INCREMENTAL LEARNING FOR ROBOTICS WITH CONSTANT UPDATE COMPLEXITY

Arjan Gijsberts

INCREMENTAL LEARNING FOR ROBOTICS WITH CONSTANT UPDATE COMPLEXITY

by

Arjan Gijsberts

A thesis submitted in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Doctoral Course on Robotics, Neuroscience, and Nanotechnologies

Department of Communication, Computer, and System Sciences
University of Genoa

and

Department of Robotics, Brain, and Cognitive Sciences
Italian Institute of Technology

Supervisors: Prof. Giulio Sandini and Prof. Giorgio Metta

ABSTRACT

The field of robotics is moving increasingly toward applications in unstructured and non-stationary human environments. These environments often cannot be foreseen in all relevant details and require therefore robots that are able to adapt to changing conditions and to learn from experience. This thesis presents the incremental Random Fourier Regularized Least Squares and Sparse Spectrum Gaussian Process Regression algorithms that are targeted for use in this context. Contrary to related work, the primary design objectives of these incremental learning methods are (1) a theoretical foundation, (2) computational efficiency, and (3) practical convenience. Rather than developing a novel algorithm from the ground up, the methods are based on the thoroughly studied Regularized Least Squares and Gaussian Process Regression algorithms, therefore ensuring a solid theoretical foundation. Non-linearity and a bounded update complexity are achieved simultaneously by means of a finite dimensional random feature mapping that approximates a kernel function. Furthermore, the computational cost for each update is predictable, allowing the methods to be used in real-time applications. Finally, their algorithmic simplicity and support for automated hyperparameter optimization ensures practical convenience when employing these methods in practice.

Empirical validation on a number of synthetic and real-life sensorimotor learning problems confirms that the generalization performance of these methods is competitive with state of the art batch learning algorithms and superior with respect to competing incremental algorithms. This performance is maintained in the challenging situation of dependent sampling distributions, which are common in learning problems that are grounded in the physical world. Moreover, their computational requirements are found to be significantly lower with respect to competing methods, such that the methods are particularly advantageous for large scale learning or when limited computational resources are available. As additional empirical validation, a dynamics model learned by means of incremental Sparse Spectrum Gaussian Process Regression is used to implement contact detection for the arm of the iCub humanoid robot.

ACKNOWLEDGMENTS

This thesis is the final product of my work in the Robotics, Brain, and Cognitive Sciences Department at the Italian Institute of Technology. I would like to thank Giulio Sandini and Giorgio Metta for letting me be a part of the "iCub" family and for providing the means to do research in this diverse environment. In this regard, I would like to thank all people at the institute with whom I have had the pleasure of being acquainted over these years. In particular, I want to mention Andrew Dankers, Vadim Tikhanoff, Francesco Rea, Tim Coles, and Stephen Hart for countless interesting discussions (not always about robotics or science), either while working at our desk or while enjoying a coffee. A special word of thanks goes to Francesco Orabona, who has been helpful on numerous accounts regarding many details of machine learning.

Clearly, I could never forget to mention my family and to express my gratitude. Especially when it comes to my education no effort has ever been too big for them. I am very grateful to my parents, my brother, and my aunts for their support. One person that has enriched my stay here in Italy is my girlfriend Roberta Basili. I would like to thank her for her understanding of the obligations toward my PhD as well as all the wonderful moments that we have spent together.

My research was financially supported by European Commission project ITALK (ICT-214668). I am thankful for the financial support I received and for the interesting and multidisciplinary project meetings.

NOTATION

$\boldsymbol{z}$	vector
$\boldsymbol{Z}$	matrix
$\boldsymbol{z}^{\mathrm{T}}, \boldsymbol{Z}^{\mathrm{T}}$	transpose of vector $\boldsymbol{z}$ or matrix $\boldsymbol{Z}$
$\boldsymbol{x} \in \mathcal{X}$	input and input space
$\boldsymbol{X}$	matrix where each row is an input sample $\boldsymbol{x}$
$y, \boldsymbol{y} \in \mathcal{Y}$	scalar or vector output and output space
$\boldsymbol{y}, \boldsymbol{Y}$	vector or matrix where each row is an output sample y or $\boldsymbol{y}$
m	number of (training) samples
n	dimensionality of input space $\mathcal{X}$
p	dimensionality of output space $\mathcal{Y}$
$\mathcal{S}$	set of m input-output pairs $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{m}$
$\langle \cdot, \cdot \rangle$	inner product
$\|\cdot\|_p$	p-norm, with $p = 2$ by default
$f(\cdot) \in \mathcal{F}$	function and function space
$k(\cdot, \cdot)$	kernel function
$\boldsymbol{K}$	kernel matrix obtained by evaluating k for each pair of samples in $\boldsymbol{X}$
$\mathcal{H}_K$	reproducing kernel Hilbert space
$\phi(\cdot)$	feature mapping
$\boldsymbol{\Phi}$	design matrix obtained by evaluating ϕ on each sample in $\boldsymbol{X}$
$\boldsymbol{Z} \succ 0$	positive definite matrix
$\boldsymbol{Z} \succeq 0$	positive semidefinite matrix
$\boldsymbol{I}$	identity matrix
$\mathbf{0}$	zero vector or matrix
$\hat{\boldsymbol{y}}$	predicted output
ϵ	measurement error
e	residual or prediction error $y - \hat{y}$
s^2	predicted variance
$\mathcal{L}(\cdot, \cdot)$	loss function
$P(\cdot)$	probability distribution
$p(\cdot)$	probability density
$\mathcal{N}\big(x \mid \mu, \sigma^2\big)$	Gaussian or normal distribution over x with mean μ and variance σ^2
$\mathcal{U}(a, b)$	uniform distribution in the interval $[a, b]$
$\mathcal{GP}$	Gaussian process

$\mathbb{E}[\cdot]$	expected value of a random variable
$\mathbb{V}[\cdot]$	variance of a random variable
$\mathcal{R}(\cdot)$	risk functional
$\mathbb{N}$	natural numbers
$\mathbb{R}$	real numbers
$\mathcal{O}(\cdot)$	asymptotic upper complexity bound
δ_{ij}	Kronecker delta, 1 if $i = j$ and 0 otherwise

ACRONYMS

ANN	Artificial Neural Network
ARD	Automatic Relevance Detection
ASE	Asymmetric Squared Exponential
CUSUM	cumulative sum control chart
DBN	Deep Belief Network
DoF	Degree of Freedom
ϵ-SVR	ϵ-insensitive Support Vector Regression
GLR	Generalized Likelihood Ratio
GMM	Gaussian Mixture Model
GPR	Gaussian Process Regression
i.i.d.	independent and identically distributed
KM	kernel method
KNN	Kernel Nearest Neighbor
KRLS	Kernel Regularized Least Squares
KRR	Kernel Ridge Regression
LASSO	least absolute shrinkage and selection operator
LGP	Local Gaussian Processes
LSSVM	Least Squares Support Vector Machine
LWPR	Locally Weighted Projection Regression
LWR	Locally Weighted Regression
MKL	Multi Kernel Learning
nMSE	normalized Mean Squared Error

PAC	probably approximately correct
RBD	Rigid Body Dynamics
RBF	Radial Base Function
RFRLS	Random Fourier Regularized Least Squares
RFWR	Receptive Field Weighted Regression
RKHS	Reproducing Kernel Hilbert Space
RLS	Regularized Least Squares
RN	Regularization Network
RNEA	Recursive Newton-Euler Algorithm
ROC	receiver operating characteristic
SRM	Structural Risk Minimization
SSGPR	Sparse Spectrum Gaussian Process Regression
SVM	Support Vector Machine
VC	Vapnik-Chervonenkis
WGLR	Window-Limited Generalized Likelihood Ratio

CONTENTS

LIST OF FIGURES

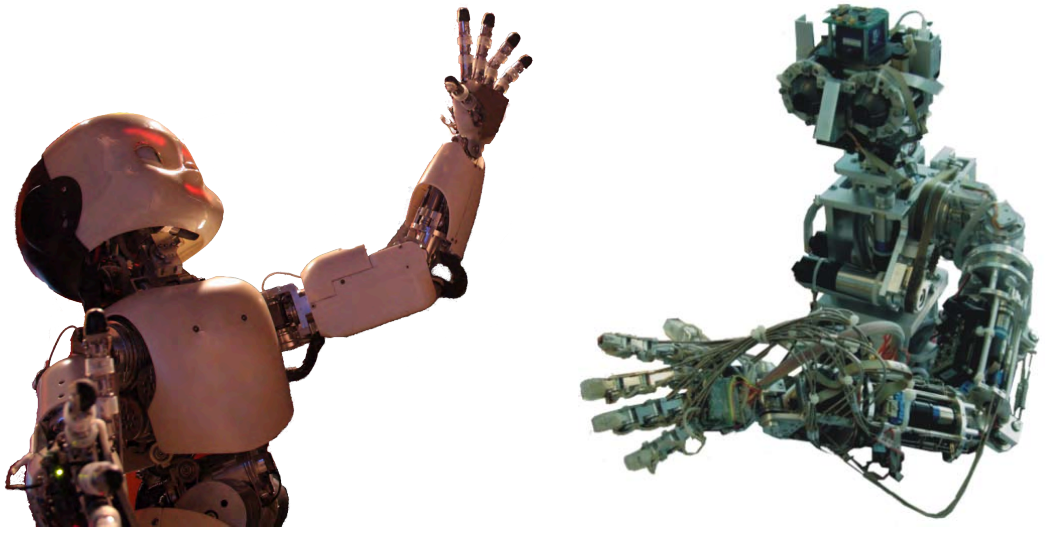
LIST OF TABLES

1 INTRODUCTION

The field of robotics is increasingly moving toward applications beyond the traditional structured environments such as manufacturing plants. The first robots have entered human environments and it is expected that service robots will find use in applications ranging from domestic chores and health care to entertainment and social companionship (for an essay on this topic, see Gates, 2007). These human-centered applications require dexterous and safe robots that are able to solve unforeseen tasks autonomously in unstructured environments. The traditional approach to robotics, consisting of precisely characterizing the robot, its environment, and the desired task, is rendered infeasible in this novel setting. The field of robotics therefore faces significant challenges in the development of a future generation of robots, both in terms of suitable mechanics and morphology, as well autonomous and intelligent behavior.

A viable and arguably inevitable direction of research to obtain these goals is the study of robots that are able to adapt to changing conditions and to learn from past experience. This strategy is particularly apposite if the environment cannot be foreseen in all relevant details, either due to its complexity or its non-stationarity. Learning mechanisms allow advanced robots to act successfully under such adverse circumstances and have shown superior performance with respect to traditional analytic approaches in a variety of applications. The field of machine learning deals with developing techniques and algorithms that help machines to "learn" from experience or, equivalently, to infer knowledge from observations. It is therefore not surprising that there is a significant mutual interest from the robotics and machine learning communities to jointly develop autonomous robots using advanced learning techniques.

The subject of this thesis is at the interjunction of these two fields; more specifically, it presents an incremental, non-linear, and computationally efficient learning algorithm for use in adaptive robotics. The motivation and specific context for this algorithm are further described in the fol-



(a) iCub **(b)** James

Figure 1.1: Two humanoid robots currently residing at the Italian Institute of Technology. The iCub has been developed as part of the RobotCub project (Metta et al., 2006). James is a predecessor of the iCub, originally developed at the Laboratory for Integrated Advanced Robotics of the University of Genoa.

lowing section. Subsequently, the approach is summarized in Section 1.2, with an emphasis on its merits with respect to related work and the contributions for the fields of robotics and machine learning. Finally, the organization of this thesis is outlined in Section 1.3

1.1 Learning and Adaptation in Robotics

Performing complicated tasks in unstructured human environments imposes strict requirements on the design and operation of robots. Not only should robots be autonomous and dexterous, in addition they should also operate safely by minimizing the possibility of inflicting damage or injuring humans. The morphology, actuation, and perception of the robot is therefore to be designed carefully to comply with these requirements. Typical design objectives include dexterous hands for fine manipulation tasks, an appropriate size that allows reaching for objects without restricting locomotion, and low weight to minimize impact when colliding with the environment. A compelling line of thought is that robots employed in human-centered environments should share the human morphology to a large extent. Figure 1.1 demonstrates two such robots in the form of two state of the art humanoids operated at the Italian Institute of Technology. The iCub robot is a full-body humanoid approximately the size of a $3\frac{1}{2}$ year old child with 53 Degrees of Freedom (DoF) (Tsagarakis et al., 2007; Metta et al., 2010). Its 22-DoF predecessor James is an upper torso humanoid roughly the size of a 10 year old boy (Jamone et al., 2006).

A consequence of the increasing mechanical complexity of these advanced robots is that they are

significantly more complicated to describe and control using analytical means. This complexity is not limited to the large number of DoF, but also regards the mechanical constructions required to actuate the robot given the constraints on size, weight, and safety. The difficulty in controlling the behavior of these robots is further amplified by the fact that human environments are unstructured and non-stationary, and that the desired skill-set for the robot can often not be precisely anticipated. Hence, the traditional approach of modeling the robot, its environment, and the desired tasks prior to deployment is no longer feasible in this setting. A more realistic approach for successful operation with these uncertainties is to create autonomous robots that are able to continuously adapt their behavior to the environment and learn from previous experience. This adaptation has to take place while the robot is operating in the environment and should be time-efficient to ensure responsive behavior. Inspired by psychological stages in human development, the paradigm of developmental robotics goes even further by requiring robots to undergo multiple stages of autonomous mental development to attain an increasingly sophisticated skill-set (Weng et al., 2001; Lungarella et al., 2003).

The purpose of learning and adaptation is to allow robots to make accurate predictions in an unknown environment. Accurate predictions of both the internal (i.e., its body) and external environment (and interactions between these) are crucial for anticipatory behavior and skillful control. Not surprisingly, machine learning techniques (see Figure 1.2 for an overview) are suitable candidates to implement such predictive capabilities. In particular, we are interested in techniques that are appropriate given the following properties of the robotics domain:

Accuracy: The quality of a prediction is measured to a large extent by its accuracy. More accurate predictions simplify control and translate in increased task efficacy and efficiency. Moreover, generalization from earlier observations is required to obtain satisfactory predictions, since it is unlikely that identical situations occur more than once.

Non-Stationarity: The physical world is not stationary and changes occur at various time scales. This includes changes in the environment due to external actors or physical phenomena, as well as internal changes in the body of the robot due to wear-and-tear, tool use, or environmental influences.

Incremental Operation: Tasks, observations, and environmental changes can often not be anticipated, and their occurrence and order are governed by the physical environment. It follows that learning is necessarily an integral and continuous process during open-ended robot operation (i.e., online), which incrementally updates knowledge of both the internal and external environment.

Autonomy: The robot and its learning process should be as autonomous as possible, thus requiring a minimal amount of prior programming and human intervention during operation.

