Department of Computer Science
Series of Publications A
Report A-2022-10

Gaussian Processes and Convolutional Neural Networks for Modeling Sensor Data

Krista Longi

Doctoral dissertation, to be presented for public examination with the permission of the Faculty of Science of the University of Helsinki, in Auditorium B123, Exactum, Kumpula on 2nd of September at 1 p.m.

University of Helsinki
Finland

Supervisor
Arto Klami, University of Helsinki, Finland

Pre-examiners
Simo Särkkä, Aalto University, Finland
Jaakko Suutala, University of Oulu, Finland

Opponent
Niklas Wahlström, Uppsala University, Sweden

Custos
Arto Klami, University of Helsinki, Finland

Contact information

Department of Computer Science
P.O. Box 68 (Pietari Kalmin katu 5)
FI-00014 University of Helsinki
Finland

Email address: info@cs.helsinki.fi
URL: http://cs.helsinki.fi/
Telephone: +358 2941 911

ISSN 1238-8645 (print)
ISSN 2814-4031 (online)
ISBN 978-951-51-8405-4 (paperback)
ISBN 978-951-51-8406-1 (PDF)
Helsinki 2022
Unigrafia

Gaussian Processes and Convolutional Neural Networks for Modeling Sensor Data

Krista Longi

Department of Computer Science
P.O. Box 68, FI-00014 University of Helsinki, Finland
Krista.Longi@helsinki.fi

PhD Thesis, Series of Publications A, Report A-2022-10
Helsinki, August 2022, 84+73 pages
ISSN 1238-8645 (print)
ISSN 2814-4031 (online)
ISBN 978-951-51-8405-4 (paperback)
ISBN 978-951-51-8406-1 (PDF)

Abstract

Different sensors are constantly collecting information about us and our surroundings, such as pollution levels or heart rates. This results in long sequences of noisy time series observations, often also referred to as signals. This thesis develops machine learning methods for analysing such sensor data. The motivation behind the work is based on three real-world applications. In one, the goal is to improve Wi-Fi networks and recognise devices causing interference from spectral data measured by a spectrum analyser. The second one uses ultrasound signals propagated through different paths to localise objects inside closed containers, such as fouling inside of industrial pipelines. In third, the goal is to model an engine of a car and its emissions.

Machine learning builds models of complex systems based on a set of observations. We develop models that are designed for analysing time series data, and we build on existing work on two different models: convolutional neural networks (CNNs) and Gaussian processes (GPs). We show that CNNs are able to automatically recognise useful patterns both in 1D and 2D signal data, even when we use a chaotic cavity to scatter waves randomly in order to increase the acoustic aperture. We show how GPs can be used when the observations can be interpreted as integrals over some region, and how we can introduce a non-negativity constraint in such cases. We also show how Gaussian process state space models can be used to learn long-

and short-term effects simultaneously by training the model with different resolutions of the data.

The amount of data in our case studies is limited as the datasets have been collected manually using a limited amount of sensors. This adds additional challenges to modeling, and we have used different approaches to cope with limited data. GPs as a model are well suited for small data as they are able to naturally model uncertainties. We also show how a dataset can be collected so that it contains as much information as possible with the limited resources available in cases where we use GPs with integral observations. CNNs in general require large datasets, but we show how we can augment labeled data with unlabeled data by taking advantage of the continuity in sensor data.

Computing Reviews (2012) Categories and Subject Descriptors:

Computing methodologies → Machine learning → Machine learning approaches → Neural networks
Computing methodologies → Machine learning → Machine learning approaches → Kernel methods
Computing methodologies → Applied computing → Physical sciences and engineering

General Terms:
Gaussian processes, convolutional neural networks, Gaussian process state-space models, time series data, integral observations

Additional Key Words and Phrases:
signal data, continuous data, approximate inference, sparse Gaussian processes, sensor placement, non-negativity constraint

Acknowledgements

First and foremost, I would like to express my appreciation to my supervisor Arto Klami for all the advice, guidance and help during my doctoral studies. I am especially grateful to him for creating a positive work environment without which this thesis would not have been completed.

This thesis would not have been possible without all of our collaborators who provided interesting real-world problems, data and valuable domain knowledge in different areas. I would like to thank Ekahau and Teemu Pulkkinen for a fruitful and easy collaboration. I am grateful to both Altum technologies and our collaborators at the physics department, including but not limited to Tom Sillanpää, Joni Mäkinen, Timo Rauhala, Ari Salmi and Edward Hæggström. I also would like to extend my gratitude to Bosch Center for Artificial Intelligence, Jakob Lindinger, Olaf Dünnbier, Melih Kandemir and Barbara Rakitsch for giving me a chance to take up a new project and learn new things at the end of my PhD journey. Especially I want to thank Barbara Rakitsch for being such a great teacher.

I would like to thank all my colleagues in the MUPI group. Especially I am grateful to Joseph Sakaya, Aditya Jitta and Jarkko Lagus for being great colleagues and friends and supporting me both in work and in my personal life. I am also grateful to my coauthors Ville Tanskanen and Chang Rajani.

I thank my pre-examiners, Simo Särkkä and Jaakko Suutala, for reviewing the thesis and providing valuable feedback. I thank Niklas Wahlström for agreeing to act as my opponent.

I would also like to extend my thanks to the Doctoral School in Computer Science (DoCS), including Sini Karppinen and Pirjo Moen, for supporting me in various ways during this journey and the graduation process. In addition, the peer-support of everyone at the DoCS pizza evenings has been invaluable and I am glad to be able to say I have made great friends that have supported me through this journey and will remain in my life also after this. I want to mention Maksym Gabielkov, Joonas Miettinen and Chen He, but there are also many others.

Finally, I would like to thank my family for always being there for me. This thesis would not have happened without the love and support of my partner Jonne. You and Kuura mean the world to me. I am forever grateful to my parents for their unconditional love and support. You are and were the best role models.

Helsinki, August 2022
Krista Longi

Contents

Original Publications

This thesis is based on the following publications that are reprinted at the end of the thesis.

I. Krista Longi, Teemu Pulkkinen, Arto Klami. Semi-supervised Convolutional Neural Networks for Identifying Wi-Fi Interference Sources In *Asian Conference on Machine Learning (ACML)*, PMLR 77:391-406, 2017.

II. Tom Sillanpää, Krista Longi, Joni Mäkinen, Timo Rauhala, Arto Klami, Ari Salmi, Edward Hæggström. Localizing a target inside an enclosed cylinder with a single chaotic cavity transducer augmented with supervised machine learning. In *AIP Advances*, 11(11), 115104, 2021.

III. Krista Longi, Chang Rajani, Tom Sillanpää, Joni Mäkinen, Timo Rauhala, Ari Salmi, Edward Hæggström, Arto Klami. Sensor Placement for Spatial Gaussian Processes with Integral Observations. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, PMLR 124:1009-1018, 2020.

IV. Ville Tanskanen, Krista Longi, Arto Klami. Non-linearities in Gaussian processes with integral observations. In *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1-6, 2020.

V. Tom Sillanpää, Timo Rauhala, Joni Mäkinen, Chang Rajani, Krista Longi, Arto Klami, Ari Salmi, Edward Hæggström. Ultrasonic Fouling Detector Powered by Machine Learning. In *2019 IEEE International Ultrasonics Symposium (IUS)*, pp. 1639-1642, 2019.

VI. Krista Longi, Jakob Lindinger, Olaf Dünnbier, Melih Kandemir, Arto Klami, Barbara Rakitsch. Traversing Time with Multi-Resolution Gaussian Process State Space Models. In *Learning for Dynamics & Control Conference (L4DC)*, PMLR 168:366-377, 2022.

Author contributions

I. The author derived and implemented the algorithm. The author carried out and analysed the experiments except for the baseline. The author also wrote the first draft of the publication except for the parts related to the WLAN domain and interference detection and the baseline. The experimental setup was designed jointly and the publication was finalised jointly.

II. The machine learning approach, the algorithms and corresponding experiments were designed together by the author and Arto Klami. The author derived and implemented the algorithms and carried out the related parts of the experiments as well as participated in writing these parts of the publication. The author participated in designing the experimental setup.

III. The publication and experiments were mainly designed and written by the author and Arto Klami. The author derived and implemented the model-based algorithms and derived the analytical solution for the RQ kernel. The author carried out and analysed the experiments.

IV. The experimental setup and algorithms were designed jointly. The author carried out and reported the experiments on Poisson regression and ultrasonic fouling detection. The publication was written jointly, but the author was mainly responsible of the related work.

V. The machine learning setup was designed by Arto Klami and the author. The machine learning algorithm was implemented and corresponding experiments carried out by the author. The author participated in writing the parts about the machine learning approach and in designing the general experimental setup.

VI. The author was responsible of implementing the algorithm as well as performing the experiments and analysing the results. The initial problem setting and idea were given by Barbara Rakitsch, while the details and experiments were designed jointly. The author contributed to writing the publication, mainly to the experiments sections and editing.

Chapter 1

Introduction

The goal of this thesis is to introduce novel machine learning approaches for solving practical problems with *time series data* collected using a set of sensors. In this chapter we describe the motivation behind the original publications as well as outline the original contributions.

1.1 Motivation

Machine learning is used to learn models about complex systems based on data. We develop methods for data containing some structure, in this case temporal or spatial continuity, and we build on existing work on convolutional neural networks and Gaussian processes and tailor them to our specific needs. The work is motivated by real applications with potential societal impact, and we have used specialised sensors to collect the datasets manually. This means that the amount of data available for training is limited, and we show how to use the limited data efficiently, how to augment labeled data with unlabeled data and how to collect the data so that it is as informative as possible.

1.1.1 Sensor Data

Nowadays, sensors are continuously collecting large amounts of data all around us. Smart watches monitor our heart rates and sleeping patterns, various sensors collect data about our environment, such as pollution levels and temperatures, and medical imaging sensors detect x-rays that have traveled through the human body. More generally, sensor is physical device that monitors its environment in the real world, detecting some specific type of input. Sensor data is the one- or multidimensional digital output per time point given by such devices, often also referred to as *signals*. The

time series data collected by these sensors can be analysed and used to for example improve industrial processes, monitor health, and ease many every day tasks.

We have three novel datasets collected using specialised sensors, and the machine learning approaches and improvements presented were motivated by these real-world problems. However, the presented approaches are in general applicable to wide range of problems with temporal or spatial data, and in some publications we demonstrate the benefits of the presented methods also on other applications than the ones mentioned here.

In `Publication` I we collected data with a spectrum analyzer that performs sweeps over the 2.4GHz spectrum to identify devices operating in this frequency range. The goal is to improve Wi-Fi networks by identifying interfering devices. In `Publications` II-V use ultrasound signals to map an environment. The path through which the signals travel changes the signals, and by analysing them we can form estimates about the environment, such as localise an object inside a container (`Publication` II) or non-invasively localise fouling inside pipes (`Publications` III-V). In `Publication` VI we have measured the raw emissions of an engine, and we want to understand the system to predict future emissions for new input configurations.

1.1.2 Convolutional Neural Networks and Gaussian Processes

Machine learning is used to learn models of real-world systems based on collected data. Models are simplifications of the system: they are based on making simplifying assumptions so that we are able to both define the model and learn the model with limited computational resources. Models can then be used to explain or learn about the system or to make predictions or decisions based on new, previously unseen data.

Various different methods exists, but they all take data as input. We focus on *supervised learning*, which means machine learning problems that reduce to learning a mapping between an input $\mathbf{x}$ and an output $\mathbf{y}$ based on examples for which the outputs are known. The models used here have different approaches to learning such mappings, but they are specifically designed for spatial or temporal data taking advantage of the special structure: continuity in the data causes the nearby values to be correlated.

Convolutional neural networks (CNN) are deep learning models (Goodfellow et al., 2016; LeCun et al., 2015), which means that they consist of multiple processing layers that learn representations of the data with different levels of abstraction. They typically find one model that best represents

the data that is then used in prediction tasks. Convolutional neural networks are based on applying filters on the data and finding regions where the difference between the filter and the data are small. These filters are automatically learned from the training data so we do not need to previously know what features in the data are important. Using these filters is also computationally more efficient than just examining individual time points and reduces the needed amount of training data. The benefit of DL methods is that they scale well to high dimensional data and long trajectories, which different signals often are. For example, CNNs can be used to efficiently analyse the approximately 3000 features per time point in the dataset in `Publication` I.

`Publications` I-II are build on convolutional neural networks, and we provide the necessary background on deep learning and CNNs in Sections 2.1 and 3.1. Chapter 4 discusses these publications in more detail and demonstrates how CNNs can be used to analyse time series data and to introduce invariance in the temporal domain.

In contrast, Gaussian processes (GP) (Rasmussen and Williams, 2006) are probabilistic or Bayesian models (Ghahramani, 2015) that use probabilities to represent uncertainty about the mapping being learned. The learned solution is a posterior distribution over all possible models, where models that are more likely according to the data have a higher probability. The natural ability to model uncertainties is important in many applications, especially when the amount of data is limited. This is why they are suitable for many applications with sensor data. GPs are based on smooth functions, which makes them a natural choice for spatial and temporal data. In addition, GPs generalise well and we can incorporate prior knowledge of the modeled system into the GP. However, the limitation of GPs is with large datasets as the computational complexity grows cubically with the amount of time points. Still, many approximations to tackle this issue have been presented, and we discuss them in Section 3.2.1.

The rest of the publications build on GPs, which we introduce in Section 3.2. `Publications` III-V, presented in Chapter 5, discuss using GPs when observations consist of integrals over an unknown function. While in traditional GP regression we observe values of certain locations, we now observe a signal that has travelled through the observation space along some path. Objects on the path of the signal affect its properties, and we extract information from the signal that can be interpreted as the value of the integral along that line. `Publication` VI, discussed in Chapter 6, is separate from the other publications building on GPs, and extends GP based state space models to datasets containing long- and short-term effects.

1.1.3 Limited Data

In our applications the amount of data is limited as it was largely collected manually using a limited amount of specialised sensors. This is typical for many sensor data applications. Specialised sensors can be expensive and thus the amount limited. Many times placing the sensors and recording data requires manual labour. Even though one sensor could be collecting data for a long time resulting in long trajectories, this still gives information of only one location. Accurately labeling the data might require manual labour and be the bottle neck in the data collection process. Sometimes less data also means less exposure to possibly harmful things, such as X-rays in CT scans. Thus, methods used to learn from sensor data often need to be able to cope with limited amount of data.

Deep learning methods are known to be data hungry, but we can cope with limited data by keeping the network structure fairly simple and taking advantage of parameter sharing in CNNs (LeCun et al., 2015), by using the data efficiently with cross-validation or by augmenting labeled data with unlabeled data as we discuss in Chapter 4. Bayesian methods naturally model uncertainties, and are often preferred over DL methods when the amount of data is small or when some events in it are rare (Ghahramani, 2015). Gaussian processes (Section 3.2) are an example of such models, and we use them in Chapters 5 and 6.

The data can also be collected efficiently so that it provides as much information about the environment as possible by smartly designing the locations for sensors (Krause and Guestrin, 2007). When modeling spatial data, such as temperatures or amount of fouling inside a pipe, the question is where should a limited amount of monitoring sensors be placed so that we can get an accurate mapping of the environment. How to best place sensors when we measure signals traveling through certain paths to model the environment using GPs is discussed in Section 5.5.

1.2 Thesis Contributions

This thesis introduces machine learning based solutions for three practical problems as well as develops novel techniques for analysing sensor data. From the perspective of the applications, this thesis introduces advances for the following practical problems:

- We provide a novel method for recognising radio frequency devices that can cause Wi-Fi interference based spectral data recorded with a specialised spectrum analyser (`Publication` I).

- We show that when combined with machine learning, a chaotic cavity can be used to increase the acoustic aperture of ultrasound signals to localise objects inside closed containers (`Publication` II).
- We provide the first machine learning method for non-invasive localisation of fouling inside pipes based on ultrasound waves propagated through the pipe (`Publications` III-V).
- We provide a novel method for learning both long- and short-term effects simultaneously when modeling emissions of a gasoline car engine (`Publication` VI).

From the perspective of machine learning algorithms, the main contributions that also generalize to other applications than the ones presented here are:

- We introduce a novel semi-supervised technique for training convolutional neural networks that assigns pseudo-labels for unlabeled data by taking advantage of continuity in time series data (`Publication` I).
- We provide the first presented techniques for optimizing sensor locations for GPs with integral observations (`Publication` III).
- We present an easy-to-use, general technique for learning non-conjugate GPs with integral observations. The method is based on probabilistic programming, and can be easily used regardless of the reason for non-conjugacy making it easy to use for practitioners. (`Publication` IV).
- We introduce an efficient algorithm for learning Gaussian process state-space models that can recognise both short- and long-term effects in time series data (`Publication` VI).

Chapter 2

Preliminaries

This chapter provides the necessary preliminaries about machine learning required to understand the rest of the thesis. We focus on general concepts, principles and learning paradigms, while the specific models used in this thesis are introduced in Chapter 3. Comprehensive overviews of machine learning principles are provided for example by Murphy (2012) and Bishop (2006).

Supervised machine learning

In machine learning data is used to learn some unknown parameters of a statistical model in cases where the problem would be too difficult to solve with rule-based algorithms. The data is typically recorded from a real life system, and the goal is to learn and make predictions about the system. Examples of such data are records of temperatures over time in different areas or the amounts of particles in emissions of a car engine.

We consider *supervised machine learning* problems, where the data consists of N input-output pairs $\mathcal{D} = \{\mathbf{y}_n, \mathbf{x}_n\}_{n=1}^N$. We mark the set of inputs or features with $\mathcal{X} = \{\mathbf{x}_n\}_{n=1}^N$, $\mathbf{x}_n \in \mathbb{R}^{D_x}$ and the set of outputs with $\mathcal{Y} = \{\mathbf{y}_n\}_{n=1}^N$, $\mathbf{y}_n \in \mathbb{R}^{D_y}$ the outputs or the observations. If the values of $\mathcal{Y}$ are discrete we talk about *classification* and if they are real-valued we talk about *regression*. In unsupervised learning the data consists of only the inputs, but it is not in the scope of this thesis.

We assume that the dependency between the inputs and the outputs can be described with a latent function $f(\mathbf{x}) : \mathbb{R}^{D_x} \to \mathbb{R}^{D_y}$. The goal is to make inference about f based on a data set called the *training set*. Once the mapping between the inputs and outputs has been learned, the performance can be evaluated on previously unseen inputs called the *test set*. Being able to make predictions about previously unseen data is called

generalization, and it is an important quality of any machine learning model as the training data is always limited.

Parametric and non-parametric models

Models can be divided into two groups depending on how f is parametrized: *parametric* and *non-parametric* models. In parametric models $f(\mathbf{x})$ is assumed to have a certain structure with finite amount of parameters θ. Possible prior beliefs are incorporated by placing a prior distribution on θ. Examples of such models are linear regression, where the form of f is very limited, and neural networks (Section 2.1) (Goodfellow et al., 2016), where the amount of parameters θ is large and very complex functions f are possible. In non-parametric models the function f is not explicitly parametrized, but the amount of parameters grows with the amount of data. This can be seen to correspond to having an infinite amount of parameters. An example of such model is Gaussian process (GP) (Section 3.2) (Rasmussen and Williams, 2006). Parametric models are forced to make stronger assumptions about the data than non-parametric models, but this also means that they are usually computationally more efficient.

