of unimportant

variables (features), for example, k-NN [Hastie et al., 2009]. Variable selection methods can be very time consuming especially if we consider the wrapper class of methods [Sorjamaa et al., 2007]. Thus, the motivation for publication III is the desire to avoid computationally expensive variable selection.

3.3.2 Strategies for long-term time series prediction

There are three main strategies for long-term time series prediction.

Recursive strategy

The Recursive strategy for long-term time series prediction is a simple and intuitive strategy. The goal is to build the model which estimates the next value by using r previous values. Here r, which is the regressor size, is a hyperparameteter of a model, and can be determined via cross-validation or other methods for selection of hyperparameters. Thus, on the first step the model computes the following estimation:

$$\hat{y}_{t+1} = f(y_t, y_{t-1}, \dots y_{t-r+1}) \tag{3.17}$$

To predict the second value, the first predicted value is introduced into the model:

$$\begin{aligned} \hat{y}_{t+2} &= f(\hat{y}_{t+1}, y_t, \dots y_{t-r+2}) \\ &\vdots \\ \hat{y}_{t+r+1} &= f(\hat{y}_{t+r}, \hat{y}_{t+r-1}, \dots \hat{y}_{t+1}) \end{aligned} \tag{3.18}$$

The process can be continued until we predict as many values as needed. It is clear that prediction of $t+r+1$-th value is based only on estimations $\hat{y}_{t+r}, \cdots \hat{y}_{t+1}$, and does not depend on any original values of the time series data. Since each prediction has some error, errors accumulate with the increase of the prediction steps.

Direct strategy

In the Direct strategy, the regressor size r is also a hyperparameter of the model. The goal is to directly predict p steps ahead using the same set of regressors $y_t, y_{t-1}, \dots y_{t-r+1}$. Later in this article p is called prediction horizon. Hence, for

every next future value $\hat{y}_{t+k}$, training of a separate model is required:

$$\begin{aligned}
\hat{y}_{t+1} &= f_1(y_t, y_{t-1}, \dots y_{t-r+1}) \\
\hat{y}_{t+2} &= f_2(y_t, y_{t-1}, \dots y_{t-r+1}) \\
&\vdots \\
\hat{y}_{t+p} &= f_p(y_t, y_{t-1}, \dots y_{t-r+1})
\end{aligned} \tag{3.19}$$

It is clear that predictions are always based on true values of time series, but the time lag between regressors and prediction value grows with p. This often causes a gradual growth of prediction error. However, the Direct strategy is usually more accurate than Recursive [Sorjamaa et al., 2007].

DirRec strategy

The DirRec strategy [Sorjamaa and Lendasse, 2006] combines both Recursive and Direct strategies. The number of regressors is no longer constant. On the first step, DirRec strategy coincides with the Direct strategy, then all predicted values serve as additional regressors and the order of the model grows:

$$\begin{aligned}
\hat{y}_{t+1} &= f_1(y_t, y_{t-1}, \dots y_{t-r+1}) \\
\hat{y}_{t+2} &= f_2(\hat{y}_{t+1}, y_t, y_{t-1}, \dots y_{t-r+1}) \\
&\vdots \\
\hat{y}_{t+p} &= f_p(\hat{y}_{t+p-1}, \dots, \hat{y}_{t+1}, y_t, y_{t-1}, \dots y_{t-r+1})
\end{aligned} \tag{3.20}$$

As in the Direct strategy, for every future prediction, the corresponding model needs to be trained. So, the complexity of the training is proportional to the number of values to be predicted p. It has been shown [Sorjamaa and Lendasse, 2006] that in general DirRec strategy with variable selection has superiority over the other two strategies when the model f is nonlinear.

The goal of publication III is to show that this statement holds without variable selection when the model f is an OP-ELM. The motivation for this is that OP-ELM intrinsically performs a variable selection in a hidden space.

3.3.3 Experimental evaluation of OP-ELM with different prediction strategies

In the original publication III the experiments are done for three sets of time series data. In this chapter, the complete results are presented only for one time series, however, the description of all three is included. Sea-water temperature [Corona and Lendasse, 2007], Sun Spots [Center, 2012] and Santa Fe

Sun spots time series				
	Linear Model	**LS-SVM**	**Mean and std of 100 independent OP-ELMs (ensemble)**	**Ensemble of OP-ELMs (Average)**
	Regressor size = 28, prediction horizon = 12			
Recursive	496.811	1661.7 1705.574±23.090	491.047 ± 11.2947	471.874
Direct	**493.389**	**1413.4** 1464.967±29.881	487.127 ± 5.908	**456.372**
DirRec	493.485	1764.5 1804.640±20.696	**482.166 ± 4.494**	467.993
	Regressor size = 28, prediction horizon = 24			
Recursive	785.610	2063.3 2053.930±30.014	748.210 ± 30.327	716.536
Direct	**772.982**	**1527.3** 1570.166±23.591	739.334 ± 8.893	**692.122**
DirRec	773.778	2068.1 2086.731±16.874	**734.116 ± 8.245**	713.702
	Regressor size = 28, prediction horizon = 28			
Recursive	891.363	2206 2170.886±45.850	832.184 ± 39.071	791.281
Direct	**874.878**	**1554** 1595.346±29.829	825.926 ± 11.614	**773.803**
DirRec	876.144	2149.8 2165.569±15.847	**824.160 ± 10.671**	801.817

Table 3.5. Mean Square Errors(MSE) for Sun spots dataset. Different regressor sizes and prediction horizons are considered. Results of different models are given in column-wise. In bold font best MSEs for each column (and each regressor size and prediction horizon) are presented. In small font bagging results for LS-SVM are presented. (Publication III)

A [Gershenfeld and Weigend, 1994]. The first one is a weekly measurement of sea water temperature during several years, there are 875 measurements in total. The second is one of the oldest time series in history; it provides monthly averages of a number of dark spots on the sun from the year 1749 until 2012, there are 3161 measurements in total. Santa Fe A is a dataset recorded from a far-infrared-laser in a chaotic state and it is explicitly divided into a training set (1000 points) and test set (9093 points). We would like to emphasize that these time series datasets are taken from completely different domains, so our method is applied to time series with completely different properties and behavior.

Usually, in time series prediction the number of regressors to use is unknown and it has to be estimated. For the Sun Spots dataset number of regressors equals 28 and is estimated by the following procedure; a linear model is trained for a various number of regressors and the number with minimal leave-one-out validation error is taken.

Predictions are calculated for each subsequence of a test set where the length

is equal to the regressor size. In other words, for a regressor size r, for each $k = 1,...,r$ consecutive values of a test set, the predictions are calculated up to the prediction horizon. For a certain number of prediction steps ahead, Mean Squared Error (MSE) are averaged over all subsequences of size r, and finally obtained MSEs are averaged over all numbers of steps ahead up to the prediction horizon. Therefore, for an experiment with a single OP-ELM (or linear model) one number is obtained - twice averaged MSE which characterizes the prediction accuracy.

Least-Squares Support Vector Machine (LS-SVM) is a competitive technique which has been intensively used for nonlinear modeling Suykens [2002]. Experiments have been performed by a famous LS-SVM Matlab Toolbox Pelckmans et al. [2002]. LS-SVM has an advantage that training converges to solving a linear system in dual space. So, it is especially interesting to compare it against our method. However, for a good performing model, hyperparameters need to be tuned. There are two hyperparameters C - for regularization and σ - for the Gaussian kernel. They have been adjusted by ten-fold cross-validation and a grid search as implemented in the toolbox. In addition, bagging Breiman [1996] with 50 bootstrap samples is applied to LS-SVM and results are listed in the same column in the smaller font. Bagging for the linear model has been tried but the performance is very close to the complete data performance as mentioned in the original bagging paper, therefore they are not shown here.

Results of experiments are given in Table 3.5. Due to the random generation of neurons in the OP-ELM model, many instances of OP-ELMs need to be run in order to estimate its performance. For every set of parameters 100 OP-ELM s are run and the MSE is computed for each of those. Averages and standard deviations of these MSEs are presented in the third column of the table. In addition, arithmetic mean between forecasts of 100 OP-ELM s and its MSE are calculated and depicted in the fourth columns. This is called *ensemble method* van Heeswijk et al. [2011].

Experimental conclusions

- Average MSE of DirRec strategy (e.g. third column of Table 3.5) is better than the best MSE among all strategies for linear ordinary least squares model.

  This statement holds for all time series under investigation and all sets of parameters, except for one experiment.

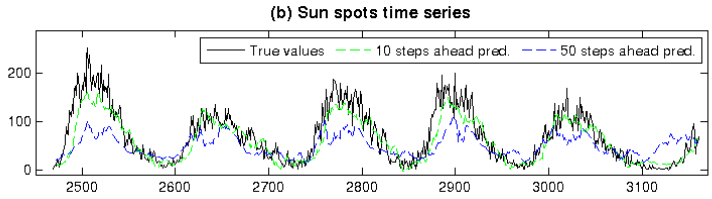


Figure 3.5. Visualization of predictions from ensemble of OP-ELMs. Ten steps ahead predictions as well as fifty steps ahead predictions are plotted for Sun Spots time series. Regions for predictions are taken from the end of each time series and consist of aroud 700 points. Regressor size - 28. (Publication III)

For the Santa Fe time series, Direct strategy significantly outperforms other strategies, the standard deviation of DirRec strategy is less than standard deviations of other strategies. This indicates that in a single run OP-ELM with DirRec strategy tends to be the most accurate.

- For other strategies, there are no such straightforward results as in the previous item.

  Comparing only OP-ELM with three strategies, it is seen that there exist cases where each one of them is the best. Thus, DirRec is not generally the best strategy but is it almost always better than the best linear model.

- LS-SVM outperforms OP-ELM only for Santa Fe time series.

  This is a highly nonlinear time series which can be seen from Figure 3.5. For the other two, time series performance of LS-SVM is significantly worse, even worse than the linear model. Therefore, in contrast to OP-ELM, LS-SVM, in general, is not able to perform well without a variable selection procedure and is a bad model for unfavorable time series. In any case, in the next section, it is shown that computational time for LS-SVM is several times longer than OP-ELM.

- Using an ensemble method can improve the results dramatically.

- OP-ELM has a fast computational time which is a highly desirable property for some applications.

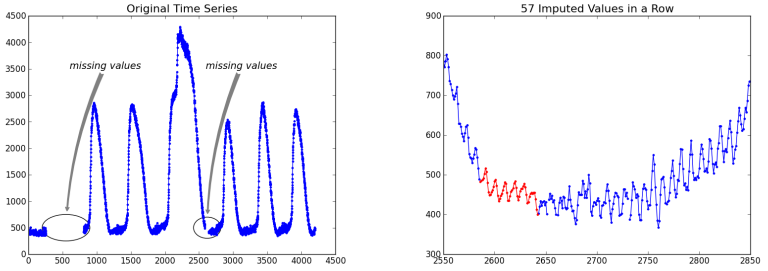
3.3.4 Example of predicting star photometric outbursts

Time series prediction with extreme learning machines has also been applied in publication V. In this work the long-term forecasting of photometric observations of the star V363 Lyr has been performed. The original data obtained by the KEPLER satellite is depicted in the Figure 3.6 (a). It covers roughly 86 days and contains 4201 measurements.

The increased resolution of these photometric observations in contrast to the previous studies has revealed different patterns from other stars of this type. For example, the variable periodicity of the outbursts and one super outburst which is present in the current data raise doubts on the previous classification of the star V363 Lyr. There is also a presence of the short-term periodicities which are visible in Figure 3.6 (b).

In this work properties of this time series have been analyzed with respect to predictability. It is demonstrated that a black-box model, like a neural network, can predict the future outbursts more accurately then than the predictions made by the periodic predictions which follow from the spectral analysis.

The time series contains missing values. There are also long missing-value gaps in the data which are indicated in the Figure 3.6. Before time series forecasting the missing values have been imputed by a Gaussian Mixture model method [Eirola and Lendasse, 2013].

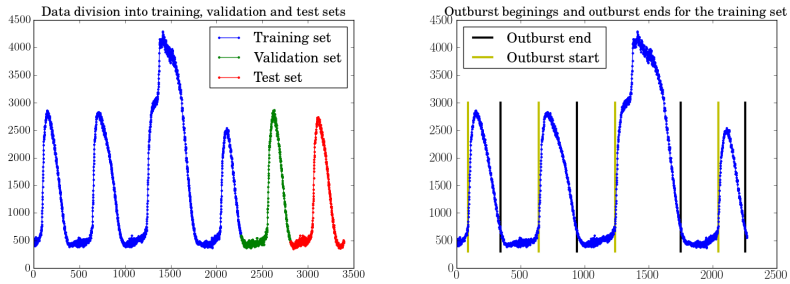


(a) Original time series **(b)** Example of the longest imputation

Figure 3.6. Photometric time series of the star V363 Lyr (Publication V)

Forecasting methodology

The main goal of this work is to forecast the next outburst of the time series. We define the start of an outburst as the point where the time series exceeds 750. In the forecasting setup, we assume that the last outburst has just ended and the starting point of forecasting is the value in the *valley* of the time series. In other words, the starting point of forecasting can not be on the previous outburst. We have divided our time series into three sets: training, validation, and test; this division is shown in Fig. 3.7 (a). We want to estimate how well we can predict the outburst depending on the starting time point of forecasting. For instance, consider the validation set. First, we assume that no points of the validation set are observed and we try to forecast the next outburst. Then we assume that only one point of the validation set is observed and again forecast the outburst. Finally, we assume that all points in the validation set before the outburst are observed and we check whether our forecasting method can predict the outburst in the next point in time. The marks where the outbursts start and end on the training set are drawn in Fig. 3.7 (b).