Efficiency: Timeliness is a secondary measure of the quality of predictions. Timely predictions

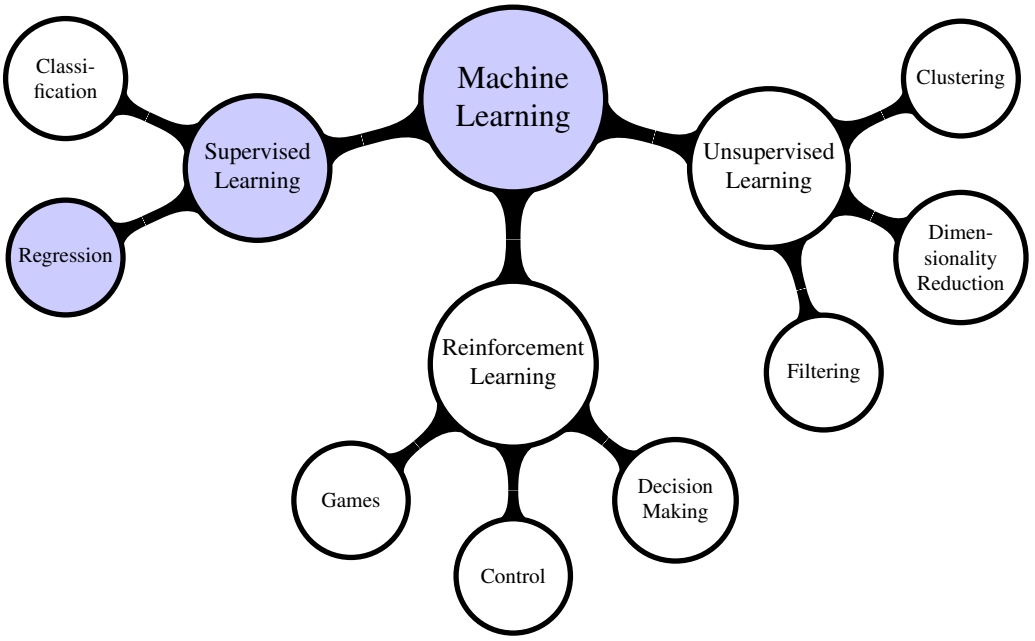


Figure 1.2: Overview of the primary paradigms within the field of machine learning. The context considered in this thesis is marked in blue.

are crucial for responsive behavior and dynamic interactions with the environment. In addition, efficiency is generally desired in order to make optimal use of the limited computational resources of the robot.

A multitude of machine learning approaches have been used to allow learning in complex robot systems, among which reinforcement learning and regression methods have found the most prominent use. The topic of this thesis will therefore be constrained to (supervised) regression methods that conform to abovementioned properties. It should be noted, however, that all primary machine learning paradigms (cf. Figure 1.2) are to some extent relevant in the development of intelligent robots that act autonomously in unstructured environments.

1.1.1 Model Learning

The limitation to regression problems might initially be perceived as rather restrictive. However, regression is a fundamental technique for learning associations between motor actions and corresponding sensory responses (i.e., sensorimotor learning). These associations are at the center of action and perception of the robot and therefore a core component for the control and behavior of robots. This is supported by the hypothesis that learning sensorimotor associations is among the first stages in human development (Piaget, 1952). Two types of models are commonly learned, namely forward and inverse models. Forward models predict the perceptual consequences associated with motor actions. For example, a forward kinematic model computes the (typically Cartesian) position of a manipulator for a given joint configuration. In contrast, inverse models

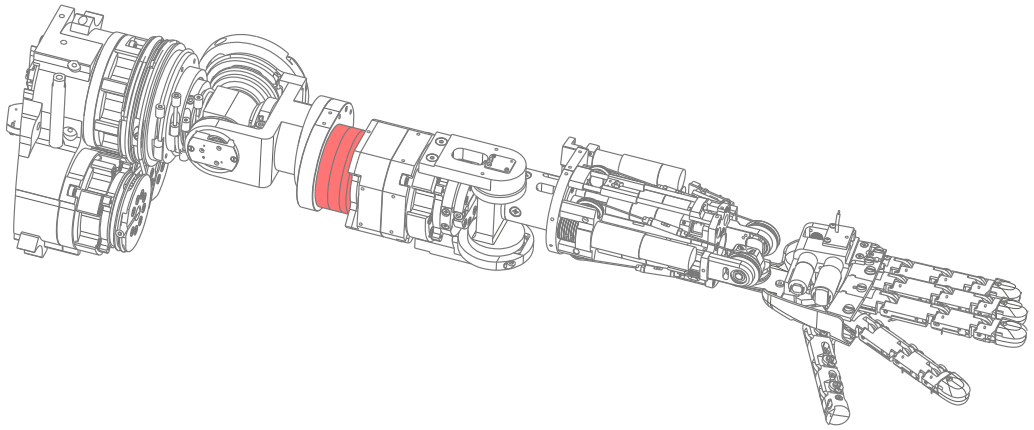


Figure 1.3: Position of the proximal force/torque sensor (indicated in red) in the left arm of the iCub humanoid.

attempt to predict the action that results in desired perceptual changes. Following the earlier example, an inverse kinematics model thus computes the joint configuration that is required to attain a desired Cartesian position.

One class of internal models that will be used extensively in the experimental validation in Chapter 5 is the robot dynamics. There are two primary reasons for prominently featuring this particular type of internal model. First, the availability of (inverse) dynamics models is important for compliant control and contact detection, which themselves are crucial techniques for safe robot operation in human environments. Second, dynamics modeling is a well-posed supervised learning problem, as opposed to kinematics, since the desired outputs can be measured directly using a force/torque sensor. Figure 1.3 demonstrates a proximal force/torque sensor mounted in the arm of the iCub humanoid. The relationship between manipulator configuration (i.e., joint positions, velocities, and accelerations) and forces/torques can be described analytically using the Rigid Body Dynamics (RBD) formula

$$\boldsymbol{\tau} = \boldsymbol{D}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\ddot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) \ ,$$

where $\boldsymbol{D}$, $\boldsymbol{C}$ and $\boldsymbol{g}$ are the inertial, Coriolis, and gravity terms, respectively (Spong et al., 2006; Siciliano and Khatib, 2008). Despite being analytically correct, this formulation has limited applicability on real-life robots due to the difficulty of accurately determining the various kinematic and dynamic parameters. Moreover, many modern lightweight robots (such as James and iCub) have significant additional nonlinear dynamics beyond the RBD model, such as actuator dynamics (e.g., due to gearboxes), routing of cables, and protective covers. Additional arguments to favor learning over analytical modeling will be given in Section 2.8.

1.2 Proposed Method and Contributions

The need for regression methods that can be used in the described setting has been acknowledged in the robotics community and this has led to the development of a number of domain-appropriate learning methods (e.g., Schaal and Atkeson, 1998; Vijayakumar et al., 2005; Nguyen-Tuong et al., 2009). These methods typically involve comparatively complicated mechanisms to conform with the domain specific properties and are less supported by theory than recent machine learning techniques. A detailed treatment of these methods will follow in Section 3.3.

This thesis presents an alternative learning method that competes directly with these previous methods. Contrary to previous work, this method has been developed (1) to conform directly with the requirements outlined in the previous section, (2) using established and proven techniques, and (3) to be convenient in practical use. The result is a learning method that combines efficient linear learning algorithms with a kernel approximation for non-linearity. The contribution of this method for the robotics and machine learning communities can be summarized as follows:

Theoretical Foundation: The method is based on established and rigorously studied methods, rather than devising an entirely novel algorithm from the ground up. This theoretical foundation ensures excellent generalization performance, convergence to an optimal solution given sufficient observations, and numerical stability when implemented on finite-precision hardware.

Computational Efficiency: The method improves over competing methods in terms of computational requirements (as measured in wall time) and is only rivaled by an analytical RBD model when learning robot dynamics. Furthermore, the time and space complexity per update are constant with respect to the number of observations and usage in a real-time setting is feasible due to accurate predictability of the computational cost. In addition, fewer observations are required to attain satisfactory predictive accuracy.

Practical Convenience: An important property of the method is its algorithmic simplicity, which signifies that it can be understood, implemented, and deployed easily. Moreover, the number of hyperparameters is kept at a minimum and these can be tuned automatically using a principled methodology. The tradeoff between computation and predictive accuracy can be controlled directly using a single hyperparameter.

The relative merits of the method are validated empirically both in batch as well incremental settings using synthetic and realistic robotic problems. Furthermore, the practical use of the method is further emphasized by applying it for robotic contact detection using a learned dynamics model. Although the method is presented primarily in a robotics context, it is by no means limited to this specific context and can generally be used for incremental learning problems.

1.3 Organization

The remainder of the thesis continues with a general introduction in the theory of machine learning and kernel methods in Chapter 2. The relationship between robotics and machine learning will receive additional attention in its final section. Subsequently, an overview of state of the art incremental learning methods is given in Chapter 3, which includes dedicated sections on incremental kernel methods and learning techniques that have been presented primarily for use in robotics. The proposed method is subsequently described in Chapter 4, which ends with a thorough discussion on its properties and merits with respect to related work. The empirical validation is subdivided in two sections; Chapter 5 contains comparative experiments on a number of synthetic and real robotics datasets, while Chapter 6 describes how the proposed method can be used to successfully implement contact detection on the iCub humanoid. Finally, conclusions and directions for future work are given in Chapter 7.

MACHINE LEARNING AND KERNEL METHODS

2

The field of machine learning is concerned with designing algorithms that allow computers to evolve behavior based on empirical observations. A primary focus in the study of learning algorithms is on generalization, namely the capacity to extend knowledge from past observations to future behavior; much similar to the human capacity to generalize. For instance, a human is often able to classify novel instances of objects in its environment by matching it with certain known features of similar objects (e.g., structure, color, context, or functionality). Learning algorithms aim to replicate this capacity in a computational sense.

Several frameworks have been proposed to formalize the learning problem from a theoretical perspective. Among these are probably approximately correct (PAC) learning (Valiant, 1984), Bayesian inference (MacKay, 2002), and statistical learning theory (Vapnik, 1995). The latter framework, developed over the last four decades by Vapnik and others, was essential for the development of the popular Support Vector Machines (SVMs) (Drucker et al., 1996). Another fundamental aspect of this learning method is the use of kernel functions, which implicitly map input data onto a hypothetical, high dimensional feature space. The learning method is applied in this feature space, such that non-linearity is obtained while avoiding a large computational overhead. Although popularized within SVM, this technique can be used with various linear algorithms and the resulting algorithms are collectively known as kernel methods (KMs). This class of algorithms has gained considerable popularity within the machine learning community due to a rigorous theoretical foundation and excellent empirical performance (e.g., Schölkopf and Smola, 2001; Zhang et al., 2007; Gijsberts et al., 2010a). However, despite this apparent success, there are both practical and theoretical concerns with respect to applications in (developmental) robotics. These regard primarily the computational complexity of many kernel-based algorithms, their batch operation, and violations of key assumptions of the theoretical learning framework.

The first section of this chapter introduces the most relevant aspects of statistical learning theory. These concepts are subsequently used as mathematical background throughout the remaining sections. KMs are initially presented at the hand of the linear Regularized Least Squares (RLS) algorithm in Section 2.2, after which kernel functions are introduction in Section 2.3. The kernel-based variant of RLS is described in Section 2.4 together with some related methods (e.g., SVM). Subsequently, a treatment of Gaussian Process Regression (GPR) is given in Section 2.5. Although the latter algorithm is developed in a Bayesian framework, its functional form shows remarkable similarities with RLS. Considerations for employing these methods in practice, such as kernel selection and hyperparameter optimization, is covered in Section 2.6, followed by an example learning scenario in Section 2.7. Finally, this chapter is concluded in Section 2.8 with a discussion on practical and theoretical concerns with respect to the robotics application domain.

2.1 Learning Theory

In order to describe statistical learning theory, it is useful to formalize the learning process in mathematical terms. Suppose that we observe a process that generates an output y given a certain input $\boldsymbol{x}$, where each observation consists of an n-dimensional input $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^n$ and the corresponding output $\boldsymbol{y} \in \mathcal{Y}$. In case $\mathcal{Y}$ denotes a discrete set of classes (e.g., $\mathcal{Y} \subseteq \{-1, 1\}$), then the problem is considered a classification problem (cf. Figure 1.2). However, in the following we will limit ourselves to regression problems, such that $\mathcal{Y} \subseteq \mathbb{R}$. Let us assume that we have collected a finite set of m observations, denoted as $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$. The goal in machine learning is to use these observations to "learn" a function $f : \mathcal{X} \mapsto \mathcal{Y}$, such that $f(\boldsymbol{x})$ is a good approximation of the output y for an arbitrary input $\boldsymbol{x}$. Obviously, in order to find such a function, we must require that the collected observations are actually representative for possible future observations. In statistical learning theory, this is guaranteed by requiring that the observations are independent and identically distributed (i.i.d.) according to a fixed but unknown distribution P on $\mathcal{X} \times \mathcal{Y}$. This joint distribution of inputs and outputs can be decomposed as $P(y, \boldsymbol{x}) = P(y|\boldsymbol{x})P(\boldsymbol{x})$. Note that we may recognize two phases in the generation of samples: first, an input $\boldsymbol{x}$ is generated according to the marginal (or prior) probability $P(\boldsymbol{x})$; second, the corresponding output y is generated according to the conditional probability $P(y|\boldsymbol{x})$ given the input $\boldsymbol{x}$. These probability distributions imply that (1) we cannot control the generation process of the samples and (2) the mapping from inputs to outputs is stochastic rather than deterministic.

Learning methods use these observations to produce a function that captures the relationship between inputs and outputs, or in formal terms $\mathcal{X} \times \mathcal{Y} \mapsto f$. The approximation quality of a function f is measured at the hand of a non-negative loss function $\mathcal{L}(y, f(\boldsymbol{x}))$, where $\boldsymbol{x}$ and y are implicitly assumed to come from a single input-output pair. The optimal function f^* can therefore be defined as the one that minimizes the loss over the weighted space of inputs and outputs. In other words,

we seek to find a function f that minimizes the *expected risk* (Vapnik, 1995), defined as

$$\mathcal{R}_{\text{exp}}(f) = \int_{\mathcal{X}\times\mathcal{Y}} \mathcal{L}(y, f(\boldsymbol{x}))\, \mathrm{d}P(y, \boldsymbol{x}) \ . \tag{2.1}$$

Direct minimization of this integral is infeasible, since the joint distribution $P(y, \boldsymbol{x})$ is unknown. The only available information regarding the process is the set of observations $\mathcal{S}$. It may seem reasonable to use the average loss on these observations instead

$$\mathcal{R}_{\text{emp}}(f) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(y_i, f(\boldsymbol{x}_i)) \ , \tag{2.2}$$

which is known as the *empirical risk*. However, direct optimization of the empirical risk is error prone due to the finite number of samples. This can be understood by considering a function that memorizes all samples in $\mathcal{S}$ and returns an arbitrary constant value for all other samples. Although this function clearly minimizes the empirical risk in Equation (2.2), it will almost surely perform poorly in terms of the expected risk in Equation (2.1). This phenomenon is known as overfitting, in which a learning method produces a function that models the possibly noisy observations in $\mathcal{S}$ too closely. As a result, the function does not generalize well on future (unseen) observations.