Point estimates vs. posterior distributions

There are also two alternative ways to handle the parameters θ. One way is to find the values for θ that best describe the data, called a *point estimate*. Training is done by adjusting the values of the parameters by optimizing an objective function that measures the error or distance between the predicted output and the actual output of the model, such as the squared error over the training set or the likelihood $p(\mathcal{Y}|\theta)$. The resulting model is then considered to be the best available representation of the considered system, and can be used in further applications such as getting the point estimate of outputs $\mathcal{Y}^*$ at test points $\mathcal{X}^*$. Most deep learning methods fall into this category. They automatically learn to recognise patterns from data sets by having multiple non-linear layers with multiple parameters that allow learning very complicated functions from large amounts of data. Thus, these methods have been very successful in several domains where large data sets are available (LeCun et al., 2015). Deep learning methods are introduced in Section 2.1.

However, in these traditional deep learning methods we do not have estimates of the certainty of the mapping, though recent works have introduced some solutions to this (Gal, 2016). There will always be uncertainties about the system and understanding when the model is not confident about

the predictions is important. Even large amounts of data can not always capture the whole complexity of the system, especially when there are rare events present. Bayesian framework provides a natural way to deal with uncertainty, as instead of point estimates, full *posterior distributions* $p(\theta|\mathcal{D})$ are provided for the parameters of the model and the predicted outputs. In addition, it allows us to utilize expert information by defining a prior distribution that corresponds to our believes. The believes then get updated through the Bayes rule after we introduce new observations. Gaussian processes are a flexible tool for performing Bayesian inference about functions. An introduction of Bayesian inference is given in Section 2.2.

2.1 Deep Learning

Deep Learning (DL) (LeCun et al., 2015) refers to machine learning algorithms that stack multiple simple, non-linear layers into deep networks that learn representations with multiple levels of abstraction. Such algorithms are often referred to as (deep) neural networks ((D)NN). Neural networks derive their name from being loosely inspired by human neurons. However, most consider neural networks to be just function approximators and do not even aim to model human neurons or the brain (Goodfellow et al., 2016).

While NNs are not a new invention, but the ideas stem from the 50s, they have only fairly recently become extremely popular (LeCun et al., 2015; Schmidhuber, 2015). DL methods have brought significant improvements in several fields such as speech recognition (Hinton et al., 2012), language processing (Devlin et al., 2019; Sutskever et al., 2014) and image processing (Krizhevsky et al., 2012). This sudden success is often attributed to advances in computer hardware, such as GPUs (Raina et al., 2009), availability of large amounts of data and advances in computational methods as well as appearance of several good open source software libraries (Abadi et al., 2016; Al-Rfou et al., 2016) that have made development easy. For extensive introduction to deep learning the readers are directed to the books by Goodfellow et al. (2016) and Zhang et al. (2014).

DL methods iteratively get better when seeing new data, and thus can often require very large datasets to succeed. On the other hand, the methods are able to utilize these large data sets as very little manual engineering is required for feature extraction. While for a long time machine learning techniques were based on manual labour by human engineers and domain experts to carefully construct a feature vector (Andrew, 2001), deep learning methods are able to do this automatically. Taking raw data as input and

automatically learning the relevant representations with a general-purpose procedure is called *representation learning*. In deep learning, the multiple layers are expected to learn different levels of representations.

While DL methods have achieved undeniable success in many areas, it is good to remember that there are also limitations as there are to all methods. NN models are often called black box models as the layers do not necessarily learn concepts or features that are interpretable for humans. The lack of interpretability and the need for explainable AI in many applications has been one of the main sources of critisims for DL and also an area of active research (Gilpin et al., 2018). A related issue is the lack of uncertainty quantification. Traditionally DL models have been criticised of making just point predictions and not quantifying the uncertainty of their predictions. While various methods, including probabilistic and Bayesian neural networks (Gal, 2016), have been developed to address this issue, many of them still fall short (Ovadia et al., 2019). The success of DL is often based on very large data sets combined with large NN architectures. This means, that the training also requires massive computational resources that are both financially and environmentally expensive (Strubell et al., 2020). In addition, DL models are susceptible to adversarial attacks created to mislead the model (Goodfellow et al., 2015).

2.1.1 Structure of Deep Neural Networks

The goal in DL is to learn to approximate a function f that maps the input $\mathbf{x}$ of the algorithm into an output $\mathbf{y}$. Thus, a neural network is basically a mapping $\mathbf{y} = f_\theta(\mathbf{x})$ where the function parameters θ are learned from training data. A characteristic property of DNNs is that the amount of learnable parameters θ is large, often hundreds of millions. The values of the parameters are adjusted by optimizing an *objective function* (also referred to as *loss function*) that expresses how different the current outputs are from the expected outputs.

Neural networks consists of nodes or neurons that map an N dimensional input $\mathbf{x}$ into an M dimensional output $\mathbf{y}$. A node f multiplies each incoming input dimension i with a weight $\mathbf{w}_i$ and outputs their sum combined with a bias term b:

$$f(\mathbf{x}) = g(\sum_{i=1}^{N} \mathbf{x}_i \mathbf{w}_i + b) = g(\mathbf{w}^T \mathbf{x}),$$

where $\mathbf{w}$ is a vector containing the weights and biases. The outputs are mapped through an *activation function* g. In the case of a linear node, this

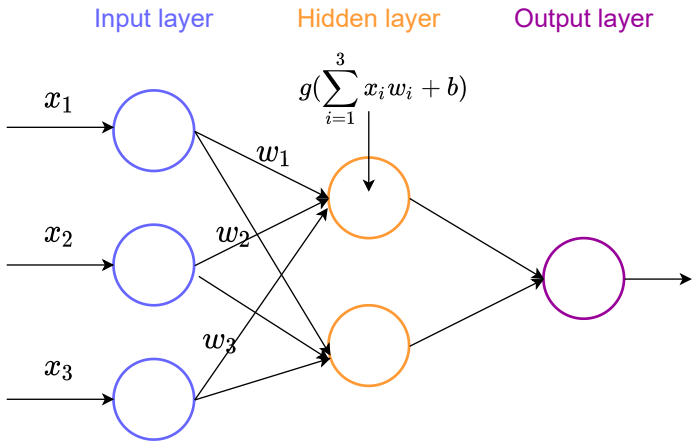


Figure 2.1: Illustration of a simple fully connected feed-forward neural network with one hidden layer.

corresponds to the identity function. However, in most cases g is defined to be some non-linear function, often the rectified linear unit function (ReLU) defined as $g(z) = max\{0, z\}$ (Glorot et al., 2011; Jarrett et al., 2009; Nair and Hinton, 2010). Linear functions are limited in their expressiveness, and adding non-linearities allows learning more complex functions.

Deep neural networks (DNN) consist of multiple layers of multiple nodes. In *feed-forward networks* the output of the previous layer will be used as an input to the next layer. For example, a feed-forward network with three layers f_1, f_2, f_3 is defined as $f(\mathbf{x}) = f_3(f_2(f_1(\mathbf{x})))$. Having multiple layers of non-linear modules further increases the flexibility of the model. Figure 2.1 shows an example of a simple feed-forward network consisting of the input layer, the output layer and one hidden layer.

Traditionally neural networks were *fully connected*, which means that each neuron is connected to all neurons in the previous layer. Later, more specialised structures that use weight sharing to reduce the memory and computational requirements by taking advantage of special structure in the data were introduced. Convolutional neural networks (LeCun et al., 1995), discussed in Section 3.1, are one example of such structure.

2.1.2 Learning Deep Neural Networks

Parameters of a neural network are optimized iteratively using gradient-based optimizers, as is often done with parametrized machine learning models. The learning algorithm computes the gradient of the objective

function with respect to the parameters to indicate which direction the parameters should be adjusted to decrease the error until a (locally) optimal solution has been found. In DL most objective functions become non-convex because of the non-linearities, so optimizers do not necessarily find the globally optimal solution. The gradients in DL models can be efficiently computed using a method called backpropagation (Linnainmaa, 1970; Werbos, 1982).

The values of the weights are adjusted based on some objective function, that can be defined to be the average of losses over training samples:

$$\mathcal{L}(\theta) = \sum_{n=1}^{N} L\left(\mathbf{y}_n, f_\theta\left(\mathbf{x}_n\right)\right)$$

The loss is some non-negative function that describes the difference between the outputs $f_\theta(\mathbf{x}_n)$ and the expected outputs $\mathbf{y}_n$. The loss should be chosen to reflect the problem in hand. For example, for a regression task the squared error can be an appropriate distance measure whereas for classification cross-entropy is often used.

To tackle the computational challenges related to large data sets, DL models are usually learned using stochastic gradient descent (SGD) (Bottou et al., 1998; Robbins and Monro, 1951), where the gradients are computed and the weights adjusted iteratively based on only a small, each time different subset of the data (often called a *minibatch*).

2.1.3 Practicalities of Deep Neural Networks

In practice, there are several design choices that need to be made when implementing deep learning models, as there naturally are with other machine learning models too. The point of this section is not to introduce all possible options as it is out of the scope of this thesis, but to just point out the most important parts that we need to pay attention to when using deep neural networks in an application. An emphasis is put on the choices that were made on `Publications` I and II. While there are some general guidelines, designing neural networks is an area of active research and often the best choices can be found only through trial and error by evaluating the performance of a model on a validation set.

One important choice is to choose the objective function as well as the output of the model. This depends on the task the network is designed to do, and the output layer transforms the output of the last hidden layer into a form that is appropriate to completing the task. For example, in classification it is typical for the output to just give the predicted class,

typically one-hot encoded, while for localisation tasks we might want to directly output the continuous coordinates. In tasks where we want the outputs to be means of Gaussian distributions, it is common to use a linear layer as the output layer. This choice makes using maximum likelihood as the objective function equivalent to using mean squared error. On the other hand, in classification problems a common choice is to use sigmoid function to achieve Bernoulli output distributions.

DNNs are trained using stochastic gradient descent that does not have convergence guarantees when applied to nonconvex cost functions, but is sensitive to the initial values for the model parameters. This is why we also need to pay attention to initialisation when training the models. Several suggestions for this also exists, but often a good technique is to use small random values. Most DL libraries also have different initialisations implemented (Abadi et al., 2016). In addition to SGD, other more efficient optimization methods have also been introduced. Out of these, ADAM (Kingma and Ba, 2015) has been popular and is also used in `Publications` I and II in this thesis. In addition to the algorithm, we need to make choices, such as the amount of iterations used or define a stopping criterion. These algorithms also have different tunable parameters, such as the *learning rate* and the size of the used mini-batches. Learning rate describes the magnitude of the adjustments we make to the parameters in each iteration.

The overall structure of the deep network also matters: how many layers, how many units per layer and how should the units be connected? Are feed-forward networks a good choice for the task, or should we use structures designed for specific tasks? *Dropout* can be used for further regularisation (Srivastava et al., 2014). Again, books by Goodfellow et al. (2016) and Zhang et al. (2021) discuss also these practical choices for DL algorithms in more detail and presents multiple different options.

2.2 Bayesian Inference

Bayesian or probabilistic machine learning uses probabilities to express uncertainty (Gelman et al., 2013; Ghahramani, 2015). Uncertainty in modeling stems from many sources: measurements are noisy, many values for model parameters might be consistent with the data and many different models can be used to describe the data. Benefits of Bayesian machine learning include natural ability to keep track of uncertainties in the model and the resulting automatic complexity trade-off.

In contrast to choosing one best model with point estimates, in Bayesian approach several models can be considered to be consistent with the data.

Thus, the goal is to find the posterior probability distribution over unknown model parameters θ conditioned on the data $\mathcal{D}$: $p(\theta|\mathcal{D})$. Predictions can then be made by computing expectations over this distribution.

Learning from data happens by updating a prior probability distribution, which describes our believes before observing any data, into a posterior distribution. The posterior distribution can be found using the Bayes theorem

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{p(\mathcal{D})}.$$

Here $p(\theta)$ is the prior, $p(\mathcal{D}|\theta)$ is the likelihood and $p(\mathcal{D}) = \int p(\mathcal{D} \mid \theta)p(\theta)d\theta$ is the normalizing constant that ensures that the posterior is a valid probability distribution. The prior can be seen to represent the uncertainty before observing any data, and allows incorporating prior knowledge and formalising the assumptions made. However, even though the ability of incorporating prior knowledge is usually seen as an advantage, it can be also seen as a disadvantage; in Bayesian approach a prior always needs to be specified, even when we do not have any present knowledge.

The task of computing the posterior distribution is known as inference. Unfortunately, in many cases, especially when the dimensionality of the parameters θ is high, computing the integral in $p(\mathcal{D})$ becomes complicated or intractable in practice. By carefully designing the model it is sometimes possible to ease the computations so that we can avoid evaluating $p(\mathcal{D})$ completely. This is usually when we use so called *conjugate* priors. The prior is called a conjugate prior for the likelihood when the posterior and prior distributions are in the same probability distribution family. This is the case for example in Gaussian process regression: the convenient closed-form solutions that have made GPs a popular tool exist only when we use conjugate priors (Rasmussen and Williams, 2006).

However, using conjugate priors means that there is only a limited set of likelihoods and priors that can be used. This allows only a restricted set of models and forces strong assumptions that do not necessarily hold. Approximate inference refers to techniques that can be used to approximately solve the Bayes theorem when more complex models are needed. For example, even though GPs can be solved in closed-form in the standard case, adding constraints or using non-Gaussian additive errors can make the model more flexible and better suited for the problem, but also makes approximate inference necessary.

There are a wide range of solutions for approximate inference that can roughly be divided into two main categories: sampling based methods turn the problem into one of sampling from the posterior distribution, whereas

optimization based methods replace inference with optimization by approximating the posterior with a simpler, tractable distribution. In the following subsections we will shortly describe both of these approaches as we will use them later in this thesis.

2.2.1 Sampling-based Approximations

Sampling-based methods, as the name implies, are based on taking samples from the posterior distribution $\theta_s \sim p(\theta|D)$, which are then used to approximate some needed quantity. This is called a Monte Carlo estimate. More specifically, the expectation of some quantity f can be approximated with S samples as

$$\mathbb{E}_{p(\theta|\mathcal{D})}[f] \approx \frac{1}{S}\sum_{s=1}^{S} f(\theta_s).$$

More comprehensive overview of sampling-based approximations is available for example in Neal (1993), Brooks et al. (2011) and Angelino et al. (2016).

A critical question in sampling based approximate inference is how can we efficiently obtain independent samples from high dimensional distributions. There are various ways that differ in efficiency and accuracy. The broad class of the most popular sampling algorithms used is called Markov chain Monte Carlo (MCMC). Monte Carlo methods refer to non-iterative methods generating independent samples, such as rejection sampling and importance sampling. However, these do not work well in high dimensions. Instead, it is common to use Markov chain Monte Carlo methods that are based on generating samples from a carefully designed Markov Chain to scale better with the dimensionality of the sample space and to reduce the correlation between samples.

Traditionally MCMC methods, such as Gibbs sampling (Geman and Geman, 1984) or random-walk Metropolis (Metropolis et al., 1953), have been seen to be computationally heavy as they require a long time to converge to the target distribution, mostly because of inefficient random walks (Neal, 1993). Hamiltonian Monte Carlo (HMC) is able to avoid the random walk behaviour and increase the efficiency by utilising gradients of the log joint density $log\ p(\mathcal{D}, \theta)$. HMC has hyperparameters that affect the efficiency of the algorithms, and no-U-Turn Sampler (NUTS) (Hoffman and Gelman, 2014) can be used eliminate the need for tuning the values by hand.

2.2.2 Optimization-based Approximations

Optimization-based methods introduce a new, simpler distribution $q_\lambda(\theta)$ that is used to approximate the true posterior $p(\theta|\mathcal{D})$ so that $q_\lambda \approx p$. Its parameters λ are tuned by minimizing an objective function that measures the distance between these two distributions. More formally, Let $\mathcal{Q}$ be a class of tractable distributions called the approximating family. Then the goal is to find $q_\lambda \in \mathcal{Q}$ that minimizes a quantity $\mathbb{D}(q_\lambda \| p(\theta|\mathcal{D}))$ that measures the closeness between these two distributions.

We focus on variational inference (VI) (Jordan et al., 1999; Wainwright and Jordan, 2008). A good overview on variational inference is available for example in Blei et al. (2017), and more recent advances have been reviewed by Zhang et al. (2019). VI uses Kullback-Leibler (KL) divergence as $\mathbb{D}$ as it has convenient properties and it can be optimized using a surrogate objective function. Other divergences are also possible, though commonly the used divergences belong to a family of measures called f-divergence (Csiszár, 1967). The KL divergence is used to measure the differences in information contained within the two distributions, and it is defined as

$$\mathrm{KL}(q\|p) = \int q_\lambda(\theta) \log \frac{q_\lambda(\theta)}{p(\theta|\mathcal{D})}\, d\theta.$$

KL divergence is asymmetric, meaning that $\mathrm{KL}(q\|p) \neq \mathrm{KL}(p\|q)$, and the presented formulation of KL divergence is used in VI.

However, evaluating this measure directly is not possible as it requires knowing p, and thus it can not be directly optimized. Instead, KL can be decomposed into two parts: a part that can be computed, and a constant that is not needed for optimization. Thus, instead of directly optimizing KL, in variational inference we optimize another objective called the evidence lower bound

$$ELBO = \mathbb{E}_q[\log p(\mathcal{D}, \theta)] - \mathbb{E}_q[\log q_\lambda(\theta)].$$

Maximizing this is equivalent to minimizing $\mathrm{KL}(q\|p)$.

There are various different ways that can be used to optimize the ELBO, though often gradient-based optimization techniques are used. In general the techniques for obtaining the gradients can be divided into two categories: score function gradients (Paisley et al., 2012; Ranganath et al., 2014) and reparametrization gradients (Kingma and Welling, 2014; Rezende et al., 2014).

A critical question in variational inference is how to choose the approximating family $\mathcal{Q}$, as it should be flexible enough to be able to find a density

that is close to $p(\theta|\mathcal{D})$ but still simple enough for the optimization. A common choice is to use *mean-field variational family* where the variational family is assumed to be fully factorized and the model parameters θ to be independent

$$q_\lambda(\theta) = \prod_i q_{\lambda_i}(\theta_i).$$

The individual factors can still be defined in multiple ways. Mean-field assumption is efficient, but more complex families have also been suggested (Kingma and Welling, 2014; Saul and Jordan, 1996).

Chapter 3

Machine Learning Models for Sensor Data

The goal of this chapter is to give the necessary background on the methods used in the publications in this thesis. At the same time, this functions as an introduction to methods that can be used to analyse data collected with different sensors, though the list is not exhaustive by any means. All data sets in this thesis are temporal and/or spatial data, which can be seen to have a special structure: the nearby values are known to be correlated, and all presented methods take advantage of this. For more information on these methods as well as other methods suitable for time series analysis interested readers are referred to for example the following books: Billings (2013), Box et al. (2015), Goodfellow et al. (2016), Prandoni and Vetterli (2008), Rasmussen and Williams (2006) and Särkkä (2013).

The model for each application should be chosen to reflect our prior knowledge of the system, the quality and the type of the data and the question we want to answer. A simple linear regression might be an appropriate choice when the data is expected to be linear or we want a simple and quick solution. When only little is known about the system we are modeling, a more flexible model is often better. For example, deep learning can be used to automatically find important features in the data, and CNNs do this more efficiently when the data is structured. Gaussian processes are good for modeling smooth functions, such as spatial or temporal data, and can naturally model uncertainties. When choosing a model we always make assumptions about the system. Modeling complex systems perfectly is usually impossible, but we are required to make assumptions that are inconsistent with the data to make the computations feasible.