(a) Data division into training, validation and test sets

(b) Outburst beginnings and outburst ends for the training set

Figure 3.7. Time series forecasting setup (Publication V)

There are two different approaches to forecasting the next outburst which we investigate in this paper:

- **Directly predict the time point where the next outburst happen.** This can be done by building a regression model between the last observed time window of length d (this is called *regressor size*) and the position of the next outburst. This approach we further divide into two sub-methods:
  - Explicitly include in the model a variable whose value is the distance from the end of the previous outburst; if the time series is strongly periodic then this variable alone may provide a very good estimation of the next outburst.

Table 3.6. MAE of outburst position forecast

Periodicity variable	**Data**	**Linear Model**	**OP-ELM**	**TROP-ELM**	**Random Forest**
Excluded	**Valid.**	80.37	$44.69 \pm (1.05)$	$43.59 \pm (0.87)$	$\mathbf{42.42 \pm (0.002)}$
	Test	66.86	$35.28 \pm (1.38)$	$32.27 \pm (0.95)$	$\mathbf{19.76 \pm (0.13)}$
Included	**Valid.**	**3.27**	$3.41 \pm (0.05)$	$3.62 \pm (0.05)$	$3.53 \pm (0.01)$
	Test	**29.0**	$29.66 \pm (0.04)$	$29.33 \pm (0.05)$	$29.25 \pm (0.02)$

  - Do not include this *periodicity variable* and build a regression model where regressors are just the previous values of the time series.

- **Conduct time series prediction and monitor where the predictions indicate an outburst.** At first, this seems to be a more complex problem because we forecast not only the position of outburst but also the time series values. However, as shown later, this approach may be more beneficial than the first one under certain conditions.

Forecasting the outburst position directly

The regression models we compare are: Linear regression with Tikhonov Regularization, OP-ELM, TROP-ELM . Also, we compare Random Forests (RF) as a state-of-the-art nonlinear regression model.

The error measure for forecasting we compute in the following way. For both validation and test sets, we know the position of true outburst. So, for every starting point of forecasting, a regression model predicts the position of the next outburst and we compute the absolute error between the prediction and the true outburst. Since the starting point roll from the beginning of the validation (or test) set up until one point before the true outburst, we average all the predictions. Therefore, the error is the Mean Absolute Error (MAE). Because all regression models except the linear model involve randomness we repeat the experiment 100 times (20 for Random Forest) and average the result.

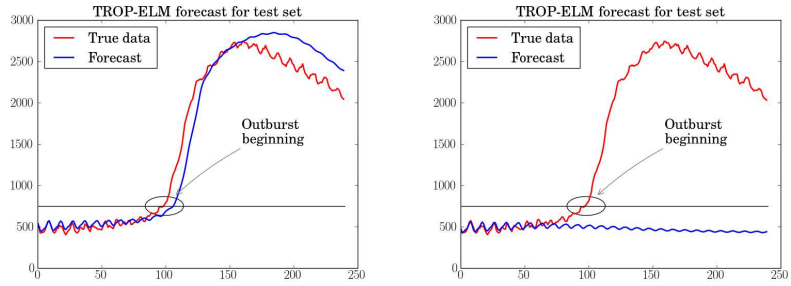
Hyperparameters are optimized on the validation set. Hyperparameters of the Linear model, OP-ELM and TROP-ELM are the regressor size (time delay embedding dimension) and regularization parameter. Random Forests also requires the number of samples in leaves and the number of randomly selected variables used to build trees.

Looking at Table 3.6, one can notice that when the periodicity variable is included, the best model is the Linear model. This indicates that outbursts are quite periodic. However, test error is much larger than validation error because

the last outburst (which belongs to the test set) is closer to the previous outburst than the outburst distances in the rest time series. Hence, even though the outburst periodicity is quite apparent, it is not constant. When the periodicity variable is not included, the best model on validation and test sets is Random Forests. This agrees with the fact that it is a state-of-the-art regression model. It produces the lowest MAE on the test set – significantly lower than the linear model with the periodicity variable.

Forecasting the outburst by time series prediction

In this approach, time series prediction is performed and when the predicted values become larger than 750 (outburst detection threshold) we assume that the outburst is starting Fig. 3.8 (a). We can quantify the error between the true outburst position and forecast position similarly as in the previous case - by the absolute error of the difference of two positions. However, it may happen that predicted values are never larger than the threshold as shown in Fig. 3.8 (b). Therefore, we must treat this special case separately. So, we assume that the outburst forecasting error is infinity and denoted by *inf*.



(a) Outburst beginning is estimated (non-infinite error)

(b) Forecast does not indicate an outburst (infinite error)

Figure 3.8. Successful and unsuccessful time series forecasts

We construct a time delay embedding matrix of width 20 (20 was selected using the validation set) and for each time window we define the next time series value as a value to estimate. Using training data we make a regression model and then we use the *Recursive prediction strategy* 3.3.2. There are other long-term forecasting strategies such as *Direct* and *DirRec* but by experimenting on the validation set we observed that recursive strategy works the best for this time series. Regression models which are used for time series prediction are the same as in Section 3.3.4, namely *Linear*, OP-ELM, TROP-ELM and Random Forests. We also tested a K-NN approach. The TROP-ELM performed the best amongst the tested models (details are in publication V)

Comparison of direct outburst forecasting and forecasting by TS prediction

Table 3.7. Average outburst MAE

	average MAE (over all starting points)	**average MAE** (over last 30 starting points)
Direct outburst forecasting (Random Forest)	$\mathbf{19.76 \pm (0.13)}$	12.65 ± 0.17
Forecasting outburst by TS prediction (TROP-ELM)	26.86 ± 5.84	$\mathbf{4.14 \pm 1.37}$

We have selected the best models from each approach based on validation results. Now we want to compare these two approaches on the test set. It is worth mentioning again that the time series prediction approach can not provide outburst forecast at all for some starting points in which case can we assume that the forecasting error is infinite. So, the error we provide for this approach is calculated only for those values for which the forecast is available. The MAE of outburst forecasting is presented in Table 3.7.

The difference between the two error columns in Table 3.7 is that the first error column is an average MAE given all the starting points and the second error column is an average MAE given the previous 30 starting points which are adjacent to the true outburst position. From Table 3.7 we see that that direct forecasting of outburst position by Random Forests is more accurate than forecasting by time series prediction. However, the third column of Table 3.7 shows us that in the vicinity of the true outburst the time series forecasting model becomes more accurate with a noticeable gap.

4. Linear state-space models and Gaussian Processes

4.1 Introduction and motivation

In the previous chapter Time Series Prediction (TSP) with Extreme Learning Machines has been described. The latter is a nonlinear model and a time series is modeled in an auto-regressive fashion. In this chapter, we consider linear models for time series modeling and forecasting. Interestingly, the performance of linear models is not worse in practical forecasting. This has been demonstrated in an NN3 time series forecasting competition [Crone et al., 2011].

To achieve the accuracy of a nonlinear model, a linear model requires more careful data preprocessing and model selection efforts. Additionally, forming ensembles of linear models which are itself larger linear models is a fruitful way to improve forecasting accuracy.

The most famous and prominent linear forecasting methodology is Auto Regressive Integrated Moving Average (ARIMA) modeling [Box et al., 2008]. ARIMA is an extension of the ARMA model of a stationary random process [Hayes, 2009]:

$$y_t + \sum_{i=1}^{p} a_i y_{t-i} = \sum_{j=0}^{q} b_j \epsilon_{t-j} \quad .$$

The time series y_t is modeled as a linear combination of the previous values y_{t-i} and the previous (and current) values of the white noise ϵ_{t-j}. This is an ARMA(p, q) model. If the time series is not stationary, then its first difference is closer to be stationary [Box et al., 2008], the second difference is even closer and so forth. Auto Regressive Integrated Moving Average model can also be

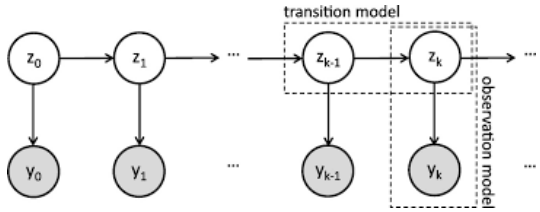


Figure 4.1. Graphical model of probabilistic state-space model

expressed in the state-space form:

$$\begin{aligned} \mathbf{z}_k &= \boldsymbol{A}_{k-1}\mathbf{z}_{k-1} + \mathbf{q}_k; \quad \text{where: } \boldsymbol{q}_k \sim \mathbb{N}(0, \boldsymbol{Q}_k); \quad \text{(state / transition equation)} \\ y_k &= \boldsymbol{H}_k\mathbf{z}_\mathbf{k} + \epsilon_k; \quad \text{where: } \epsilon_k \sim \mathbb{N}(0, \boldsymbol{R}_k); \quad \text{(measurement equation)} \end{aligned} \tag{4.1}$$

for some matrices $\boldsymbol{A}_k, \boldsymbol{H}_k, \boldsymbol{Q}_k, \boldsymbol{R}_k$. The state variables $\mathbf{z}_k$ are latent variables i.e. they are not observed. The observed variables are y_k. The corresponding probabilistic graphical model is presented in Figure 4.1. [Box et al., 2008] present ARIMA time series modeling and forecasting. In particular, initial pre-processing, model fitting, model evaluation and selection are described. The strength of the methodology is its completeness and the accuracy is state-of-the-art provided it is properly applied.

There are other strong advantages of the state-space approach. It is possible to handle missing data quite naturally and to model completely uneven sampled time series data. This is achieved by formulating the state-space model in continuous time. Examples of such approach are demonstrated in the next section. Another strength is the possibility to incorporate prior information like a trend, various periodicities and cycles into the model. These are called structural time series (STS) models [Harvey, 1990] and described in more detail below.

There exist several extensions of linear state-space models. For example, the Recurrent Neural Network (RNN) which is widely used in text and speech modeling is are a nonlinear generalization of (4.1). Non-Gaussian observational error ϵ_n is another popular modification. It allows modeling e.g. count data or binary data. Heavy-tailed distribution of ϵ_n makes the model more robust with respect to outliers. Nevertheless, linear state-space models serve as a basis for inference and learning in extended models.

Linear state-space models are also closely related to temporal Gaussian Process Regression (GPR) [Hartikainen and Särkkä, 2010]. In temporal Gaussian processes, the input is only one-dimensional - point in time. Temporal GPR with

almost any[1] covariance function is equivalent to a certain state-space model of the form (4.1). Matrices $\boldsymbol{A}_n, \boldsymbol{H}_n, \boldsymbol{Q}_n, \boldsymbol{R}_n$ are determined by the structure of the covariance matrix and values of the hyperparameters. The main point of converting temporal GPR to the state-space representation is that inference scales linearly with respect to the number of data points instead of cubic scalability for the standard temporal GP. Similar state-space transformation is also possible for spatio-temporal models [Särkkä et al., 2013]. Publication VII discusses the state-space representation of temporal GPR, expressed through the inverse of the covariance matrix which is sparse in this case.

Conversely, it is possible to convert state-space models to temporal GPR form. Popular state-space models are represented as a GP covariance function in publication VI. The motivation for this conversion is the construction of new covariance functions with the desire to reuse an available GP code, and new insights into the problem at hand.

4.2 Structural time series models

Auto Regressive Integrated Moving Average modeling approach and its success has been described in the introductory section. The ARIMA model can also be represented in the state-space form [Hayes, 2009]. The structural time series (STS) model can also be represented in the state-space form Eq. (4.1). The STS model is defined in terms of components which have a direct interpretation. These include, for instance, trend component, seasonal component, cyclic component etc.

Modeling the distinct components is a strength of the STS approach. It is up to the researcher to incorporate these components. The choice of which components to include depend on the prior knowledge available to the researcher and desired properties of the forecast. In contrast, the ARIMA approach is a *black box* where the model structure is determined only from the data, and there is no way to influence this structure.

4.2.1 State-space models

In this dissertation, the focus is on linear Gaussian state-space models. These are useful for time series modeling and forecasting of real valued time series and serve as a basis for nonlinear or non-Gaussian models (Sec. 4.1). The linear Gaussian state-space model has been stated in Eq. (4.1) and is repeated here for

[1] All standard covariance functions and their standard combinations

convenience:

$$\begin{aligned} &\mathbf{z}_k = \boldsymbol{A}_{k-1}\mathbf{z}_{k-1} + \mathbf{q}_k; \quad \text{where: } \boldsymbol{q}_k \sim \mathbb{N}(0, \boldsymbol{Q}_k); \quad \text{(state / dynamic equation)} \\ &y_k = \boldsymbol{H}_k\mathbf{z}_\mathbf{k} + \epsilon_k; \quad \text{where: } \epsilon_k \sim \mathbb{N}(0, \boldsymbol{R}_k); \quad \text{(measurement equation).} \end{aligned} \tag{4.2}$$

The variables $\mathbf{z}_k$ are latent variables and $\mathbf{y}_k$ - observed variables. A detailed overview of state-space models is given in [Särkkä, 2013].

The main inference tasks are computing of filtering and smoothing distributions $p(\mathbf{z}_k|y_1,\cdots,y_k)$ and $p(\mathbf{z}_k|y_1,\cdots,y_N)$ respectively. N is the total number of the observed data points. The classical algorithms to perform these inference tasks are the Kalman Filter (KF) and the Rauch-Tung-Striebel (RTS) smoother. Starting from the initial distribution $p(\mathbf{z}_0)$ the Kalman Filter sequentially computes $p(\mathbf{z}_{k+1}|y_1,\cdots,y_{k+1})$ given the distribution on the previous step $p(\mathbf{z}_k|y_1,\cdots,y_k)$. RTS smoother starts from the last filtering distribution $p(\mathbf{z}_N|y_1,\cdots,y_N)$ and computes in the reverse order all the distributions $p(\mathbf{z}_k|y_1,\cdots,y_N)$.