The problem of overfitting can be alleviated by restricting the optimization to a limited function space $\mathcal{F}$. This produces satisfactory results if $\mathcal{F}$ is chosen sufficiently small to prevent overfitting. Unfortunately, identifying an appropriate space $\mathcal{F}$ requires intimate knowledge of $P(y, \boldsymbol{x})$ and generalization performance may suffer if $\mathcal{F}$ is chosen too restrictive. It is therefore more desirable to choose $\mathcal{F}$ sufficiently large and to penalize "complex" functions, as demonstrated graphically in Figure 2.1. In other words, we want to minimize the regularized risk functional

$$\mathcal{R}_{\text{reg}}(f) = \mathcal{R}_{\text{emp}}(f) + \lambda G(f) \qquad \text{s.t. } f \in \mathcal{F} \ , \tag{2.3}$$

where $G(f)$ is a regularizer that penalizes complex functions and λ is the a constant that regularizes the tradeoff between both terms. There is a broad range of choices for the regularization term $G(f)$. As we will see later, different loss functions and regularization terms lead to different learning methods. Two common choices for the regularization term are L_1 and L_2 regularization (it is assumed that $\mathcal{F}$ is an inner product space), defined respectively as $\|f\|_1$ and $\|f\|_2^2$. The former leads to sparse solutions, whereas the L_2 norm promotes smoothness by penalizing wild oscillations.

2.2 Regularized Least Squares

A compelling choice is to use the L_2 norm for both the loss function and the regularization terms, such that $G(f) = \|f\|^2$ and $\mathcal{L}(y, f(\boldsymbol{x})) = (y - f(\boldsymbol{x}))^2$. Let us consider this configuration while

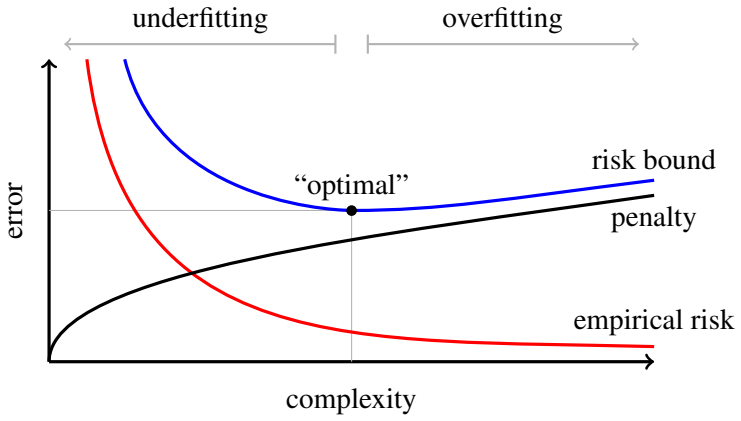


Figure 2.1: Demonstration of overfitting and underfitting with respect to model complexity. The risk bound is the sum of both the empirical risk and the complexity penalty.

constraining $\mathcal{F}$ to the set of linear functions of the form

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle \quad , \tag{2.4}$$

where $\boldsymbol{w}$ is a weight vector. Inserting the squared loss and L_2 regularization in Equation (2.3) results in a convex optimization problem of $\boldsymbol{w}$, for which the objective function is given by

$$\begin{aligned} J(\boldsymbol{w}, \lambda) &= \frac{\lambda}{2} \|\boldsymbol{w}\|^2 + \frac{1}{2} \sum_{i=1}^{m} (y_i - f(\boldsymbol{x}_i))^2 \\ &= \frac{\lambda}{2} \|\boldsymbol{w}\|^2 + \frac{1}{2} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2 \quad . \end{aligned}$$

Note that the bottom equation uses matrix notation by defining an $m \times n$ matrix of input samples $\boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_m]^{\mathrm{T}}$ and an m-dimensional vector of outputs $\boldsymbol{y} = [y_1, ..., y_m]^{\mathrm{T}}$. Furthermore, the additional factor $\frac{1}{2}$ is solely for mathematical convenience and does not affect the optimal solution. Setting the partial derivative of J with respect to $\boldsymbol{w}$ to zero, such that

$$\frac{\partial J}{\partial \boldsymbol{w}} = \boldsymbol{w} \left(\lambda \boldsymbol{I} + \boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \right) - \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y} = 0 \quad ,$$

we obtain the optimal solution given by

$$\arg\min_{\boldsymbol{w}} J(\boldsymbol{w}, \lambda) = \left(\lambda \boldsymbol{I} + \boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \right)^{-1} \boldsymbol{X}^{\mathrm{T}} \boldsymbol{y} \quad . \tag{2.5}$$

Note that this system of linear equations is well-posed if $\lambda > 0$, thus guaranteeing a unique optimal solution. It is therefore not surprising that this technique, known as Tikhonov regularization, was originally proposed to improve conditioning of ill-posed linear systems of equations (Tikhonov and Arsenin, 1977). Since then, the regularized optimization problem in Equation (2.5) has been introduced in the statistical community as ridge regression (Hoerl and Kennard, 1970). An equiv-

alent formulation with respect to Artificial Neural Networks (ANNs) is known as weight decay (Hinton, 1989), while it has also been studied extensively within the framework of Regularization Networks (RNs) (Poggio and Girosi, 1990; Girosi et al., 1993). In the remainder, this algorithm will be referred to as (linear) RLS (following Rifkin et al., 2003), to emphasize (1) the link to both regularization and the least squares method and (2) that it can and has been used successfully for both regression and classification problems.

Obtaining the solution for Equation (2.5) requires solving an n-dimensional system of linear equations. As we will see in later chapters, explicit inversion of the covariance matrix $\left(\lambda \boldsymbol{I} + \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)$ is usually not necessary. Regardless, the overall time complexity for training is cubic in the number of input features and linear in the number of samples. Once an optimal weight vector $\boldsymbol{w}$ has been found, the function in Equation (2.4) can be used to predict the output for a given input vector $\boldsymbol{x}$. It is clear from Equation (2.4) that the time complexity per prediction is linear in n and independent of the number of training samples m.

The output variable y has thus far been assumed scalar for simplicity; however, it is relatively easy to consider the case of a p-dimensional output vector $\boldsymbol{y}$[1] (Bishop, 2006, Section 3.1.5). Substituting the vector variant in Equation (2.4) and Equation (2.5), we observe that the weight vector becomes an $n \times p$ weight matrix $\boldsymbol{W}$, where each column contains the weight vector corresponding to one of the p regression problems. The advantage of this formulation is of computational nature: solving Equation (2.5) is dominated by inversion of the covariance matrix. Since this matrix does not depend on the output variables, the inversion step can easily be "reused" to solve multiple regression problems simultaneously at negligible additional cost. Obviously, a requirement is that all problems share identical input variables $\boldsymbol{X}$ and regularization parameter λ.

2.3 Kernel Trick

The set of linear functions is commonly too restrictive for real-life problems, which are often found to be non-linear. A well-known approach to apply linear algorithms on non-linear problems is to map the input features into a feature space. The input-output relation can consequently be represented in linear form in this feature space. Let us consider a mapping function

$$\phi : \mathcal{X} \mapsto \mathcal{H} \ ,$$

where Hilbert space $\mathcal{H}$ is referred to as the feature space. Substituting mapping function ϕ in Equation (2.4), we obtain

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \phi(\boldsymbol{x}) \rangle \ ,$$

[1] The distinction between using $\boldsymbol{y}$ to denote multiple outputs of a single sample or the scalar output of multiple samples will be clear from context.

and the optimal solution from Equation (2.5) becomes

$$\arg\min_{\boldsymbol{w}} J(\boldsymbol{w}, \lambda) = \left(\lambda \boldsymbol{I} + \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{y} \ , \tag{2.6}$$

where we defined (slightly abusing notation) the design matrix $\boldsymbol{\Phi} = \phi(\boldsymbol{X})$. In other words, the algorithm operates explicitly in the feature space and consequently $\boldsymbol{w} \in \mathcal{H}$. Various types of non-linear feature mappings, also known as basis functions, have been used in practice, such as Radial Base Functions (RBFs), polynomial basis functions, splines, sigmoidal functions, and Fourier bases (for details, see Bishop, 2006; Hastie et al., 2009, and references therein).

The time complexity of Equation (2.6) is cubic in the number of features and may therefore be computationally infeasible if we choose a large number of basis functions. The kernel trick helps to avoid this computational limitation and even allows linear methods to be applied in infinite dimensional feature spaces. This kernel concept was originally introduced in the pattern recognition community by Aizerman et al. (1964), and subsequently popularized in the context of SVMs (Boser et al., 1992; Cortes and Vapnik, 1995; Cristianini and Shawe-Taylor, 2000; Schölkopf and Smola, 2001). In order to explain kernels, let us first consider the dual representation of the RLS algorithm. The optimal solution in Equation (2.6) can be rewritten in terms of $\boldsymbol{w}$ as

$$\boldsymbol{w} = \frac{1}{\lambda} \boldsymbol{\Phi}^{\mathrm{T}} \left(\boldsymbol{y} - \boldsymbol{\Phi} \boldsymbol{w}\right) = \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha} = \sum_{i=1}^{m} \alpha_i \phi(\boldsymbol{x}_i) \ , \tag{2.7}$$

showing that $\boldsymbol{w}$ can be rewritten as a linear combination of the training samples. It follows that the m-dimensional coefficient vector $\boldsymbol{\alpha}$ is given by

$$\boldsymbol{\alpha} = \frac{1}{\lambda} \left(\boldsymbol{y} - \boldsymbol{\Phi} \boldsymbol{w}\right) = \left(\boldsymbol{K} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{y} \ ,$$

where $\boldsymbol{K} = \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}}$. The optimal solution for this dual representation therefore requires solving an m-dimensional system of linear equations, as opposed to an n-dimensional system for the primal formulation in Equation (2.5). Hence, this alternative formulation is computationally advantageous for training if $m < n$. More importantly, however, is that the training data only occurs within inner products. Note that the matrix $\boldsymbol{K} = \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}}$ can be described component-wise as $\boldsymbol{K}_{ij} = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$ and that the prediction function can be written as

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle = \left\langle \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha}, \phi(\boldsymbol{x}) \right\rangle = \sum_{i=1}^{m} \alpha_i \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}) \right\rangle \ .$$

This leads to the key observation that only the inner product in $\mathcal{H}$ is required to compute the RLS solution. The kernel trick "exploits" this observation by directly specifying a kernel function

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle_{\mathcal{H}} \ , \tag{2.8}$$

thus avoiding explicit mapping of the input samples into feature space. As a consequence, the kernel is often considered as the object of primary interest, and its corresponding feature space is only of secondary importance.

2.3.1 Admissible Kernels

A relevant question that arises is which functions $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ actually correspond to an inner product in a feature space $\mathcal{H}$. The answer is given by Mercer's condition (Mercer, 1909), which in short[2] requires that

$$\int_{\mathcal{X}\times\mathcal{X}} k(\boldsymbol{x}_i, \boldsymbol{x}_j) f(\boldsymbol{x}_i) f(\boldsymbol{x}_j) \,\mathrm{d}\boldsymbol{x}_i \,\mathrm{d}\boldsymbol{x}_j \geq 0 \qquad \text{for all } f \in L_2(\mathcal{X}) \ .$$

This condition is easier understood at the hand of a dataset of finite size. In this case the integral reduces to a summation and we can write for any $\boldsymbol{v} \in \mathbb{R}^m$

$$\begin{aligned}
\boldsymbol{v}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{v} &= \sum_{i,j=1}^{m} v_i v_j \boldsymbol{K}_{i,j} = \sum_{i,j=1}^{m} v_i v_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \\
&= \sum_{i,j=1}^{m} v_i v_j \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle \\
&= \left\langle \sum_{i=1}^{m} v_i \phi(\boldsymbol{x}_i), \sum_{j=1}^{m} v_j \phi(\boldsymbol{x}_j) \right\rangle \\
&= \left\| \sum_{i=1}^{m} v_i \phi(\boldsymbol{x}_i) \right\|^2 \geq 0 \ ,
\end{aligned} \tag{2.9}$$

from which we can conclude that the symmetric kernel matrix

$$\boldsymbol{K} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^{m} \tag{2.10}$$

is positive semidefinite (denoted as $\boldsymbol{K} \succeq 0$). Consequently, the kernel matrix has non-negative eigenvalues and a unique Cholesky factorization. Furthermore, the kernel function k is positive semidefinite if it produces a positive semidefinite kernel matrix for any finite set $\boldsymbol{x}_i \in \mathcal{X}$ for $1 \leq i \leq m$; this property will be used later on in Chapter 4. Kernels that satisfy this condition will subsequently be referred to as admissible kernels, or simply kernels.

[2] A detailed and mathematically rigorous treatment of kernels is outside of the scope of this thesis; the interested reader is therefore referred to one of many excellent textbooks on the subject (e.g., Cristianini and Shawe-Taylor, 2000; Schölkopf and Smola, 2001; Shawe-Taylor and Cristianini, 2004; Steinwart et al., 2009).

2.3.2 Reproducing Kernel Hilbert Space

Given an admissible kernel k, we can construct a corresponding Hilbert space $\mathcal{H}$. Recall that the weight vector $\boldsymbol{w}$ is an element in the feature space. By combining Equation (2.7) and Equation (2.8) we can thus construct a feature space that contains $\boldsymbol{w}$ as

$$\mathcal{H} = \left\{ \sum_{i=1}^{m} \alpha_i k(\boldsymbol{x}_i, \cdot) : m \in \mathbb{N}, \boldsymbol{x}_i \in \mathcal{X}, \alpha_i \in \mathbb{R}, 1 \leq i \leq m \right\} \ .$$

Note that this feature space is a set of points that are in fact functions, for which the dot indicates the position of the argument of the function. Let $f, g \in \mathcal{H}$, such that

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) \quad \text{and} \quad g(\boldsymbol{z}) = \sum_{j=1}^{o} \beta_i k(\boldsymbol{z}_j, \boldsymbol{x}) = \quad ,$$

then the inner product in this space is defined as

$$\langle f, g \rangle = \sum_{i=1}^{m} \sum_{j=1}^{o} \alpha_i \beta_i k(\boldsymbol{x}_i, \boldsymbol{z}_j) = \sum_{i=1}^{m} \alpha_i g(\boldsymbol{x}_i) = \sum_{j=1}^{o} \beta_j f(\boldsymbol{z}_j) \ .$$

This demonstrates that $\langle f, g \rangle$ is real-valued, symmetric and bilinear. The final requirements to satisfy the properties of an inner product is that

$$\langle f, f \rangle \geq 0 \qquad \text{for all } f \in \mathcal{H} \ ,$$

which can easily be proved since

$$\langle f, f \rangle = \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\alpha}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{\alpha} \geq 0 \qquad \text{using Equation (2.9)} \ .$$

The feature space $\mathcal{H}_K$ constructed in this manner is also referred to as the Reproducing Kernel Hilbert Space (RKHS) corresponding to the kernel function k (Wahba, 1990). This terminology is due to the reproducing property

$$\langle f, k(\boldsymbol{x}, \cdot) \rangle = \sum_{i=1}^{m} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) = f(\boldsymbol{x}) \ .$$

This property is an important characteristic of the kernel function k, as it guarantees that the kernel is positive semidefinite. The interpretation of the feature mapping corresponding to an RKHS is not entirely intuitive, as each input sample is turned into a function on the domain $\mathcal{X}$. In other words, each input sample is represented in the RKHS by its similarity with all other points in the input space $\mathcal{X}$. Regardless, it is important to realize that although there is a single RKHS

corresponding to a kernel k, there may be other Hilbert and Euclidean spaces for which the kernel k describes the inner product (e.g., see Section 4.1 for the polynomial kernel).