3.1 Convolutional Neural Networks

Convolutional neural network (CNN) (LeCun et al., 1989) is a specialised type of neural network that is designed for structured data with spatial or temporal components. Fully connected neural networks do not scale well for large images or signals, as the amount of parameters quickly gets too large when the input resolution is large. The special structure in the data makes it possible to limit the number of parameters.

In spatial and temporal data, such as images, local groups of values are highly correlated (i.e pixels representing a carrot in a picture are similar to each other). In addition, the data is invariant to translations as these correlated groups of values can be located anywhere (there is still a carrot in the picture no matter where it is). Because of these properties, CNNs have been designed so that parameters are shared by different units to detect the same pattern in different parts of the data array. A major benefit of parameter sharing is that it reduces the need for memory as well as training data by reducing the number of learnable parameters.

Depending on the dimensionality of the data and the application, convolutional layers can be used to introduce invariance over one or multiple dimensions. For example, 1D convolutions are useful for analysing signals, 2D convolutions for natural images and 3D convolutions for videos and some medical images such as computed tomography (CT) scans. CNNs have been especially successful with 2D spatial data, leading to significant improvements in many tasks. The CNN based ImageNet (Krizhevsky et al., 2012), for example, almost halved the state of the art error rate for object recognition when it was published. However, CNNs are equally applicable to 1D data and can thus be used to analyse time series data as is done in this thesis.

The name comes from the fact that CNNs use a mathematical operation called convolution in at least one of their layers. It is an operation on two functions x and w, and the output $x * w$ describes how the shape of one is modified by the other. In CNNs the first function x is referred to as the input, and the second function w as the *kernel*. In other words, convolution indicates the places where a kernel and the input data are similar so it can be described as moving a filter window over the data. The output of the operation is often referred to as the *feature map*.

In practice, we are dealing with discrete convolution as the time index t in data sets is discretized using

$$s(t) = (x \star w)(t) = \sum_{a=-\infty}^{a=\infty} x(a)w(t-a).$$

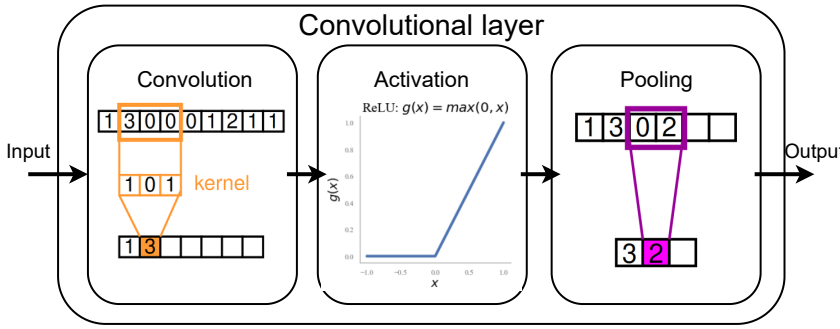


Figure 3.1: Illustration of a convolutional layer that consists of a 1D convolution stage, a non-linear activation function and max pooling stage. **Convolution:** Here is pictured a kernel of size three. The corresponding value is computed as $\sum_{i=1}^{3} w_i x_i = 1 \times 3 + 0 \times 0 + 0 \times 1 = 3$. **Activation:** The activation function pictured here is ReLU, which is defined as $g(x) = max(0, x)$. Other activation functions are possible. **Pooling:** Here is pictured max pooling with window size two. Max pooling gives the maximum value within the window, here 2.

As the amount of elements both in the input and the kernel is always finite, this can be implemented as a summation over finite set of elements by assuming that the functions are zero elsewhere. Given a 2-dimensional kernel $\mathbf{w}$ and a 2-dimensional input $\mathbf{x}$, the feature map or the output $\mathbf{s}$ of the convolution operation is defined as

$$\mathbf{s}[i, j] = (\mathbf{x} * \mathbf{w})(i, j) = \sum_{m=1}^{M} \sum_{n=1}^{N} \mathbf{x}(m, n)\mathbf{w}(i - m, j - n).$$

The indexes of rows and columns of the feature map are marked with i and j and the dimensionality depends on the size of the input image and the kernel as well as the used *stride* and *padding*. Here the dimensionality of the kernel is marked with M and N. Stride defines how much a filter is shifted at each step, and padding refers to additional borders of zeros that are sometimes added to the input. Here the convolution is defined in two dimensions, but naturally both the data and the kernel can have varying amount of dimensions depending on the application. Figure 3.1 illustrates the building blocks of a convolutional layer in 1D case, including the convolution operation.

In addition, almost all CNNs contain an operation called pooling, which is used to introduce invariance to local translations of the input. Pooling replaces an output with a summary of the nearby outputs. Perhaps the

most commonly used summary is to take the maximum value of a neighbourhood. Other options are for example average or minimum. Pooling can also be used to improve computational efficiency as it can reduce the dimensionality of the data by combining all the outputs of a neighbourhood into one single output.

The outputs of convolutional layers are extracted features, and CNNs are commonly extended with traditional fully connected layers that learn to perform the task in hand, such as classification. The same practical issues as described in Section 2.1.3, such as regularisation and initialisation, need to be considered.

3.2 Gaussian Processes

Gaussian Processes (GPs) (MacKay, 1998; O'Hagan, 1978; Rasmussen and Williams, 2006) are non-parametric, Bayesian models that define a probability distribution over functions f. They are Bayesian as the inference is based on updating the prior $p(f)$ into the posterior $p(f|\mathcal{D})$ based on data $\mathcal{D}$. Each of the infinitely many potential functions that can fit the data is assigned a probability. They are non-parametric as there are no explicitly defined parameters θ, but $p(f|\mathcal{D})$ is computed directly without any parameters and the prior $p(f)$ is placed directly on the function.

GPs have been widely used, perhaps due to their many desirable properties: they provide uncertainty estimations of the result, allow incorporating prior information, do not suffer from overfitting, allow computing the predictive posterior distribution in closed form and have convenient model selection procedures. GPs are most commonly used to perform non-linear regression, where the goal is to fit a (non-linear) curve to data. A comprehensive overview of GPs from machine learning perspective is available in the book by Rasmussen and Williams (2006).

More formally, a GP is a collection of random variables for which any finite set of function values have a joint Gaussian distribution. Gaussian process $f(\mathbf{x}) \sim GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$ is a random function in a d-dimensional function space. It is specified by a prior mean function $m : \mathbb{R}^d \to \mathbb{R}$ and a prior symmetric positive-definite covariance function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, also known as the *kernel*:

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})], \\ k\left(\mathbf{x}, \mathbf{x}'\right) &= \operatorname{cov}\left(f(\mathbf{x}), f\left(\mathbf{x}'\right)\right) = \mathbb{E}\left[\left(f(\mathbf{x}) - m(\mathbf{x})\right)\left(f\left(\mathbf{x}'\right) - m\left(\mathbf{x}'\right)\right)\right]. \end{aligned}$$

A major benefit with GPs is that the predictive posterior can be computed in closed form when the observation noise is assumed to be Gaussian.

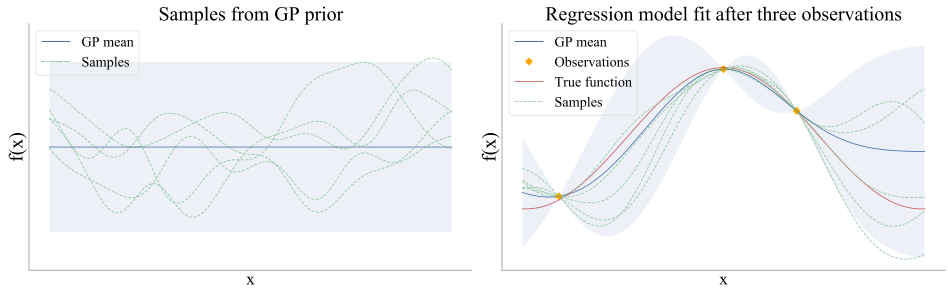


Figure 3.2: A visualisation of a Gaussian process regression with a RBF kernel where we are modeling a one-dimensional function. The solid blue line represents the mean and the shaded blue area the 95% confidence interval of the GP. Green dashed lines show samples from the process. **Left**: A GP not conditioned on any datapoints. **Right**: The posterior after conditioning on three observed data points. The red line represents the true function from which the data points have been observed from.

Let $\mathcal{D} = \{\mathcal{X}, \mathbf{y}\} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ be a data set consisting of inputs $\mathcal{X}$ and noisy scalar outputs of form $y_i = f_i + \epsilon_i$, where $\epsilon_i \sim \mathcal{N}(\mu, \sigma^2)$ and $f_i = f(\mathbf{x}_i)$. We denote all the training outputs with $\mathbf{y}$ and the corresponding training latent function values with $\mathbf{f}$. The predictive distribution of the output $y^* = f^* + \epsilon^*$ at a test location $\mathbf{x}^*$ is a multivariate Gaussian distribution

$$\mathrm{p}(\mathbf{y}^* \mid \mathbf{x}^*, \mathcal{X}, \mathbf{y}) \sim \mathcal{N}(\mathbf{y}^* | \mu, \Sigma), \tag{3.1}$$

where

$$\begin{aligned} \mu &= k(\mathbf{x}^*, \mathcal{X})^\top [k(\mathcal{X}, \mathcal{X}) + \sigma^2 \mathrm{I}]^{-1} \mathbf{y}, \\ \Sigma &= \mathrm{k}(\mathbf{x}^*, \mathbf{x}^*) + \sigma^2 - k(\mathbf{x}^*, \mathcal{X})^\top [k(\mathcal{X}, \mathcal{X}) + \sigma^2 \mathrm{I}]^{-1} k(\mathbf{x}^*, \mathcal{X}). \end{aligned} \tag{3.2}$$

Figure 3.2 illustrates a GP before observing any data as well as after observing three data points.

A part of the appeal of GPs is the ease of encoding prior information into the model through the mean and covariance functions. They describe how the random functions behave and how different points in the input space depend on each other. The prior mean function is almost always assumed to simply be zero everywhere without any loss of generality, and we also do this in this thesis. Information regarding the mean behaviour as well as assumptions about for example smoothness and periodicity can be encoded into the kernel, allowing a wide range of different modeling assumptions. Most kernels have hyperparameters θ, such as the variance and length scale, which determine the correlation structure. These are

typically learned from the data by maximizing the log marginal likelihood, as it also has a closed form expression

$$\log p(\mathbf{y} \mid \mathcal{X}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^\top [k(\mathcal{X}, \mathcal{X}) + \sigma^2 \mathrm{I}]^{-1}\mathbf{y} - \frac{1}{2}\log \left|[k(\mathcal{X}, \mathcal{X}) + \sigma^2 \mathrm{I}]\right| - \frac{N}{2}\log 2\pi.$$

Exact closed-form posterior inference is possible only when the GP prior is combined with a Gaussian likelihood, but in some cases there is a need to use non-Gaussian predictive likelihoods. A classical example of when this is not true is classification, here the process is squashed through a sigmoidal inverse-link function and a Bernoulli likelihood conditions the data on the transformed function values. Additional examples are count data with a Poisson likelihood and constraints achieved through non-linear transformations, such as a non-negativity constraint. In these cases either sampling or optimization approximation strategies described in Section 2.2 need to be used.

Kernels

The choice of a kernel is important in GPs as it describes the characteristics of the function we want to predict. A wide range of possible choices of kernel functions exists leading to a wide range of models. In fact, many common models, such as linear regression or Kalman filters, are examples of GPs with a specific kernel.

A kernel (also known as the covariance function, kernel function) is a function that receives two inputs $(\mathbf{x}_1, \mathbf{x}_2)$ and returns a scalar describing the similarity of those inputs as an output. The kernel is required to be positive semidefinite in order to produce valid covariance matrices. Usually the inputs are points in a Euclidean space, but they can also be for example sets of points (Garnett et al., 2010), lines (Jidling et al., 2018; Purisha et al., 2019), images or text.

The kernel defines the prior covariance between two function values, and different kernels correspond to different assumptions about the function. Sometimes it is possible to design a specialised kernel with detailed domain knowledge incorporated (Wahlström et al., 2013), but usually more general kernel functions are used. Table 3.1 lists some common kernels that we also discuss in this thesis as well as a couple of additional demonstrative examples. More complex structures can be achieved by combining kernels. Duvenaud (2014) provides a more comprehensive overview on kernel functions and their combinations.

Common kernel functions can roughly be divided into two categories: stationary and non-stationary kernels. In stationary kernels the covariance between two function values depends only on the difference between

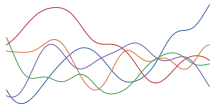
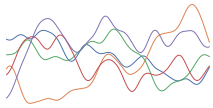
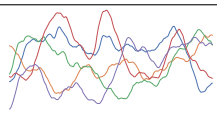
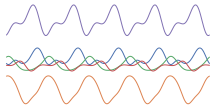
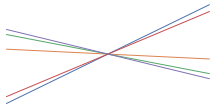
Kernel name	$\mathbf{k}(\mathbf{x}, \mathbf{x}')$	**Samples**
Radial Basis Function Squared Exponential Gaussian kernel	$\sigma_f^2 \exp\left(-\frac{r^2}{2\ell^2}\right)$	
Rational Quadratic	$\left(1 + \frac{r^2}{2\alpha\ell^2}\right)^{-\alpha}$	
Matérn ($\nu = 3/2$)	$\sigma^2(1 + \frac{\sqrt{3}r}{\ell}) \exp(-\frac{\sqrt{3}r}{\ell})$	
Periodic	$\sigma_f^2 \exp\left(-\frac{2}{\ell^2} \sin^2\left(\pi \frac{r}{p}\right)\right)$	
Linear	$\sigma_f^2 (x - c)\,(x' - c)$	

Table 3.1: Examples of common kernels. Most of these are also used in the publications included in this thesis. Here $r = |x - x'|$ denotes the distance between inputs. Parameters σ^2 and ℓ are the variance and the lengthscale. Samples shows examples of samples taken from the prior.

them $\mathbf{r} = \mathbf{x} - \mathbf{x}'$, and the kernel is often expressed as a function of $\mathbf{r}$: $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{r})$. Examples of stationary kernels are periodic kernel and radial basis function (RBF) kernel, while the linear kernel is non-stationary. The definitions of these are given in Table 3.1.

Most kernels have a small number of parameters that usually are referred to as hyperparameters. Common parameters are the lengthscale ℓ that specifies the width of a kernel or the smoothness of the functions, and the signal variance σ^2 that determines the variation of function values from their mean. Lengthscale determines how far it is possible extrapolate from the training data, and with small lengthscales the function values can change quickly while large values lead to smooth functions. Small variance leads to the functions staying close to their mean value whereas larger values allow more variation.

3.2.1 Efficient Computation

Though Gaussian processes have many desirable features, they unfortunately become computationally intractable with large data sets as the computational and memory requirements increase as the cube and square of the amount of data points N. This is due to matrix operations like the need to invert the covariance matrix $k(\mathcal{X}, \mathcal{X}) + \sigma^2\mathrm{I}$ in Equation 3.1, which is an $O(N^3)$ operation. So even though it is possible to analytically write down the posterior, approximate methods are needed because of the cost of computation.

The main idea in solutions that scale to larger data sets is to use only $M < N$ highly informative quantities to learn the GP. Many different sparse approximations that replace the full covariance matrix by a lower M-rank representation have been proposed. These reduces the computational complexity from $O(N^3)$ to $O(NM^2)$ and storage requirements from $O(N^2)$ to $O(NM)$. Intuitively, in regions where there are a lot of observation, there is often redundant information, and approximating this with a smaller set of locations does not loose much information.

The common approach is to use M points in the same domain as the inputs, called inducing points (Quiñonero-Candela and Rasmussen, 2005). Another way is to rely on Fourier features that lie in the spectral domain of the process (Hensman et al., 2017; Solin and Särkkä, 2020). Rasmussen and Williams (2006) provide on overview on earlier methods and Quiñonero-Candela and Rasmussen (2005) and Bui et al. (2017) discuss many methods based on inducing points providing different unifying views on them.

More recently, Wang et al. (2019) also demonstrated that near exact methods with large data set have become possible because of increased computation power and algorithmic advances such as the Blackbox Matrix-Matrix multiplication inference procedure (Gardner et al., 2018).

Here we briefly describe two different approaches: inducing points and Hilbert space approximations. It turns out, that Hilbert space approximations are useful when dealing with integral observations (Jidling et al., 2018; Purisha et al., 2019), which we briefly discuss in Section 5.2. Inducing points we use in `Publications` V and VI to allow scaling the presented method for larger data sets also.

Hilbert space approximation

Hilbert space approximation of a covariance function, originally presented by Solin and Särkkä (2020), is a useful reduced-rank approximation for GP regression for stationary kernels. The computational cost can be reduced

to $O(NM^2)$ by representing the covariance function k with $M < N$ basis functions as

$$k\left(\mathbf{x}, \mathbf{x}'\right) \approx \sum_{j=1}^{M} S\left(\sqrt{\lambda_j}\right) \phi_j(\mathbf{x}) \phi_j\left(\mathbf{x}'\right).$$

Here $S(\boldsymbol{\omega})$ is the spectral density, which is the Fourier transform of the kernel. For example, for the RBF kernel it is defined as

$$S(\boldsymbol{\omega}) = \sigma_f^2 (2\pi)^{d/2} l^d \exp\left[-\frac{l^2 \|\boldsymbol{\omega}\|_2^2}{2}\right],$$

where d is the dimensionality of $\mathbf{x}$. The other terms, λ_j and the basis functions ϕ_j are the jth eigenvalue and the eigenfunction of the Laplace operator in a given domain. These also have simple closed-form expressions. The expressions, details and derivations can be found in Solin and Särkkä (2020).

This approximation is especially useful as the convariance function parameters are independent from the basis functions, which allows for efficient hyperparameter learning ($O(M^3)$). It also decouples the arguments $\mathbf{x}$ and $\mathbf{x}'$, which makes more efficient computations possible in the case of linear functional evaluations, such as integral observations (Jidling et al., 2018; Purisha et al., 2019).

Inducing points

Most of the sparse GP methods are based on inducing points. Specifically, a set of M points $\{\mathbf{z}_m\}_{m=1}^M$, called the *inducing inputs*, is introduced in the input space and $\mathbf{f}$ is summarized with *inducing points* or *inducing variables* $\mathbf{u} = f(\mathbf{z})$. Inducing points can be selected to be a representing subset of the dataset, but they can also be auxiliary pseudo-inputs (Bui et al., 2017; Quiñonero-Candela and Rasmussen, 2005; Snelson and Ghahramani, 2005). The locations of these pseudo-inputs can be optimized, though this sometimes can lead to overfitting (Quiñonero-Candela and Rasmussen, 2005). However the locations of the inducing inputs are chosen, they should cover the the same region as the data. Also, for efficiency M should naturally be smaller than the number of data points N.

We can recover $p(\mathbf{y}, \mathbf{f})$ by integrating out $\mathbf{u}$ from the joint GP prior $p(\mathbf{y}, \mathbf{f}, \mathbf{u})$

$$p(\mathbf{y}, \mathbf{f}) = \int p(\mathbf{y}, \mathbf{f}, \mathbf{u}) \mathrm{d}\mathbf{u} = \int p(\mathbf{y}, \mathbf{f}|\mathbf{u}) p(\mathbf{u}) \mathrm{d}\mathbf{u}.$$

The methods used to solve this can roughly be divided into two categories: a) approximating the model with a simpler one (Quiñonero-Candela and

Rasmussen, 2005) and b) keeping the original model but deriving an approximate inference method (Qi et al., 2010; Titsias, 2009).