These models require hyperparameters. These can be some elements or full matrices $\boldsymbol{A}_k, \boldsymbol{H}_k, \boldsymbol{Q}_k, \boldsymbol{R}_k$, noise covariances $\boldsymbol{Q}_k, \boldsymbol{R}_k$ and/or mean and covariance of the initial state $\mathbf{m}_0, \boldsymbol{P}_0$. Standard approaches can be used for learning the hyperparameters: maximum likelihood, maximum a posteriori (MAP) or fully Bayesian. For any of these approaches the marginal likelihood (or just the likelihood) must be computed:

$$ll(\boldsymbol{\Theta}) = \log p(y_1,\cdots,y_N|\boldsymbol{\Theta}) = \sum_{k=1}^{N-1} p(y_k|y_1,\cdots,y_{k-1}) \tag{4.3}$$

Conveniently, in the linear Gaussian case, the marginal likelihood can be computed as a by-product of the Kalman Filter. Each term is the normalization constant of the corresponding update step of a Kalman Filter.

4.2.2 Combining state-space models

The structural time series framework is a way to construct state-space models and incorporate the desired properties or prior information into those. These properties are, for instance, fixed level, trend, periodicity, and quasi-periodicity (cyclicity) [Durbin and Koopman, 2012; Harvey, 1990]. A certain state-space model corresponds to each aforementioned property. These models can be combined additively. The multiplicative combination can be done similarly after a logarithmic transformation. Suppose that

$$y_k = z_k^{trend} + z_k^{periodic} + \epsilon_k, \tag{4.4}$$

so y_n is a sum of trend and periodic components. It is possible to write this combination as a single state-space model:

$$\begin{aligned}\begin{bmatrix}\mathbf{z}_k^{(tr)}\\ \mathbf{z}_k^{(per)}\end{bmatrix} &= \begin{bmatrix}\boldsymbol{A}_{k-1}^{(tr)} & 0\\ 0 & \boldsymbol{A}_{k-1}^{(per)}\end{bmatrix}\begin{bmatrix}\mathbf{z}_{k-1}^{(tr)}\\ \mathbf{z}_{k-1}^{(per)}\end{bmatrix} + \begin{bmatrix}\mathbf{q}_k^{(tr)}\\ \mathbf{q}_k^{(per)}\end{bmatrix}\\ y_k &= \begin{bmatrix}\boldsymbol{H}_k^{(tr)} & \boldsymbol{H}_k^{(per)}\end{bmatrix}\begin{bmatrix}\mathbf{z}_{k-1}^{(tr)}\\ \mathbf{z}_{k-1}^{(per)}\end{bmatrix} + \epsilon_k\end{aligned} \tag{4.5}$$

It is clear that $\mathbf{z}_k^{(tr)}$ and $\mathbf{z}_k^{(per)}$ are uncorrelated random processes if their noise terms are uncorrelated. In this case the covariance function of y_n is:

$$\begin{aligned}\mathtt{Cov}[y_k, y_{k+n}] &= \boldsymbol{H}_k^{(tr)}\mathtt{Cov}[z_k^{(tr)}, z_{k+n}^{(tr)}](\boldsymbol{H}_k^{(tr)})^\top +\\ &+ \boldsymbol{H}_k^{(per)}\mathtt{Cov}[z_k^{(per)}, z_{k+n}^{(per)}](\boldsymbol{H}_k^{(per)})^\top + \delta_{(n=0)}\sigma_\epsilon^2 \quad .\end{aligned} \tag{4.6}$$

Here $\delta_{(n=0)}$ is a Kronecker delta which equals one when $n = 0$, and zero otherwise. So, the covariance is a sum of two covariances (matrices $\boldsymbol{H}$ are often 1) and a white noise term from the measurement equation.

4.2.3 Introduction to Gaussian processes

A Gaussian Process (GP) is a random process $f(t)$ where for arbitrary selected time points $t_1, t_2, \cdots, t_N$ the probability distribution $p[\, f(t_1), f(t_2), \cdots f(t_N)\,]$ is a multivariate Gaussian.

To define a Gaussian Process (GP) it is necessary to define a mean function $m(t) = \mathbb{E}[f(t)]$ and covariance function $K(t,t') = \mathtt{Cov}[t,t'] = \mathbb{E}[\,(f(t) - m(t))(f(t') - m(t'))\,]$. The covariance function encodes the information about the smoothness of the random process.

Defining a covariance function $K(t,t'|\boldsymbol{\theta})$ is equivalent to defining a prior over functions where a set of functions are all the realizations of the random process. A covariance function also depends on a set of hyperparameters $\boldsymbol{\theta}$. GP prior is denoted as:

$$f(t) \sim \mathbb{GP}(0, K(t,t'|\boldsymbol{\theta}))$$

In Gaussian Process Regression the observations are obtained from the following model:

$$y_k = f(t_k) + \epsilon_k, \quad \epsilon_k \sim \mathbb{N}(0, \sigma_n^2)$$

Having observed a dataset $\{(y_k, t_k)\}_{k=1}^N = \{\boldsymbol{y}, \boldsymbol{t}\}$, the mean and covariance of any

new observed data $\mathbf{y}^\star$ at new time moments $\mathbf{t}^\star$ can be found:

$$\begin{aligned} m(\boldsymbol{t}^\star|\boldsymbol{t}) &= K(\boldsymbol{t}^\star,\boldsymbol{t})\left[K(\boldsymbol{t},\boldsymbol{t})+\sigma_n^2 I_N\right]^{-1}\boldsymbol{y} \\ K(\boldsymbol{t}^\star,\boldsymbol{t}^\star|\boldsymbol{t}) &= K(\boldsymbol{t}^\star,\boldsymbol{t}^\star)-K(\boldsymbol{t}^\star,\boldsymbol{t})\left[K(\boldsymbol{t},\boldsymbol{t})+\sigma_n^2 I_N\right]^{-1}K(\boldsymbol{t}^\star,\boldsymbol{t})^\top+\sigma_n^2 I_N \end{aligned} \tag{4.7}$$

The logarithm of the marginal likelihood $MLL(\boldsymbol{\theta}) = \log p(\boldsymbol{y}|\boldsymbol{t},\boldsymbol{\theta}) = \log \int p(\boldsymbol{y}|\boldsymbol{f},\boldsymbol{t})p(\boldsymbol{f}|\boldsymbol{t})d\boldsymbol{f}$ which is used for inferring the hyperparameters (Sec. 2.3.7):

$$MLL(\boldsymbol{\theta}|bmy) = -\frac{1}{2}\boldsymbol{y}^\top\left[K+\sigma_n^2 I_N\right]^{-1}\boldsymbol{y}-\frac{1}{2}\log\det[K+\sigma_n^2 I_N]^{-1}-\frac{N}{2}\log 2\pi$$

4.3 Popular state-space models as Gaussian processes

Basic state-space model (SS model) models are usually presented as discrete time models with Gaussian errors, e.g. in the books [Durbin and Koopman, 2012; Harvey, 1990]. The structural time series framework allows the combination of several basic state-space models into a more complex model. There are generalizations of discrete-time SS models to continuous time [Harvey, 1990, Chap. 9], which may be converted back to discrete time. Since errors in the basic SS model are assumed to be Gaussian, state-space models are also Gaussian Process models. However, a direct systematic connection to Gaussian Processes used in machine learning is not widely used. The goal of publication VI is to provide explicit connections between GP models and structural time series models.

Gaussian Process are an important class of models in machine learning [Rasmussen and Williams, 2005]. Modeling of time series has been widely addressed by the GP community [Roberts et al., 2013]. However, the modeling principles differ significantly from the state-space models. Modeling is done in continuous time and the main object to model is the covariance function (and optionally the mean function). Knowledge about the connection between a continuous-discrete state space model and Gaussian process exists [Hartikainen and Särkkä, 2010]. The advantage of representing the GP in SS model form is that the inference can be done in $\mathcal{O}(N)$ time where N is the number of data points, while the classic GP regression requires $\mathcal{O}(N^3)$ operations. However, if the amount of data points is relatively small $N < 10000$, or we use some modification of standard GP, the difference in computational time can become negligible [Ambikasaran et al., 2014] on modern computers.

In publication VI we derive several GP covariance functions which correspond to structural time series models. This explicit connection is useful for researches

with different backgrounds: state-space modelers can see that their methods are equivalent to certain Gaussian Process, therefore they can use various extensions developed in the GP literature. GP specialists, on the other hand, can analyze the covariance functions corresponding to state-space models take advantage of approaches from the state-space model domain. More importantly, this connection allows to derive new covariance functions for Gaussian Processes as demonstrated in Section 4.3.5.

4.3.1 Bayesian linear regression

Recall Bayesian Linear Regression (BLR) in the state-space form. Assume that we have N measurements $\mathbf{y} = [\, y_1, y_2, \ldots, y_N \,]^\top$, which are observed at time points $\mathbf{t} = [\, t_1, t_2, \ldots, t_N \,]^\top$. Further, assume that there is a linear dependency between measurements and time:

$$\begin{aligned} y_k &= \theta t_k + \epsilon_k \\ \theta &\sim \mathbb{N}(m_0, P_0) \quad \text{- prior of the parameter } \theta \\ \epsilon_k &\sim \mathbb{N}(0, \sigma_0^2) \quad \text{- Gaussian white noise,} \end{aligned} \tag{4.8}$$

where θ is a parameter of the model with the prior $\theta \sim \mathbb{N}(m_0, P_0)$ and ϵ_k is Gaussian white noise. In this formulation, BLR provides the posterior distribution of θ. Furthermore, BLR provides the posterior predictive distribution which for any set of time points $t_1^\star, t_2^\star, \ldots, t_M^\star$ yields the distribution of corresponding measurements. It is well known [Rasmussen and Williams, 2005] that the same posterior predictive distribution can be obtained by Gaussian Process Regression (GPR) with the kernel:

$$\mathbf{y} \sim \mathbb{GP}\left(m_0 \mathbf{t}, P_0 \mathbf{t}\mathbf{t}^\top + \sigma_0^2 I\right) \quad . \tag{4.9}$$

We are interested in representing the BLR model in the state-space form because it allows using the model in the sequential regime - when data arrives one-by-one. Moreover, Kalman Filter type of inference, which is the standard for the linear state-space models, scales linearly with the number of samples, while Gaussian Process or batch BLR scales cubically [Rasmussen and Williams, 2005]. There are several ways to express BLR in the state-space form [Särkkä,

2013, p. 37]:

$$\begin{cases} \begin{bmatrix} x_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} 1 & \Delta t_{k-1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ \theta_{k-1} \end{bmatrix} \\ y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ \theta_k \end{bmatrix} + \epsilon_k, \quad \text{where:} \end{cases} \tag{4.10}$$

$$x_0 = 0 \sim \mathbb{N}(0,0), \quad \theta_0 \sim \mathbb{N}(m_0, P_0), \quad \epsilon_k \sim \mathbb{N}(0, \sigma_0^2)$$

$$\Delta t_{k-1} = t_k - t_{k-1}, \quad \text{and it is assumed that } t_0 = 0.$$

The equivalence of Bayesian Linear Regression and the state-space model in Eq. (4.10) can be shown. We see that $\theta_k = \theta_{k-1}$ for all k, so it does not change with time. Since $t_0 = 0$ and $x_0 = 0$ we have that:

$$x_1 = t_1\theta_0 = \theta t_1$$

$$x_2 = x_1 + (t_2 - t_1)\theta_1 = t_2\theta_1 + t_1(\theta_0 - \theta_1) = t_2\theta_1 = \theta t_2$$

$$\vdots$$

$$x_k = x_{k-1} + (t_k - t_{k-1})\theta_{k-1}$$

$$= t_k\theta_{k-1} + t_{k-1}(\theta_{k-2} - \theta_{k-1}) = t_k\theta_{k-1} = \theta t_k \quad .$$

We see that $x_k = t_k\theta$, substituting the obtained result into the equation for y_k we obtain: $y_k = \theta t_k + \epsilon_k$, equivalent to the original BLR formulation. Using the obtained state-space model we can find the covariance matrix of y_k, that will be the same as the one in Eq. (4.9). The next section will show how to explicitly derive the covariance function for the more general state-space model.

In this section we have shown the equivalence of GPR with the covariance matrix in Eq. (4.9) and the state-space model formulation in Eq. (4.10). These two models are also equivalent to the Bayesian Linear Regression model.

4.3.2 General state-space model

In this section we derive the covariance function for a more general state-space model than in the previous section. In the literature this model is called the Local Linear Trend Model (LLLM). It is shown that this general state-space model under a special setting of parameters becomes equivalent to the well-known time series models e.g. Local Level Model (LLM), Bayesian Linear Regression. Additionally, there is a close connection to the quasi-periodic (cyclic) model.

Derivation of the covariance function provides a useful connection to Gaussian Process Regression for the aforementioned models. The general state-space model is defined as:

$$
\begin{cases}
\begin{bmatrix} x_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} 1 & \Delta t_{k-1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} q_k^{(1)} \\ q_k^{(2)} \end{bmatrix} \\
y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ \theta_k \end{bmatrix} + \epsilon_k, \quad \text{where:} \quad \epsilon_k \sim \mathbb{N}(0, \sigma_0^2)
\end{cases}
$$

$$
\begin{aligned}
&\Delta t_{k-1} = t_k - t_{k-1}, \\
&\begin{bmatrix} x_1 \\ \theta_1 \end{bmatrix} \sim \mathbb{N}\left(\begin{bmatrix} c_0 \\ m_0 \end{bmatrix}, \begin{bmatrix} K_0 & 0 \\ 0 & P_0 \end{bmatrix} \right) \\
&\begin{bmatrix} q_k^{(1)} \\ q_k^{(2)} \end{bmatrix} \sim \mathbb{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} q_0^2 \Delta t_{k-1} & 0 \\ 0 & g_0^2 \Delta t_{k-1} \end{bmatrix} \right).
\end{aligned}
\tag{4.11}
$$

In this dissertation, the notation is slightly different from the original notation in publication VI. The difference being that here the initial state is assumed to be x_1 not x_0. The new notation is easier to present.