2.4 Kernel Regularized Least Squares

For completeness, let us fully iterate over the RLS algorithm when using kernel functions. In this case, the prediction function is written as

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i) = \langle \boldsymbol{\alpha}, \boldsymbol{k} \rangle \quad , \tag{2.11}$$

where $\boldsymbol{k} = [k(\boldsymbol{x}, \boldsymbol{x}_1), \cdots, k(\boldsymbol{x}, \boldsymbol{x}_m)]^{\mathrm{T}}$. The linear combination of kernel evaluations in Equation (2.11) will be referred to as the kernel expansion. The representer theorem, originally due to Wahba (1990) and subsequently generalized by Schölkopf et al. (2001), guarantees that the optimal solution in the possibly infinite dimensional RKHS can be expressed as a finite dimensional kernel expansion. The optimal coefficients are then given by

$$\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y} \quad , \tag{2.12}$$

where the kernel matrix $\boldsymbol{K}$ is defined as in Equation (2.10). This algorithm is known under multiple names in the machine learning community. It was originally proposed by Saunders et al. (1998) as Kernel Ridge Regression (KRR), though an identical algorithm has been studied in the RN framework as well (see Evgeniou et al., 2000, and the references therein). The algorithm has also been used for classification problems under the names RLS Classification (Rifkin et al., 2003) and Proximal SVM (Fung and Mangasarian, 2001), which is an interesting choice as it uses the squared loss function. Albeit controversial, the algorithm defined by Equation (2.11) and Equation (2.12) will subsequently be referred to Kernel Regularized Least Squares (KRLS) following the earlier terminology of linear RLS. However, note that linear RLS is a special case of KRLS, where the kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is simply the inner product in the original input space $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$.

2.4.1 Related Methods

The kernel trick can be applied to any algorithm for which the solution can be described in terms of inner products. Departing from KRLS, several related KMs can be obtained by relatively minor modifications to the optimization objective. For instance, the extension with an additional bias term b in the predictions function, such that

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i) + b \quad ,$$

is known as a Least Squares Support Vector Machine (LSSVM)[3] (Suykens et al., 2002b). The optimal solution is obtained by solving a system of $m + 1$ linear equations, where the bias term accounts for the additional row and column. This LSSVM formulation is not equal to KRLS with an additional constant term in the input dimension. In the latter case the bias term is regularized (and thus driven to zero), while in the former case it is not.

The popular SVM is also closely related to KRLS and can be obtained by changing the loss function in Equation (2.2). In case of regression, the squared loss function $\mathcal{L}(y, f(\boldsymbol{x})) = (y - f(\boldsymbol{x}))^2$ is replaced with the ϵ-insensitive loss

$$\mathcal{L}(y, f(\boldsymbol{x})) = |y - f(\boldsymbol{x})|_\epsilon = \max\left(|y - f(\boldsymbol{x})| - \epsilon, 0\right) \ .$$

The interpretation of this loss function is that small deviations (i.e., up to ϵ) are tolerated, while larger deviations are penalized in linear fashion. A consequence is that the kernel expansion becomes sparse, as only samples for which the error is larger than ϵ will result in non-zero coefficients α. These samples are referred to as the support vectors and the algorithm is therefore known as ϵ-insensitive Support Vector Regression (ϵ-SVR) (Drucker et al., 1996; Smola and Schölkopf, 2004). In contrast, the squared loss used in KRLS and LSSVM is almost surely non-zero for all samples and the kernel expansion will therefore include all training samples.

A similar sparsity property can be found in SVM for classification problems (Boser et al., 1992; Burges, 1998), which uses the hinge loss

$$\mathcal{L}(y, f(\boldsymbol{x})) = |1 - yf(\boldsymbol{x})|_+ = \max\left(1 - yf(\boldsymbol{x}), 0\right) \ .$$

In this case, incorrect predictions as well as predictions that are "marginally" correct are penalized, as the loss is non-zero if $yf(\boldsymbol{x}) < 1$ (as opposed to $yf(\boldsymbol{x}) < 0$). This particular loss function forces SVM to maximize a margin between both classes. Analogous to the regression case, correct predictions outside of this margin will have zero coefficients and can therefore safely be omitted from the kernel expansion.

A different approach to enforce sparsity is by replacing the L_2 regularizer with the L_1 regularizer in Equation (2.3). The latter is defined as $\|f\|_1$ and linearly penalizes non-zero components in the weight vector $\boldsymbol{w}$. This induces sparsity in the solution and effectively constitutes a form of feature selection. Combined with the squared loss function, this method is known as least absolute shrinkage and selection operator (LASSO) regression, proposed for the linear case by Tibshirani (1996) and subsequently generalized to the kernel framework by Roth (2004). This approach to sparsity may lead to problems if there are many correlated input features. In these cases, it may be beneficial to combine the properties of L_1 and L_2 regularization. Examples thereof include bridge regression (Fu, 1998), which uses the L_p norm as regularizer. This norm preserves sparsity if p is

[3] This could be considered a misnomer, as the algorithm shares more similarities with KRLS (or Kernel Ridge Regression) than with Support Vector Machines.

chosen only slightly larger than 1. Elastic nets, on the other hand, use a linear combination of L_1 and L_2 regularization (Zou and Hastie, 2005; Mol et al., 2009). The contribution of each term is given by a predefined parameter β, such that the regularizes is written as $(1-\beta)\left\|f\right\|_1 + \beta\left\|f\right\|_2^2$. The extreme cases $\beta = 0$ or $\beta = 1$ correspond to standard L_1 and L_2 regularization, respectively.

2.5 Gaussian Process Regression

Most statistical models can be considered both within the frequentist as well as the Bayesian approach. The KMs described thus far were formulated within the former paradigm. GPR, on the other hand, is a closely related KM that approaches the learning problem from a Bayesian perspective (Rasmussen and Williams, 2005). The primary conceptual difference between frequentist and Bayesian learning methods is that in the former paradigm one searches for a single unknown prediction function f that underlies the observations. In contrast, Bayesian methods define a probability distribution over all prediction functions in a given function space $\mathcal{F}$. An initial prior belief of this distribution is formulated by the practitioner, which is subsequently refined based on the evidence (i.e., the observed samples) to obtain a posterior belief.

Let us investigate the Bayesian approach at the hand of the generalized linear model (Bishop, 2006; Rasmussen and Williams, 2005). In this model, we assume that the outputs can be described by a linear model in the inputs[4], where the observed outputs are possibly corrupted by noise. We can thus write

$$y = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \epsilon \ ,$$

where it is assumed that the additive noise ϵ is i.i.d. and follows a Gaussian distribution with zero mean and variance σ_n^2, such that $\epsilon \sim \mathcal{N}\big(0, \sigma_n^2\big)$. Furthermore, it is assumed that the outputs $\boldsymbol{y}$ are zero-mean[5]. The likelihood of the observed outputs given the input samples and the weight vector can then be written as

$$p(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{X}) = \mathcal{N}\big(\boldsymbol{y}|\boldsymbol{X}\boldsymbol{y}, \sigma_n^2 \boldsymbol{I}\big) \ .$$

Furthermore, let us define a prior over the weight vector $\boldsymbol{w}$ as a multi-variate Gaussian with zero mean and covariance matrix $\boldsymbol{\Sigma}_p$, or

$$\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma_p}) \ .$$

This prior expresses our initial belief about the parameters before observing any samples. In the

[4] All equations in the treatment of the linear model can trivially be extended with a finite dimensional feature map by replacing $\boldsymbol{x}$ with $\phi(\boldsymbol{x})$.

[5] This can be ensured by subtracting the sample mean or, alternatively, by appending an additional constant bias term in the input representation.

Bayesian framework, inference is based on the posterior distribution over the weight vector $\boldsymbol{w}$, given the observations $\boldsymbol{X}$ and $\boldsymbol{y}$. Applying Bayes' rule on the prior and likelihood, we obtain the posterior (see Rasmussen and Williams, 2005)

$$\begin{aligned} p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{y}) &= \frac{p(\boldsymbol{y}|\boldsymbol{w},\boldsymbol{X})p(\boldsymbol{w})}{p(\boldsymbol{y}|\boldsymbol{X})} \\ &\sim \mathcal{N}\left(\boldsymbol{w}\middle|\bar{\boldsymbol{w}} = \frac{1}{\sigma_n^2}\boldsymbol{A}^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}, \boldsymbol{A}^{-1}\right) \ , \end{aligned} \tag{2.13}$$

where the posterior covariance matrix

$$\boldsymbol{A} = \frac{1}{\sigma_n^2}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \boldsymbol{\Sigma}_p^{-1} \ .$$

Note that we conveniently used the fact that the normalizing constant $p(\boldsymbol{y}|\boldsymbol{X})$ (i.e., the marginal likelihood) is independent of $\boldsymbol{w}$. Moreover, we observe that the mean prediction of this Bayesian linear model is identical to the optimal RLS solution in Equation (2.5), provided that $\boldsymbol{\Sigma}_p = \frac{\lambda}{\sigma_n^2}\boldsymbol{I}$. This is in fact no coincidence, since minimizing the squared loss in RLS results in a maximum likelihood estimate, which coincides with the mean of the Gaussian maximum a posteriori estimate of the Bayesian model. In other words, regularization in RLS is functionally equivalent to an (implicit) prior on the weight vector $\boldsymbol{w}$.

Model predictions are obtained by integrating over all possible $\boldsymbol{w}$ with respect to their posterior distribution in Equation (2.13). In other words, rather than selecting one weight vector that describes the model (i.e., frequentist), we consider a distribution over all possible linear models. The result is a predictive distribution, which in the linear model is given by

$$\begin{aligned} p(y|\boldsymbol{x},\boldsymbol{X},\boldsymbol{y}) &= \int p(y|\boldsymbol{x},\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{X},\boldsymbol{y})\,\mathrm{d}\boldsymbol{w} \\ &= \mathcal{N}\left(y\middle|\frac{1}{\sigma_n^2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y},\ \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{x}\right) \ , \end{aligned}$$

where $(\boldsymbol{x}, y)$ is an individual test sample, as opposed to the training data $(\boldsymbol{X}, \boldsymbol{y})$. Not surprisingly, the mean of this predictive distribution coincides with the RLS prediction function.

2.5.1 Extension with Kernels

Generalization of the Bayesian linear model to the kernel framework follows similar lines as those described for KRLS in Section 2.3. There are two equivalent interpretations, namely the weight space view and the function space view (Rasmussen and Williams, 2005). The latter interpretation will be explained here in more detail. Recall from the formulation of RKHSs (cf. Section 2.3.2) that we replaced weight vector $\boldsymbol{w}$ with a function, such that the feature space is in fact a function space. In the Bayesian interpretation, we thus perform inference directly in this function space.

Using the fact that the weight prior is a zero-mean Gaussian with covariance matrix $\mathbf{\Sigma}_p$, the mean and variance of the prior in this function space are thus given by

$$\begin{aligned}\mathbb{E}[f(\cdot)] &= \mathbb{E}[\langle \boldsymbol{w}, \phi(\cdot)\rangle] \\ &= \mathbb{E}[\boldsymbol{w}]^{\mathrm{T}}\phi(\cdot) \\ &= 0\end{aligned}$$

and

$$\begin{aligned}\mathbb{E}\Big[f(\cdot_1)^{\mathrm{T}} f(\cdot_2)\Big] &= \mathbb{E}\Big[\langle \boldsymbol{w}, \phi(\cdot_1)\rangle^{\mathrm{T}} \langle \boldsymbol{w}, \phi(\cdot_2)\rangle\Big] \\ &= \phi(\cdot_1)^{\mathrm{T}}\mathbb{E}\big[\boldsymbol{w}\boldsymbol{w}^{\mathrm{T}}\big]\phi(\cdot_2) \\ &= \phi(\cdot_1)^{\mathrm{T}}\mathbf{\Sigma}_p\phi(\cdot_2) \ ,\end{aligned} \tag{2.14}$$

where ϕ is the non-linear mapping and the dot indicates the position of the parameter. In other words, the function values $f(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_j)$ are jointly Gaussian with mean 0 and covariance $\phi(\boldsymbol{x}_i)^{\mathrm{T}}\mathbf{\Sigma}_p\phi(\boldsymbol{x}_j)$ for any $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. Noting that the covariance $\mathbf{\Sigma}_p$ is necessarily positive semidefinite, Equation (2.14) can be described in the form of the kernel function

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left\langle \mathbf{\Sigma}_p^{\frac{1}{2}}\phi(\boldsymbol{x}_i), \mathbf{\Sigma}_p^{\frac{1}{2}}\phi(\boldsymbol{x}_j)\right\rangle \ , \tag{2.15}$$

where $\mathbf{\Sigma}_p^{\frac{1}{2}}$ is the square root of $\mathbf{\Sigma}_p$. It is therefore common to interpret the kernel function as a covariance function in the GPR framework. The distribution in this function space is a Gaussian Process[6], since it is jointly Gaussian for any finite number of inputs. Gaussian Processes can be described entirely by its mean and covariance; in this case, the prior on the functions can therefore be written as

$$f(\boldsymbol{x}) \sim \mathcal{GP}(\mathbf{0}, k(\boldsymbol{x}_i, \boldsymbol{x}_j)) \ .$$

The specification of the kernel function thus defines a distribution over functions, prior to observing any samples. Although important for understanding the approach, it is usually of limited use to sample functions from this prior $\mathcal{GP}$. Instead, we are interested in the posterior distribution that incorporates knowledge from the training data. Given the m training samples $(\boldsymbol{X}, \boldsymbol{y})$ and an individual test sample $\boldsymbol{x}$, we can write the joint distribution as

$$\begin{bmatrix} f(\boldsymbol{X}) \\ f(\boldsymbol{x}) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \boldsymbol{K} + \sigma_n^2\boldsymbol{I} & \boldsymbol{k} \\ \boldsymbol{k}^{\mathrm{T}} & k(\boldsymbol{x}, \boldsymbol{x}) \end{bmatrix}\right) \ ,$$

[6] A Gaussian Process $\mathcal{GP}$ is a collection of random variables, any finite number of which have a joint Gaussian distribution.

where $\boldsymbol{K} = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j=1}^{m}$ and $\boldsymbol{k} = [k(\boldsymbol{x}, \boldsymbol{x}_1), \cdots, k(\boldsymbol{x}, \boldsymbol{x}_m)]^{\mathrm{T}}$. Furthermore, $f(\boldsymbol{X})$ should be considered as latent variables, since only the noise corrupted outputs $\boldsymbol{y}$ are observed. This is accounted for by adding the noise variance σ_n^2 to the diagonal of the kernel matrix. The posterior distribution can be obtained by conditioning the joint prior distribution on the observations. An intuitive interpretation is that we restrict the joint distribution to only those functions that are in agreement with the observations. After conditioning, the posterior distribution becomes (Rasmussen and Williams, 2005)

$$f(\boldsymbol{x})|\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{y} \sim \mathcal{N}\Big(\boldsymbol{k}^{\mathrm{T}} \left(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y},\ k(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{k}^{\mathrm{T}} \left(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}\right)^{-1} \boldsymbol{k} + \sigma_n^2\Big) \ . \quad (2.16)$$

The additional noise variance σ_n^2 in the predictive variance is based on the reasonable assumption that test samples (as well as training samples) are corrupted by noise. This additional term may be omitted when test samples are known to be free of noise (e.g., synthetic datasets). Comparing Equation (2.16) and Equation (2.12), we observe that the predictive mean of GPR is identical to KRLS. This demonstrates again that regularization in the frequentist paradigm is equal to the specification of a prior in the Bayesian paradigm. Regardless, there are two primary differences between KRLS and GPR. First, GPR produces a predictive distribution rather than a point prediction, and this distribution can be used to quantifies the uncertainty in the predictions. The second advantage is that the Bayesian approach in GPR allows for principled ways of model selection, as will be explained in more detail in the following section.