Approximating the model usually means making different independence assumptions for the conditional distribution $p(\mathbf{y}, \mathbf{f}|\mathbf{u})$. The most common approach is to assume that $\mathbf{y}$ and $\mathbf{f}$ are conditionally independent given $\mathbf{u}$ (Quiñonero-Candela and Rasmussen, 2005)

$$p\left(\mathbf{y}, \mathbf{f}\right) \simeq q\left(\mathbf{y}, \mathbf{f}\right) = \int q\left(\mathbf{y}|\mathbf{u}\right) q(\mathbf{f}|\mathbf{u}) p(\mathbf{u}) \mathrm{d}\mathbf{u}.$$

Different methods use different approaches to approximate the conditionals $p(\mathbf{y}|\mathbf{u})$ and $p(\mathbf{f}|\mathbf{u})$, including the subset of regressors (Smola and Bartlett, 2001), DTC (Seeger et al., 2003), FITC (Snelson and Ghahramani, 2005) and PITC (Quiñonero-Candela and Rasmussen, 2005). The fully independent training conditional (FITC) (Snelson and Ghahramani, 2005) assumes that the function values $\mathbf{f}$ are independent conditioned on the inducing points: $q(\mathbf{f}|\boldsymbol{u}) = \prod_{n=1}^{N}(\mathbf{f}_n|\boldsymbol{u})$ and $q(\mathbf{y}|\boldsymbol{u}) = p(\mathbf{y}|\boldsymbol{u})$. FITC simplifies the inference and has recently been shown to have good empirical performance (Jankowiak et al., 2020; Rossi et al., 2021).

Instead of approximating the model, sparse variational GP methods reformulate the posterior inference problem as variational inference (Titsias, 2009). A key difference is that pseudo-inputs are considered to be parameters of the approximation, and they are jointly selected with the hyperparameters by minimizing the KL divergence between a variational GP and the true posterior GP. The method was originally introduced by (Titsias, 2009), but since then others have extended the work for example by reformulating the variational objective and by extending it to non-conjugate models and Gaussian process state space models (Frigola et al., 2014; Hensman et al., 2013, 2015; Matthews et al., 2016).

3.2.2 Sensor Placement

Strategies for choosing the locations of sensors in standard Gaussian processes have been well studied (Garnett et al., 2010; Krause and Guestrin, 2007; Krause et al., 2008; Seo et al., 2000; Zhu and Stein, 2006). Depending on the application, the sensor locations can be chosen completely before making any measurements (Krause et al., 2008), or sequentially based on observations made by previously selected sensors (Krause and Guestrin, 2007). In model-based approaches, when the kernel function is fully known, we do not loose anything by performing the sensor selection before making any measurements as the posterior variance does not depend on the actual

observed values. Pre-selecting all sensors is also the only option when using geometric approaches as they are model independent. However, in most cases we do not know the kernel parameters, but they are optimised, and in such cases choosing the sensors sequentially leads to better results. The same applies if non-Gaussian likelihoods are needed.

Whether using pre-selection or sequential techniques, the best results are typically obtained by model-based methods searching for sensor locations informative of the underlying function (Krause and Guestrin, 2007; Krause et al., 2008), but also geometric approaches covering the space maximally have been considered (González-Banos, 2001). Next, we briefly describe both of these strategies for standard GPs.

Model-based approaches

Model-based approaches are based on optimizing some measure of information content, such as mutual information or entropy (Krause et al., 2008). However, the main issue with these methods is that finding an optimal placement is NP-complete (Ko et al., 1995; Krause et al., 2008). Thus, typically the placement algorithms choose the locations greedily, i.e choosing one sensor at a time and placing the next sensor to the location where the change in the wanted measure is the highest at that point. This also means that in practice we can consider only a finite subset of possible locations $\mathcal{V}$ from which we choose a subset $\mathcal{A}$ of locations where the sensors are placed.

An intuitive choice for the measure of information is to use the entropy criterion H (Cressie, 2015; Shewry and Wynn, 1987)

$$\begin{aligned}\mathcal{A}^* &= \text{argmin}_{\mathcal{A}\subset\mathcal{V}:|\mathcal{A}|=N} H\left(\mathcal{V}\backslash\mathcal{A}|\mathcal{A}\right) \\ &= \text{argmax}_{\mathcal{A}\subset\mathcal{V}:|\mathcal{A}|=N} H(\mathcal{A}).\end{aligned}$$

Minimizing the uncertainty of unobserved locations after placing the sensors is equivalent to finding a set of sensors that are most uncertain about each other. Considering only the selected locations usually results into choosing locations that are as far from each other as possible, and unfortunately sensors tend to be placed along the borders of the area (Ramakrishnan et al., 2005). As mentioned, optimizing the entropy criterion is an NP-hard problem (Ko et al., 1995), and in practice sensors y are chosen greedily by selecting the location y^* from $\mathcal{V}\backslash\mathcal{A}$ that maximally increases the conditional entropy (Cressie, 2015; Krause et al., 2008; McKay et al., 1979)

$$y^* = \text{argmax}_y H(y|\mathcal{A}) = \text{argmax}_y \frac{1}{2}\log(2\pi e\sigma^2_{y|\mathcal{A}}).$$

Using mutual information $I(\cdot,\cdot)$ (MI) (Caselton and Zidek, 1984) fixes some of the issues related to the entropy criterion and leads to better accuracies, as it looks for locations that are most informative about the unused locations using

$$\mathcal{A}^* = \text{argmax}\, I(\mathcal{A}; \mathcal{V}\backslash\mathcal{A}).$$

In practice, we solve for (Krause et al., 2008)

$$\begin{aligned} y^* &= \text{argmax}_y\, H(y|\mathcal{A}) - H(y|\mathcal{V}\backslash(\mathcal{A}\cup y)) \\ &\propto \frac{\sigma^2_{y|\mathcal{A}}}{\sigma^2_{y|\mathcal{V}\backslash(\mathcal{A}\cup y)}}. \end{aligned}$$

However, using MI is computationally more demanding and becomes unfeasible with large sets of possible locations. Krause et al. (2008) present some computational tricks to make the computation more efficient, such as using lazy evaluations with priority queues or by utilizing the locality of the kernels.

Geometric approaches

Geometric approaches refers to algorithms that take advantage of the geometric properties of the are of interest. They only consider the area and not the model, thus the same result can be used with various different models and we do not need to assume a GP when designing the placement.

In cases where the observations are points in the measurement space, the geometric options are limited. The intuitive and standard approach is to assume that each sensor can observe an area with a fixed radius. The locations are then be optimized so that the area can be covered with as little sensors as possible (González-Banos, 2001). This is also referred to as the disk model (Kershner, 1939). Geometric approaches become more interesting when the observations are for example lines in the measurement space, as discussed in Section 5.5.

3.3 Gaussian Process State Space Models

State Space Model (SSM) (Särkkä, 2013) is a common tool for modeling time-series data with dynamical behaviour, i.e. data where there are non-trivial correlations between different data points. Examples of such applications are forecasting the daily number of new cases in epidemiology (Zimmer and Yaesoubi, 2020) or predicting the emissions of a car engine (Yu

et al., 2020). SSMs are popular in fields where dynamical time series data is common, such as signal processing, control, ecology and finance.

Gaussian process state space model (GPSSM) (Eleftheriadis et al., 2017; Frigola et al., 2013, 2014; Ko and Fox, 2009; Turner et al., 2010; Wang et al., 2007) allows for a Bayesian approach to SSMs leading to the many benefits described in Section 3.2. Especially, the natural ability of Bayesian methods to model uncertainty makes them well suited for physical sensing applications where amounts of data can be limited or interesting phenomena might be rare and only present in a small subset of the data. We start this section by introducing the general state-space model, after which we focus on the special case of GPSSMs.

3.3.1 State Space Models

The key assumption in SSMs is that there exists some *latent state*, denoted by $\mathbf{x}_t \in \mathbb{R}^{D_x}$, at a particular time point t, that compresses the history of the system and fully describes the system at each time point. To allow more efficient learning of the state-space model, the states are defined so that once $\mathbf{x}_t$ is known, the previous states $\mathbf{x}_0, ..., \mathbf{x}_{t-1}$ don't add any new information about the later states $\mathbf{x}_{t+1}$. This is called a *Markov property* and can more formally be written as

$$p\left(\mathbf{x}_{t+1} \mid \ldots, \mathbf{x}_{t-1}, \mathbf{x}_t\right) = p\left(\mathbf{x}_{t+1} \mid \mathbf{x}_t\right).$$

In some cases, the states can be physically interpretable, such as a direction, speed and acceleration of a car. These are sufficient for predicting the position of the car, and knowledge of past values does not provide any additional information. More often, we are just interested in learning the dynamics from data and making predictions, while the states do not need to have any specific meaning but to just act as a summary of the past.

Even though the system states are a key assumption in SSMs, it is not possible to directly observe these as the term latent implies. Instead, we assume that we can only make partial, noisy observations $\mathbf{y}_t \in \mathbb{R}^{D_y}$ at time point t.

More specifically, a discrete-time SSM can be written as

$$\begin{aligned} \boldsymbol{x}_0 &= p(\mathbf{x}_0), \\ \boldsymbol{x}_{t+1} &= f\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right) + \boldsymbol{\epsilon}_t, \\ \boldsymbol{y}_t &= g\left(\boldsymbol{x}_t\right) + \boldsymbol{\gamma}_t, \end{aligned} \tag{3.3}$$

where $\boldsymbol{\epsilon}_t$ is the process noise and $\boldsymbol{\gamma}_t$ is the measurement noise. The relation between the states is described by a *transition function* f, and the

observation function g describes the relation between the states and the observations. In addition, a density $p(\mathbf{x}_0)$ for the initial state $\mathbf{x}_0$ needs to be defined. Here the states are assumed to be continuous, and a corresponding model with discrete states is usually called a hidden Markov model.

Sometimes, in addition to the outputs $\mathbf{y}_t$, we can also observe an input signal $\boldsymbol{u}_t$ at time t that can influence the system. This is used especially in the automatic control context, and an example of such input is for example turbulence affecting an aircraft. To simplify the notation, it is typical to leave the inputs out from the equations. After this section we will exclude the inputs from all equations to avoid clutter, but note that they could be easily included when needed.

Inference and learning in linear and non-linear SSMs

SMMs can be used to model different kinds of problems. In some cases, like when the states correspond to known physical properties such as a speed of an object, the model, i.e the transition function f and observation function g, might be completely known and the only goal is to infer the states $\mathbf{x}_{1:T}$. Perhaps more often the real physical process is not fully known and in addition to the states, the functions f and g and their parameters need to be determined.

In principle there is no difference between the states and the model parameters as both are unknown quantities of the model. However, in many cases the dimensionality of function parameters is fairly low, whereas the state sequence can be long and grows with the size of the data $\mathbf{y}$. Thus, because of efficiency the inference of these are often handled differently. Finding the posterior distribution of the latent states $\mathbf{x}_{1:T}$ based on the data is often referred to as inference while inferring f and g is called learning the model.

Inference and learning in linear SSMs, where the transition and observation functions f and g are linear functions, is a well studied area and many efficient algorithms exist. In fact, many problems can be solved in closed form for example with a Kalman filter/smoother (Kalman, 1960). However, linear SSMs have limited expressiveness and are often too simplistic for many real world applications. Nonlinear extensions of SSMs are more flexible and thus often more useful in practice. Typically they assume some parametric form for the transition and observation functions f and g. Both probabilistic and deterministic approaches to solving non-linear SSMs exists. A comprehensive introduction to many methods such as non-linear Kalman filtering and smoothing and particle filtering and smoothing can be found in Särkkä (2013).

3.3.2 Gaussian Process Prior on the Transition Function

Gaussian process state space models (GPSSM) combine GPs and SSMs into probabilistic, non-linear SSMs (Deisenroth and Rasmussen, 2011; Wang et al., 2005). The Bayesian non-parametric approach brings many benefits that have made GPSSMs an interesting model for example in reinforcement learning (Curi et al., 2020; Dörr, 2021), human motion tracking (Pöhler et al., 2019) and modeling financial time-series data (Wu et al., 2014).

There are multiple different dynamical system models based on GPs, of which Frigola (2015) gives a good overview. For example, it is possible to place a GP prior on the observation function g (Ferris et al., 2007), the transition function f (Frigola, 2015; Wang et al., 2005), or both the observation and the transition function (Frigola et al., 2014; Turner et al., 2010).

In this thesis GPSSMs refer to state space models with a GP prior on the transition function: $f \sim GP(m(\cdot), k(\cdot,\cdot))$. This allows capturing complex dependencies in dynamical systems and encoding prior knowledge about the system, such as assumptions about the smoothness of the states, into the GP. For example, $m(\cdot)$ can be based on knowledge of the underlying physics of the model.

More specifically, we discuss models where a GP prior is placed on the transition function f of the SSM model defined in Equation 3.3. The model can then be written as

$$\begin{aligned} f(\mathbf{x}) &\sim \mathcal{GP}\left(m(\mathbf{x}), k\left(\mathbf{x}, \mathbf{x}'\right)\right), \\ \mathbf{x}_0 &\sim p\left(\mathbf{x}_0\right), \\ \mathbf{x}_{t+1} \mid \mathbf{x}_t &\sim \mathcal{N}\left(f\left(\mathbf{x}_t\right), Q\right), \\ \mathbf{y}_t \mid \mathbf{x}_t &\sim \mathcal{N}\left(C\mathbf{x}_t + \mathbf{b}, \Omega\right). \end{aligned}$$

The covariance Q describes the process noise and Ω describes the observational noise. It is common to choose a simple, parametric observation model g since it simplifies the computations but does not reduce the generality of the model when the latent state is sufficiently large (Frigola, 2015). Here we have defined the observation function to be linear $g(\mathbf{x}_t) = C\mathbf{x}_t + \mathbf{b}$. However, more complex observation models are naturally also possible. Another common design choice for reducing computational complexity is to use independent GPs for each latent dimension (Curi et al., 2020; Doerr et al., 2018).

Inference in GPSSMs cannot be performed analytically because of the non-linear dynamics. Some works have presented solutions for inferring the latent states when assuming that the system is already known (Deisenroth

and Rasmussen, 2011; Ko and Fox, 2009). However, information about the system is rarely available, and we are interested in the case where the latent state distribution is optimized together with the system model. The solutions mostly build on the approximate inference techniques introduced in Section 2.2. Sampling-based solutions are accurate but too inefficient for large data sets (Frigola et al., 2013).

The variational inference solutions differ on the independence assumptions they make in the form of the approximate posterior to improve computational efficiency. Many methods introduce independence between the latent states and the transition function (Frigola et al., 2014), between the states (Mattos et al., 2016) or assume a linear latent state structure (Eleftheriadis et al., 2017). While these increase the computational tractability, they oversimplify the model which can lead to biased, sub-optimal solutions. More recent solutions have introduced non-factorised approximate posteriors that allow preserving the temporal correlations of the prior distribution (Curi et al., 2020; Doerr et al., 2018; Ialongo et al., 2019).

Sparse GPSSM

A standard solution to learning GPSSM models is to use variational inference to optimize a Monte Carlo estimate of the lower bound (Kingma and Welling, 2014). However, computing the lower bound scales cubically with the number of time points $O(T^3)$. This is because complete latent state trajectory $\mathbf{x}_{1:T}$ needs to be sampled, which requires incrementally building a $T \times T$ matrix inverse. Sparse GPs can be used to reduce the computational complexity to $O(T)$ (Curi et al., 2020; Doerr et al., 2018; Ialongo et al., 2019).

Doerr et al. (2018) use the FITC approximation (Snelson and Ghahramani, 2005) described in Section 3.2.1, where the function values are assumed to be independent conditioned on a set of inducing points $\mathbf{f}_M$. This leads to the model

$$\begin{aligned} \mathbf{f}_M &\sim \mathcal{N}(\mathbf{f}_M|0, \mathrm{K}_{MM}), \\ \mathbf{f}_t|\mathbf{f}_M &\sim \mathcal{N}(\mathbf{f}_t|\mu(\mathbf{x}_t), \Sigma(\mathbf{x}_t)), \\ \mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{f}_t &\sim \mathcal{N}(\mathbf{x}_{t+1}|\mathbf{x}_t + \mathbf{f}_t, \mathrm{Q}), \end{aligned} \tag{3.4}$$

where $\mathbf{f}_t$ are the GP predictions at time index t with mean $\mu(\mathbf{x}_t)$ and covariance $\Sigma(\mathbf{x}_t)$ as defined in Equation 3.2.

Another benefit of this formulation, in addition to reducing the computational complexity, is that it allows decomposition of the lower bound between the time points. This makes it possible to use mini-batches (Hensman et al., 2013; Salimbeni and Deisenroth, 2017), which further reduce the

computational complexity by reducing the length of the latent state trajectories T that needs to be sampled. Furthermore, long trajectories converge poorly because of vanishing and exploding gradients Pascanu et al. (2013), and using shorter mini-batches reduces this issue also.

Chapter 4

Convolutional Neural Networks for Signals

In Section 3.1 we described how CNNs have been designed for analysing data where spatial or temporal invariance is needed. Many signal processing applications are based on such data, and CNNs have been successfully used for example in speech recognition (Palaz et al., 2015) and EEG signal classification (Cecotti and Graeser, 2008). In this chapter we discuss using CNNs to analyse signal data collected by physical sensors in two practical applications that have been presented in `Publication` I and `Publication` II.

Signals are sequences of data points over time, and commonly we are looking for features in these signals that are invariant of time, i.e can be located anywhere in the signal. Some signals are multidimensional and we might need invariance over other domains also. Figure 4.1 gives examples of different types of signal data. Whether to use convolutions only over the time domain (Lee et al., 2009), or to use higher order convolutions, such as 2D convolutions to find features that are invariant over both frequencies and time, depends on the application in hand. Sometimes, CNNs can be used to induce invariance only over other domains than time, while using other techniques to model temporal continuity (Abdel-Hamid et al., 2012, 2014).

In our applications we use convolutions over the temporal domain in both cases, as the features we are looking for can be located anywhere in time. In `Publication` II, the signals are one dimensional, so 1D convolutions are all we can utilize. In `Publication` I, the data is two dimensional and, in fact, we do need invariance over the frequency domain also.

In both publications the data has been recorded with one receiving sensor and collected manually. This means that the amount of data we

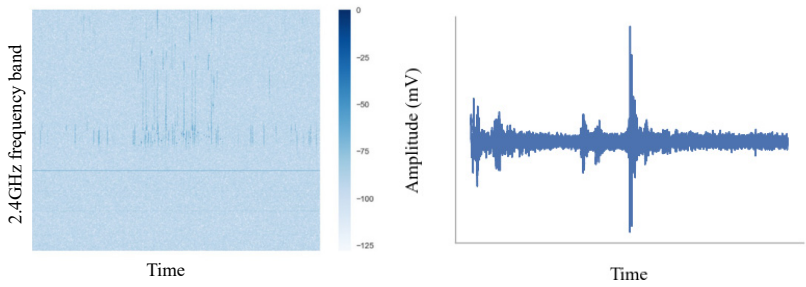


Figure 4.1: Examples of different types of signal data. **Left:** Example of data measurements pictured in `Publication` I. This one is a measurement of microwave oven at 800w. This is a three minute measurement where the microwave oven was turned on for one minute in the middle. We use 2D convolutions in the analysis as we need invariance in both the frequency (y axis) and the temporal domain (x axis). **Right** Example of a measured pulse-echo signal used in `Publication` II. The data in case is one-dimensional so we use 1D convolutions to find patterns in the signals.

have is limited. Deep learning methods on the other hand are known to work well especially in cases where amounts of data are extremely large (Krizhevsky et al., 2012). This is why in Section 4.1 we discuss ways to use CNNs with small amounts of data.