The model in Eq. (4.11) is also more general than the BLR model. One difference consists of extra noise terms in the dynamic (or state) equation. Another difference is a non-zero prior distribution for the initial state variable x_1: $x_1 \sim \mathbb{N}(c_0, K_0)$.

We can easily compute the mean of $\mathbf{z}_k = \begin{bmatrix} x_k \\ \theta_k \end{bmatrix}$: using notation $A[\Delta t] = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}$ and taking into account the fact that the mean $\mathbb{E}[\mathbf{q}_i] = 0$ and expanding the expressions for Δt_i:

$$
\mathbb{E}[\mathbf{z}_k] = A[\Delta t_{k-1} + \Delta t_{k-2} + \ldots + \Delta t_0]\mathbb{E}[\mathbf{z}_0] = A[t_k - t_1] \begin{bmatrix} c_0 \\ m_{0.} \end{bmatrix} \tag{4.12}
$$

Noise in the dynamic Equation

In this subsection the extra noise terms which appear in the dynamic equation are briefly discussed. The two dimensional noise term $\mathbf{q} = \begin{bmatrix} q_k^{(1)} \\ q_k^{(2)} \end{bmatrix}$ comprises two independent and Gaussian distributed components. Consider, for example, the first component $q_k^{(1)} \sim \mathbb{N}(0, q_0^2 \Delta t_{k-1})$. It is a classical *Wiener process* [Rasmussen and Williams, 2005] also called *standard Brownian motion*

and is a generalization of a simple random walk to the continuous time when time measurements are not necessarily equidistant. Its covariance function is $\mathrm{Cov}[\, q_k^{(1)}(t_1), q_k^{(1)}(t_2)\,] = K_0 + q_0^2 \min(t_1, t_2)$ and it is a basic example of a non-stationary Gaussian Process.

Covariance function of the general state-space model

The full derivation of the covariance of the general state-space model (4.11) is presented in publication VI; here we present the end results. The notation is slightly different from the original publication because the initial state here has index 1, not 0, as noted earlier.

Consider the covariance function between y_k and y_{k+n} which is unknown and that we wish to find:

$$\begin{aligned}
&\mathrm{Cov}[\, y_k, y_{k+n}\,] = \mathbb{E}[\,(y_k - \mathbb{E}[\, y_k\,])(y_{k+n} - \mathbb{E}[\, y_{k+n}\,])\,] = \\
&= \mathbb{E}[\,(x_k + \epsilon_k - \mathbb{E}[\, x_k\,])(x_{k+n} + \epsilon_{k+n} - \mathbb{E}[\, x_{k+n}\,])\,] = \\
&= \mathrm{Cov}[\, x_k, x_{k+n}\,] + \delta_{(n=0)}\sigma_0^2 \quad .
\end{aligned} \tag{4.13}$$

Therefore, we see that in order to find the covariance function of y_k it is enough to find the covariance function of x_k and to add the Kronecker symbol.

We present an example of the covariance function for the case of three observed data points: t_1, t_2, t_3 and y_1, y_2, y_3. The covariance of the latent variables is:

$$\begin{aligned}
&\begin{bmatrix}
\mathrm{Cov}[\mathbf{z}_1, \mathbf{z}_1] & \mathrm{Cov}[\mathbf{z}_1, \mathbf{z}_2] & \mathrm{Cov}[\mathbf{z}_1, \mathbf{z}_3] \\
\mathrm{Cov}[\mathbf{z}_2, \mathbf{z}_1] & \mathrm{Cov}[\mathbf{z}_2, \mathbf{z}_2] & \mathrm{Cov}[\mathbf{z}_2, \mathbf{z}_3] \\
\mathrm{Cov}[\mathbf{z}_3, \mathbf{z}_1] & \mathrm{Cov}[\mathbf{z}_3, \mathbf{z}_2] & \mathrm{Cov}[\mathbf{z}_3, \mathbf{z}_3]
\end{bmatrix} = \\
&= \mathscr{A}\{T\}\, D_0\, (\mathscr{A}\{T\})^\top .
\end{aligned} \tag{4.14}$$

The notation in the above formula is the following:

$$\mathscr{A}\{T\} = \begin{bmatrix}
A[0] & 0 & 0 \\
A[\Delta t_1] & A[0] & 0 \\
A[\Delta t_2 + \Delta t_1] & A[\Delta t_2] & A[0]
\end{bmatrix}, \tag{4.15}$$

$$D_0 = \begin{bmatrix}
\mathrm{Cov}[\mathbf{z}_1, \mathbf{z}_1] & 0 & 0 \\
0 & \mathrm{Cov}[\mathbf{q}_1, \mathbf{q}_1] & 0 \\
0 & 0 & \mathrm{Cov}[\mathbf{q}_2, \mathbf{q}_2]
\end{bmatrix}. \tag{4.16}$$

Finally, the covariance of the variables x consist of two parts:

$$\begin{bmatrix} \mathtt{Cov}[x_1,x_1] & \mathtt{Cov}[x_1,x_2] & \mathtt{Cov}[x_1,x_2] \\ \mathtt{Cov}[x_2,x_1] & \mathtt{Cov}[x_2,x_2] & \mathtt{Cov}[x_2,x_3] \\ \mathtt{Cov}[x_3,x_1] & \mathtt{Cov}[x_3,x_2] & \mathtt{Cov}[x_3,x_3] \end{bmatrix} = \mathtt{Cov}_1[\cdot] + \mathtt{Cov}_2[\cdot], \tag{4.17}$$

where

$$\mathtt{Cov}_1[x_k, x_{k+n}] = K_0 + q_0^2 \min(x_k, x_{k+n}) \tag{4.18}$$

and

$$\begin{aligned} \mathtt{Cov}_2[\cdot] &= TDT^\top \\ T &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ \Delta t_1 & 0 & 0 & 0 \\ \Delta t_2 + \Delta t_1 & \Delta t_2 & 0 & 0 \end{bmatrix} \\ D &= \begin{bmatrix} P_0 & 0 & 0 \\ 0 & g_0^2 \Delta t_0 & 0 \\ 0 & 0 & g_0^2 \Delta t_1 \end{bmatrix}. \end{aligned} \tag{4.19}$$

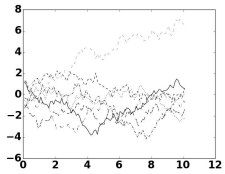
Matrix T can also be represented as:

$$T = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ \Delta t_0 & 0 & 0 \\ \Delta t_1 & \Delta t_1 & 0 \end{bmatrix}. \tag{4.20}$$

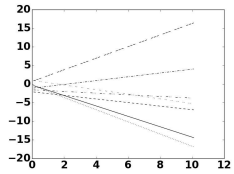
Eq. (4.17) is the solution for the covariance function of the model stated in Eq. (4.11). Given time points $\mathbf{t} = [t_1, t_2, \ldots, t_N]^\top$ we can compute the covariance function, the mean function which is given in (4.12) and use Gaussian Process Regression in a regular way. The sample paths from GP with this covariance function are presented in Fig. 4.2.

4.3.3 Example: Local Level Model

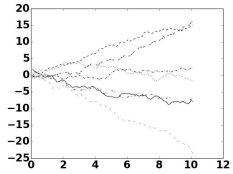
Local Level Model (LLM) is the simplest model among the structural time series models [Durbin and Koopman, 2012]. Its standard representation in the



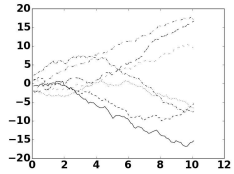
(a) Brownian motion: $K_0 = 1, \underline{P_0 = 0}, \underline{q_0^2 = 1}, \underline{g_0^2 = 0}$



(b) Linear Regression: $K_0 = 1, P_0 = 1, \underline{q_0^2 = 0}, \underline{g_0^2 = 0}$



(c) $K_0 = 1, P_0 = 1, \underline{q_0^2 = 1}, \underline{g_0^2 = 0}$



(d) $K_0 = 1, P_0 = 1, q_0^2 = 1, g_0^2 = 1$

Figure 4.2. GP sample paths of general covariance (LLLM) for various parameter values. The underline emphasizes parameters which equal zero. Publication VI

literature is:

$$\begin{cases} x_k = x_{k-1} + q_k; & q_k \sim \mathcal{N}(0, q_0^2) \\ y_k = x_k + \epsilon_k; & \epsilon_k \sim \mathcal{N}(0, \sigma_0^2) \end{cases} \tag{4.21}$$

$$x_0 \sim \mathbb{N}(c_0, K_0) \quad .$$

As we can see this is a random walk expressed by the dynamic variable x_k additionally submerged into the white noise ϵ_k. The covariance of this model is: $\mathrm{Cov}[y_k, y_m] = K_0 + q_0^2 \min(k, m) + \sigma_0^2 \delta(m = k)$. If we generalize this model to arbitrary time intervals it can be written as:

$$\begin{cases} \begin{bmatrix} x_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} 1 & \Delta t_{k-1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} q_k^{(1)} \\ q_k^{(2)} \end{bmatrix} \\ y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ \theta_k \end{bmatrix} + \epsilon_k, \quad \text{where:} \quad \epsilon_k \sim \mathbb{N}(0, \sigma_0^2) \end{cases}$$

$$\Delta t_{k-1} = t_k - t_{k-1} \tag{4.22}$$

$$\begin{bmatrix} x_1 \\ \theta_1 \end{bmatrix} \sim \mathbb{N}\left(\begin{bmatrix} c_0 \\ \boxed{0} \end{bmatrix}, \begin{bmatrix} K_0 & 0 \\ 0 & \boxed{0} \end{bmatrix} \right)$$

$$\begin{bmatrix} q_k^{(1)} \\ q_k^{(2)} \end{bmatrix} \sim \mathbb{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} q_0^2 \Delta t_{k-1} & 0 \\ 0 & \boxed{0} \end{bmatrix} \right) \quad .$$

The parameters that are nullified with respect to the general model Local

Linear Trend Model (LLLM) are denoted by boxes, see also Fig 4.2a. We can see that the equation for θ_k is redundant because θ_0 is initialized with zero and the corresponding noise term is also zero. The covariance function could also be obtained by using the formula for the general covariance function and setting the corresponding coefficients to zero.

At the end of this section it is worth mentioning that although the LLM is the simplest structural time series model, it can be successfully applied to real world data [Durbin and Koopman, 2012, p. 16].

4.3.4 Damped trend model

We can also extend the previous model to make the trend damped. It is similar to the general model (4.11), except that a slope gradually decreases. Here we present only the dynamic equation for this model because the remainder is the same as in (4.11).

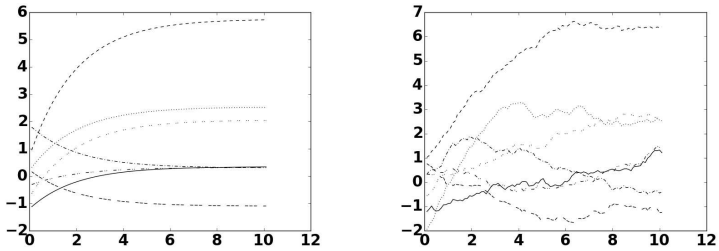
$$\begin{bmatrix} x_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} 1 & \Delta t_{k-1} \\ 0 & \boxed{\phi} \end{bmatrix} \begin{bmatrix} x_{k-1} \\ \theta_{k-1} \end{bmatrix} + \begin{bmatrix} q_k^{(1)} \\ q_k^{(2)} \end{bmatrix} \tag{4.23}$$

The damping factor is ϕ highlighted by a box in (4.23). It must satisfy $0 < \phi < 1$ to correctly dampen the trend.

Next we present the covariance function of this damping trend. The first part of the covariance $\texttt{Cov}_1[\cdot]$ is the same as in (4.18). The second is also similar to (4.19) except that matrix T must be substituted with:

$$T = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & \phi & 0 & 0 \\ 1 & \phi & \phi^2 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ \Delta t_0 & 0 & 0 & 0 \\ \Delta t_1 & \Delta t_1 & 0 & 0 \\ \Delta t_2 & \Delta t_2 & \Delta t_2 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\phi} & 0 & 0 \\ 0 & 0 & \frac{1}{\phi^2} & 0 \end{bmatrix}. \tag{4.24}$$

It is worth noting that the model is not completely adapted to continuous time as the damping factor ϕ does not depend on the time interval Δt_{k-1} between two consecutive measurements y_k. So, strictly speaking, the covariance (4.24) is valid only when all Δt_i are the same. It is possible to extend the derived covariance to cover the general case, however, for simplicity of presentation and space constraints it is not done here. Sample paths from the GP with a damped trend covariance are given in Fig. 4.3.



(a) No noise: $\phi = 0.95$ $K_0 = P_0 = 1$, $q_0^2 = g_0^2 = 0$

(b) Non-Zero Noise: $\phi = 0.95$ $K_0 = P_0 = 1$, $q_0^2 = g_0^2 = 0.05$

Figure 4.3. Damped sample paths: publication VI

4.3.5 Periodic and quasi-periodic modeling

In the structural time series framework there are several models for periodicities and cycles (quasi-periodicities). We consider here the most popular model which is frequently used for cyclic modeling [Durbin and Koopman, 2012, p. 44]:

$$
\begin{cases}
\begin{bmatrix} x_k \\ x_k^\star \end{bmatrix} = \begin{bmatrix} \cos(\omega_c \Delta t_{k-1}) & \sin(\omega_c \Delta t_{k-1}) \\ -\sin(\omega_c \Delta t_{k-1}) & \cos(\omega_c \Delta t_{k-1}) \end{bmatrix} \begin{bmatrix} x_{k-1} \\ x_{k-1}^\star \end{bmatrix} + \\
\qquad\qquad + \begin{bmatrix} q_k^{(1)} \\ q_k^{(2)} \end{bmatrix} \\
y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_k \\ x_k^\star \end{bmatrix} + \epsilon_k, \quad \text{where:} \quad \epsilon_k \sim \mathbb{N}(0, \sigma_0^2)
\end{cases}
\tag{4.25}
$$

$$
\Delta t_{k-1} = t_k - t_{k-1}
$$

$$
\begin{bmatrix} x_1 \\ x_1^\star \end{bmatrix} \sim \mathbb{N}\left(\begin{bmatrix} m_0 \\ m_0 \end{bmatrix}, \begin{bmatrix} P_0 & 0 \\ 0 & P_0 \end{bmatrix} \right)
$$

$$
\begin{bmatrix} q_k^{(1)} \\ q_k^{(2)} \end{bmatrix} \sim \mathbb{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} g_0^2 \Delta t_{k-1} & 0 \\ 0 & g_0^2 \Delta t_{k-1} \end{bmatrix} \right).
$$

Eq. (4.25) is the generalization of a discrete time model [Durbin and Koopman, 2012; Harvey, 1990] to continuous time. The Δt_i are used to express the uneven time sampling. If the sampling is even all the Δt_i equal one.