2.6 Hyperparameter Optimization

Thus far, we have seen how a number of learning methods are formulated and how these can be extended to non-linear problems with kernel functions. An important practical concern has conveniently been ignored so far, namely, how to choose a kernel function and the hyperparameters (e.g., λ in RLS or σ_n in GPR). The kernel function can be interpreted as a similarity measure of the samples and its appropriate choice depends therefore strongly on the problem domain. Similarly, hyperparameters are used to incorporate prior beliefs regarding what constitutes an appropriate solution. For instance, the λ parameter in RLS regulates the desired level of smoothness of the solution and its misspecification may lead to severe underfitting or overfitting. Consequently, the generalization performance depends critically on our choice of kernel function and hyperparameters.

2.6.1 Selecting the Kernel

A key property of KMs is that linear algorithms are performed in an implicit feature space. The underlying assumption is that the problem is actually linear in the feature space induced by the kernel. For this assumption to hold, it is necessary to select an appropriate kernel for the learning

problem under consideration. Although this may seem as a daunting task, there are particular kernels that have been shown to perform well on a wide variety of practical learning problems. The two most commonly used families of kernels are polynomial and RBF kernels. The former, (inhomogeneous) polynomial kernels, are of the form

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = (R + \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle)^d \quad , \tag{2.17}$$

for $R \geq 0$ and degree $d \in \mathbb{N}_+$. The corresponding feature space is finite dimensional and consists of all monomials up to the d-th degree, weighted according to a function of the constant R (cf. Equation (4.1)). The special case $R = 0$ and $d = 1$ is referred to as the linear kernel and corresponds to the inner product of the samples in the original input space. Clearly, increasing the degree d also increases the capacity of the underlying learning method and may therefore result in overfitting.

The class of RBF kernels is probably the most widely used in machine learning literature. Kernels belonging to this family take the Gaussian form

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2} \quad , \tag{2.18}$$

where $\gamma > 0$ controls the bandwidth and thus the sensitivity of the similarity measure. A large value of γ results in a diagonal kernel matrix and causes therefore severe overfitting. Conversely, small values of γ reduce the capacity of the underlying learning method, which will therefore not be able to fit the data. It can be shown that the RBF kernel corresponds to an infinite dimensional feature space (Schölkopf and Smola, 2001). Both polynomial and RBF types of kernels yield universal approximators; any training set can be learned without error (assuming noise-less observations), given sufficiently large degree or sufficiently small kernel width. Note, however, that this should not be confused with these kernels being universally optimal.

The RBF kernel in Equation (2.18) is isotropic and assigns an equal weight to all n input dimensions. In practical settings, it is therefore often necessary to scale the input dimensions to reflect the relative importance of the input features. This preprocessing step can be integrated in the definition of the kernel to form an anisotropic RBF kernel, such that

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x}_j)^{\mathrm{T}} \boldsymbol{M} (\boldsymbol{x}_i - \boldsymbol{x}_j)} \quad , \tag{2.19}$$

where $\boldsymbol{M}$ is typically an $n \times n$ diagonal matrix with $\boldsymbol{M}_{ii} = \ell_i^{-2}$ and $\ell_i > 0$ for $1 \leq i \leq n$. The vector of characteristic length scales $\boldsymbol{\ell}$ describe both the overall bandwidth and the relative importance of each input dimension, and may even be used to cancel out irrelevant dimensions (i.e., $\ell_i \to \infty$). Optimization of these length scales therefore constitutes a form of Automatic Relevance Detection (ARD) (MacKay, 1995). Unfortunately, a consequence is that the number of kernel parameters increases from 1 to n. Within the GPR framework this kernel is also known as

the Asymmetric Squared Exponential (ASE) covariance function and written as[7]

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_f^2 \, e^{-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{x}_j)^{\mathrm{T}} \boldsymbol{M} (\boldsymbol{x}_i - \boldsymbol{x}_j)} \quad . \tag{2.20}$$

The additional factor σ_f denotes the signal variance (or amplitude), such that the signal to noise ratio is given by $\frac{\sigma_f}{\sigma_n}$. This ratio corresponds is functionally identical to the regularization parameter λ in RLS. Considering both noise terms separately, however, is relevant in GPR for the variance estimation.

An interesting corollary of the characterization of kernel functions in Section 2.3.1 is that certain operations can be shown to preserve admissibility (i.e., positive semidefiniteness). For instance, this holds for linear combinations as well as products of kernels (Shawe-Taylor and Cristianini, 2004). In other words, these operations manipulate and combine simple kernels to obtain more complex and (hopefully) better fitting composite kernels. Recently, this had led to an increasing interest in so-called Multi Kernel Learning (MKL) (Lanckriet et al., 2004; Sonnenburg et al., 2006; Gijsberts et al., 2010a; Orabona et al., 2010). Moreover, kernel functions need not be restricted to vectorial inputs and special kernels have been designed for diverse objects and structures (Shawe-Taylor and Cristianini, 2004). Examples thereof include string kernels for text classification (Joachims, 1998; Lodhi et al., 2002) or DNA barcoding (Sonnenburg et al., 2007), graph kernels for drug discovery or web data mining (Vishwanathan et al., 2010), and even kernels modeling the response of the visual cortex (Smale et al., 2010).

2.6.2 Hyperparameter Optimization

Complementary to the kernel function, hyperparameters pose an important mechanism to include prior knowledge of the problem domain in the learning algorithm. Examples of these hyperparameters include the regularization parameter λ and the kernel parameters (e.g., the bandwidth γ). Optimization of these parameters is commonly known as model selection or simply hyperparameter optimization. Unfortunately, while most learning algorithms are specifically formulated to result in a convex optimization problem, the hyperparameter optimization problem is typically non-convex and characterized by multiple local minima.

The most common approach to hyperparameter optimization is to empirically evaluate the performance of a predefined number of configurations using cross validation. In cross validation, an independent subset of the observations is used to estimate the expected risk of a model. A particularly common combination is to discretize hyperparameters on a grid and to estimate the performance using 10-fold cross validation. However, this strategy is likely to be suboptimal, since the discretization is often coarse due to computational limitations. Furthermore, optimizing the parameters of a larger number of hyperparameters (e.g., an anisotropic RBF kernel with n

[7] Commonly in GPR literature, the noise variance is described as part of the kernel using an additional $\sigma_n^2 \delta_{ij}$ term, with δ_{ij} being Kronecker's delta. It is omitted here for coherency with the overall treatment of KMs.

parameters) is computationally infeasible, since grid search scales exponentially with the number of hyperparameters. Several approaches have been proposed to alleviate this problem to a certain extent. These include gradient based optimization schemes (Cristianini et al., 1999; Chapelle et al., 2002; Keerthi et al., 2007), which require the kernel function and performance validation measure to be differentiable. More generic is the application of optimization methods that do not require gradients, such as Pattern Search (Momma and Bennett, 2002), Evolutionary Algorithms (Friedrichs and Igel, 2004; Lessmann et al., 2006; Gijsberts et al., 2010a), or Particle Swarm Optimization (Escalante et al., 2009).

2.6.3 Marginal Likelihood Optimization

The approaches explained thus far attempt to minimize an estimate of the generalization error. Bayesian methods offer an alternative approach, as these allow to describe the relationship between the observations in the hyperparameters in probabilistic terms. Following a full Bayesian treatment, one should place a prior on the hyperparameters and integrate them out, or

$$p(\boldsymbol{y}|\boldsymbol{X}) = \int p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})p(\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{\theta} \ , \tag{2.21}$$

where $\boldsymbol{\theta}$ is a vector containing the hyperparameters and $p(\boldsymbol{\theta})$ is appropriately referred to as a hyperprior. Evaluation of the integral in Equation (2.21), however, is often found to be intractable in practice (Bishop, 2006; Rasmussen and Williams, 2005). As an approximation, we may discard the hyperprior and instead optimize the marginal likelihood

$$p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) = \int p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{w})p(\boldsymbol{w}|\boldsymbol{\theta})\,\mathrm{d}\boldsymbol{w} \ ,$$

also known as the evidence. This approximation for second level inference[8] is known as evidence approximation (MacKay, 1992), empirical Bayes, or type II maximum likelihood (see Bishop, 2006, and references therein). For regression problems, this likelihood is given by

$$p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) = \mathcal{N}\big(\boldsymbol{y}|\boldsymbol{0},\boldsymbol{X}\boldsymbol{\Sigma}_p\boldsymbol{X}^{\mathrm{T}} + \sigma_n^2\boldsymbol{I}\big) \ ,$$

where we recognize that $\boldsymbol{X}\boldsymbol{\Sigma}_p\boldsymbol{X}^{\mathrm{T}}$ is in fact the kernel matrix $\boldsymbol{K}$ for the linear kernel (cf. Equation (2.15)). It follows that we can equivalently write

$$p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) = \mathcal{N}\big(\boldsymbol{y}|\boldsymbol{0},\boldsymbol{K} + \sigma_n^2\boldsymbol{I}\big) \ .$$

Model selection in this Bayesian framework therefore equates to finding the hyperparameter configuration that maximizes the marginal likelihood. For practical reasons, however, it is more con-

[8] In this naming scheme, optimization of the weight vector $\boldsymbol{w}$ or kernel coefficients $\boldsymbol{\alpha}$ is referred to as first level inference.

venient to minimize the negative log marginal likelihood. The optimization problem thus becomes

$$\arg\min_{\boldsymbol{\theta}} \ \frac{1}{2}\tilde{\boldsymbol{K}}^{-1}\boldsymbol{y} + \frac{1}{2}\ln\det\tilde{\boldsymbol{K}} + \frac{m}{2}\ln 2\pi \ , \tag{2.22}$$

where $\tilde{\boldsymbol{K}} = \left(\boldsymbol{K} + \sigma_n^2\boldsymbol{I}\right)$. The corresponding partial derivatives with respect to each hyperparameter θ is given by

$$-\frac{\partial}{\partial\theta}\ln p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) = -\frac{1}{2}\operatorname{tr}\left(\left(\boldsymbol{\alpha}\boldsymbol{\alpha}^{\mathrm{T}} - \tilde{\boldsymbol{K}}^{-1}\right)\frac{\partial\tilde{\boldsymbol{K}}}{\partial\theta}\right) \ ,$$

where $\boldsymbol{\alpha} = \tilde{\boldsymbol{K}}^{-1}\boldsymbol{y}$. The resulting minimization problem can be optimized using standard gradient based optimization routines, although the optimization problem is not convex and different starting points may lead to different local optima. Nonetheless, a practical advantage of evidence based hyperparameter optimization is that is scales well with the number of hyperparameters, as opposed to grid search. Furthermore, the marginal likelihood automatically incorporates a tradeoff between model fit and model complexity (Rasmussen and Williams, 2005; MacKay, 1999). This can be observed in Equation (2.22), where the first term measures data fit and the second term penalizes model complexity. Consequently, the risk of overfitting (or underfitting) is relatively low.

2.7 Example: Learning Inverse Dynamics

The popularity of KMs can be explained by their solid theoretical foundation and excellent performance on many practical problems. These include applications in fields as diverse as computer vision (e.g., Comaniciu et al., 2003; Lampert, 2009), bio-informatics (e.g., Saigo et al., 2004; Schölkopf et al., 2004), text categorization (e.g., Joachims, 1998; Silva and Ribeiro, 2009), and robotics (e.g., Rani et al., 2006; Nguyen-Tuong et al., 2008b; Gijsberts et al., 2010b). In this section, a comparison with other methods on the problem of modeling manipulator dynamics (i.e., a regression problem) demonstrates that kernel-based methods can indeed result in significant performance gains.

As explained previously in Section 1.1.1, modeling inverse dynamics is relevant for a number of tasks, such as compliant robot control, zero gravity control, and contact detection. The dynamics can often be described using an analytical model of the Rigid Body Dynamics (RBD), assuming that both kinematic and dynamic parameters are known or identifiable with high accuracy. However, the accuracy of these analytical models is limited in practice by potential non-linear effects that are not explicitly taking into account in the model. Supervised machine learning approaches constitute a viable alternative to analytical modeling, as these attempt to learn all dependencies that are present in the observations. Fumagalli et al. compare an analytical RBD model with two learning methods, namely LSSVM and a feedforward ANN (Haykin, 1994; Hagan and Menhaj,

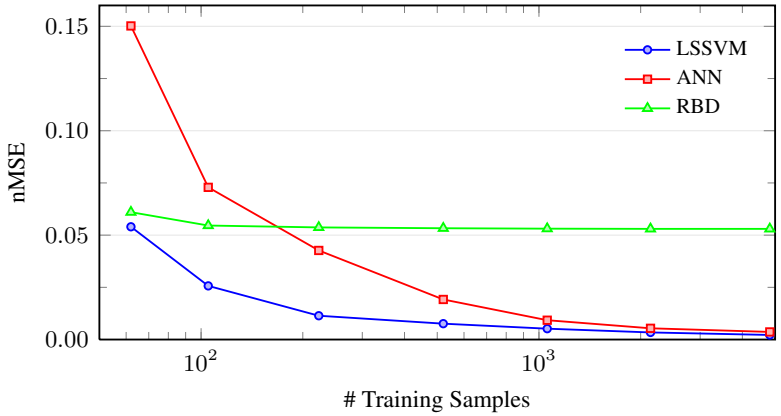


Figure 2.2: Test error with respect to the number of training samples for LSSVM, ANN, and RBD when predicting the internal dynamics of James.

1994), for the estimation of internal forces and torques in the arm of the humanoid robot James[9] (cf. Section 1.1).

A large dataset was collected from the robot and subsequently randomly subsampled in training, validation, and test sets. Figure 2.2 demonstrates the test error when varying the number of training samples. As can be seen clearly, the analytical model does not benefit from an increased number of samples. The explanation is that this model uses observations exclusively for parameter identification. The data-driven learning methods, on the other hand, improve drastically when more observations are available. LSSVM in particular shows superior performance and outperforms the analytical model even when trained on less than 100 samples. The ANN method requires much more training samples to attain performance similar to LSSVM, although both methods eventually converge to similar performance given a sufficient amount of training samples (cf. 5000 training samples).

2.8 Machine Learning for Robotics

The successful application of LSSVM in the previous section marks the benefits of learning approaches with respect to analytical models. This is an interesting result, as one might expect that analytical models describe the underlying model perfectly and should therefore always be superior. However, a problem that should not be underestimated is that these models are idealized. For example, let us consider the RBD model from the previous section, which describes the forces and torques given a manipulator configuration described in terms of joint angles, positions, and velocities. This analytical model assumes the following:

- absence of friction, backlash, and deformability;

[9] Section 5.3 contains a more detailed description of this learning problem.