Even though we are able to solve the applications presented in this section using CNNs, it is good to note that they are by no means the only method that could be used. For example, recurrent neural networks have been designed for time series data, and have worked great in cases where invariance over time domain is needed, such as language modelling (Mikolov et al., 2010) and speech recognition (Graves et al., 2013).

4.1 Training CNNs with Limited Data

In many real world sensing applications collection of data can be expensive or impractical. Often the amount of sensors is limited, sensors can be expensive and placing sensors as well as making measurements requires manual labour. DL methods on the other hand are traditionally data hungry (Krizhevsky et al., 2012). In this section we present two alternative approaches to deal with limited amount of data in the context of deep learning. The first one is based on efficiently using all available data by utilising cross-validation. The second one is based on using unlabeled data

in addition to labeled data (Chapelle et al., 2006), as data collection in general is often easier than accurately labeling the data.

There are naturally multiple other ways to deal with limited data. One popular approach is to augment labeled data with transformations (Krizhevsky et al., 2012). This is especially popular in natural image classification, where for example by turning, shifting, reversing, adding noise, modifying contrast or the lighting etc, it possible to get more training data that is still accurately labeled. While it is easy to imagine that these transformations work for natural images and can improve the classification accuracy, for signal data augmentation is not as simple. In many cases, like with the data from `Publication` I and `Publication` II, it is not straight forward or even possible to design rich transformations that would be guaranteed to preserve the labels.

This is by no means an exhaustive list of ways to deal with limited data, but only provides examples as well as shortly describes the methods we have chosen to use in our publications. Sometimes, the best technique is to choose a model family that is better suited for limited data than DL is. For example, Bayesian methods are traditionally considered to be better as it is possible to incorporate prior information and we get better estimates about the uncertainty of the model (Ghahramani, 2015).

Cross-validation

In some cases, we have no additional data that can be used in the training, but we need to cope with the limited amount we have in hand. This is the case in `Publication` II. There we have manually collected data from 150 different locations. Here collecting unlabeled data would not be any easier, and data augmentation not possible. In this case, in order to have enough data for training, testing and validation, we have used cross-validation. Cross-validation has been used extensively in the past with many different machine learning models. It is based on training the model multiple times by dividing the data into subsets, and each time using some of the subsets for training and some for evaluation. This allows maximizing the amount of used data in training while still being able to evaluate the performance on multiple data points.

Using unlabeled data

Sometimes, acquiring labels for the data is the bottle neck in the process, while recording unlabeled data is easier. In these cases, it is possible to use pre-training with unlabeled data (Erhan et al., 2010; Lee et al., 2009) or

semi-supervised learning, which refers to using both labeled and unlabeled data in the training process (Chapelle et al., 2006). Unsupervised pre-training, however, is a hard task on its own, often requiring a generative model for the data. This mostly means, that the amount of unsupervised data is required to be substantially larger than the amount of labeled data before considering designing a generative model for the domain makes sense.

Semi-supervised learning uses both labeled and unlabeled data in training a model. The standard and perhaps most intuitive way to do this is to alternate between inferring the labels for unlabeled samples and then using both the labeled data as well as the unlabeled data with inferred labels to again update the model (Blum and Mitchell, 1998; Ghahramani and Jordan, 1994).

In deep learning context predicting labels for unlabeled samples has been referred to as using pseudo-labels (Lee et al., 2013), and we build on this work in our publication. Similar technique has also been used for example in image classification with small data sets by augmenting the data with images from online image search (Kolář et al., 2016). We add to these by including a component that takes advantage of the continuity assumption in our data to make assigning pseudo-labels more reliable. We describe this in more detail below.

More advanced techniques for semi-supervised learning naturally exists also, including pseudo-ensemble agreement (Bachman et al., 2014) and semi-supervised learning with ladder networks (Rasmus et al., 2015) that directly have improved the results achieved by Lee et al. (2013). However, we do not go into details on other methods in this thesis, as the simple pseudo-label technique was easy to extend and enough to solve the problem in `Publication` I.

4.1.1 Structured Pseudo-labels

In `Publication` I we extend the concept of pseudo-labels (Lee et al., 2013) into structured pseudo-labels. The idea is to take advantage of the structure in the data, i.e the continuity in time, when assigning the pseudo-labels for unlabeled data points.

Let N and N' be the number of labeled and unlabeled samples, f_m and $f_m^{'}$ the outputs of labeled and unlabeled data and y_m and $y_m^{'}$ the true labels of labeled data and the pseudo-labels. The optimization problem with pseudo-labels is then defined as

$$\mathcal{L}(\theta, y') = \frac{1}{N}\sum_{m=1}^{N} L(y_m, f_m) + \alpha(t)\frac{1}{N'}\sum_{m=1}^{N'} L(y_m^{'}, f_m^{'}). \tag{4.1}$$

The coefficient $\alpha(t)$ can be used to scale the weight of pseudo-labels during the training. It can be given a constant value or it can be slowly increased during the training process when the predictions get better (Lee et al., 2013).

Since we have temporal data, we encourage the predictions to be smooth over the sample space instead of independently labeling each sample. The same idea can be used also for other types of data with continuity. We encourage continuity by adding a new term to the loss function that adds a penalty when consecutive samples are assigned a different label. The penalty term to be added to Equation 4.1 is

$$\lambda \sum_{m=2}^{n'} \mathbb{I}[y_i'^m \neq y_i'^{m-1}],$$

where λ can be used to scale the weight of the penalty and $\mathbb{I}$ is the identity function that evaluates to 1 when the consecutive labels are not equal and to 0 otherwise.

When we add this term to the loss function, the assignment of pseudo-labels can not be done independently as usually. Instead, we use a dynamic programming algorithm corresponding to the Viterbi algorithm used for Hidden Markov models (HMMs). The same algorithm can be used to encourage continuity also in the test set or the actual predictions. Pseudocode and details can be found in `Publication` I.

4.2 Application: Identifying Sources of Wi-Fi Interference

In `Publication` I we use CNNs to identify radio frequency devices from signal data. The goal is to improve wireless local area networks by detecting interference sources. For example, every day devices such as baby monitors, microwave ovens and Bluetooth devices operate in the same frequency range as Wi-Fi and can have a negative impact on the performance of a Wi-Fi network.

We have data measured with a device that performs sweeps over the 2.4GHz spectrum, providing roughly 3000 features for every data point measured, which is once per 40ms. An example of such raw data is given in Figure 4.1 (left). We have collected data from 14 different possible sources of interference. We split the data into partially overlapping two-second windows, each corresponding to 50 consecutive time points, making the input two-dimensional spectrograms where one dimension is over the frequencies and the other over time.

In this case, we need invariance over both domains so we use 2D convolutions. Naturally, there can be devices on anywhere in time. Different devices also operate in different frequencies, and some devices, such as bluetooth, can switch frequencies, so we need to be able to identify similar patterns everywhere in the frequency domain also. We build a simple CNN with two convolutional layers and one fully connected layer with dropout. The output layer outputs the classification label one-hot encoded. We use categorical crossentropy as the objective function.

Previously to our work, Rayanchu et al. (2011) used manually designed features and decision trees to accurately recognise 8 different devices. The drawback of this is that a separate classifier needs to be trained for each device, and the features need to be hand picked. In our work we compare against a baseline motivated by this system, though all features can not be incorporated as our data is too sparse in time compared to the data these features were designed for.

When using all available data, we can solve the classification task with 97% accuracy, while the baseline only reached a 79% accuracy. We also demonstrate that the features the CNN learns generalize over devices and the model can thus be fairly easily be extended to new devices: we show that when adding a new device to be classified, it is sufficient to only train the last layer of the network to recognise the new device.

As the data was collected manually, accurate labeling of the data is challenging especially near the time points where devices were turned on or off. Using semi-supervised learning, and especially enforcing temporal continuity when assigning pseudo-labels, increases the classification accuracy when we have only little accurately labeled training data available. An interesting extension to the method could allow modifying the labeled data when it looks like they have been given a wrong label. Labeling data is a challenging task and mistakes happen. In this case, especially, precise labeling is not possible as even if we know a device was turned on, there might not have been any detectable interference.

4.3 Application: Using Ultrasound to Localise a Target

Ultrasound signals are used in many environmental sensing applications, and in `Publication` II we use them to locate a target inside a container. We place a transducer that sends and receives ultrasound signals outside the container, and based on the received signals that have travelled through the container we can deduct the location of a target.

With one transducer the acoustic aperture is limited, and we use a chaotic cavity (CC) to disperse the field (Montaldo et al., 2005). Chaotic cavity works by scattering the waves randomly. This increases the acoustic aperture but also the complexity of the problem due to the chaotic scattering that makes signal interpretation by conventional means difficult. Thus, we use CNNs to automatically detect the needed features. The inputs are the recorded signals, an example of which is shown in Figure 4.1 (right).

We trained a fairly simple CNN consisting of two 1D convolutional layers with the recorded signals as inputs and the corresponding locations of the target as outputs. The loss function to be minimized was the mean squared error between the predicted location and the true location.

The main goal was to show that using a chaotic cavity increases the area inside the container we are able to map, while still allowing us to infer useful information about the locations of targets inside the container despite of the added complexity to the signals. Figure 4.2 illustrates the experimental setting and the improvements we achieve by using a chaotic cavity.

We use Gaussian process regression to visualise the accuracy of the model in different parts of the container: we fit a GP with the distances between true and predicted locations, and the mean of this GP is shown with and without a chaotic cavity. These plots demonstrate that without a chaotic cavity it is possible to observe targets only in the middle of the container, directly in front of the transducer. A chaotic cavity increases the accuracy of localisation around the edges: the area inside the 30 mm contour line (black) is 49% larger with a chaotic cavity than without it.

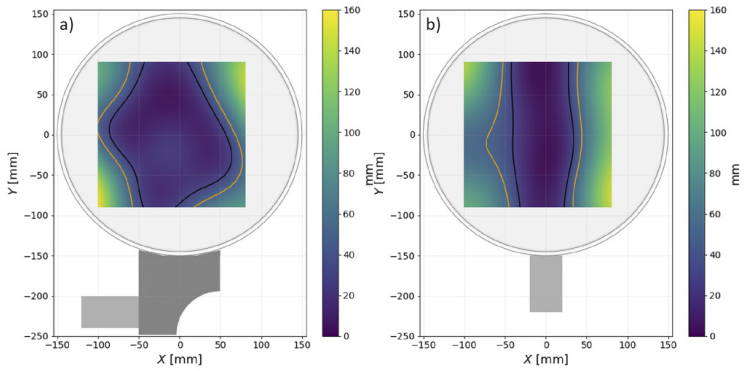


Figure 4.2: **Advantage of using a chaotic cavity.** Mean of a GP estimate of the distance between the true locations and predicted locations with a chaotic cavity (left) and without a chaotic cavity (right). The black and orange contour lines represent the 30mm and 50mm distances. The diameter of the target was 25mm. The light gray box represents the transducer while the darker gray component on the left represents the chaotic cavity. Picture from `Publication` II.

Chapter 5

Gaussian Processes with Integral Observations

As explained in Section 3.2, Gaussian processes (Rasmussen and Williams, 2006) are a convenient tool for modeling functions as it is possible to infer the posterior distribution in closed form. GPs have been popular especially in regression modeling, where the goal is to learn a mapping between a scalar input and an output. However, the closed-form inference applies also after linear functional transformations, meaning that GPs are applicable to a wide set of diverse observations, such as values of integrals or derivatives over functions.

In this chapter we discuss using GPs when we can only observe integrals over some area of interest. In other words, instead of directly observing values of a function, the observations correspond to averages or sums over parts of it. For example, when modeling daily rates of incidences of a disease based on incidences per week or month, the observations are sums over time. Observations can also be sums or averages over spatial areas, such as census data over administrative districts on a map (Law et al., 2018) or signal attenuations along linear paths through objects. For example, in X-ray computed tomography (CT) imaging, where X-rays are used to map the internal structure of an object non-invasively, the observations correspond to values of the integrals over the paths the X-rays have traveled (Purisha et al., 2019). We present a novel application where ultrasound signals are used to non-invasively map fouling inside industrial pipelines.

One major limitation of GPs is that the convenient closed-form solutions exist only when the likelihood and the the noise are Gaussian. Several solutions exists for traditional GPs (see Section 3.2), but applying these with integral observations is not straight forward. In Section 5.4 we discuss possible solutions, including our contribution from `Publication` IV.

Finally, to make solving practical inverse problems easier, we discuss sensor placement for different physical sensing configurations in this context in Section 5.5. In `Publication` V we provide the first comprehensive overview on the topic, and give two practical ways to optimize sensor locations.

5.1 Related Work

Integral observations appear in many physical sensing applications, though only a few existing works have yet applied GPs in solving these. Integral observations with GPs have been shown to work in the context of laser scanners (O'Callaghan and Ramos, 2011) and tomographic reconstruction (Jidling et al., 2018; Purisha et al., 2019), while potential applications exist also for example in ultrasonic structural health monitoring (Lu and Michaels, 2009). In computed tomography (CT), the goal is to reconstruct a 3D object based on signal attenuation along linear paths through the object (Jidling et al., 2018; Purisha et al., 2019). In `Publications` III-V we also introduce and focus on a new application area based on ultrasonic signals: fouling detection in pipes, which we describe in Section 5.6.

In addition to different physical sensing problems, integral observations occur when the data of interest is collected in lower granularities than we want to model it. Such data is often referred to as *aggregated observations* (Purisha et al., 2019; Tanaka et al., 2019b; Yousefi et al., 2019) or *binned data* (Smith et al., 2018a,b). For aggregated data, GPs have been applied to model for example infectious diseases, air pollution and poverty rates on a finer scale based on data aggregated by larger areas, such as administrative districts, or by coarser time resolutions, such as hours or days (Law et al., 2018; Tanaka et al., 2019b; Yousefi et al., 2019). Integral observations of binned data can also be utilized to achieve differentially private regression (Smith et al., 2018b).

GP based methods for aggregated data have been studied especially in geostatistics, where GPs are known as *kriging* (Goovaerts, 2010; Goovaerts et al., 1997; Kyriakidis, 2004; Xavier et al., 2018). Learning more fine-scaled estimates from aggregated spatial data is there known as *downscaling*, *disaggregation* or *change of support* (Gotway and Young, 2002; Xavier et al., 2018; Zhang et al., 2014). Recent works in machine learning have focused on multi-task learning where they aim to combine data collected from multiple sources and in multiple granularities (Hamelijnck et al., 2019; Tanaka et al., 2019a; Yousefi et al., 2019)

5.2 Gaussian Processes for Linear Operators

Like Gaussian distributions are closed under linear transformations, GPs are closed under linear operators (Rasmussen and Williams, 2006; Särkkä, 2011). We define $\mathcal{L}_x$ to be a linear operator applied to a function $f(x)$, and consider cases where $f(x)$ is modeled as a Gaussian process $f(x) \sim GP(m(x), k(x, x'))$. Then $\mathcal{L}_x f(x)$ is GP with

$$\mathbb{E}[\mathcal{L}_x f(x)] = \mathcal{L}_x m(x),$$

$$\text{cov}(\mathcal{L}_x f(x), \mathcal{L}_{x'} f(x')) = \mathcal{L}_x \mathcal{L}_{x'}^T k(x, x').$$

In principle, $\mathcal{L}$ could be any linear operator, such as derivative (Solak et al., 2002; Wahlström, 2015) or finite summation (Smith et al., 2018a), but we focus on cases where the linear operator is an integral (Jidling et al., 2018; Law et al., 2018).

The observations are a collection of N region-response pairs $(v_i, y_i)_{i=1}^N$, where the region v_i is a subset of the input space, and y_i is the value of the response variable for that region, i.e the integral over it. All the regions we deal with are regular shapes, i.e intervals in 1D space or lines in 2D space, but in principle these regions could be anything, such as administrative districts on maps (Tanaka et al., 2019a).

More specifically, we consider setups where the observations y_i are noisy values for integrals evaluated along a region v_i computed using

$$y_i = f(v_i) + \epsilon_i = \frac{1}{|v_i|} \int_{\mathbf{z} \in v_i} u(\mathbf{z}) d\mathbf{z} + \epsilon_i.$$

Here $u(\mathbf{z}) \sim GP(0, k(z, z'))$. In `Publications` III and V we assume $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ so that we can solve the analytical posterior, while in `Publication` IV we allow also non-Gaussian additive errors. Figure 5.1 gives an example of GP regression when the observations v_i are over lines in a 2D space.

5.3 Efficient Computation of the Kernel

In addition to computing the covariance between two points like in traditional GP regression, with integral observations we need to also define the covariance between a region v_i and a point x_i^* and between two regions v_i and v_j. For kernel function k, the covariance between a region and a point is defined to be

$$k(v_i, \mathbf{x}_i^*) = \frac{1}{|v_i|} \int_{\mathbf{z} \in v_i} k(\mathbf{z}, \mathbf{x}_i^*) d\mathbf{z}, \tag{5.1}$$

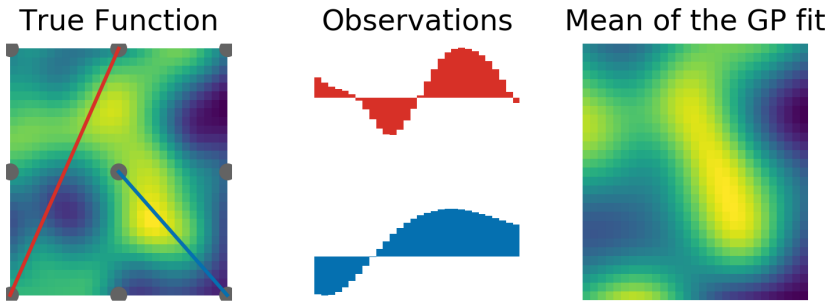


Figure 5.1: Mapping an unknown function $f(x)$ when the observations are integrals over lines. **Left:** The true function as well as nine sensors that form 36 line integral observations v_i between them. **Middle:** Two examples of what a slice of the function looks like along an observation line. **Right:** Mean of the GP fit after conditioned on the noisy observations of all 36 line integrals. Reproduced from `Publication` IV.

and between two regions

$$k(v_i, v_j) = \frac{1}{|v_i|\,|v_j|} \int_{\mathbf{z}\in v_i} \int_{\mathbf{s}\in v_j} k\left(\mathbf{z}, \mathbf{s}\right) d\mathbf{s} d\mathbf{z}. \tag{5.2}$$

The posterior GP can then be computed as shown in Equation 3.2, but that requires solving the integral in Equation 5.1 and the double integral in Equation 5.2. Computing these depends on the kernel function k, and in general these are not analytically tractable. This is perhaps one reason why integral observations have not been commonly used with GPs.

General methods for approximating the integrals

Numerical integration is the most flexible way to approximate the integrals as it can be used with any kernel function and can be extended also to non-linear cases as we show in `Publication` V. However, it is also computationally expensive. Still, it has been used by most of the earlier works (Kyriakidis, 2004; O'Callaghan and Ramos, 2011; Tanaka et al., 2019b; Yousefi et al., 2019).