Notice that the model is completely symmetric with respect to the vector $\mathbf{x} = [x_k, x_k^\star]$; the initial conditions are symmetric and the noise is symmetric. If we assume no noise in the model then it is straightforward to show that the

covariance function of x_k is a periodic covariance function:

$$Cov[x_k, x_{k+n}] = P_0^2 \cos[\omega_c (t_{k+n} - t_k)] \quad . \tag{4.26}$$

The process x_k can be considered as a random process where randomness originates only from the initial conditions. This process is also wide-sense stationary since the covariance function depends on the difference of the time points. Again, if we suppose that the noise vector is absent from the dynamic model, i.e. $q_0^2 = 0$, then the x_n variable is just a cosine wave. This can be deduced by considering x_1, which is a sum of cosine and sine, with coefficients with initial values $x_1, x_1^\star$. This sum can be represented as a cosine wave where the phase depends on those coefficients. Hence, without extra the white noise the x_n is a cosine wave. However, with the presence of a white noise, deviations from the strict periodicity are possible.

Quasi-Periodic (cyclic) covariance function

The covariance function derived in publication VI consist of two parts as in Eq. (4.17). The two parts $\mathtt{Cov}_1$ and $\mathtt{Cov}_2$ are:

$$\mathtt{Cov}_1[\cdot] = \mathscr{L}\{Cos\{T\}\}\, D\, (\mathscr{L}\{Cos\{T\}\})^\top \tag{4.27}$$

$$\mathtt{Cov}_2[\cdot] = \mathscr{L}\{Sin\{T\}\}\, D\, (\mathscr{L}\{Sin\{T\}\})^\top \quad , \tag{4.28}$$

where matrices T and D are exactly the same as in Eq. (4.17). In Eq. (4.27) there are two new matrix operations which are nested: $\mathscr{L}\{\cdot\}$ leaves the lower triangular part (including the main diagonal) of the argument matrix intact, and put zeros to the upper-triangular part; $Cos\{\cdot\}$ applies the cos function element-wise to the matrix.

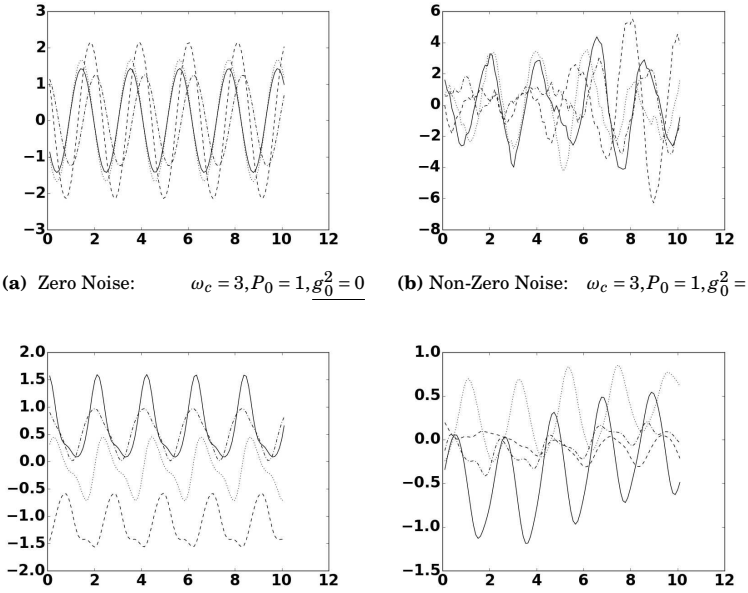
In Eq. (4.28) $Sin\{\cdot\}$ is used instead of $Cos\{\cdot\}$ with a similar meaning - element-wise application of sin function to the argument matrix.

Thus, we have obtained the expression for the covariance matrix of the quasi-periodic model (4.25). Hence, it is now possible to model this cyclic state-space model as a Gaussian Process with the obtained covariance function. The GP sample paths with cyclic covariance function are shown in Fig. 4.4a and 4.4b.

If the data contains several frequencies or periodicities then the corresponding state-space models can be combined in the measurement equations for y_k. In GP regression this is equivalent to the summation of covariance functions.

Also, if the periodic pattern in the data is not close to a cosine wave we need to introduce more harmonics to model the more complicated pattern. Harmonics

for several frequencies are combined $\omega_c, 2\omega_c, \cdots, k\omega_c$ (k harmonics) as described in the previous paragraph.



(a) Zero Noise: $\omega_c = 3, P_0 = 1, g_0^2 = 0$

(b) Non-Zero Noise: $\omega_c = 3, P_0 = 1, g_0^2 = 1$

(c) Standard Periodic Covariance Rasmussen and Williams [2005]:

(d) Quasi-Periodic Covariance Rasmussen and Williams [2005]:

Figure 4.4. Quasi-periodic (cyclic) sample paths. Publication VI

Gaussian periodic covariance function

It is interesting to compare the periodicity modeling approach proposed above with the approach used in Gaussian Process Regression (GPR). In GPR there exists a periodic covariance function [Rasmussen and Williams, 2005, p. 92] which is expressed as:

$$Cov[t_1, t_2] = \exp\left(-\sin^2\left(\frac{\omega_c(t_1 - t_2)}{2}\right)\right). \tag{4.29}$$

This is a periodic covariance function with the frequency ω_c. Sample paths from the GP with periodic covariance are presented in Fig. 4.4c. Since the covariance function is periodic it is possible to represent it as a Fourier series with harmonics $\omega_c, 2\omega_c, 3\omega_c, \cdots$. This example is exactly the case that can be represented by a combination of state-space models, which is described in the previous subsection. Thus, the periodic covariance function used in GPR can be represented by an equivalent random process in the state-space form as is done in [Solin and Särkkä, 2014].

In the same paper of [Solin and Särkkä, 2014] the question of representing

the quasi-periodic covariance function is also discussed. The quasi-periodic covariance function is a multiplication of some stationary covariance functions (e.g. Matern covariance) [Solin and Särkkä, 2014] and the periodic one in (4.29). The random process which is modeled by a quasi-periodic covariance function has no fixed period, the period length fluctuates. Sample paths of quasi-periodic covariance functions are shown in Fig. 4.4d. By using the noise $\mathbf{q_k}$ we also deviate from strict periodicity, however there is no direct correspondence between the model in Eq. (4.25) and the quasi-periodic covariance function in [Solin and Särkkä, 2014]. This question requires further investigation and is not addressed here.

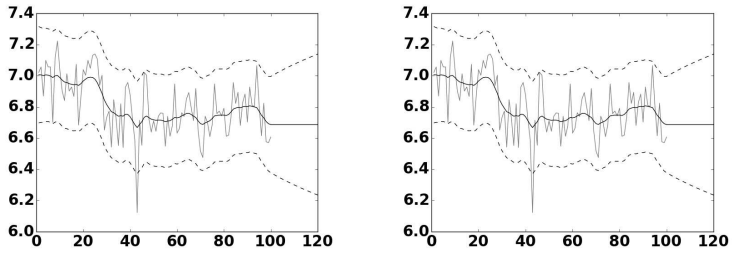
4.3.6 Experiments

In publication VI a number of basic experiments have been performed in order to demonstrate that the derived covariance functions in Sec. 4.3.2, 4.3.4 and 4.3.5 are applicable in the Gaussian Process Regression (GPR) framework and to show that the results are equivalent to state-space modeling. The proposed kernels are applied to several artificially generated datasets and it is shown that GPR results are meaningful. Here we present only the comparison of the GP regression approach and the state-space approach for the Nile Water Level Cobb [1978] dataset which is frequently used in the time series literature. It is shown by the simple example of an Local Level Model model from Section 4.3.3 that the modeling results are equivalent.

All new kernels proposed in this paper have been implemented as add-ons to the *GPy toolbox*. This a powerful toolbox for Gaussian Process modeling and inference [The GPy authors, 2012–2015]. A crucial part of GP inference is finding hyperparameters of a kernel. A standard way to do this is to find the maximum of the marginal log-likelihood (MAP estimate) [Rasmussen and Williams, 2005, p. 112]. In the subsequent experiments, the maximum is searched for by the L-BFGS [Byrd et al., 1995] algorithm. Since the marginal log-likelihood is a non-convex function, each optimization procedure is run ten times with different random initial conditions. The hyperparameters which produce the highest marginal log-likelihood are considered as the solution.

We want to demonstrate that the Gaussian Process Regression approach complemented with the kernels proposed in this paper is completely equivalent with the state-space modeling approach. The state-space inference uses a Kalman Filter (KF) and Rauch-Tung-Striebel (RTS) smoother. We have taken the simple Local Level Model from (4.21). Often this model is used as a starting point for time series analysis. The results of the modeling and forecasting of the Nile

dataset are presented in Figure 4.5; from the figure it is impossible to see any difference between approaches. Analysis of numerical data, which is not presented here also shows that the difference is negligible. Hence, we have shown experimentally for one model that the state-space approach and GP regression model can be used interchangeably depending on the modeler's preferences and other relevant considerations.



(a) State-space model **(b)** GP regression

Figure 4.5. Comparison of time series forecasting of GP regression and state-space model. Publication VI

4.4 Sparse inverse Gaussian process regression

In the introductory Section 4.1 we have considered the structural time series framework which combines certain components to obtain a larger linear Gaussian state-space model. It has also been mentioned that Gaussian processes with all common covariance functions can be represented as state-space models.

In Section 4.3 the derivation of covariance functions for popular structural time series components has been presented. Hence, it is possible to use regular Gaussian Process machinery for inference in the state-space models. From the computational point of view, this transition is not beneficial, since inference in vanilla GP scales as $\mathcal{O}(N^3)$ while in the state-space model inference is linear $\mathcal{O}(N)$.

In publication VII we have considered the speed-up of inference beyond linear time. The computational complexity cannot, in general, be lower than $\mathcal{O}(N)$ because the inference algorithm has to read every data point at least once. However, with finite N, by using parallelization we can indeed obtain sublinear computational complexity in a "weak sense"; provided that the original algorithm is $\mathcal{O}(N)$, and it is parallelizable, the solution can be obtained with time complexity which is strictly less than linear. Although the state-space formulation together with a Kalman Filter (KF) and Rauch-Tung-Striebel (RTS)

smoothers provides the means to obtain linear-time complexity, the Kalman Filter and RTS smoother are inherently *sequential* algorithms, which makes them difficult to parallelize.

In publication VII we show how the state-space formulation can be used to construct linear-time inference algorithms that use the sparseness property of precision matrices of Markovian processes. The advantage of these algorithms is that they are parallelizable and hence allow for sublinear time complexity (in the above weak sense). Similar ideas have also been proposed earlier in the robotics literature Anderson et al. [2015]. We discuss practical algorithms for implementing the required sparse matrix operations and test the proposed method using both simulated and real data.

We assume that the model is similar to (4.2) with one small difference, i.e. for simplicity matrix $\boldsymbol{H}$ is assumed to be constant which is the case in most practical models:

$$\begin{aligned}
&\mathbf{z}_k = \boldsymbol{A}_{k-1}\mathbf{z}_{k-1} + \mathbf{q}_k; \quad \text{where: } \boldsymbol{q}_k \sim \mathbb{N}(0, \boldsymbol{Q}_k); \quad \text{(state / dynamic equation)} \\
&y_k = \boldsymbol{H}\mathbf{z}_\mathbf{k} + \epsilon_k; \quad \text{where: } \epsilon_k \sim \mathbb{N}(0, \boldsymbol{R}_k); \quad \text{(measurement equation)} \quad ,
\end{aligned} \tag{4.30}$$

also, $\mathbf{z}_1 \sim \mathbb{N}(0, \boldsymbol{P}_0)$. The difference in the theory is not prohibitive if $\boldsymbol{H}$ also depends on index k.