- perfect identification of kinematic and dynamic parameters;
- noise-free and perfectly synchronized sensor and encoder measurements.

Unfortunately, these assumptions are often found to be overly optimistic in modern complex robots. Although the individual impact of violating any of the assumptions may be limited, the combination of violating multiple assumptions will result in significant deviations between the idealized and observed response. Supervised learning methods are much less susceptible to this type of errors, as they construct a model based on actual observations of the underlying phenomenon rather than an idealized abstraction thereof.

Learning methods are usually easy to employ on practical problems and only require limited prior knowledge as compared to analytical models. Nonetheless, the methods as described in this chapter are not appropriate for the robotics paradigm as outlined in Chapter 1. In that section, the approach to robot learning was specified as incremental (or sequential) in nature and possibly open-ended. Conversely, the KMs in this chapter are so-called batch learning methods, which train on the entire set of observations in a single pass. The trained model can subsequently be used to predict the output values for unlabeled observations. In other words, training and testing are therefore distinct phases in the batch learning paradigm. Incorporating additional observations to a trained model requires retraining a model on both previous and new observations, and is therefore computationally prohibitive.

Moreover, a consequence of the kernel trick is that the solution is described as a linear combination of kernel evaluations with respect to (a subset of) the training data (i.e., the kernel expansion). The size of the solution is thus directly related to the number of training observations. Even for SVM, which is explicitly formulated to produce sparse solutions, it can be shown that the size of the kernel expansion is bounded from below by a linear function in the size of the training set (Steinwart, 2003). Increasing prediction times as more training data becomes available effectively rules out the possibility of open-ended learning, which requires both the temporal and spatial complexity of each learning step to be constant[10] (i.e., $\mathcal{O}(1)$).

A last remark regards typical assumptions in learning theory on the observations, such as i.i.d. or Gaussianity assumptions. In effect, also these assumptions are often violated in realistic applications and this is particularly evident in domains that are grounded in the physical world. Successive observations in many robotic problems (e.g., the earlier dynamics problem) are inherently dependent and possibly non-stationary. The problem of generalization in this difficult non-i.i.d. setting is unfortunately still an open research question in machine learning. Nonetheless, there are recent theoretical results that prove that generalization is possible if the i.i.d. assumption is weakened (e.g., Pan and Xiao, 2009). Not surprisingly, these results indicate that dependent observations contain less information than i.i.d. observations. Intuitively, it seems therefore a reasonable approach to (1) use the maximum amount of training data to deal with dependence and (2) use the

[10] If not, the problem will become computationally intractable at some point.

most recent data available to react to non-stationarity of the distribution[11]. In other words, this argument is in favor of incremental learning approaches that incorporate all available information and adapt continuously to react to potential changes.

[11] It is implicitly assumed that distribution changes are either gradual or infrequent.

3 Related Work

The previous chapter identified two primary obstacles for applying standard kernel methods in the considered robotics paradigm, namely (1) a linear increase of the prediction time with respect to the number of training samples and (2) the inability to incrementally update a solution as new observations become available. Multiple approaches have been proposed that target either one or both of these problems. Regarding the linear increase of the kernel expansion, a common strategy is to replace the kernel matrix with a reduced rank approximation. As a result, the kernel expansion is described in terms of a sparse subset of the training samples.

Incrementally updating a solution on arrival of new observations is related to reducing the complexity of kernel methods (KMs), as both techniques are typically required to scale these methods to problems that are too large to solve in a single batch. Another justification for incremental updating is given by problems in which the data arrives sequentially by nature, which are common in (developmental) robotics. Although the term "incremental" is preferred in this thesis, similar concepts are known under various terms in other fields, such as stream learning (in data mining, e.g., Gaber et al., 2005), recursive updates (in adaptive filtering, e.g., Sayed, 2008), or simply online learning (e.g., Kivinen et al., 2004).

In this chapter, an overview is given of related work in overcoming these two obstacles. Various techniques to limit the kernel expansion are explained in Section 3.1, after which approaches to incremental updates of KMs follow in Section 3.2. This latter section includes online algorithms that are developed in the regret minimization framework. Finally, a number of incremental regression algorithms that have been targeted particularly for use within the field of robotics is presented in Section 3.3.

3.1 Limiting the Kernel Expansion

The solution in KMs is described in terms of a weighted summation of kernel evaluations with respect to the training data. An obvious approach to reduce this linear complexity is to limit the expansion to only a subset of the training samples. In this strategy, a main concern is the appropriate selection of this subset, referred to as the active set or support set, such that the impact on the prediction function is minimal. Different criteria can be used to this extent, such as removing samples corresponding to near-zero coefficients (Suykens et al., 2002a) or taking unbiased random subsamples (Lee and Mangasarian, 2001). Alternatively, Downs et al. (2002) propose to project linear dependent samples onto each other in order to sparsify the solution of a Support Vector Machine (SVM) without loss of accuracy. Their argument is that linear dependent samples can be described completely in terms of the remaining samples and can therefore safely be omitted from the kernel expansion. To verify whether a new input sample $\boldsymbol{x}$ is linearly dependent with respect to the original m samples, we can write[1]

$$\begin{aligned} \|\delta\|^2 &= \min_{\boldsymbol{d}} \left\| \sum_{i=1}^{m} d_i \phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}) \right\|^2 \\ &= \min_{\boldsymbol{d}} \left(\sum_{i,j=1}^{m} d_i d_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) - 2 \sum_{i=1}^{m} d_i k(\boldsymbol{x}_i, \boldsymbol{x}) + k(\boldsymbol{x}, \boldsymbol{x}) \right) \leq \nu \ , \end{aligned} \tag{3.1}$$

where ν regulates the desired approximation accuracy. The optimal coefficients $\boldsymbol{d}$ can subsequently be used to project the sample on the remaining training samples. In case of exact linear dependence ($\nu = 0$), the sparse solution produces identical results with respect to the original kernel expansion, while approximate linear dependence ($\nu > 0$) induces more sparsity at the cost of accuracy. Nonetheless, these reduction methods require a full solution to be computed first and therefore aim to reduce prediction time while increasing training time.

In most situations it is desirable to reduce the training complexity as well, to allow KMs to scale to large datasets. A key concept in KMs is that training samples enter the algorithm only through their entries in the kernel matrix. The strategy in most reduction approaches is therefore to target the kernel matrix $\boldsymbol{K}$ directly and replace it with a low-rank approximation. This approximation generally takes the form

$$\tilde{\boldsymbol{K}} = \boldsymbol{G}\boldsymbol{G}^{\mathrm{T}} \approx \boldsymbol{K} \ ,$$

or similarly

$$\tilde{\boldsymbol{K}} = \boldsymbol{G}\boldsymbol{W}\boldsymbol{G}^{\mathrm{T}} \approx \boldsymbol{K} \ ,$$

[1] There are more efficient procedures to verify linear dependence within the entire training set at once.

where $\boldsymbol{G}$ is a rectangular $m \times l$ matrix, $\boldsymbol{W}$ is a square $l \times l$ matrix, and $l < m$ indicates the reduced rank. The optimal solution and the prediction function can subsequently be described more efficiently in terms of vector and matrix products with $\boldsymbol{G}$ and $\boldsymbol{W}$, rather than using the full rank matrix $\boldsymbol{K}$. Note that in this formulation all samples are used during training, although only a subset of them is used to describe the solution. Several techniques have been used to implement these low-rank decompositions (Quiñonero-Candela and Rasmussen, 2005), such as the Nyström method (Williams and Seeger, 2001; Zhang et al., 2008), randomized singular value decomposition (Drineas and Mahoney, 2005), or incomplete Cholesky decomposition (Fine and Scheinberg, 2002; Bach and Jordan, 2005). Furthermore, there are various strategies to select the active set of l samples, such as unbiased uniform sampling (Williams and Seeger, 2001), greedy posterior maximization (Smola and Bartlett, 2001), maximum information gain (Seeger et al., 2003), matching pursuit (Keerthi et al., 2006), or using likelihood optimization in Bayesian frameworks (Titsias, 2009). Interestingly, Snelson and Ghahramani (2006) lift the typical restriction that the active set is a subset of the training samples and allow for so-called pseudo-inputs to be optimized. The time and space complexity for training low-rank approximations is typically $\mathcal{O}\big(ml^2\big)$ and $\mathcal{O}(ml)$, respectively, such that significant gains can be expected if $l \ll m$.

The Core Vector Machine follows an entirely different approach, as it approximates an SVM by reducing it to a minimum enclosing ball problem in feature space (Tsang et al., 2005a, 2006, 2005b, 2007). Subsequently, an efficient algorithm is employed to find a $(1+\epsilon)$-approximation[2] for the latter problem (Bădoiu and Clarkson, 2008), which by virtue of the reduction translates to an approximate solution for the SVM problem. Similar to the previous approaches, the time complexity of this algorithm is linear in the number of training samples. More interestingly, however, is that the size of the kernel expansion only depends on the desired accuracy ϵ and not on the number of training samples.

3.2 Incremental Methods

The strategies to reduce the kernel expansion described in the previous section require all samples to be available during training. Furthermore, once an optimal solution is obtained, it cannot readily be extended with additional training samples. Several strategies have been proposed to update solutions incrementally without the need to completely retrain the model. These methods can roughly be subdivided in three primary categories. The first category of incremental procedures performs an exact update routine to obtain the batch solution after adding (or removing) a training sample (e.g., Cauwenberghs and Poggio, 2001; Martin, 2002; Ma et al., 2003; Liang and Li, 2009). Not surprisingly, the size of the resulting kernel expansion is identical to the batch solution and the time complexity for each update is $\mathcal{O}\big(m^2\big)$. The advantage of these procedures is that the exact

[2]The radius of the approximate solution to the minimum enclosing ball problem is guaranteed to be $(1+\epsilon)$ times the true optimal radius; $\epsilon > 0$ is typically chosen very small (e.g., $1 \cdot 10^{-6}$).

batch solution is obtained after each individual update. However, these procedures are not suitable for large scale problems due to high computational requirements.

In the second category of algorithms this high update complexity is alleviated by computing approximate updates. A popular algorithm in this class is LASVM (Bordes et al., 2005), which is closely related to the Sequential Minimal Optimization procedure for batch training of SVMs (Platt, 1999). At each iteration, the algorithms seeks to add and remove samples from the active set by optimizing the coefficients of a pair of samples. First, the "process" subroutine attempts to insert an input sample in the kernel expansion using a direction search. This subroutine therefore involves at least one input sample that is currently not in the active set, which is typically the newly arrived sample in an online setting. The subsequent "reprocess" subroutine selects again a pair of inputs samples and potentially eliminates these from the active set by verifying if their weight coefficients can be set to zero. Although the solution is not identical to the SVM solution at each time step, it can be shown that LASVM converges to the SVM solution in the limit. Empirical results suggest that competitive performance can be achieved after only a single pass over the training data (Bordes et al., 2005), though this requires the samples to arrive in randomized order.

The size of the kernel expansion in LASVM scales linearly with the number of training samples, despite regular removal of samples from the active set. This problem can be alleviated by projecting approximate linear dependent samples onto each other, as described in Equation (3.1). In Sparse Online Greedy ϵ-insensitive Support Vector Regression (ϵ-SVR) (Engel et al., 2002), this measure is used in the context of an approximate SVM for regression (Vijayakumar and Wu, 1999), while Kernel Recursive Least Squares uses the same technique to incrementally compute a kernel-based least squares solution with restricted kernel expansion (Engel et al., 2004). A similar method is used in a Bayesian context by Csató and Opper (2002) to formulate Sparse Online Gaussian Process Regression, although in this method the linear dependence measure is subsequently multiplied with a likelihood-dependent term for the sample. Assuming a compact input space $\mathcal{X}$, it can be proved that the approximate linear dependence condition with $\nu > 0$ leads to a bounded kernel expansion (e.g., Engel et al., 2004). However, its exact size (and therefore computational requirements) are data-dependent and cannot be computed a priori. Csató and Opper (2002) therefore impose a strict upper bound on the size of the kernel expansion; once this limit is reached, a sample is forcefully projected on the remaining samples. Ranganathan and Yang (2008), on the other hand, propose Online Sparse[3] Gaussian Process Regression, in which the computational complexity is constrained by sparse matrix calculations and by removing the oldest sample once a predefined budget has been reached. Regardless, all these methods combine the desirable properties of incremental processing of incoming training samples and a bounded kernel expansion.

[3]Note the different ordering of "sparse" and "online".

3.2.1 Online Learning with Regret Minimization

The third and last category consists of online algorithms developed in the regret minimization framework (Littlestone, 1988; Vovk et al., 2005; Cesa-Bianchi and Lugosi, 2006). This framework considers an online learning scenario, in which the learning method adapts its hypothesis (i.e., the prediction function) at each time step. Given an initial hypothesis f_0 in some function space $\mathcal{F}$, the following steps are performed for $1 \leq t \leq T$:

1. the environment presents an input sample $\boldsymbol{x}_t$;
2. the learner responds with $\hat{y}_t = f_{t-1}(\boldsymbol{x}_t)$;
3. the environment presents the corresponding output y_t;
4. the learner incurs a loss $\mathcal{L}(y_t, \hat{y}_t)$ and updates its hypothesis accordingly to f_t.

Note that there are no restrictions on the distribution of the data. This paradigm is explained as competing with a set of experts, which is typically chosen as the set of all possible functions in $\mathcal{F}$. The goal of a learning algorithm is to produce a sequence of functions $f_1, \ldots, f_T$ that minimizes the cumulative regret with respect to the best expert (i.e., the best fixed hypothesis) in retrospect. The cumulative regret can thus be defined as

$$R_T = \sum_{t=1}^{T} \mathcal{L}(y_t, f_t(\boldsymbol{x}_t)) - \min_{f \in \mathcal{F}} \sum_{t=1}^{T} \mathcal{L}(y_t, f(\boldsymbol{x}_t)) \ ,$$

which is the difference between the cumulative loss of the learning algorithm and the cumulative loss of the optimal fixed prediction function. This difference captures how much better the learning algorithms could have performed if it would have followed the advice of the best expert, therefore explaining the terminology "regret". Intuitively, regret minimization seems closely related to risk minimization as described in Section 2.1. Under certain conditions, the cumulative regret can indeed be linked with generalization performance in terms of expected risk (Littlestone, 1989; Cesa-Bianchi et al., 2004; Zhang, 2005; Cesa-Bianchi and Gentile, 2008; Kakade and Tewari, 2009). One might expect that the final prediction function f_T (i.e., the one that depends on the entire sequence of T samples) is also the one that optimizes the risk. However, this is often not the case and so-called "online-to-batch" conversions are required to convert the sequence $f_1, \ldots, f_T$ into a single prediction function f with low risk. An intuitive explanation is that the sequence of hypotheses produced by the algorithm depends on the data and is therefore subject to variance. A typical online-to-batch conversion in case of independent and identically distributed (i.i.d.) sampling it therefore to return the *average* hypothesis $\bar{f}$ (Cesa-Bianchi and Gentile, 2008).