In numerical integration the integral is approximated by discretizing the observation regions v_i into J points where the integrand is evaluated, and the approximation is given by a weighted sum of these. Any numerical integration method can be used to choose the discretization points and weights. The simplest way is to use Riemann sums with evenly spaced

discretization points $x_{ij} \in \mathbf{v}_i := [x_{i1}, \ldots, x_{iJ}]$, resulting in

$$\int_{v_i} f(x)dx \approx \sum_{j=1}^{J} \Delta_{v_i} f(x_{ij}). \tag{5.3}$$

Numerical integration leads to quadratic complexity for the region-vs-region case in Equation 5.2 in terms of the number of evaluation points J, and thus requires us to trade off between computation time and accuracy. The main challenge with this approach is that it becomes infeasible with large data sets, however, for many real world sensing problems the method is still practical.

Methods based on Hilbert space approximations (Solin and Särkkä, 2020) are applicable to all stationary kernels and allow avoiding numerical integration completely by approximating the kernel with a finite set of basis functions as described in Section 3.2.1. The method decouples both the model parameters from the integrals as well as the input arguments from each other. This makes it possible to analytically solve the integrals (Jidling et al., 2018; Purisha et al., 2019). While the method can not be used with all possible kernels, in practice most popular kernels in use are stationary, like all the kernels used in this thesis. Even though the number of required basis functions grows exponentially with the problem dimensionality, the method is an excellent practical solution for problems in 2D feature spaces.

Kernel specific methods

It is also possible to develop methods that are specialised for specific kernels by analysing their exact functional form. This way we can sometimes solve the integrals either fully analytically or partially analytically. The drawback here is that clear indicators on when this can be done do not exists but each kernel has to be separately analysed. Whether on analytical solution exists or not can also depend on the shape of the region v_i.

For the frequently used RBF kernel, Hendriks et al. (2018) showed that it is possible to solve the integral in Equation 5.1 analytically when the observed regions are lines. They also provided a semi-analytic solution for the double integral in Equation 5.2 using numerical integration only for the outer integral reducing the computational cost to be linear in J. In the case of one dimensional feature space, these integrals can be computed fully analytically, and with certain shapes of regions, specifically rectangles, cuboids or hyperrectangles, kernels in higher dimensions can be expressed as the product of one dimensional kernels and the analytical result can be applied to higher dimensional feature spaces also (Smith et al., 2018a).

One of the contributions of this thesis is to extend these results on the RBF kernel into the rational quadratic kernel (RQ). In Table 3.1 we gave the equation of the kernel and showed how the samples from the prior look like. In `Publication` III we show that a fully analytical solution for line integrals exists for the RQ kernel with $\alpha = 2$. The exact derivations can be found in the publication. The benefit of kernel specific analytical methods is that they are exact and fast. We show that even with small amounts of evaluation points in numerical integration the analytic approach is about 40 times faster than fully numerical integration.

However, the exact results presented are still somewhat tedious and only applicable for very specific cases. In practice, the methods based on Hilbert space approximations allow switching between different stationary kernels more conveniently, provide fast ways to optimize for the hyperparameters and still provide accurate results in most cases (Jidling et al., 2018; Purisha et al., 2019). For small data sets the numerical methods are efficient enough, and they are the easiest to implement, work with arbitrarily shaped regions and can be easily used with any kernel.

5.4 Non-Conjugate GPs

Like in traditional GP regression, also with integral observations the convenient analytical solutions hold only with Gaussian likelihood models. Sometimes, to get more flexible models, we need to resort to non-Gaussian likelihoods. This is often done by assuming that the observations are integrals of a GP that is mapped through a non-linear function $g(\cdot)$ with

$$y_i = \frac{1}{|v_i|} \int_{\mathbf{z} \in v_i} g(u(\mathbf{z}))d\mathbf{z} + \epsilon_i.$$

Here $g(\cdot)$ should be selected to match the needs of an application. Another reason for non-conjugacy is to assume a non-Gaussian additive error model.

First we start with some motivating examples. Many real world data sets consist of only non-negative values, such as amount of fouling on a surface or the height of a person. While we can use conjugate GPs to model such data sets, in many cases introducing a non-negativity constraint improves the results. In Figure 5.2 we show a motivating toy example of modeling a non-negative latent function based on binned data. Traditional GP regression places most of the posterior mass to be on negative values, while the constraint ($g(x) = \log(1 + e^{f(x)})$) helps to correctly place the mean at zero. Another example is modeling count data with a Poisson likelihood. Several potential aggregate data sets consist of counts: number

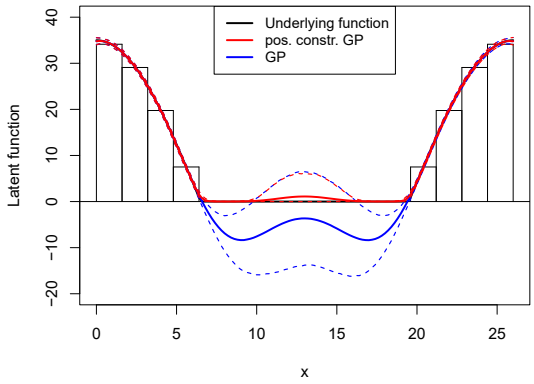


Figure 5.2: Illustration of the benefit of introducing a non-negativity constraint into Gaussian process modeling of aggregated data when the latent-function is known to be non-negative. The bars represent aggregated observations we have of the black underlying true function. The blue and red line represent the means of the GP fit without and with a non-negativity constraint introduced by modeling the latent function as $\log(1 + e^{f(x)})$ for $f(x)$. The dashed lines show the 5% and 95% quantiles. Picture originally published in `Publication` IV.

of new incidences of an infectious disease or number of votes in a voting district. In `Publication` IV we provide an example of daily Google search counts.

In this section we briefly discuss previously presented solutions for non-conjugate GPs with integral observations, most of which rely on extending traditional solutions based on variational inference. We then introduce a general sampling based approach from `Publication` IV that can be always applied regardless of the transformation or the likelihood used.

Related work

One approach to introducing constraints without having to deal with non-conjugacy, is to place some virtual observations that fulfill the desired condition in the input space. Riihimäki and Vehtari (2010) used this idea to achieve approximately monotonic GP posterior means. Smith et al. (2018a) build on this work by using virtual points to enforce non-negativity when modeling binned data. However, the method changes the posterior as well

as the estimates of the hyperparameters, and it might not be practical with all data sets (Smith et al., 2018a).

A more principled approach was proposed by Yousefi et al. (2019) and Law et al. (2018) based on variational approximations for exponential family likelihoods. However, both of these methods are limited to some specific cases. Yousefi et al. (2019) only allows non-linearities after the aggregation and the method by Law et al. (2018) works for specific likelihoods and only with some transformations that are computationally tractable.

General sampling-based solution

`Publication` IV presents a more general solution for the problem, allowing for any arbitrary transformations with any likelihoods. It is based on numerical integration and MCMC sampling. Integrals in Equations 5.1 and 5.2 are computed by evaluating the value of $g(f(\mathbf{x}))$ for a evenly spaced collection of J points $\mathbf{x}$ in v_i

$$\int_{\mathbf{x}\in v_i} g(f(\mathbf{x}))d\mathbf{x} \approx \sum_{j=1}^{J} \Delta_{v_i} g(f(x_{ij})).$$

In `Publication` IV we use simple Riemann sums, but more advanced quadratures could also be used.

For inference we use a Markov chain Monte Carlo based method (see Section 2.2.1) called the No-U-Turn Sampler (NUTS) (Hoffman and Gelman, 2014) implemented in the `Stan` probabilistic programming language (Carpenter et al., 2017). This gives the presented method also many advantages compared to the existing solutions. The method is easy to implement using probabilistic programming tools, making it easy to use for practitioners. The generality of the method is also beneficial in many signal processing tasks as it makes it possible to use any arbitrary functions. This allows us to plug in possible existing physical knowledge of the non-linearities.

The obvious limitation of this method is the lack of computational efficiency making it slower than the other approaches and thus not applicable for some large applications. Due to the discretization, the amount of observations is NM in practice. Despite of this, the method is still perfectly adequate for many physical sensing problems where data sets are small. In addition, the computations can be made more efficient by using inducing points as described in Section 3.2.1. `Publication` IV contains more details on how to specifically do that in this context.

5.5 Sensor Placement

In many physical sensing applications with integral observations the data is collected by transmitting and receiving sensors, and a critical question in modeling is how to place these sensors. For example, in our case of monitoring structural health and locating fouling inside of industrial pipelines we want to minimize the number of required sensors while being able to map the inside of the pipes as well as possible. Individual sensors are expensive and placing them requires manual labour and may for example require removing the coating of the pipe at the sensor locations.

More specifically, we consider cases where we are trying to map an unknown function inside of some area of interest I using transmitting and receiving sensors. The observations correspond to signals traveling through the area between two sensors. We assume that the underlying function has a point-wise effect on the signal that can be interpreted as an integral over the function. Examples of such effects are modulating the signal's amplitude, phase or velocity. In other words, we assume that from these signals we can extract information about the integral over the path it has travelled. The goal is to place the sensors in a way that we can get as much information about the function as possible with a limited amount of sensors.

To the best of our knowledge, no other works have previously studied this question in the context of integral observations and GPs, and in `Publication` III we provide two different approaches that extend previously presented solutions for point observations. We describe three types of sensors that are also pictured in Figure 5.3:

1. Paired Sensors: Each receiver listens to only one transmitter.

2. Separate Transmitters and Receivers: The sensors are split into two groups and each receiver listens to all transmitters.

3. Universal Sensors: We have a collection of N sensors, each acting in turn both as transmitter and receiver, listening to all other sensors.

The first case is fairly simple, as adding a pair of sensors only adds one more observation and the problem reduces to placing point-observation sensors. However, in the two other cases adding a new sensor introduces multiple new observations that the evaluation depends on. The optimization methods we present optimize the locations for the sensors via a set of measurements induced by them.

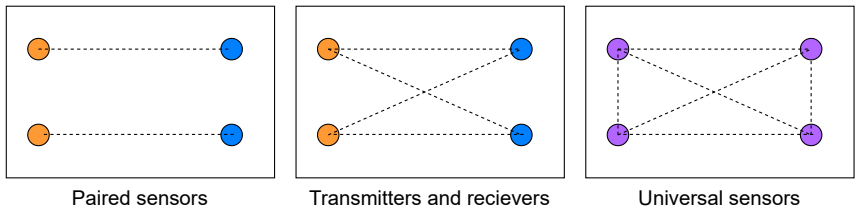


Figure 5.3: Illustration of different sensor types. Here transmitting sensors are orange, receiving sensors are blue and sensors capable of both transmitting and receiving are purple. The dashed lines represent the observations between the sensors.

Geometric approach

Though some geometric approaches have also been proposed for point observations (González-Banos, 2001; Kershner, 1939), line integral observations make the idea more interesting. We can take advantage of the concept of line arrangement (Agarwal and Sharir, 2010) and place the sensors so that the intersections (vertex) of the lines and the segments (edges) and polygons (faces) induced by the lines cover the area of interest as well as possible. This can be done using black box optimization to optimize some fitness function, such as minimizing the distance between vertices and points in the area.

Model-based approach

The state of the art methods for optimizing sensors for point observations are based on maximizing some measure of information content. The most common choices are using entropy to find sensors that are most uncertain of each other or mutual information to find sensors that are most informative about unsensed locations, as described in Section 3.2.2. In most cases solving this is NP-complete, which requires resorting to greedy algorithms that choose the next location from a finite set of possible locations (Krause et al., 2008).

In `Publication` III we directly build on these methods. However, with line observations there are some additional points to consider. First, we can now have three types of sensors: transmitters, receivers and universal sensors that are able to both send a signal to all other sensors as well as receive the signal form all other sensors. Second, one new sensor can induce multiple new line observations if every receiver records signals from all transmitters, or if the sensors are universal.

It is possible to compute the reduction in uncertainty for all the newly induced lines individually. When choosing where to place the new sensor, we need to decide how to summarize the information from individual lines. We denote with l_m^s the lines induced when adding a new sensor s, and $\mathcal{B}$ is the set of lines induced by the sensor locations that have already been chosen. There are the three approaches to choose the next sensor s when using the entropy criterion H:

$$s_a^* = \text{argmax}_s \, \text{argmax}_l \, H(l_m^s|\mathcal{B}),$$
$$s_b^* = \text{argmax}_s \, \text{argmin}_l \, H(l_m^s|\mathcal{B}),$$
$$s_c^* = \text{argmax}_s \, \frac{1}{M'} \sum_m^{M'} H(l_m^s|\mathcal{B}).$$

These favour slightly different configurations. For example, the first ones selects a set of line that contains at least one new observation that we are uncertain of. This can be similarly done using the mutual information. For details we refer the readers to the publication.

Comparison of the approaches

Both of the presented approaches have advantages and disadvantages, and they can work in different situations. One of the main differences is that the geometric approach places sensors on continuous areas whereas the model-based approach requires a discrete set of points. In addition, the model-based algorithm is greedy, and as a result might not always find the optimal solution. Figure 5.4 illustrates these differences. In this case the geometric approach is able to recover a regular polygon. This also means that for regularly shaped areas of interest we can use existing configurations from the literature, and in some cases perhaps only optimize the orientation of the configuration as this is much faster. In this case, due to the discretization as well as the greedines of the model-based algorithm, it is not able to recover the same shape.

The drawback of the geometric approach is that it can be slow and thus not necessarily applicable for large amounts of sensors, and defining the possible locations for transmitters and receivers can be tedious for unusual configurations. Model-based approach can easily be modified for any shapes or configurations. It can also easily be used to decide how many transmitters and receivers we should use, whereas for the geometric approach with current specifics these need to specified. In addition, the model-based approach can be extended to sequential selection in applications where this is appropriate. Such active learning scenarios will always lead to better or

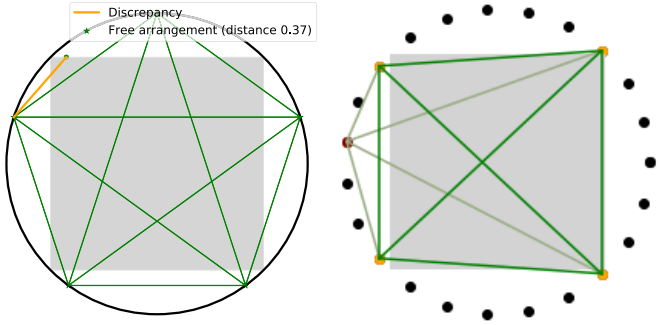


Figure 5.4: Comparing model based and geometric approaches for sensor placements for universal sensors. The gray square represents the area of interest and the sensors are placed on a circle around it. **Left: Geometric approach** Sensors are placed on a continuous area, which here is the circle around the area of interest. Here we minimize the distance from vertices to any point in the area. Picture from `Publication` III. **Right: Model-based approach** The discrete set of possible sensor locations are represented with black dots. The placement here is achieved using the mean summary, and the 5th sensor chosen is shown in maroon and the newly induced lines in lighter green.

equal results to pre-selecting the sensors when the hyperparameters are not known in advance (Krause and Guestrin, 2007).

5.6 Application: Fouling Detection in Pipes

We apply the methods presented in this chapter in a practical problem of localising fouling in pipes using ultrasound signals. The goal of being able to form a map of the fouling inside a pipe has been the motivation behind `Publications` III-V. We demonstrate that integral observations can be used to estimate unknown functions also in practice. We show that taking physical properties of the function into account, such as non-negativity of the function, improves the results. We also show some recommended placements for sensors for collecting the data.

Figure 5.5 illustrates the problem setting. In the picture there are two sensors, one transmitter and one receiver, that are marked with blue both on the surface of a pipe as well as in a setting where the pipe is pictured flattened on a surface. The goal is to form a map of the amount of fouling inside the pipe. In the simplified setting in the picture, we only have fouling on the area pictured by the orange ellipse.

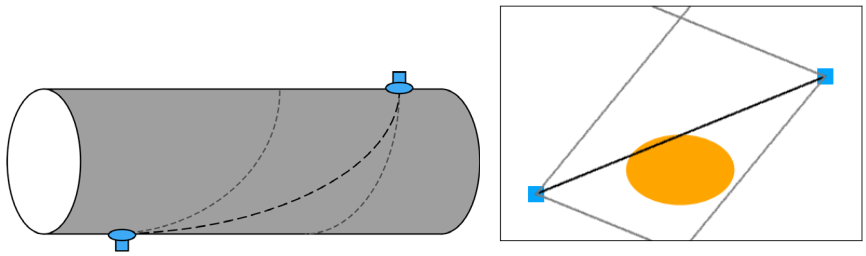


Figure 5.5: Figure illustrating the problem setting. On the left we have a picture of a pipe, and on the right we are showing the same setting flattened on a surface. Sensors, one transmitter and one receiver, are marked with blue. The first order helical path is marked with black and the second order paths in grey. The goal is to localize the orange fouling inside the pipe using information provided by signal propagating through the paths.

Because of the pipe geometry, one pair of a transmitter and receiver actually introduces multiple observations as we obtain an observation for each helical path along the surface. The transmitter sends ultrasonic land waves that propagate along the pipe, and the group velocity depends on properties of the pipe such as the thickness and the material. Thus, any possible fouling on the surface of the pipe on the path of the signal has an effect on it's arrival time at the receiver. The differences in these arrival times between a clean and a fouled pipe can be interpreted as integrals over the fouling function of the pipe. As this thesis concentrates on the machine learning methods rather than the physics behind the application, the readers are directed to `Publication` V for more details.

We demonstrate the methods on artificially simulated data. We have defined a simple fouling function that is pictured in Figure 5.6 (left), and the observations are noisy values of the integrals over the lines shown in the same picture. These are formed by one transmitter and 12 receivers, and we included first and second order helical paths.

The picture in the middle in Figure 5.6 demonstrates that by using integral observations combined with GPs, it is possible to approximately locate the fouling in the pipe. We also demonstrate similar results on real data in `Publication` V. The picture on the right shows that adding a simple non-negativity constraint improves the accuracy by more than 40%.

Figure 5.7 shows the optimal sensor placements achieved using different strategies. It turns out that having approximately balanced number of transmitters and receivers is optimal. Using these placements the geometric approach matches the accuracy of best model-based approach when

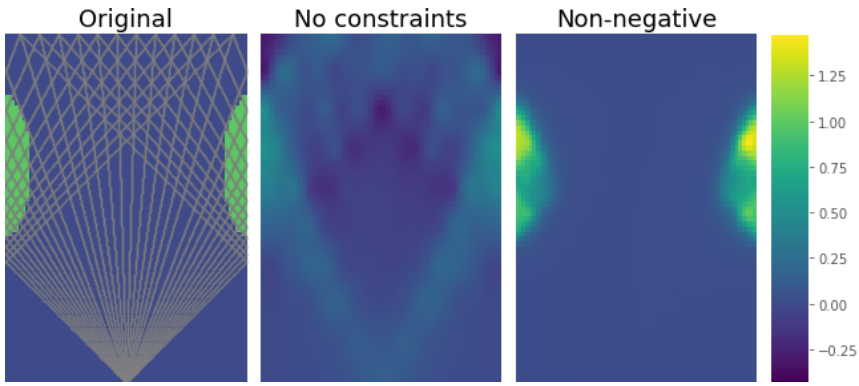


Figure 5.6: Using ultrasound to locate fouling inside a pipe. **Left:** Fouling function pictured on a surface of a flattened pipe. The gray lines represent the paths that the signals corresponding to our observations travel. The yellow ellipse represents the fouling in otherwise clean (blue) pipe. **Middle:** Prediction of the location of the fouling when no constraints are used. **Right:** Prediction of the location of the fouling when a non-negativity constraint is used. Picture from `Publication` IV.

predicting the mean function using six sensors. All strategies perform better than our original setup consisting of only one transmitter and several receivers.