For instance, the state-space model for the Gaussian process with the Matérn$\left(\nu = \frac{3}{2}\right)$ covariance function $k_{\nu=3/2}(t) = \sigma^2\left(1 + \frac{\sqrt{3}\Delta t}{\ell}\right)\exp\left(-\frac{\sqrt{3}\Delta t}{\ell}\right)$ with two hyperparameters ℓ - lengthscale and σ^2 - magnitude is:

$$\begin{aligned}
&\boldsymbol{A}_{n-1} = \boldsymbol{A}[\Delta t_n] = \texttt{expm}\left(\begin{bmatrix} 0 & 1 \\ -\phi^2 & -2\phi \end{bmatrix}\Delta t_n\right), \qquad \phi = \frac{\sqrt{3}}{\ell} \\
&\boldsymbol{P}_0 = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \frac{3\sigma^2}{\ell^2} \end{bmatrix}, \qquad \boldsymbol{H} = \begin{bmatrix} 1 & 0 \end{bmatrix} \\
&\boldsymbol{Q}_n = \boldsymbol{P}_0 - \boldsymbol{A}_{n-1}\boldsymbol{P}_0\boldsymbol{A}_{n-1}^\top.
\end{aligned} \tag{4.31}$$

4.4.1 Sparse precision matrix

It has already been shown in Eq. (4.14) with a three-time-points example, that the covariance function $\mathcal{K}(\mathbf{t},\mathbf{t})$ of latent variables $\boldsymbol{z}$ at time points $\boldsymbol{t} = [t_1, t_2, \cdots, t_N]^\top$ is:

$$\mathcal{K}(\mathbf{t},\mathbf{t}) = \mathbf{A}\mathbf{Q}\mathbf{A}^\top. \tag{4.32}$$

In this expression:

$$\mathbf{A} = \begin{bmatrix} I & 0 & 0 & \cdots & 0 \\ A[\Delta t_1] & I & 0 & & 0 \\ A[\Delta t_1 + \Delta t_2] & A[\Delta t_2] & I & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ A[\sum_{i=1}^{N} \Delta t_i] & A[\sum_{i=2}^{N} \Delta t_i] & A[\sum_{i=3}^{N} \Delta t_i] & \cdots & I \end{bmatrix}, \tag{4.33}$$

$$\mathbf{Q} = \begin{bmatrix} P_0 & 0 & 0 & \cdots & 0 \\ 0 & Q_1 & 0 & & 0 \\ 0 & 0 & Q_2 & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & Q_N \end{bmatrix}. \tag{4.34}$$

There exist the following theorem (e.g. Anderson et al. [2015]) which shows the sparsity of the inverse of the covariance matrix:

$$\mathscr{K}^{-1} = \mathbf{A}^{-T}\mathbf{Q}^{-1}\mathbf{A}^{-1}. \tag{4.35}$$

.

Theorem 4.1. *The inverse of the kernel matrix $\mathscr{K}$ from Eq.* (4.32) *is a block-tridiagonal (BTD) and therefore is a sparse matrix.*

The above result can be obtained by noting that the $\mathbf{Q}^{-1}$ is block diagonal (denote the block size as b) and that

$$\mathbf{A}^{-1} = \begin{bmatrix} I & 0 & 0 & \cdots & 0 \\ -A[\Delta t_1] & I & 0 & & 0 \\ 0 & -A[\Delta t_2] & I & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & -A[\Delta t_N] & I \end{bmatrix}, \tag{4.36}$$

which is easily checked by the direct multiplication.

It is easy to find the covariance of observation vector $\mathbf{y} = [y_1, y_2, \cdots y_N]^\top$. In the state-space model (4.30) observations y_i are linear transformations of state vectors $\mathbf{z}_i$, hence according to the properties of linear transformations of Gaussian variables, the covariance matrix in question is:

$$\boldsymbol{G}\mathscr{K}\boldsymbol{G}^\top + \boldsymbol{I}_N\sigma_n^2, \tag{4.37}$$

where $\boldsymbol{G} = (\boldsymbol{I}_N \otimes \boldsymbol{H})$ and the symbol $\otimes$ denotes the Kronecker product.

Note that in the state-space model Eq. (4.30), $\boldsymbol{H}$ is a row vector because y_i are 1-dimensional. The summand $\boldsymbol{I}_N \sigma_n^2$ correspond to the observational noise can be ignored when we refer to the covariance matrix of the model. Therefore, we see that the covariance matrix of $\mathbf{y}$ has the inner part $\mathscr{K}$ with a sparse (block-tridiagonal) inverse. This property can be used for computational convenience via the *matrix inversion lemma*.

4.4.2 Sparse inverse Gaussian process (SpInGP)

The sparseness of the inverse covariance can be utilized in the temporal Gaussian Process Regression (GPR). We express [2] the covariance function of a GP as:

$$\boldsymbol{K}(\mathbf{t},\mathbf{t}) \approxeq \boldsymbol{G}\,\mathscr{K}(\mathbf{t},\mathbf{t})\boldsymbol{G}^\top, \tag{4.38}$$

which is the same as (4.37) without the noise contributions. Then, all the GP formulas can be rewritten using the matrix inversion lemma, where we redefine the noise $\boldsymbol{\Sigma} = \sigma_n^2 \boldsymbol{I}$:

$$\begin{aligned}&[\boldsymbol{G}\mathscr{K}(\mathbf{t},\mathbf{t})\boldsymbol{G}^\top + \boldsymbol{\Sigma}]^{-1}\\ &= \boldsymbol{\Sigma}^{-1} - \boldsymbol{\Sigma}^{-1}\boldsymbol{G}[\mathscr{K}(\mathbf{t},\mathbf{t})^{-1} + \boldsymbol{G}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{G}]^{-1}\boldsymbol{G}^\top\boldsymbol{\Sigma}^{-1}.\end{aligned}$$

After substituting this expression into the formula which calculates the means of the GP at new time points (4.7) we get:

$$\begin{aligned}m(\mathbf{t}^\star) = \boldsymbol{G}_M\mathscr{K}(\mathbf{t}^\star,\mathbf{t})\boldsymbol{G}^\top[\boldsymbol{\Sigma}^{-1}\mathbf{y} - \boldsymbol{\Sigma}^{-1}\boldsymbol{G}\times\\ \underbrace{[\mathscr{K}(\mathbf{t},\mathbf{t})^{-1} + \boldsymbol{G}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{G}]^{-1}(\boldsymbol{G}^\top\boldsymbol{\Sigma}^{-1}\mathbf{y})}_{\text{Computational subproblem 1}}],\end{aligned} \tag{4.39}$$

where $\boldsymbol{G}_M = (\boldsymbol{I}_M \otimes \boldsymbol{H})$. The inverse of the matrix $\mathscr{K}(\mathbf{t},\mathbf{t})$ is available analytically from Theorem 4.1. It is a block-tridiagonal (BTD) matrix. The matrix $\boldsymbol{G}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{G}$ is a block-diagonal matrix which follows from:

$$\boldsymbol{G}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{G} = \sigma_n(\boldsymbol{I}_N \otimes \boldsymbol{H}^\top)(\boldsymbol{I}_N \otimes \boldsymbol{H}) = \sigma_n^2(\boldsymbol{I}_N \otimes \boldsymbol{H}^\top\boldsymbol{H}).$$

However, the formula (4.39) might still be prohibitive to use. Even though the matrix $\mathscr{K}^{-1}(\mathbf{t},\mathbf{t})$ is sparse (block-tridiagonal), the inverse matrix is dense. So, the matrix $\mathscr{K}(\mathbf{t}^\star,\mathbf{t})$ is dense as well. The total computational complexity of

[2] For some covariance functions this is an approximation, but this approximation can be made arbitrarily small.

the above formula, by using the proper inversion algorithm for BTD matrices is $\mathcal{O}(MN)$. However if M is large then the matrix $\mathcal{K}(\mathbf{t}^\star, \mathbf{t})$ may not fit into computer memory because it is dense. Of course, it is possible to apply (4.39) in batches taking each time a small number of new time points, but this is cumbersome in implementation.

There is an alternative approach to compute the means of GP predictions at time points $\boldsymbol{t}^\star$. We need to combine the training and test points in one vector $T = [\mathbf{t}^\star; \mathbf{t}]$ of the size $L = M + N$ which is sorted accordingly. Now consider the GP covariance formula for the full covariance $K(T,T)$. Note that according to the previous discussion $\boldsymbol{K}(T,T) = \boldsymbol{G}\mathcal{K}(T,T)\boldsymbol{G}^\top$ and we try to express the covariance $\mathcal{K}(T,T)$ through the inverse $\mathcal{K}^{-1}(T,T)$ which is sparse. The mean formula for T is then:

$$m(T) = \boldsymbol{G}_L \, \mathcal{K}(T,T) \, \boldsymbol{G}_L^\top \left[\boldsymbol{G}_L \, \mathcal{K}(T,T) \, \boldsymbol{G}_L^\top + \boldsymbol{\Sigma}' \right]^{-1} \mathbf{y}'. \tag{4.40}$$

Here the $\boldsymbol{G}_L$ by analogy equals $(\boldsymbol{I}_L \otimes \boldsymbol{H})$, and $\boldsymbol{\Sigma}'$ contains σ_n^2 on those diagonal positions which correspond to training points and ∞ on those which correspond to new points (infinity must be understood in the limit sense). The vector $\mathbf{y}'$ is similarly augmented with zeros.

By applying the following matrix identity:

$$\boldsymbol{A}^{-1}\boldsymbol{B}[\boldsymbol{D} - \boldsymbol{C}\boldsymbol{A}^{-1}\boldsymbol{B}]^{-1} = [\boldsymbol{A} - \boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{C}]^{-1}\boldsymbol{B}\boldsymbol{D}^{-1},$$

we arrive to:

$$m(T) = \boldsymbol{G}_L \underbrace{[\mathcal{K}^{-1} + \boldsymbol{G}^\top(\boldsymbol{\Sigma}')^{-1}\boldsymbol{G}]^{-1}\boldsymbol{G}_L^\top(\boldsymbol{\Sigma}')^{-1}\mathbf{y}'}_{\text{Computational subproblem 1}}. \tag{4.41}$$

This formula involves the inversion of the of BTD matrix of the size L, but avoids dealing with potentially large covariance matrices $\mathcal{K}(\mathbf{t}^\star, \mathbf{t})$.

Similarly, by combining the training and test time points in the single vector T, and using similar principles we derive the formula for temporal GP covariance computation :

$$\boldsymbol{S}(T,T) = \boldsymbol{G}_L \underbrace{[\mathcal{K}^{-1} + \boldsymbol{G}_L^\top(\boldsymbol{\Sigma}')^{-1}\boldsymbol{G}_L]^{-1}\boldsymbol{G}_L^\top}_{\text{Computational subproblem 2}}. \tag{4.42}$$

Typically, we are only interested in the diagonal of the covariance matrix (4.42), therefore not all the elements need to be computed. Section 4.4.4 describes this and other computational subproblems in more details.

4.4.3 Marginal Likelihood

The marginal log-likelihood (or evidence) (Sec. 2.3.7) is required to perform the hyperparameter inference in the Gaussian Process framework (Sec. 4.2.3). The formula for the marginal log-likelihood of GP is:

$$\log p(\mathbf{y}|\mathbf{t}) = \underbrace{-\frac{1}{2}\mathbf{y}^\top [\boldsymbol{K}+\boldsymbol{\Sigma}]^{-1}\mathbf{y}}_{\text{data fit term: ML\_1}} \underbrace{-\frac{1}{2}\log\det[\boldsymbol{K}+\boldsymbol{\Sigma}]}_{\text{determinant term: ML\_2}} - \frac{N}{2}\log 2\pi \tag{4.43}$$

Computation of the *data fit term* is very similar to the mean computation in Section 4.4.2. For computing the *determinant term* the *Determinant Inverse Lemma* must be applied. It allows the determinant computation to be expressed as:

$$\begin{aligned} &\det[\boldsymbol{K}+\boldsymbol{\Sigma}] = \det[\boldsymbol{G}\mathscr{K}\boldsymbol{G}^\top + \boldsymbol{\Sigma}] = \\ &= \underbrace{\det[\mathscr{K}^{-1} + \boldsymbol{G}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{G}]}_{\text{Computational subproblem 3}} \det[\mathscr{K}]\det[\boldsymbol{\Sigma}] \quad . \end{aligned} \tag{4.44}$$

We assume that $\boldsymbol{\Sigma}$ is diagonal, therefore $\det[\boldsymbol{\Sigma}]$ is easy to compute: $\det[\boldsymbol{\Sigma}] = (\sigma_n^2)^N$ (N - number of data points). $\det[\mathscr{K}] = 1/\det[\mathscr{K}^{-1}]$, so we need to know how to compute the determinants of BTD matrices. This constitutes the third computational problem to be addressed.

4.4.4 Summary of computational subproblems

Let's consider briefly computational subproblems defined in the previous sections. They all involve solving numerical problems with a **symmetric** block-tridiagonal (BTD) matrces, e.g. in Eq. (4.41).

The mean computation in SpInGP formulation requires solving **computational subproblem 1** which is emphasized in Eq. (4.41). It is a block-tridiagonal linear system of equations:

$$\begin{bmatrix} \boxed{?} \\ \boxed{?} \\ \vdots \\ \boxed{?} \end{bmatrix} = \begin{bmatrix} A_1 & B_1 & \cdots & 0 \\ C_1 & A_2 & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_n \end{bmatrix}^{-1} \begin{bmatrix} x \\ x \\ \vdots \\ x \end{bmatrix} \tag{4.45}$$

This is a standard subproblem which can be solved with classical algorithm - Thomas algorithm which is sequential; it performs block LU factorization of

a given matrix. A parallel version has been developed. Unfortunately, block matrix algorithms are rarely found in numerical libraries.

The **computational subproblem 3** in Eq. (4.44) involves computing the determinant of the same symmetric block-tridiagonal matrix. In general, any direct (non-iterative) solver performs some version of LU (Cholesky in the symmetric case) decomposition, therefore subproblem 3 is usually solved simultaneously with subproblem 1.

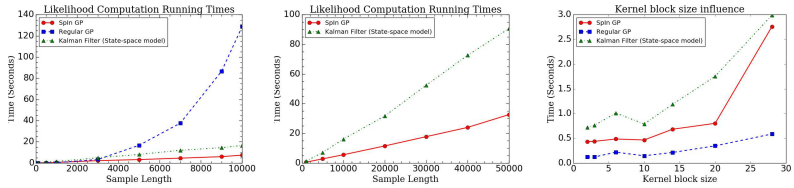
$$\begin{bmatrix} \boxed{?} & x & \cdots & x \\ x & \boxed{?} & & x \\ \vdots & \vdots & \ddots & \vdots \\ x & x & \cdots & \boxed{?} \end{bmatrix} = \begin{bmatrix} A_1 & B_1 & \cdots & 0 \\ C_1 & A_2 & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_n \end{bmatrix}^{-1} \times \begin{bmatrix} X & X & \cdots & 0 \\ X & X & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & X \end{bmatrix} \tag{4.46}$$

Alternatively, we can tackle subproblems 1 and 3 by using general band matrix solvers or sparse solvers. There are several general purpose **direct** sparse solvers available: *Cholmod*, MUMPS, PARDISO etc. The restriction is that the solver must be direct follows from the need to compute the determinant in subproblem 3.