Various online[4] learning methods have been proposed and analyzed within the regret minimization framework. A significant part of these algorithms are closely related to the original perceptron algorithm and its kernel-based variant (Rosenblatt, 1958; Freund and Schapire, 1999). Inspired by SVM, a typical modification of the perceptron is to enforce a large margin (Shalev-Shwartz and Singer, 2005; Crammer et al., 2006). The problem of an increasing kernel expansion is prominent in this type of algorithms, as each update usually involves adding a sample to the active set. Several variants have therefore been proposed to limit the size of the kernel expansion, for instance by removing a sample at random (Cavallanti et al., 2007), removing the oldest sample in the active set (Dekel et al., 2008; Kivinen et al., 2004), or by projecting samples using approximate linear dependence (Orabona et al., 2009). Another approach is to remove the sample that remains recognized with the largest margin (Crammer et al., 2004). This margin can be viewed as an indirect estimate of the impact of the removal on the overall performance of the hyperplane. Instead, one might also opt to remove the sample that results in the least increase in misclassification rate of (a subset of) the samples seen thus far (Weston et al., 2005).

A number of regret minimization algorithms are directly derived from batch methods (e.g., SVM) and perform a stochastic optimization step of the corresponding objective function for each new sample (Kivinen et al., 2004; Smale and Yao, 2006; Vishwanathan et al., 2006). For example, the NORMA algorithm optimizes a regularized KM with convex loss function using stochastic gradient descent (for details, see Kivinen et al., 2004). Each update consists of adding the sample to the kernel expansion with an appropriate coefficient, such that at time step t

$$\boldsymbol{X}_t := \begin{bmatrix} \boldsymbol{X}_{t-1}^{\mathrm{T}} \\ \boldsymbol{x}_t^{\mathrm{T}} \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{\alpha}_t := \begin{bmatrix} (1 - \eta_t \lambda)\, \boldsymbol{\alpha}_{t-1} \\ -\eta_t \partial \mathcal{L}(y_t, f(\boldsymbol{x}_t)) \end{bmatrix} ,$$

where $\eta < \frac{1}{\lambda}$ is the learning rate. Like many other online learning algorithms, these update rules are easy to implement and extremely efficient. Given i.i.d. sampling, it can be shown that this method converges to the batch solution in the limit. However, the convergence rate is strongly dependent on the i.i.d. assumption; violation of this assumption will result in considerably slower convergence, if at all.

A final note is that regret minimization algorithms are presented in this section as being distinct from other incremental algorithms. However, this distinction is foremost based on whether the algorithm has been proposed and analyzed primarily within the regret minimization or risk minimization framework. These frameworks are not mutually exclusive and algorithms can be analyzed within both frameworks. A primary example of such an algorithm is Kernel Regularized Least Squares (KRLS) (and therefore also Regularized Least Squares (RLS)), which has been studied in either framework. More detailed treatment of these results will follow in Section 4.4.

[4]Online learning has become synonymous with learning in the regret minimization framework. To avoid confusion, the term "online" will therefore strictly refer to algorithms developed and analyzed within this framework, while the term "incremental" will be used in a more broad sense for algorithms that learn one sample at a time.

3.3 Methods Proposed for Robotics

The importance of incremental and computationally efficient regression methods has been acknowledged early on in the robotics community. In effect, this has led to the development of a number of methods targeted specifically for use in this application domain. One of the first methods that gained widespread acceptance is Receptive Field Weighted Regression (RFWR) proposed by Schaal and Atkeson (1998). RFWR is an incremental Locally Weighted Regression (LWR) variant, in which the function under consideration is approximated by piecewise linear models. The predicted output is then given by a weighted summation of the individual contributions of these local models (i.e., the receptive fields), much like in the mixture of experts approach (e.g., Jacobs et al., 1991). For K receptive fields, the prediction function is written as

$$f(\boldsymbol{x}) = \frac{1}{Z} \sum_{i=1}^{K} a_i(\boldsymbol{x}) f_i(\boldsymbol{x}) \ ,$$

where Z is a normalization factor defined as $Z = \sum_{i=1}^{K} a_i(\boldsymbol{x})$ and $a_i(\boldsymbol{x})$ indicates the level of activation of the i-th receptive field with respect to an input $\boldsymbol{x}$. Typically, $a_i(\boldsymbol{x})$ is chosen such that each receptive field is active only within a localized region of the input space. The canonical example of an activation function that implements this localization is the Radial Base Function (RBF) given by

$$a_i(\boldsymbol{x}) = e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{c}_i)^{\mathrm{T}} \boldsymbol{D}_i (\boldsymbol{x}-\boldsymbol{c}_i)} \ ,$$

where $\boldsymbol{c}_i$ is the center of the receptive field and $\boldsymbol{D}_i$ is a positive definite distance matrix. The local prediction functions f_i, on the other hand, are simple linear regression models

$$f_i(\boldsymbol{x}) = \boldsymbol{w}_i^{\mathrm{T}} (\boldsymbol{x} - \boldsymbol{c}_i) + b_i \ ,$$

where $\boldsymbol{w}_i$ are the linear weight coefficients and b_i is a bias term. Each receptive field is thus essentially described by a center $\boldsymbol{c}_i$, a distance matrix $\boldsymbol{D}_i$, and the model parameters $\boldsymbol{w}_i$ and b_i. An example of receptive fields with diagonal distance matrices covering a 2-dimensional input space is demonstrated in Figure 3.1. In the incremental paradigm, an incoming sample is assigned to the receptive field with maximum activation, which then uses the sample to update its local model. Once a receptive field has seen enough inputs, its distance matrix D is optimized using an incremental gradient descent based on stochastic leave-one-out cross validation (for details, see Schaal and Atkeson, 1998; Vijayakumar et al., 2005). Conversely, new receptive fields are allocated as needed if existing receptive fields are not activated sufficiently, as determined using a minimum activation threshold a_{gen}. The number of receptive fields that will be allocated therefore depends on this threshold, the input space $\mathcal{X}$, and the complexity of the function (through adaptations to the distance matrices $\boldsymbol{D}_i$). Furthermore, it follows from the well-known curse of dimensionality

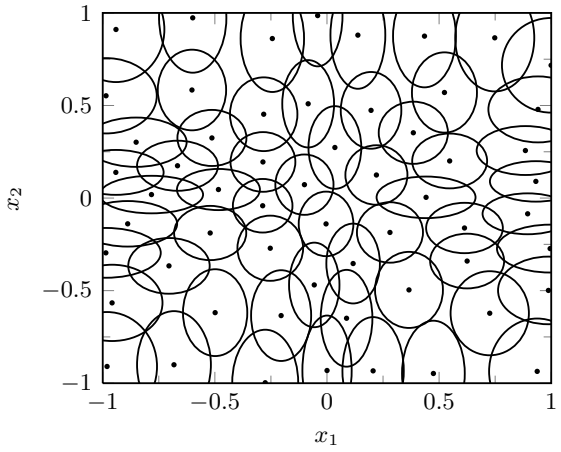


Figure 3.1: Example of receptive fields in RFWR and LWPR on the Cross 2D dataset (cf. Section 5.1).

that the number of receptive fields scales exponentially with the input dimensionality (Duda et al., 2001). This exponential increase is alleviated to some extent in Locally Weighted Projection Regression (LWPR) (Vijayakumar et al., 2005), which uses Partial Least Squares in the local models to reduce the impact of redundant and irrelevant input dimensions (Wold et al., 2001). This improvement has led LWPR to gain significant popularity for use in incremental robotics learning problems. Since LWPR can be considered a generalization of RFWR, references to LWPR in the following will refer implicitly both to LWPR and RFWR.

At this point it is important to stress the fundamental difference between LWPR and KMs: the latter methods perform global linear regression in an implicit high-dimensional feature space, whereas the former method combines individual linear regression models that operate within a (small) region of the original input space. An intrinsic property of LWPR is that learning takes place at two levels, namely (1) finding an appropriate structuring of the input space and (2) fitting linear models to each subspace. This interplay between both levels of learning is governed using a large number of hyperparameters, among which the initial distance matrix, the initial learning rate and meta learning rate, activation thresholds for receptive field creation and pruning, a projection threshold for PLS, and a penalty term preventing infinite shrinkage of the receptive fields.

LWPR can achieve similar generalization performance as KMs in ideal conditions, although this requires considerable efforts tuning a large number of hyperparameters. Furthermore, it typically needs a large number of training samples to achieve this performance. In most empirical validations, the performance therefore lacks behind methods such as Gaussian Process Regression (GPR) and ϵ-SVR, while requiring more involvement on part of the practitioner. Nguyen-Tuong et al. (2009) propose Local Gaussian Processes (LGP) in an attempt to improve the accuracy of LWPR by replacing the linear models with GPR models. Fewer receptive fields are necessary in this approach, as GPR models are more complex than linear models and can therefore accurately model a larger region of the input space. A nearly identical approach has been proposed by Schneider and

Ertel (2010), the primary difference being that they employ a variant of GPR with a heteroskedastic noise model (Kersting et al., 2007). Nonetheless, the performance of both approaches is still inferior with respect to a global GPR and the only advantage is a reduction in computational requirements.

Several other approaches have been proposed for incremental learning in robotics, among which Sparse Incremental ϵ-SVR and Sparse Incremental GPR (Nguyen-Tuong and Peters, 2010, 2011, respectively). Both methods are incremental KMs with a bounded kernel expansion due to approximate linear dependence projection and an additional fixed budget constraint. Aside from minor differences, these two methods can be considered identical to respectively Sparse Online Greedy ϵ-SVR and Sparse Online GPR (cf. Section 3.2). Furthermore, Cederborg et al. (2010) propose to use a Gaussian Mixture Model (GMM) (Ghahramani and Jordan, 1994) applied within the LWR framework. In this method, predictions are obtained by training a GMM on a small number of training samples in the vicinity of the test sample.

Surprisingly for methods that are proposed for incremental and real-time learning, the computational complexity in all these methods is described in subjective terms (e.g., "efficient") rather than formal terms (e.g., asymptotic complexity or an explicit upper bound). It is important to realize that localization does not necessarily decrease complexity, as the problem of an increasing kernel expansion is essentially identical to an increasing number of local models. Although this increase can be bounded in the latter case using adequate measures (e.g., $a_{\text{gen}} < 1$ and $\boldsymbol{D} > \mathbf{0}$), in practice it may still cause computational intractability in case of a large number of samples. Furthermore, there is little theoretical justification for these methods and there are no known generalization or regret bounds.

4 ALGORITHM

The previous chapter identified a number of incremental kernel methods (KMs) that satisfy the requirement of a bounded time and space complexity. The common aspect of most of these methods is that growth of the kernel expansion is restricted using an additional sparsification procedure on top of a standard incremental KM. Typically, this procedure introduces a significant computational overhead and additional hyperparameters that may not be trivial to tune (e.g., the measure of linear dependence ν). Furthermore, in many of these methods (including Locally Weighted Projection Regression (LWPR)) it is difficult to exactly quantify the computational cost a priori, which is inconvenient when employed in a strict real-time environment. This can be circumvented by imposing a fixed budget for the kernel expansion, though this requires an additional selection procedure to explicitly remove training data.

In this chapter, an alternative algorithm is proposed that attempts to avoid these drawbacks. The crucial realization is that the kernel trick and the representer theorem are at the core of the problem. As noted by Shawe-Taylor and Cristianini (2004), a key property that is required of a kernel function for an application is that its evaluation should require significantly less computation than would be needed for explicit evaluation of the corresponding feature mapping. This is unlikely in case of a large or potentially infinite number of training samples. It is therefore computationally advantageous to avoid the kernel machinery and its consequences by performing the feature mapping explicitly. Unfortunately, for some kernels (e.g., the Radial Base Function (RBF) kernel) this feature mapping is infinite dimensional and therefore computationally intractable.

Rahimi and Recht (2008a) demonstrate many kernel functions can be approximated to any arbitrary accuracy using a finite dimensional random feature mapping. Though this technique has originally been proposed for large scale batch learning, it can be applied with any given KM. The contribution of this thesis is to combine this kernel approximation with incremental variants

of Regularized Least Squares (RLS) and Gaussian Process Regression (GPR). The advantages of these particular base learning algorithms are (1) the availability of exact update routines with constant time and space complexity, (2) a solid theoretical and empirical foundation, and (3) ease of implementation and use. The latter argument holds in particular for GPR, since it provides a predictive variance and hyperparameters can be tuned automatically using log marginal likelihood optimization.

A treatment of the random finite dimensional approximation of kernels is given in Section 4.1, which presents the technique in the perspective of RLS. Section 4.2 continues along this line and applies the kernel approximation within a Bayesian framework in the form of GPR. The incremental variants of both algorithms are subsequently discussed in Section 4.3, after which this chapter is concluded with an in-depth discussion of the overall method in Section 4.4.

4.1 Approximating Kernels using Random Features

As mentioned, the kernel expansion is only strictly necessary in absence of a finite dimensional representation of the feature space of a given kernel. If such a representation exists, however, it can be advantageous to avoid the kernel machinery by explicitly mapping the samples in this finite dimensional feature space. An example is the polynomial kernel from Equation (2.17), which corresponds to a finite dimensional vector containing all possible monomials up to a given degree (Shawe-Taylor and Cristianini, 2004). The exact formulation of this feature mapping is

$$\begin{aligned} k(\boldsymbol{x}_i, \boldsymbol{x}_j) &= (R + \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle)^d \\ &= \sum_{s=0}^{d} \binom{d}{s} R^{d-s} \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle^s \quad \in \mathbb{R}^{\binom{n+d}{d}} \ . \end{aligned} \tag{4.1}$$

This mapping becomes computationally infeasible when either the degree d or the original input dimensionality n is moderately large. Regardless, considerable gains can be made in application domains where the number of samples exceeds the dimensionality of the mapping. Consider for instance open-ended learning problems of relatively low dimensionality (e.g., learning robot dynamics or kinematics).