The main challenges in solving the problem now lie in prepossessing the signal to extract the integral information for real data. In `Publication` V we use simple peak detection methods to find the arrival times of different helical paths in a clean and a fouled pipe, and used the difference between these as the value of the integral. However, even though this can approximately work in some cases, in reality the problem is much more complex. The effect of the fouling on the signal depends on the type of signal used, and thus it does not necessarily even always slow it down. In addition, the differences between a clean and a fouled pipe can be so small that the noise from the peak detection hides all differences. Thus, even though we have demonstrate the effectiveness of the method on simulated data, and performed some initial experiments on real data, most questions concerning real data are still currently unsolved, and the work on it continues. An interesting approach is to focus on the amplitudes of the peaks instead of the times of arrival.

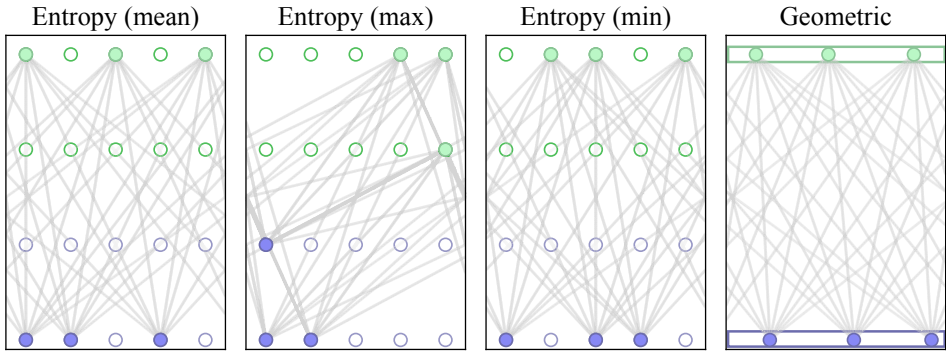


Figure 5.7: Optimal placements for ultrasonic fouling detection in pipes for six sensors using model-based and geometric approaches. The purple circles at the bottom of the pipe represent the locations where we are allowed to place transmitters, whereas the green circles on top are the locations where we can place the receivers. We assume that each transmitter signal can be recorded by all receivers, and we have included first and second order helical paths that are shown in gray. For the geometric approach, the rectangles represent the continuous areas where transmitters and receivers can be placed. Picture from `Publication` III.

Chapter 6

Multi-Resolution Gaussian Process State Space Models

Learning GPSSMs is a difficult task, where one of the main challenges is scaling up to long time series. Even the efficient solutions using sparse GPs scale linearly with the length T of the time sequence as the gradients need to be propagated back through the whole trajectory. In addition to the computational requirements, using long sequences in training can lead to vanishing and exploding gradients, which effectively prevents updating the parameters and leads to poor convergence (Pascanu et al., 2013).

Dividing the data into smaller sequences of size $B < T$, called mini-batches, that are used in training solves both of the described issues, and has led to good results in applying GPSSMs to large data (Doerr et al., 2018). However, a big issue with training the model with mini-batches is that long-term phenomena that evolve slower than the size of one mini-batch can no longer be learned (Williams and Zipser, 1995). Another approach would be to downsample the data, i.e include only every R:th observation. Downsampling the original data allows us to see longer trajectories in a shorter sequence, but sparse observations loose information about the short-term effects.

In `Publication` VI we present a new multi-resolution GPSSM (MR-GPSSM) that combines both of these approaches. It consists of L components that are trained on different resolutions of the data to capture effects on different time scales. Figure 6.1 illustrates the issue of learning long- and short-term effects simultaneously, while Figure 6.2 illustrates how downsampling the data before splitting it to mini-batches of fixed size increases the amount of the history we are able to learn.

Next, we will describe the building blocks needed for this new architecture. For efficiency, we build on a sparse GPSSM based on the FITC

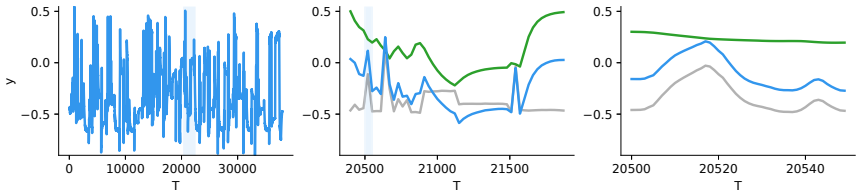


Figure 6.1: An illustration of the problem with using mini-batches when the data contains long- and sort-term effects. The dataset with fast and slowly varying dynamics (blue) was created as a sum of two functions: gray represents a function with fast varying dynamics and green a function with slowly varying dynamics. The picture is from `Publication` VI. **Left:** The full dataset. **Middle:** A mini-batch of size $B = 50$ where the data has been downsampled to include every 30th observation. The slowly varying dynamics (green) can now be learned, but for the fast dynamics (gray) the observations are too sparse. **Right:** A mini-batch of size $B = 50$ on the original data. The function with fast varying dynamics can be learned, but there is not enough information about the slow dynamics (green) in this short mini-batch.

approximation given in Equation 3.4 that allows using mini-batches during training as well as building a connection to stochastic differential equations. We re-interpret the GP transition function f as a discretized SDE, which allows using different resolutions in training. Finally, by combining results from multiple GPSSM components trained with different resolutions, we get the new MR-GPSSM architecture.

6.1 Training a GPSSM with Different Resolutions

Before introducing our multi-resolution GPSSM in Section 6.2, we discus training a standard GPSSM with different resolutions. The multi-resolution architecture is based on combining multiple such GPSSM components.

GPSSMs as discretized SDEs

The motivation behind the new architecture is based on the observation that the GP transition function f of a GPSSM model can be interpreted as discretized stochastic differential equation (SDE). SDEs are differential equations with a stochastic component that are used to model systems with dynamical behaviour. Typically, randomness is entered via Brownian motion that is here marked with $W_t \in \mathbb{R}^{D_x}$. We consider systems that are

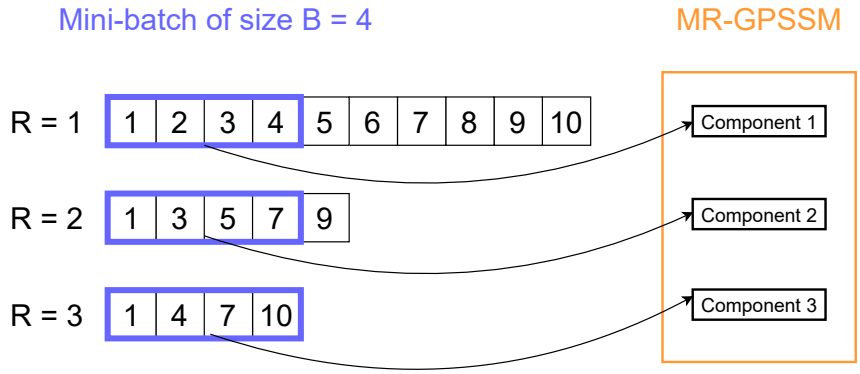


Figure 6.2: An illustration of the data processing for different resolutions for different components of MR-GPSSM. In the original data ($R = 1$), a mini-batch of size $B = 4$ covers only a small portion of the whole sequence. Using higher resolutions allows us to see a longer time span in one mini-batch. In fact, using $R = 3$ allows fitting the whole time span into one mini-batch. However, this also makes the data so sparse that we might not able to model all short-term effects anymore. Being able to use a combination of different resolutions allows learning varying dynamics.

governed by a Markov process, and a corresponding SDE is given by

$$d\mathbf{x}_t = f(\mathbf{x}_t)dt + \sqrt{\mathrm{Q}^{\Delta}}dW_t,$$

where $\mathbf{x}_t \in \mathbb{R}^{D_x}$ is the state vector and $f(\mathbf{x}_t)$ the drift component, which is the state evaluation given by the GP predictions $f(\mathbf{x}_t) \sim \mathcal{N}(\mu^{\Delta}(\mathbf{x}_t), \Sigma^{\Delta}(\mathbf{x}_t))$. The diffusion term $\sqrt{\mathrm{Q}^{\Delta}}$ gives the magnitude of the Brownian motion. SDE is continuous in time, and the drift component f transforms the states $\mathbf{x}_t$ forward while the Brownian motion adds random fluctuations in them. We use Δ to differentiate the SDE components from the corresponding components of the GP transition function.

However, SDEs do not in general have analytical solutions, and we use the Euler-Maruyama method (see for example (Särkkä and Solin, 2019)) to get a numerical approximation

$$\mathbf{f}_j|\mathbf{f}_M \sim \mathcal{N}(\mathbf{f}_j|\mu^{\Delta}(\mathbf{x}_j), \Sigma^{\Delta}(\mathbf{x}_j)), \tag{6.1}$$

$$\mathbf{x}_{j+1}|\mathbf{x}_j, \mathbf{f}_j \sim \mathcal{N}(\mathbf{x}_{j+1}|\mathbf{x}_j + R\Delta_t\mathbf{f}_j, R\Delta_t\mathrm{Q}^{\Delta}). \tag{6.2}$$

Here $\mathbf{f}_M$ are a set of inducing points, Δ_t is the time interval between two adjacent observations in the original data, and R is the resolution. We

index the GP outputs $\mathbf{f}_j$ with j instead of t, so that the time index t equals a time $t\Delta_t$ after the starting time, whereas the index j is a time $jR\Delta_t$ after the starting time. The method converges to the true solution when the step size $R\Delta_t$ gets smaller.

Learning a GPSSM with a different resolution

The Euler-Maruyama approximation of SDEs leads to a convenient formulation that resembles the GPSSM model. In the extended version of `Publication` VI, we show that when $R = 1$ we can find such setting of variational and model parameters that the variational objectives of the discretized SDE and the GPSSM are equal. By interpreting the transition function f of a GPSSM as a discretized SDE as described here, we can use Equation 6.2 with $R \geq 1$ to learn the dynamics. By marginalising out the local latent variables f_j we get the transition function

$$p^{\Delta}(\mathbf{x}_{j+1} \mid \mathbf{x}_j, \mathbf{f}_M) = \mathcal{N}(\mathbf{x}_{j+1} \mid \mathbf{x}_j + R\Delta_t \mu^{\Delta}(\mathbf{x}_j), \\ (R\Delta_t)^2 \Sigma^{\Delta}(\mathbf{x}_j) + R\Delta_t \mathrm{Q}^{\Delta}).$$

This formulation allows us to use different values for R during training and during prediction. When choosing $R > 1$ during training, we can fit a longer history in one mini-batch by only including every Rth observation while still being able to get prediction at the same resolution as the original data by choosing $R = 1$ during prediction.

As described in Section 3.3, inference in GPSSMs cannot be performed analytically, and different approximations have been suggested. We follow the existing work and use variational inference to optimize a Monte Carlo estimate of the lower bound to the log marginal likelihood. We use the same variational family as Doerr et al. (2018), which means we build on the sparse GPSSM based on the FITC approximation given in Equation 3.4. There are multiple Monte Carlo schemes for obtaining samples from the variational posterior, and we employ the one presented by Ialongo et al. (2019). However, a GPSSM with $R \neq 1$ and the multi-resolution model architecture can be learned using any these sampling schemes, and this is not the main focus of our work. We also expect that future improvements in inference schemes for GPSSMs can be used with our MR-GPSSM. Thus, we will not discuss the inference and the sampling schemes here more closely, but the details on how we perform inference as well as a discussion of different possible Monte Carlo schemes can be found in `Publication` VI and its extended version in Longi et al. (2021).

6.2 Multiple Components with Different Resolutions

We introduce a latent state that consists of L independent multi-dimensional latent states, that we call *components*: $\mathbf{x}_t = \{\mathbf{x}_t^{(l)}\}_{l=1}^{L}$, where $\mathbf{x}_t^{(l)} \in \mathbb{R}^{D_x}$. The transition function of each component is given a GP prior, and a linear combination of all latent states $\mathbf{x}_t$ defines the mean of the Gaussian emission probability. We build on the sparse GPSSM model in Equation 3.4 and extending that to multiple components:

$$\begin{aligned}
\mathbf{x}_0^{(l)} &\sim \mathcal{N}(\mathbf{x}_0^{(l)} \mid \mathbf{m}_0^{(l)}, Q_0^{(l)}),\\
\mathbf{f}_M^{(l)} &\sim \mathcal{N}(\mathbf{f}_M^{(l)} | 0, K_{MM}^{(l)}),\\
\mathbf{f}_t^{(l)} | \mathbf{f}_M^{(l)} &\sim \mathcal{N}(\mathbf{f}_t^{(l)} | \mu^{(l)}(\mathbf{x}_t^{(l)}), \Sigma^{(l)}(\mathbf{x}_t^{(l)})),\\
\mathbf{x}_{t+1}^{(l)} | \mathbf{x}_t^{(l)}, \mathbf{f}_t^{(l)} &\sim \mathcal{N}(\mathbf{x}_{t+1}^{(l)} | \mathbf{x}_t^{(l)} + \mathbf{f}_t^{(l)}, \mathrm{Q}^{(l)}),\\
\mathbf{y}_t | \mathbf{x}_t &\sim \mathcal{N}(\mathbf{y}_t | \sum_{l=1}^{L} g^{(l)}(\mathbf{x}_t^{(l)}), \Omega).
\end{aligned}$$

The different components are trained iteratively with a backfitting algorithm (Breiman and Friedman, 1985). In each step of the algorithm, the parameters $\theta^{(l)}$ of one component l are updated while the parameters of all other components are kept fixed. This sequential learning scheme makes it possible to learn the parameters $\theta^{(l)}$ of each component with a different resolution $R^{(l)}$. The details and the pseudocode can be found in `Publication` VI and its extended version in Longi et al. (2021).

6.3 Application: Engine Modeling

The motivation behind the new architecture came from modeling emissions of a car engine. With `Publication` VI, we introduce a new dataset consisting of over 500,000 measurements of raw emissions of a gasoline car engine recorded with 10Hz. As the dataset consists of long time sequences, using full trajectories to train a GPSSM is computationally infeasible, and using mini-batches is necessary. However, the dataset contains varying dynamics. For example, as our experiments also show, slow dynamics are in particular predominant for the output temperature and the dynamics can only be learned with a large enough resolution. On the other hand, nitrogen oxide concentration can be modeled only with small resolutions, including the standard resolution $R = 1$. This is why we need to be able to train the model using multiple resolutions.

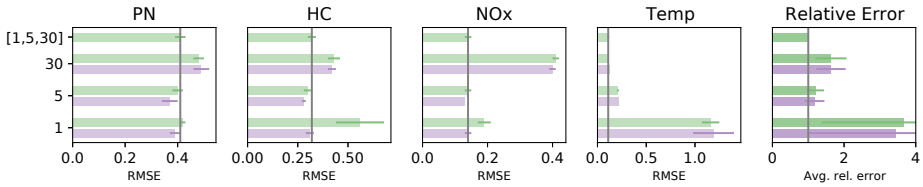


Figure 6.3: RMSE on the four outputs of the engine modeling task: particle numbers (PN), hydrocarbon concentration (HC), nitrogen oxide concentration (NOx) and engine temperature (Temp). We also show the relative error with respect to our method, averaged over all outputs. The green bars represent the multi-component model and purple lines the regular GPSSM. We report the mean value and its standard error over 5 repetitions. The multi-resolution model was trained using three different resolutions: 1, 5 and 30. Other green lines represent a multi-component architecture, where the latent states consists of three components that are each trained with the same resolution to differentiate between using different resolutions and the multi-component architecture. Picture form `Publication` VI.

We show that MR-GPSSM is able to achieve competitive results on all outputs. A regular GPSSM is able to learn the dynamics for each output only if appropriate resolution is used. Using the standard resolution $R = 1$ is completely unable to learn the dynamics for temperature. MR-GPSSM is the only one that works if the dataset contains varying dynamics. Otherwise, it performs equally to a regular GPSSM with an appropriately chosen resolution. Figure 6.3 illustrates these results.

Chapter 7

Conclusions

We have presented practical solutions for three real-world applications that all have potential to have societal impact. For example, non-invasive localisation of fouling can have significant positive environmental impact, while being able to recognise devices causing Wi-Fi interference is important in improving Wi-Fi networks. At the same time, we have made contributions to machine learning literature by improving the used models and tailoring them to our specific needs.

First, we presented a solution based on convolutional neural networks to analyse spectral data in order to identify radio frequency devices that can cause Wi-Fi interference. As these devices can have an effect on the performance of Wi-Fi networks (Chan et al., 2010), being able to identify them is important both in planning the initial network configuration as well as in troubleshooting when the performance of the network is poor. Collecting accurately labeled data is challenging, and we used a semi-supervised training method, where unlabeled data were assigned pseudo-labels by taking advantage of the continuity in the data. We were able to identify the recorded devices with near perfect accuracy. One of the main limitations of the method is that collecting labeled data from hundreds of possible devices is impossible. While we showed that the model can be easily adapted to new devices without having to train it from scratch, an unsupervised approach would be an interesting task for future work.

Next, we used ultrasound to localise targets inside closed containers in two different settings. In the first we had placed only one sensor outside a container, and used a chaotic cavity to increase the acoustic aperture of the signal. A chaotic cavity also makes it impossible to use traditional methods to analyse the signals, and we built a convolutional neural network based solution that was able to localise a small pipe inside the container. The main motivation behind the work was to show that even when using a

chaotic cavity, it is still possible to extract meaningful information from the signals to localise objects. The goal was achieved, but the accuracy of the method could most likely be further improved by collecting more data from different settings and by more carefully designing the model. For example, a recurrent neural network based solution could be an interesting question for the future.

In the second setting we used ultrasound to localise fouling inside pipes. This method has potential to have great societal impact if it allows us to recognise the structural health or fouled areas in industrial pipelines in a non-invasive way. It could have a significant environmental effect by reducing the global industrial energy consumption and global CO_2 emissions (U.S. Energy Information Administration), and by reducing the amount of hazardous chemicals currently used in industrial cleaning.

Fouling inside a pipe causes variations in wave propagation properties, and having multiple propagation paths through a pipe allows us to locate the fouling. We interpreted the observations to be integrals over the propagation paths, and showed how Gaussian processes can be used with such integral observations. Our contributions included being the first ones to present strategies for designing optimal placement for sensors in such cases, discussing how we can efficiently compute the integrals over the kernel function and giving a general, easy-to-use method for dealing with non-linearities in such cases.

While we have demonstrated that ultrasound can be used to localise fouling inside a pipe in laboratory conditions, there is still a lot to be done before the method can be used in the real world. We interpreted the delays in the arrival times of signals as the values for the integrals over the paths the signals had traveled. While this worked in laboratory conditions, the effect of the fouling on the arrival times is small and can be difficult to observe. In addition, the effect on arrival times varies depending on many things, suchs as the type of the signal and the type of the fouling, and more research is needed. How to accurately transform the measured signals into information about the value of the integral is perhaps the most important question for the future. However, we have demonstrated on simulated data that with accurate information about the integrals, GPs are well suited for localisation based on integral observations, and adding a non-negativity constraint helps significantly.

In addition, the data for `Publication` V was collected before the work considering optimal locations for sensors. A better measuring configuration would have likely led to better results on real data, as we have showed that an optimal configuration leads to better results on simulated data.

We also discussed efficient ways to compute the needed integrals over the kernel function, and showed how this can be done analytically in the case of rational quadratic (RQ) kernel. While the method is efficient, the benefit is likely to be small. Using Hilbert space approximations (Solin and Särkkä, 2020) has turned out to be a very useful approximation in context of integral observations (Jidling et al., 2018; Purisha et al., 2019), and it can be seen as a better solution than exact analytical derivations as it is also efficient, usually accurate enough and more general as it works with any stationary function. The analytical derivation can still be useful when the evaluations need to be performed multiple times, such as in planning a good sensor configuration for data collection.