The **computational subproblem 2** in Eq. (4.42) is a less general form of **computational problem 4**. Computational subproblem 4 has not been mentioned in the previous sections. It appears when taking derivatives of the log part of the log-marginal likelihood (4.43), as shown in publication VII. The scheme of subproblem 4 is demonstrated in Eq. (4.46). In this scheme small x means any single element and X any block. In short, the BTD matrix is inverted and the right-hand side (rhs) is also a BTD matrix. Right-hand side blocks do not have to be square, the only requirement is that the dimensions match. Since the inversion of the block-tridiagonal matrix is a dense matrix the solution to this problem is also a dense matrix. However, we are not interested in the whole solution but only in the diagonal of the solution.

It can be noted that subproblem 4 can be solved by computing the BTD part of the inverse of the matrix and then multiplying by the right-hand side. This is true since the right-hand side is a BTD matrix. Hence, only the BTD part of the inverse is needed to compute the required diagonal; computing only some elements in an inverse matrix is called *selective inversion*. Some direct sparse solvers implement selective inversion as a parallel algorithm. Hence, subproblem 4 can also be solved by general sparse solvers. However, developing the specialized numerical algorithms for BTD matrices may bring better performance Petersen et al. [2009].

4.4.5 Experiments



(a) Scaling of SpInGP w.r.t. number of data points. Marginal likelihood and its derivatives are computed.

(b) Scaling of SpInGP and KF w.r.t. number of data points. Marginal likelihood and its derivatives are computed.

(c) Scaling of SpInGP w.r.t block size of precision matrix. 1000 data points. Marginal likelihood and its derivatives are computed.

Figure 4.6. Scaling of SpInGP. Publication VII

Implementation

The sequential algorithms for SpInGP based on the Thomas algorithm have been implemented [3]. The implementation of parallel versions has been postponed for the future because it requires fine-tuning (in the case of using available libraries) or substantial efforts if implemented from scratch.

Numpy and *Scipy* numerical libraries in Python are used for implementation. The code is also integrated with the *GPy* [The GPy authors, 2012–2015] library where many state-space kernels and a rich set of GP models are implemented. Results are obtained on a regular laptop computer with Intel Core i7 CPU @ 2.00GHz × 8 and with 8 Gb of RAM.

4.4.6 Simulated Data Experiments

We have generated artificial data which consists of two sinusoids immersed into Gaussian noise. The speed of computing the marginal log-likelihood (MLL) along with its derivatives are presented in Fig. 4.6a. SpInGP is compared with the state-space form of temporal Gaussian Process Regression (also implemented in GPy). The state-space model inference uses a Kalman Filter: referred to as the KF model. The block size is $b = 12$ which corresponds to Matérn($\nu = \frac{3}{2}$) plus Exponentiated quadratic (RBF) with 10-th order approximation. The same test, but for a larger number of data points, is done to verify linear memory consumption, results in Fig. 4.6b.

As expected the SpInGP and KF solution scales linearly with increasing number of data points while standard GP scales cubically. The SpInGP is faster

[3] Source code available at: `https://github.com/AlexGrig/SpInGP.git`

than KF here, but this is due to the implementation.

The next test we have conducted is the scaling analysis of the same models with respect to the precision matrix block sizes b. From the same artificially generated data, 1,000 data points have been taken and the marginal likelihood and its derivatives have been calculated. Different kernels and kernel combinations have been tested so that block sizes of sparse covariance matrices change accordingly. The results are shown in Fig. 4.6c. In this figure, we see the superlinear scaling of SpInGP and KF inference ($\mathcal{O}(b^3)$ in theory). The 1,000 data points is a relatively small number so the regular GP is much faster in this case. These experiments demonstrate the computational complexity: the linear scalability of sequential SpInGP inference and cubic scalability with respect to the dimensionality of the latent space. SpInGP framework opens a way to the sublinear scalability of temporal Gaussian processes and state-space models in parallel computational environments.

5. Discussion

5.0.1 Randomly-weighted neural networks

Extreme Learning Machine (ELM) has deep connections with kernel methods. Simple example with linear regression can elucidate the situation. Linear regression can be expressed in the dual space as a kernel *ridge* regression with the linear kernel. In general, a kernel algorithm can be interpreted as mapping the data into possibly infinite dimensional space called *Reproducing Hilbert Space* [Rasmussen and Williams, 2005, ch. 6]. It has been shown [Rahimi and Recht, 2008] that a Gaussian kernel can be approximated with certain probabilistic guarantees by a finite number of basis functions with random weights. After this approximation, the problem can be converted back to the primal space and solved by ordinary least squares. Extreme Learning Machine is essentially doing the same, however, the corresponding kernel of ELM is more similar to neural network kernel [Parviainen et al., 2010]. Obviously, the approximation of the kernel with finite-dimensional random features allows balancing the accuracy and computational complexity. Currently, it is still an area of active research.

There are many extensions of the original Extreme Learning Machines. There are deep ELMs [Tissera and McDonnell, 2016], autoencoder ELMs [Zhu et al., 2015], and so forth. The common features of all this models are randomly generated weights and simple optimization algorithms. In modern deep learning, the idea is the opposite in some sense. Weights are optimized by a stochastic optimization. Because of the complex optimization surface and randomness of stochastic optimization, there is no clear understanding of the properties of the resulting model and optimization process. Synthesis of the ELM ideas and modern deep learning can help to better understand the strengths and weaknesses of the latter [Yang et al., 2014], [Moczulski et al., 2015].

We can also view at Extreme Learning Machine from the Bayesian neural

networks point of view. Bayesian neural networks treat weights as random variables, assign a prior to them and compute a posterior. Variational inference has recently advanced [Kucukelbir et al., 2017] to fit these models very efficiently. ELM in turn has two types of weights: randomly generated and obtained by the least-square solution. These structure has not been explored in the Bayesian neural networks context.

5.0.2 Temporal Gaussian processes

These days Gaussian Process research is significantly driven by successes in deep learning. There are deep Gaussian Process and new inference algorithms for them [Salimbeni and Deisenroth, 2017] which are highly scalable. This line of research is facilitated by the use of automatic differentiation libraries like TensorFlow [Abadi et al., 2016], Pytorch [Paszke et al., 2017] etc. These libraries also serve as a basis for developing GP-specific toolboxes like GPflow [Matthews et al., 2016] and GPyTorch [Gardner et al., 2018].

Probabilistic temporal models are also influenced by neural networks research. Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] networks are *de facto* standard for learning temporal dependencies in neural networks community. Those have been merged with probabilistic models e.g. Recurrent GPs [Mattos et al., 2016], Deep Kalman Filter [Krishnan et al., 2015]. I suppose that the valuable goal in this research would be to have benefits of both: interpretability of the state-space models and the flexibility of the neural networks. Also, understanding better the inference and uncertainty propagation in deep temporal models in relation to simpler models seems to be the desirable aim.

Complex spatio-temporal modeling with fast inference is a very important tool in practice. The efforts are being devoted to this direction (e.g. [John and Hensman, 2018]). Some ideas from this thesis can be used in spatio-temporal modeling. For example, the SpinGP approach would be applicable to the spatio-temporal GP with separable covariance functions.

Dealing with non-Gaussian likelihoods is also a frequently encountered situation. There are many methods in GP literature which are developed to deal with non-Gaussian likelihoods. In particular, there are: Laplace approximation, Variational Bayes, Expectation Propagation etc. [Rasmussen and Williams, 2005]. The thesis's author together with co-authors extended the temporal Gaussian Processes to the non-Gaussian likelihoods [Nickisch et al., 2018]. The main advantage of that work is linear computational complexity with respect to the number of data points.

5.0.3 Time series modeling and prediction

Several papers in this dissertation deal with time series data. So, here I briefly review the applicability of neural networks and Gaussian Process/state-space models for this task. The close relationships between Gaussian processes and state-space models (Publication VI) is the cause why these two models are considered jointly. It is worth to note that time series data is ubiquitous in our life. Hence, many scientific disciplines have developed methods for their modeling and prediction. For instance, in econometrics literature statistical methods like ARIMA and their nonlinear extensions are popular. In physics, methods based on the nonlinear dynamics have been developed [Kantz and Schreiber, 2003].

There exist several factors which influence the choice of modeling approach for time series data. These include: computational speed, presence of missing values, presence of several parallel time series, need for uncertainty estimation, exogenous variables, uneven sampling intervals, interpretability, incorporating prior information. Depending on the available time series data and on the desired properties of the resulting model the method should be selected.

Gaussian Process based methods and/or state-space models based methods are more preferable when the following properties are required: computational speed, robustness towards missing values, need for uncertainty estimation, uneven sampling intervals, interpretability, incorporating prior information. One important aspect of these models is that we can incorporate prior information about the time series. For instance, we could define that it is a periodic with a certain period and that it has certain trend. In general, the prediction problem is hard to solve when the amount of data is small or when the system rapidly forgets the previous behaviour. Using the priors is one possible solution to this problem. In this case, the prediction is not purely data-driven but also takes into account these prior beliefs.

Neural networks for time series have complementary strengths than GPs. These are purely data-driven methods, and priors are hard to incorporate. They are also slow to train and interpretability can be obtained by using extra computationally intensive methods like SHAP [Lundberg and Lee, 2017]. However, with the abundance of data and for multiple parallel time series neural networks are typically a good choice.

References

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016. URL `https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf`. 5, 92

Hirotogu Akaike. *Information Theory and an Extension of the Maximum Likelihood Principle*, pages 199–213. Springer New York, New York, NY, 1973. 19

David M. Allen. The Relationship between Variable Selection and Data Agumentation and a Method for Prediction. *Technometrics*, 16(1), 1974. 16

Sivaram Ambikasaran, Daniel Foreman-Mackey, Leslie Greengard, David W. Hogg, and Michael O'Neil. Fast direct methods for gaussian processes, 2014. URL `http://arxiv.org/abs/1403.6015`. 70

E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammarling, J. Demmel, C. Bischof, and D. Sorensen. Lapack: A portable linear algebra library for high-performance computers. In *Proceedings of the 1990 ACM/IEEE Conference on Supercomputing*, Supercomputing '90, pages 2–11, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press. ISBN 0-89791-412-0. URL `http://dl.acm.org/citation.cfm?id=110382.110385`. 48

Sean Anderson, Timothy D Barfoot, Chi Hay Tong, and Simo Särkkä. Batch nonlinear continuous-time trajectory estimation as exactly sparse gaussian process regression. *Autonomous Robots*, 39(3):221–238, 2015. 83, 84

Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statist. Surv.*, 4:40–79, 2010. 40

The GPyOpt authors. GPyOpt: A bayesian optimization framework in python. `http://github.com/SheffieldML/GPyOpt`, 2016. 18

David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, New York, NY, USA, 2012. ISBN 0521518148, 9780521518147. 5, 32

Souhaib Ben Taieb, Gianluca Bontempi, Amir F. Atiya, and Antti Sorjamaa. A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition. *Expert Systems Applications*, 39(8), June 2012. 54

James Bennett, Stan Lanning, and Netflix Netflix. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007. 10

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012. 17

James Bergstra, Dan Yamins, and David D. Cox. Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms. pages 13–20, 2013. URL `https://conference.scipy.org/proceedings/scipy2013/bergstra_hyperopt.html`. 18

Christopher Bishop and Julia Lasserre. Generative or discriminative? getting the best of both worlds. 8:3–23, January 2007. 32

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738. 5, 9, 13, 33

Gianluca Bontempi and Souhaib Ben Taieb. Conditionally dependent strategies for multiple-step-ahead prediction in local learning. *International Journal of Forecasting*, 27(3):689 – 699, 2011. 54

George Edward Pelham Box and Gwilym M. Jenkins. *Time Series Analysis: Forecasting and Control*. Wiley, 2008. ISBN 978-0-470-27284-8. 53

G.E.P. Box, G.M. Jenkins, and G.C. Reinsel. *Time Series Analysis: Forecasting and Control*. Wiley Series in Probability and Statistics. Wiley, 2008. ISBN 9780470272848. 65, 66

H.A.B. Te Braake, H.J.L. Van Can, G. Van Straten, and H.B. Verbruggen. Two-step approach in the training of regulated activation weight neural networks (rawn). *Engineering Applications of Artificial Intelligence*, 10(2):157 – 170, 1997. 39

Matthew Brand. Fast online svd revisions for lightweight recommender systems. pages 37–46, 2006. 47

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996. 58

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, October 2001. 24

Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, September 1995. ISSN 1064-8275. 81

Marshall Space Flight Center. Monthly sunspot numbers. `http://solarscience.msfc.nasa.gov/SunspotCycle.shtml`, 2012. 56

Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015. 5

Vladimir Cherkassky and Yunqian Ma. Practical selection of svm parameters and noise estimation for svm regression. *Neural Networks*, 17(1):113 – 126, 2004. 17

George W. Cobb. *Biometrika*, 65(2):243–251, 1978. 81

Francesco Corona and Amaury Lendasse. Variable scaling for time series prediction. In *Proceedings of ESTSP 2007, European Symposium on Time Series Prediction, Espoo (Finland)*, pages 69–76, 2007. 56

Sven F. Crone, Michèle Hibon, and Konstantinos Nikolopoulos. Advances in forecasting with neural networks? empirical evidence from the {NN3} competition on time series prediction. *International Journal of Forecasting*, 27(3):635 – 660, 2011. 54, 65

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, December 1989. 38

Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`. 51

Pedro Domingos. A unified bias-variance decomposition for zero-one and squared loss. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 564–569. AAAI Press, 2000. ISBN 0-262-51112-6. URL `http://dl.acm.org/citation.cfm?id=647288.721421`. 24