Gaussian process state space model (GPSSM) is a very useful tool for modeling dynamical systems with potential use cases especially in reinforcement learning (Curi et al., 2020; Dörr, 2021). We showed how to efficiently learn a GPSSM model when there are both short- and long-term effects present. Existing solutions were not able to learn effects that are not visible in one minibatch (Doerr et al., 2018), and our method demonstrated clear improvements on data sets that contained varying dynamics. This is an important improvement that can potentially have a big effect on the applicability of GPSSMs. Moreover, it can likely be combined with future improvements in learning GPSSM models. We also introduced a new dataset for modeling car engine emissions where different outputs contain either short- or long-term effects or both, and showed that our method performs consistently well on predicting all outputs.

References

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, and G. Penn. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4277–4280. IEEE, 2012.

O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu. Convolutional neural networks for speech recognition. In *IEEE/ACM Transactions on audio, speech, and language processing*, volume 22, pages 1533–1545. IEEE, 2014.

P. K. Agarwal and M. Sharir. Arrangements and their applications. In *Handbook of Computational Geometry*. Elsevier, 2010.

R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv e-prints arXiv: 1605.02688*, 2016.

A. M. Andrew. Multiple view geometry in computer vision. *Kybernetes*, 30 (9):1333–1341, 2001.

E. Angelino, M. J. Johnson, R. P. Adams, et al. Patterns of scalable Bayesian inference. *Foundations and Trends in Machine Learning*, 9(2-3):119–247, 2016.

P. Bachman, O. Alsharif, and D. Precup. Learning with pseudo-ensembles. In *Advances in neural information processing systems*, pages 3365–3373, 2014.

S. A. Billings. *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons, 2013.

C. M. Bishop. *Pattern recognition and Machine Learning*. Springer, 2006.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Conference on computational learning theory*, pages 92–100. ACM, 1998.

L. Bottou et al. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142, 1998.

G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

L. Breiman and J. H. Friedman. Estimating optimal transformations for multiple regression and correlation. *Journal of the American statistical Association*, 80(391):580–598, 1985.

S. Brooks, A. Gelman, G. Jones, and X.-L. Meng. *Handbook of Markov Chain Monte Carlo*. CRC press, 2011.

T. D. Bui, J. Yan, and R. E. Turner. A unifying framework for Gaussian process pseudo-point approximations using power expectation propagation. *The Journal of Machine Learning Research*, 18(1):3649–3720, 2017.

B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software*, 76(1):1–32, 2017.

W. F. Caselton and J. V. Zidek. Optimal monitoring network designs. *Statistics & Probability Letters*, 2(4):223–227, 1984.

H. Cecotti and A. Graeser. Convolutional neural network with embedded fourier transform for eeg classification. In *International Conference on Pattern Recognition*, pages 1–4. IEEE, 2008.

E. C. Chan, G. Baciu, and S. Mak. Effect of channel interference on indoor wireless local area network positioning. In *International Conference on Wireless and Mobile Computing, Networking and Communications*, pages 239–245. IEEE, 2010.

O. Chapelle, B. Scholkopf, and A. Zien. Semi-supervised learning. 2006. *Cambridge, Massachusettes: The MIT Press View Article*, 2, 2006.

N. Cressie. *Statistics for spatial data*. John Wiley & Sons, 2015.

I. Csiszár. Information-type measures of difference of probability distributions and indirect observation. *Studia scientiarum Mathematicarum Hungarica*, 2:229–318, 1967.

S. Curi, S. Melchior, F. Berkenkamp, and A. Krause. Structured variational inference in partially observable unstable Gaussian process state space models. In *Learning for Dynamics and Control*, pages 147–157, 2020.

M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*, pages 465–472, 2011.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 4171–4186, 2019.

A. Doerr, C. Daniel, M. Schiegg, D. Nguyen-Tuong, S. Schaal, M. Toussaint, and S. Trimpe. Probabilistic recurrent state-space models. *International Conference on Machine Learning*, pages 1280–1289, 2018.

A. Dörr. *Models for Data-Efficient Reinforcement Learning on Real-World Applications*. PhD thesis, Universität Stuttgart, 2021.

D. Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge, 2014.

S. Eleftheriadis, T. Nicholson, M. Deisenroth, and J. Hensman. Identification of Gaussian process state space models. In *Advances in neural information processing systems*, pages 5309–5319, 2017.

D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

B. Ferris, D. Fox, and N. D. Lawrence. Wifi-slam using Gaussian process latent variable models. In *International Joint Conference on Artificial Intelligence*, volume 7, pages 2480–2485, 2007.

R. Frigola. *Bayesian time series learning with Gaussian processes*. PhD thesis, University of Cambridge, 2015.

R. Frigola, F. Lindsten, T. B. Schön, and C. E. Rasmussen. Bayesian inference and learning in Gaussian process state-space models with particle mcmc. In *Advances in neural information processing systems*, pages 3156–3164, 2013.

R. Frigola, Y. Chen, and C. E. Rasmussen. Variational Gaussian process state-space models. In *Advances in neural information processing systems*, pages 3680–3688, 2014.

Y. Gal. *Uncertainty in deep learning*. PhD thesis, University of Cambridge, 2016.

J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson. Gpytorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Advances in neural information processing systems*, pages 7576–7586, 2018.

R. Garnett, M. A. Osborne, and S. J. Roberts. Bayesian optimization for sensor set selection. In *IEEE international conference on information processing in sensor networks*, pages 209–219, 2010.

A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2013.

S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

Z. Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.

Z. Ghahramani and M. I. Jordan. Supervised learning from incomplete data via an EM approach. In *Advances in neural information processing systems*, pages 120–127, 1994.

L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal. Explaining explanations: An overview of interpretability of machine learning. In *International Conference on data science and advanced analytics*, pages 80–89. IEEE, 2018.

X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *International conference on artificial intelligence and statistics*, pages 315–323, 2011.

H. González-Banos. A randomized art-gallery algorithm for sensor placement. In *Symposium on Computational geometry*, pages 232–240, 2001.

I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

P. Goovaerts. Combining areal and point data in geostatistical interpolation: Applications to soil science and medical geography. *Mathematical geosciences*, 42(5):535–554, 2010.

P. Goovaerts et al. *Geostatistics for natural resources evaluation*. Oxford University Press on Demand, 1997.

C. A. Gotway and L. J. Young. Combining incompatible spatial data. *Journal of the American Statistical Association*, 97(458):632–648, 2002.

A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *International conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

O. Hamelijnck, T. Damoulas, K. Wang, and M. Girolami. Multi-resolution multi-task Gaussian processes. In *Advances in neural information processing systems*, volume 32, pages 14025–14035, 2019.

J. N. Hendriks, C. Jidling, A. Wills, and T. B. Schön. Evaluating the squared-exponential covariance function in Gaussian processes with integral observations. *arXiv 1812.07319*, 2018.

J. Hensman, N. Fusi, and N. D. Lawrence. Gaussian processes for big data. In *Uncertainty in Artificial Intelligence*, pages 282–290, 2013.

J. Hensman, A. Matthews, and Z. Ghahramani. Scalable variational Gaussian process classification. In *Artificial Intelligence and Statistics*, pages 351–360, 2015.

J. Hensman, N. Durrande, and A. Solin. Variational Fourier features for Gaussian processes. *The Journal of Machine Learning Research*, 18(1): 5537–5588, 2017.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

M. D. Hoffman and A. Gelman. The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

A. D. Ialongo, M. Van Der Wilk, J. Hensman, and C. E. Rasmussen. Overcoming mean-field approximations in recurrent Gaussian process models. In *International Conference on Machine Learning*, pages 2931–2940, 2019.

M. Jankowiak, G. Pleiss, and J. Gardner. Parametric Gaussian process regressors. In *International Conference on Machine Learning*, pages 4702–4712, 2020.

K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *International conference on computer vision*, pages 2146–2153. IEEE, 2009.

C. Jidling, J. Hendriks, N. Wahlström, A. Gregg, T. B. Schön, C. Wensrich, and A. Wills. Probabilistic modelling and reconstruction of strain. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, 436:141–155, 2018.

M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

R. Kershner. The number of circles covering a set. *American Journal of mathematics*, 61(3):665–671, 1939.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2014.

C.-W. Ko, J. Lee, and M. Queyranne. An exact algorithm for maximum entropy sampling. *Operations Research*, 43(4):684–691, 1995.

J. Ko and D. Fox. GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. *Autonomous Robots*, 27(1): 75–90, 2009.

M. Kolář, M. Hradiš, and P. Zemčík. Deep learning on small datasets using online image search. In *Spring Conference on Computer Graphics*, pages 87–93. ACM, 2016.

A. Krause and C. Guestrin. Nonmyopic active learning of Gaussian processes: an exploration-exploitation approach. In *International conference on Machine learning*, pages 449–456, 2007.

A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9:235–284, 2008.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, volume 25, pages 1097–1105, 2012.

P. C. Kyriakidis. A geostatistical framework for area-to-point spatial interpolation. *Geographical Analysis*, 36(3):259–289, 2004.

H. C. Law, D. Sejdinovic, E. Cameron, T. Lucas, S. Flaxman, K. Battle, and K. Fukumizu. Variational learning on aggregate outputs with Gaussian processes. In *Advances in neural information processing systems*, volume 31, pages 6081–6091, 2018.

Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, volume 2, pages 396–404, 1989.

Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361 (10):1995, 1995.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

D.-H. Lee et al. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in*

representation learning, International Conference on Machine Learning, volume 3, page 896, 2013.

H. Lee, P. Pham, Y. Largman, and A. Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096–1104, 2009.

S. Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis, University of Helsinki*, 1970.

K. Longi, J. Lindinger, O. Duennbier, M. Kandemir, A. Klami, and B. Rakitsch. Traversing time with multi-resolution Gaussian process state-space models. *arXiv preprint arXiv:2112.03230*, 2021.

Y. Lu and J. E. Michaels. Feature extraction and sensor fusion for ultrasonic structural health monitoring under changing environmental conditions. *IEEE Sensors Journal*, 9(11):1462–1471, 2009.

D. J. MacKay. Introduction to Gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166, 1998.

A. G. G. Matthews, J. Hensman, R. Turner, and Z. Ghahramani. On sparse variational methods and the Kullback-Leibler divergence between stochastic processes. In *International Conference on Artificial Intelligence and Statistics*, pages 231–239, 2016.

C. L. C. Mattos, A. Damianou, G. A. Barreto, and N. D. Lawrence. Latent autoregressive Gaussian processes models for robust system identification. *IFAC-PapersOnLine*, 49(7):1121–1126, 2016.

M. D. McKay, R. J. Beckman, and W. J. Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Conference of the International Speech Communication Association*, volume 2, pages 1045–1048, 2010.

G. Montaldo, D. Palacio, M. Tanter, and M. Fink. Building three-dimensional images using a time-reversal chaotic cavity. *IEEE transactions on ultrasonics, ferroelectrics, and frequency control*, 52(9):1489–1497, 2005.

K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning*, pages 807–814, 2010.

R. M. Neal. *Probabilistic inference using Markov chain Monte Carlo methods*. Department of Computer Science, University of Toronto, ON, Canada, 1993.

S. T. O'Callaghan and F. T. Ramos. Continuous occupancy mapping with integral kernels. In *AAAI conference on artificial intelligence*, pages 1494–1500, 2011.

A. O'Hagan. Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society: Series B (Methodological)*, 40(1):1–24, 1978.

Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, and J. Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *Advances in neural information processing systems*, volume 32, pages 13991–14002, 2019.

J. Paisley, D. M. Blei, and M. I. Jordan. Variational Bayesian inference with stochastic search. In *International conference on machine learning*, pages 1363 – 1370, 2012.

D. Palaz, M. M. Doss, and R. Collobert. Convolutional neural networks-based continuous speech recognition using raw speech signal. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4295–4299. IEEE, 2015.

R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

L. Pöhler, J. Umlauft, and S. Hirche. Uncertainty-based human motion tracking with stable Gaussian process state space models. *IFAC-PapersOnLine*, 51(34):8–14, 2019.

P. Prandoni and M. Vetterli. *Signal processing for communications*. EPFL press, 2008.

Z. Purisha, C. Jidling, N. Wahlström, T. B. Schön, and S. Särkkä. Probabilistic approach to limited-data computed tomography reconstruction. *Inverse Problems*, 35(10):105004, 2019.

Y. Qi, A. H. Abdel-Gawad, and T. P. Minka. Sparse-posterior Gaussian processes for general likelihoods. In *Uncertainty in artificial intelligence*, pages 450–457. Citeseer, 2010.

J. Quiñonero-Candela and C. E. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *International conference on machine learning*, pages 873–880, 2009.

N. Ramakrishnan, C. Bailey-Kellogg, S. Tadepalli, and V. N. Pandey. Gaussian processes for active data mining of spatial aggregates. In *SIAM International Conference on Data Mining*, pages 427–438, 2005.

R. Ranganath, S. Gerrish, and D. Blei. Black box variational inference. In *Artificial intelligence and statistics*, pages 814–822, 2014.

A. Rasmus, M. Berglund, M. Honkala, H. Valpola, and T. Raiko. Semi-supervised learning with ladder networks. In *Advances in neural information processing systems*, pages 3546–3554, 2015.

C. E. Rasmussen and C. K. Williams. *Gaussian processes for machine learning*. MIT press Cambridge, 2006.

S. Rayanchu, A. Patro, and S. Banerjee. Airshark: detecting non-WiFi RF devices using commodity WiFi hardware. In *ACM SIGCOMM conference on Internet measurement*, pages 137–154. ACM, 2011.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286, 2014.

J. Riihimäki and A. Vehtari. Gaussian processes with monotonicity information. In *International conference on artificial intelligence and statistics*, pages 645–652, 2010.

H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

S. Rossi, M. Heinonen, E. Bonilla, Z. Shen, and M. Filippone. Sparse Gaussian processes revisited: Bayesian approaches to inducing-variable approximations. In *International Conference on Artificial Intelligence and Statistics*, pages 1837–1845, 2021.

H. Salimbeni and M. Deisenroth. Doubly stochastic variational inference for deep Gaussian processes. In *Advances in neural information processing systems*, pages 4588–4599, 2017.

S. Särkkä. Linear operators and stochastic partial differential equations in Gaussian process regression. In *International Conference on Artificial Neural Networks*, pages 151–158. Springer, 2011.

S. Särkkä. *Bayesian filtering and smoothing*. Cambridge University Press, 2013.

S. Särkkä and A. Solin. *Applied stochastic differential equations*. Cambridge University Press, 2019.

L. K. Saul and M. I. Jordan. Exploiting tractable substructures in intractable networks. In *Advances in neural information processing systems*, pages 486–492, 1996.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

M. W. Seeger, C. K. Williams, and N. D. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *International Workshop on Artificial Intelligence and Statistics*, pages 254–261, 2003.

S. Seo, M. Wallat, T. Graepel, and K. Obermayer. Gaussian process regression: Active data selection and test point rejection. In *Mustererkennung 2000*, pages 27–34. Springer, 2000.

M. C. Shewry and H. P. Wynn. Maximum entropy sampling. *Journal of applied statistics*, 14(2):165–170, 1987.

M. Smith, M. Alvarez, and N. D. Lawrence. Gaussian process regression for binned data. *arXiv:1809.02010*, 2018a.

M. Smith, M. Álvarez, M. Zwiessele, and N. D. Lawrence. Differentially private regression with gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 1195–1203, 2018b.

A. J. Smola and P. L. Bartlett. Sparse greedy Gaussian process regression. In *Advances in neural information processing systems*, pages 619–625, 2001.

E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264, 2005.

E. Solak, R. Murray-Smith, W. Leithead, D. Leith, and C. Rasmussen. Derivative observations in Gaussian process models of dynamic systems. In *Advances in neural information processing systems*, pages 1057–1064, 2002.

A. Solin and S. Särkkä. Hilbert space methods for reduced-rank Gaussian process regression. *Statistics and Computing*, 30(2):419–446, 2020.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for modern deep learning research. *AAAI Conference on Artificial Intelligence*, 34(09):13693–13696, Apr. 2020.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Y. Tanaka, T. Iwata, T. Tanaka, T. Kurashima, M. Okawa, and H. Toda. Refining coarse-grained spatial data using auxiliary spatial data sets with various granularities. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 5091–5099, 2019a.

Y. Tanaka, T. Tanaka, T. Iwata, T. Kurashima, M. Okawa, Y. Akagi, and H. Toda. Spatially aggregated Gaussian processes with multivariate areal outputs. In *Advances in neural information processing systems*, pages 3005–3015, 2019b.

M. Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 567–574, 2009.

R. Turner, M. Deisenroth, and C. Rasmussen. State-space inference and learning with Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 868–875, 2010.

U.S. Energy Information Administration. International Energy Outlook 2016. `https://www.eia.gov/outlooks/ieo/pdf/industrial.pdf`, last accessed on 21.02.2022.

N. Wahlström. *Modeling of magnetic fields and extended objects for localization applications*. PhD thesis, Linköping University, 2015.

N. Wahlström, M. Kok, T. B. Schön, and F. Gustafsson. Modeling magnetic fields using Gaussian processes. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3522–3526. IEEE, 2013.

M. J. Wainwright and M. I. Jordan. *Graphical models, exponential families, and variational inference*. Now Publishers Inc, 2008.

J. Wang, A. Hertzmann, and D. J. Fleet. Gaussian process dynamical models. In *Advances in neural information processing systems*, volume 18, pages 1441–1448, 2005.

J. M. Wang, D. J. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence*, 30(2):283–298, 2007.

K. Wang, G. Pleiss, J. Gardner, S. Tyree, K. Q. Weinberger, and A. G. Wilson. Exact Gaussian processes on a million data points. In *Advances in neural information processing systems*, volume 32, pages 14648 – 14659, 2019.

P. J. Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770. Springer, 1982.

R. J. Williams and D. Zipser. Gradient-based learning algorithms for recurrent. *Backpropagation: Theory, architectures, and applications*, 433, 1995.

Y. Wu, J. M. Hernández-Lobato, and Z. Ghahramani. Gaussian process volatility model. In *Advances in neural information processing systems*, volume 27, pages 1044 – 1052, 2014.

A. Xavier, M. d. B. C. Freitas, M. do Socorro Rosário, and R. Fragoso. Disaggregating statistical data at the field level: An entropy approach. *Spatial Statistics*, 23:91–108, 2018.

F. Yousefi, M. T. Smith, and M. Álvarez. Multi-task learning for aggregated data using Gaussian processes. In *Advances in neural information processing systems*, volume 32, pages 15076 – 15086, 2019.

C. Yu, M. Seslija, G. Brownbridge, S. Mosbach, M. Kraft, M. Parsi, M. Davis, V. Page, and A. Bhave. Deep kernel learning approach to engine emissions modeling. *Data-Centric Engineering*, 1:e4, 2020.

A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

C. Zhang, J. Butepage, H. Kjellström, and S. Mandt. Advances in Variational Inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):2008–2026, 2019.

J. Zhang, P. Atkinson, and M. F. Goodchild. *Scale in spatial information and analysis*. CRC Press, 2014.

Z. Zhu and M. L. Stein. Spatial sampling design for prediction with estimated parameters. *Journal of agricultural, biological, and environmental statistics*, 11(1):24–44, 2006.

C. Zimmer and R. Yaesoubi. In *Influenza Forecasting Framework based on Gaussian Processes*, pages 11671–11679, 2020.