T.J. Durbin and S.J. Koopman. *Time Series Analysis by State Space Methods: Second Edition*. Oxford Statistical Science Series. OUP Oxford, 2012. ISBN 9780199641178. 68, 70, 75, 77, 78

Bradley Efron and Trevor Hastie. *Computer Age Statistical Inference: Algorithms, Evidence, and Data Science*. Cambridge University Press, New York, NY, USA, 1st edition, 2016. ISBN 1107149894, 9781107149892. 15, 27, 42

Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Number 57 in Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, Boca Raton, Florida, USA, 1993. 28

Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *Annals of Statistics*, 32:407–499, 2004. 43

E. Eirola and A. Lendasse. Gaussian mixture models for time series modelling, forecasting, and interpolation. volume 8207 of *Lecture Notes in Computer Science*, pages 162–173, 2013. 60

Emil Eirola, Amaury Lendasse, Francesco Corona, and Michel Verleysen. The delta test: The 1-NN estimator as a feature selection criterion. In *2014 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2014. 20, 21

G. Feng, G. B. Huang, Q. Lin, and R. Gay. Error minimized extreme learning machine with growth of hidden nodes and incremental learning. *IEEE Transactions on Neural Networks*, 20(8):1352–1357, Aug 2009. 43

J. Friedman, T. Hastie, H. Höfling, and R. Tibshirani. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332, 2007. 43

Jacob R Gardner, Geoff Pleiss, Ruihan Wu, Kilian Q Weinberger, and Andrew Gordon Wilson. Product kernel interpolation for scalable gaussian processes. In *AISTATS*, 2018. 92

N. Gershenfeld and A. Weigend. The santa fe time series competition data. `http://www-psych.stanford.edu/~andreas/Time-Series/SantaFe.html`, 1994. 57

Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996. ISBN 0-8018-5414-8. 10, 40, 47, 48

Gene H. Golub, Michael Heath, and Grace Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979. 16

Jan G. De Gooijer and Rob J Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443–473, 2006. 53

Alberto Guillén, Dusan Sovilj, F. Mateo, M. Isabel, Ignacio Rojas, and Amaury and Lendasse. Minimizing the delta test for variable selection in regression problems. *International Journal of High Performance Systems Architecture*, 1(4):269–281, 2008. 20

Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, March 2003. 20

J. Hartikainen and S. Särkkä. Kalman filtering and smoothing solutions to temporal gaussian process regression models. In *Machine Learning for Signal Processing (MLSP), 2010 IEEE International Workshop on*, pages 379–384, Aug 2010. 66, 70

A.C. Harvey. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, 1990. ISBN 9780521405737. 66, 68, 70, 78

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2 edition, 2009. URL `http://www-stat.stanford.edu/~tibs/ElemStatLearn/`. 5, 16, 24, 25, 40, 45, 55

M.H. Hayes. *Statistical Digital Signal Processing and Modeling*. Wiley-India, 2009. 65, 67

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computing*, 9(8):1735–1780, November 1997. 92

Matthew D. Hoffman and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15: 1593–1623, 2014. URL `http://jmlr.org/papers/v15/hoffman14a.html`. 29

Gao Huang, Guang-Bin Huang, Shiji Song, and Keyou You. Trends in extreme learning machines: A review. *Neural Networks*, 61:32 – 48, 2015. 36

Guang-Bin Huang. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, 17(4):879–892, July 2006. 42

Guang-Bin Huang and Lei Chen. Convex incremental extreme learning machine. *Neurocomputing*, 70(16):3056 – 3062, 2007. Neural Network Applications in Electrical Engineering Selected papers from the 3rd International Work-Conference on Artificial Neural Networks (IWANN 2005). 42

Guang-Bin Huang and Lei Chen. Enhanced random search based incremental extreme learning machine. *Neurocomputing*, 71(16):3460 – 3468, 2008. ISSN 0925-2312. 38

Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1):489 – 501, 2006. Neural Networks. 35, 38, 39

ST John and James Hensman. Large-scale Cox process inference using variational Fourier features. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2362–2370, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. 92

I.T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986. 9

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001. URL `http://www.scipy.org/`. [Online; accessed <today>]. 48

Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2): 183–233, November 1999. 29

Holger Kantz and Thomas Schreiber. *Nonlinear Time Series Analysis*. Cambridge University Press, New York, NY, USA, 2003. ISBN 0521529026. 93

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. 2013. URL `http://arxiv.org/pdf/1312.6114.pdf`. 9

A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter. Learning curve prediction with Bayesian neural networks. In *International Conference on Learning Representations (ICLR) 2017 Conference Track*, April 2017. 18

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets, 2016. URL `http://arxiv.org/abs/1605.07079`. 18

Rahul G. Krishnan, Uri Shalit, and David Sontag. Deep kalman filters, 2015. URL `http://arxiv.org/abs/1511.05121`. 92

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic differentiation variational inference. *J. Mach. Learn. Res.*, 18(1):430–474, January 2017. 92

Neil Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *In NIPS*, page 2004, 2003. 33

Neil D. Lawrence. A unifying probabilistic perspective for spectral dimensionality reduction: Insights and new models. *J. Mach. Learn. Res.*, 13:1609–1638, May 2012. ISSN 1532-4435. 9

John A. Lee and Michel Verleysen. *Nonlinear Dimensionality Reduction*. Springer Publishing Company, Incorporated, 1st edition, 2007. ISBN 0387393501, 9780387393506. 9

Lisha Li, Kevin Jamieson, Giulia DeSalvo†, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. In *International Conference on Learning Representations (ICLR) 2017 Conference Track*, April 2017. 18

R.J.A. Little and D.B. Rubin. *Statistical Analysis with Missing Data*. Wiley Series in Probability and Statistics. Wiley, 2002. ISBN 9780471183860. URL `https://books.google.fi/books?id=aYPwAAAAMAAJ`. 11

Xueyi Liu, Chuanhou Gao, and Ping Li. A comparative analysis of support vector machines and extreme learning machines. *Neural Networks*, 33(Supplement C):58 – 66, 2012. 38

Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko. Scalable gradient-based tuning of continuous regularization hyperparameters, 2015. URL http://arxiv.org/abs/1511.06727. 17

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017. 93

J. Luo, C. M. Vong, and P. K. Wong. Sparse bayesian extreme learning machine for multi-classification. *IEEE Transactions on Neural Networks and Learning Systems*, 25(4):836–843, April 2014. 41

Alexander Matthews, Mark van der Wilk, Tom Nickson, Keisuke Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. Gpflow: A gaussian process library using tensorflow, 2016. URL http://arxiv.org/abs/1610.08733. 92

César Lincoln C. Mattos, Zhenwen Dai, Andreas Damianou, Jeremy Forth, Guilherme A. Barreto, and Neil D. Lawrence. Recurrent Gaussian processes. In Hugo Larochelle, Brian Kingsbury, and Samy Bengio, editors, *Proceedings of the International Conference on Learning Representations*, volume 3, Caribe Hotel, San Juan, PR, 2016. 92

Mark D. McDonnell, Migel D. Tissera, Tony Vladusich, André van Schaik, and Jonathan Tapson. Fast, simple and accurate handwritten digit classification by training shallow neural network classifiers with the 'extreme learning machine' algorithm. *PLOS ONE*, 10(8):1–20, 08 2015. doi: 10.1371/journal.pone.0134254. URL https://doi.org/10.1371/journal.pone.0134254. 35

Tucker McElroy and Marc Wildi. Multi-step-ahead estimation of time series models. *International Journal of Forecasting*, 29(3):378 – 394, 2013. 54

Yoan Miche, Antti Sorjamaa, Peter Bas, Olli Simula, C. Jutten, and Amaury Lendasse. Op-elm: Optimally pruned extreme learning machine. *IEEE Transactions on Neural Networks*, 21(1):158–162, Jan 2010. 43

Thomas P. Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, pages 362–369, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. 30

Marcin Moczulski, Misha Denil, Jeremy Appleyard, and Nando de Freitas. Acdc: A structured efficient linear layer, 2015. URL http://arxiv.org/abs/1511.05946. 91

Michinari Momma and Kristin P. Bennett. A pattern search method for model selection of support vector regression. In *Proceedings of the 2002 SIAM International Conference on Data Mining*, pages 261–274, 2002. doi: 10.1137/1.9781611972726.16. 17

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029. 5, 15, 19, 26, 27, 29

Radford M. Neal. Mcmc using hamiltonian dynamics. 2012. 29

J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965. 41

Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 841–848. MIT Press, 2002. 33

Hannes Nickisch, Arno Solin, and Alexander Grigorevskiy. State space Gaussian processes with non-Gaussian likelihood. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3789–3798, Stockholm Sweden, 10–15 Jul 2018. 92

Elina Parviainen, Jaakko Riihimäki, Yoan Miché, and Amaury Lendasse. Interpreting extreme learning machine as an approximation to an infinite neural network. In *KDIR*, 2010. 91

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 92

Kristiaan Pelckmans, Johan A. K. Suykens, T. Van Gestel, J. De Brabanter, L. Lukas, B. Hamers, B. De Moor, and J. Vandewalle. Ls-svmlab: a matlab/c toolbox for least squares support vector machines, 2002. 58

Dan Erik Petersen, Song Li, Kurt Stokbro, Hans Henrik B. Sørensen, Per Christian Hansen, Stig Skelboe, and Eric Darve. A hybrid method for the parallel computation of green's functions. *Journal of Computational Physics*, 228(14):5020 – 5039, 2009. ISSN 0021-9991. 88

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20*, pages 1177–1184. Curran Associates, Inc., 2008. 39, 91

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X. 70, 71, 73, 80, 81, 91, 92

J. Rissanen. Minimum description length principle. In S. Kotz and N. L. Johnson, editors, *Encyclopedia of Statistical Sciences, 5*, pages 523–527, New York, 1985. Wiley. 19

Stephen Roberts, M Osborne, M Ebden, Steven Reece, N Gibson, and S Aigrain. Gaussian processes for time-series modelling. *Philosophical Transactions of Royal Society A*, 371(1984):20110550, 2013. 70

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958. 5

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988. 35

Hugh Salimbeni and Marc Deisenroth. Doubly stochastic variational inference for deep gaussian processes, 2017. URL `http://arxiv.org/abs/1705.08933`. 92

Simo Särkkä. *Bayesian Filtering and Smoothing*. Cambrige University Press, 2013. ISBN 978-1-107-61928-9. 68, 71

Simo Särkkä, Arno Solin, and Jouni Hartikainen. Spatiotemporal learning via infinite-dimensional bayesian filtering and smoothing. *IEEE Signal Processing Magazine*, 30 (4):51–61, 2013. 67

W. F. Schmidt, M. A. Kraaijveld, and R. P. W. Duin. Feedforward neural networks with random weights. In *Proceedings of the 11th IAPR International Conference on Pattern Recognition. Vol.II. Conference B: Pattern Recognition Methodology and Systems*, pages 1–4, Aug 1992. 39

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6:461–464, 1978. 19

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016. 17

Timo Similä and Jarkko Tikka. *Multiresponse Sparse Regression with Application to Multidimensional Scaling*, pages 97–102. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. 22, 43

Arno Solin and Simo Särkkä. Explicit link between periodic covariance functions and state space models. In *Proceedings of the 17-th International Conference on Artificial Intelligence and Statistics (AISTATS 2014)*, volume 33, pages 904–912, 2014. 80, 81

A. Sorjamaa, J. Hao, N. Reyhani, Y. Ji, and A. Lendasse. Methodology for long-term prediction of time series. *Neurocomputing*, 70(16-18):2861–2869, October 2007. 54, 55, 56

Antti Sorjamaa and Amaury Lendasse. Time series prediction using DirRec strategy. In M. Verleysen, editor, *ESANN06, European Symposium on Artificial Neural Networks*, pages 143–148, Bruges, Belgium, April 26-28 2006. European Symposium on Artificial Neural Networks. 56

J.A.K. Suykens. *Least Squares Support Vector Machines*. World Scientific, 2002. ISBN 9789812381514. 58

The GPy authors. GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`, 2012–2015. 81, 89

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994. 43

Migel D. Tissera and Mark D. McDonnell. Deep extreme learning machines. *Neurocomputing*, 174(PA):42–49, January 2016. ISSN 0925-2312. 91

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016. URL `http://arxiv.org/abs/1609.03499`. 5

M. van Heeswijk, Y. Miche, E. Oja, and A. Lendasse. GPU-accelerated and parallelized ELM ensembles for large-scale regression. *Neurocomputing*, 74(16):2430–2437, September 2011. 41, 58

Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. Recurrent recommender networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 495–503, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4675-7. 2

A.M. Yaglom. *Correlation Theory of Stationary and Related Random Functions*. Springer, 1987. ISBN 9783540963318. 53

Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets, 2014. URL `http://arxiv.org/abs/1412.7149`. 91

Zichao Yang, Andrew Wilson, Alex Smola, and Le Song. A la Carte – Learning Fast Kernels. In Guy Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 1098–1106, San Diego, California, USA, May 2015. PMLR. 39

W. Zhu, J. Miao, L. Qing, and G. Huang. Hierarchical extreme learning machine for unsupervised representation learning. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2015. 91

Errata

Publication VI

Figure numbers in figure captions are wrong. Caption of Fig.2 must be Fig.1, Caption of Fig.4 must be Fig.2, Caption of Fig.6 must be Fig.3, Caption of Fig.8 must be Fig.4. In the text the figure numbers are correct.

Aalto-DD 144/2019

ISBN 978-952-60-8670-5 (printed)
ISBN 978-952-60-8671-2 (pdf)
ISSN 1799-4934 (printed)
ISSN 1799-4942 (pdf)

Aalto University
School of Science
Department of Computer Science
www.aalto.fi

BUSINESS +
ECONOMY

ART +
DESIGN +
ARCHITECTURE

SCIENCE +
TECHNOLOGY

CROSSOVER

DOCTORAL
DISSERTATIONS