Explicit computation of the feature mapping for the RBF kernel is intractable, since the corresponding feature space is infinite dimensional. However, Rahimi and Recht (2008a) demonstrate that the RBF kernel (and other shift invariant kernels) can be approximated to an arbitrary precision using a finite dimensional random feature mapping. Their approach utilizes Bochner's theorem, which relates positive definite functions, among which admissible kernels functions, to Fourier transforms using a finite Borel measure:

Theorem 4.1 (Bochner, 1933)**.** *A function k on $\mathbb{R}^d$ is the Fourier transform of a finite positive Borel measure μ if and only if it is positive definite and continuous.*

The relevance of Bochner's theorem has been acknowledged in the context of Regularization Networks (RNs) as early as Girosi et al. (1993), and this work has subsequently been extended to the kernel framework by Smola et al. (1998). In short, this theorem states that shift-invariant kernel functions $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = k(\boldsymbol{x}_i - \boldsymbol{x}_j)$ can be described as the Fourier transform of a unique measure μ. Consequently, if μ is a proper probability density function[1], then the kernel can be estimated by randomly sampling features according to μ (e.g., Shawe-Taylor and Cristianini, 2004, Chapter 9.7). Combining both these techniques, we can write

$$\begin{aligned} k(\boldsymbol{x}_i - \boldsymbol{x}_j) &= \int_{\mathbb{R}^d} e^{-i\boldsymbol{\omega}^{\mathrm{T}}(\boldsymbol{x}_i - \boldsymbol{x}_j)} \mu(\boldsymbol{\omega}) \, \mathrm{d}\boldsymbol{\omega} \\ &= \mathbb{E}_{\boldsymbol{\omega}}\left[e^{-i\boldsymbol{\omega}^{\mathrm{T}}(\boldsymbol{x}_i - \boldsymbol{x}_j)}\right] && \text{iff } \int \mu(\boldsymbol{\omega}) \, \mathrm{d}\boldsymbol{\omega} = 1 \\ &= \mathbb{E}_{\boldsymbol{\omega}}\left[\cos\left(\boldsymbol{\omega}^{\mathrm{T}}\left(\boldsymbol{x}_i - \boldsymbol{x}_j\right)\right)\right] && \text{as } k(\cdot), \mu(\boldsymbol{\omega}) \in \mathbb{R}^d \\ &= \mathbb{E}_{\boldsymbol{\omega}}\left[\cos\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}_i\right)\cos\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}_j\right) + \sin\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}_i\right)\sin\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}_j\right)\right] \\ &= \mathbb{E}_{\boldsymbol{\omega}}\left[z_{\boldsymbol{\omega}}(\boldsymbol{x}_i)^{\mathrm{T}} z_{\boldsymbol{\omega}}(\boldsymbol{x}_j)\right] \ , \end{aligned}$$

where

$$z_{\boldsymbol{\omega}}(\boldsymbol{x}) = \left[\cos\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}\right), \sin\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x}\right)\right]^{\mathrm{T}} \ . \tag{4.2}$$

In other words, the inner product $\langle z_{\boldsymbol{\omega}}(\boldsymbol{x}_i), z_{\boldsymbol{\omega}}(\boldsymbol{x}_j)\rangle$ gives an unbiased estimate of any shift invariant kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, given that the spectral frequency $\boldsymbol{\omega}$ is drawn according to its corresponding measure μ. The feature mapping $z_{\boldsymbol{\omega}}(\boldsymbol{x})$ in Equation (4.2) can be interpreted as a projection of $\boldsymbol{x}$ on the random direction $\boldsymbol{\omega}$ and subsequently wrapping this line on the unit circle in $\mathbb{R}^2$. Given a kernel function, the corresponding probability density function μ can be obtained by computing its inverse Fourier transform. For instance, the probability density function for the isotropic RBF kernel is Gaussian and it suffices to sample $\boldsymbol{\omega} \sim \mathcal{N}(\mathbf{0}, 2\gamma\boldsymbol{I})$. An example of a 1-dimensional RBF kernel and its approximation using these random Fourier features is shown in Figure 4.1.

The variance of the approximation can be lowered arbitrarily by averaging multiple features $z_{\boldsymbol{\omega}}$, where each instance has an individual frequency $\boldsymbol{\omega}$ drawn at random from μ. Given D random projections, we can thus write

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathbb{E}\left[\frac{1}{D}\sum_{d=1}^{D} z_{\boldsymbol{\omega}_d}(\boldsymbol{x}_i)^{\mathrm{T}} z_{\boldsymbol{\omega}_d}(\boldsymbol{x}_j)\right] \ ,$$

where $\boldsymbol{\omega}_d \sim \mu(\boldsymbol{\omega})$ for $1 \leq d \leq D$. Similarly, by concatenating multiple random features and

[1] The measure can be scaled to ensure that this is the case.

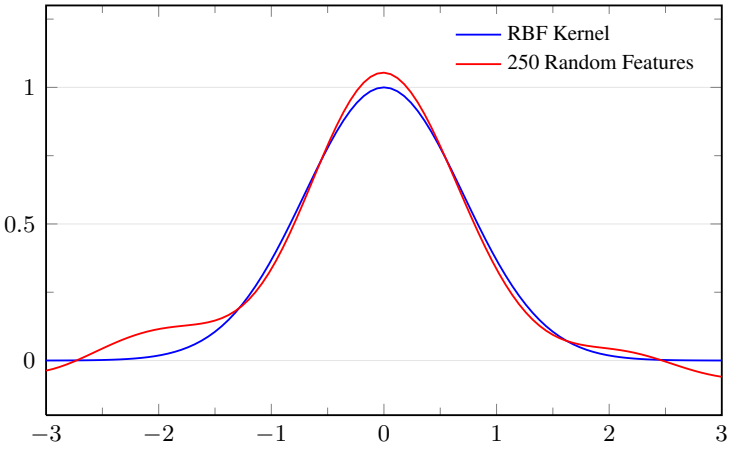


Figure 4.1: Example of the 1-dimensional RBF kernel with $\gamma = 1$ and an approximation thereof using 250 random features.

integrating the normalization term, such that

$$\phi(\boldsymbol{x}) = \frac{1}{\sqrt{D}} \left[z_{\boldsymbol{\omega}_1}(\boldsymbol{x})^{\mathrm{T}}, \cdots, z_{\boldsymbol{\omega}_D}(\boldsymbol{x})^{\mathrm{T}} \right]^{\mathrm{T}} \quad , \tag{4.3}$$

we obtain a random feature mapping $\phi : \mathcal{X} \mapsto \mathbb{R}^{2D}$ for which $k(\boldsymbol{x}_i, \boldsymbol{x}_j) \approx \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$ for sufficiently large D. The exact convergence can be bounded as follows:

Claim 4.2 (Rahimi and Recht, 2008a)**.** *Given that $\mathcal{X}$ is a compact subspace of $\mathbb{R}^n$ with diameter $\operatorname{diam}(\mathcal{X})$, then for the mapping ϕ defined in Equation* (4.3)*, we have*

$$P\left(\sup_{\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{X}} |\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle - k(\boldsymbol{x}_i, \boldsymbol{x}_j)| \geq \epsilon \right) \leq 2^8 \left(\frac{\sigma_\mu \operatorname{diam}(\mathcal{X})}{\epsilon} \right)^2 e^{-\frac{D\epsilon^2}{4(n+2)}} \quad ,$$

where $\sigma_\mu^2 = \mathbb{E}_\mu[\langle \boldsymbol{\omega}, \boldsymbol{\omega} \rangle]$ is the second moment of the Fourier transform of k.

This demonstrates that the absolute deviation between $\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$ and $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ converges exponentially fast in D. Moreover, the kernel can be approximated with constant probability to any arbitrary precision by choosing D sufficiently large.

There are alternative mappings that satisfy $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathbb{E}_{\boldsymbol{\omega}} \left[z_{\boldsymbol{\omega}}(\boldsymbol{x}_i)^{\mathrm{T}} z_{\boldsymbol{\omega}}(\boldsymbol{x}_j) \right]$. One such example is given by

$$z_{\boldsymbol{\omega},\beta}(\boldsymbol{x}) = \sqrt{2} \cos\left(\boldsymbol{\omega}^{\mathrm{T}} \boldsymbol{x} + \beta \right) \quad , \tag{4.4}$$

where the frequencies $\boldsymbol{\omega} \sim \mu(\boldsymbol{\omega})$ and the phases $\beta \sim \mathcal{U}(-\pi, \pi)$ (Rahimi and Recht, 2008a,b). An advantage of this alternative formulation is a reduced dimensionality as compared to Equation (4.2). When used in the normalized feature map defined in Equation (4.3), the resulting dimensionality will be D rather than $2D$. Depending on the underlying learning method, this more

parsimonious representation can result in significant computational savings.

Any linear learning algorithm can employ these random features when substituting the original input matrix $\boldsymbol{X}$ for the design matrix $\boldsymbol{\Phi} = \phi(\boldsymbol{X})$, where ϕ is as in Equation (4.3). Given the random features in Equation (4.4), the optimal D-dimensional weight vector $\boldsymbol{w}$ for RLS from Equation (2.5) is thus

$$\boldsymbol{w} = \left(\lambda \boldsymbol{I} + \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y} \ ,$$

and the corresponding prediction function is

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \phi(\boldsymbol{x}) \rangle \ .$$

The time and space complexity for training are $\mathcal{O}\left(mD^2\right)$ and $\mathcal{O}(mD)$, respectively, while the time complexity for predictions is $\mathcal{O}(D)$ and therefore independent of the number of training samples. In the following, the combination of RLS with random Fourier features will be referred to as Random Fourier Regularized Least Squares (RFRLS), where a possible superscript (e.g., RFRLS^{100}) is used to explicitly denote the number of features D.

4.2 Sparse Spectrum Gaussian Process Regression

As could be expected from the similarity of RLS and GPR, approximation of the kernel (or covariance) function using random features can be applied within the context of GPR. This particular use has been investigated recently[2] by Lázaro-Gredilla et al. (2010) under the name Sparse Spectrum Gaussian Process Regression (SSGPR). Similar to RLS, applying the kernel approximation with GPR corresponds to explicitly applying the feature mapping ϕ, rather than formulating the problem in terms of kernel evaluations. Recall that the linear variant of GPR is given by the generalized Bayesian linear model from Section 2.5. Replacing $\boldsymbol{x}$ with $\phi(\boldsymbol{x})$ and simplifying with respect to σ_n results in the predictive posterior

$$p(y|\boldsymbol{x}, \boldsymbol{X}, \boldsymbol{y}) = \mathcal{N}\left(\phi(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{A}^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y},\ \sigma_n^2\left(1 + \phi(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{A}^{-1}\phi(\boldsymbol{x})\right)\right) \ ,$$

where $\boldsymbol{\Phi} = \phi(\boldsymbol{X})$ and

$$\boldsymbol{A} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \sigma_n^2 \boldsymbol{I} \ , \tag{4.5}$$

where we assumed for simplicity that the weight prior $\boldsymbol{\Sigma}_p$ is diagonal. Furthermore, the predictive variance has an additional term σ_n^2 to compensate for noise present in the labels of the test samples. In case of the popular anisotropic RBF kernel from Equation (2.19), the feature mapping is given

[2] An earlier technical report implies that Lázaro-Gredilla et al. (2007) discovered this technique independently from Rahimi and Recht.

by

$$\phi(\boldsymbol{x}) = \frac{\sigma_f}{\sqrt{D}} \left[\sin\left(\left\langle \boldsymbol{\omega}_1^{\mathrm{T}}, \boldsymbol{x} \right\rangle\right), \cos\left(\left\langle \boldsymbol{\omega}_1^{\mathrm{T}}, \boldsymbol{x} \right\rangle\right), \cdots, \sin\left(\left\langle \boldsymbol{\omega}_D^{\mathrm{T}}, \boldsymbol{x} \right\rangle\right), \cos\left(\left\langle \boldsymbol{\omega}_D^{\mathrm{T}}, \boldsymbol{x} \right\rangle\right)\right]^{\mathrm{T}} \ , \quad (4.6)$$

where the frequencies $\boldsymbol{\omega} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{M})$. It is common that $\boldsymbol{M}$ is a diagonal matrix with $\mathrm{diag}\,(\boldsymbol{M}) = \left[\ell_1^2, \ldots, \ell_n^2\right]$, such that it suffices to draw $\boldsymbol{\omega}$ from a standard normal distribution $\mathcal{N}(0, 1)$ and subsequently scale these by $\boldsymbol{\ell}$. Note that this feature mapping explicitly uses the original mapping from Equation (4.2), as opposed to the more parsimonious alternative in Equation (4.4). Although both representations give an unbiased estimate of the kernel, Bayesian inference is different in both cases. The problem lies in the fact that the cosine in Equation (4.4) includes a phase β that cannot easily be integrated out, as required for a full Bayesian treatment (Lázaro-Gredilla et al., 2010).

Two main advantages of standard GPR over Kernel Regularized Least Squares (KRLS) are the availability of predictive variances and convenient hyperparameter optimization using the marginal likelihood. These advantages apply equally well in the comparison between SSGPR and RFRLS. In case of the former, the negative log marginal likelihood is given by

$$-\ln p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) = \frac{1}{2\sigma_n^2}\left(\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y} - \boldsymbol{y}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{A}^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}\right) + \frac{1}{2}\ln\det\boldsymbol{A} - \frac{D}{2}\ln\sigma_n^2 + \frac{m}{2}\ln 2\pi\sigma_n^2 \ ,$$

where $\boldsymbol{A}$ is as defined in Equation (4.5). Interestingly, considering the spectral frequencies $\boldsymbol{\omega}_d$ for $1 \leq d \leq D$ as parameters rather than constants, these can be optimized using the log marginal likelihood as well. Noting that the distribution of the spectral frequencies is closely related to the kernel via the measure μ, it is clear that optimizing these frequencies equates to tuning the kernel function to the data. Although this may result in highly efficient and sparse models, there are two disadvantages to optimization of the frequencies. First, it departs from the original motivation of approximating shift invariant kernel functions and there are no longer guarantees in terms of approximation errors with respect to the kernel equivalent. Second, the number of tunable hyperparameters increases from $n + 2$ (in case of the anisotropic RBF kernel) to $n + 2 + Dn$, therefore drastically increasing the risk of overfitting in case of a limited number of training samples.

4.3 Efficient Incremental Updates

Both RFRLS and SSGPR have demonstrated promising results for large scale batch learning. Regardless, the application domain under consideration in this thesis is the incremental learning setting. To this extent, the choice of RLS and GPR as base algorithms was not incidental, since these algorithm allows efficient and exact updates of the solution using well-known linear algebra results. This is demonstrated at the hand of RFRLS, although the procedure is nearly identical for

SSGPR due to their functional similarity. Recall that the optimal solution in RFRLS is given by

$$\boldsymbol{w} = \left(\lambda \boldsymbol{I} + \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{A}^{-1}\boldsymbol{b} \ , \tag{4.7}$$

where the solution is conveniently decomposed in terms of a $D \times D$ matrix $\boldsymbol{A} = \lambda \boldsymbol{I} + \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}$ and a D-dimensional vector $\boldsymbol{b} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$. Arrival of a new sample $(\boldsymbol{x}_t, y_t)$ at time step t can thus be described as

$$\boldsymbol{\Phi}_t = \begin{bmatrix} \boldsymbol{\Phi}_{t-1} \\ \boldsymbol{\phi}_t \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{y}_t = \begin{bmatrix} \boldsymbol{y}_{t-1} \\ y_t \end{bmatrix} \ ,$$

where $\boldsymbol{\phi}_t = \phi(\boldsymbol{x}_t)$. Integrating these update rules, the matrix $\boldsymbol{A}$ in Equation (4.7) can be formulated recursively as

$$\begin{aligned} \boldsymbol{A}_t &= \lambda \boldsymbol{I} + \boldsymbol{\Phi}_t^{\mathrm{T}}\boldsymbol{\Phi}_t \\ &= \lambda \boldsymbol{I} + \boldsymbol{\Phi}_{t-1}^{\mathrm{T}}\boldsymbol{\Phi}_{t-1} + \boldsymbol{\phi}_t\boldsymbol{\phi}_t^{\mathrm{T}} \\ &= \boldsymbol{A}_{t-1} + \boldsymbol{\phi}_t\boldsymbol{\phi}_t^{\mathrm{T}} \ , \end{aligned}$$

from which we can conclude that each additional training sample constitutes a rank-1 update. Similarly, the update of vector $\boldsymbol{b}$ can be described recursively as

$$\begin{aligned} \boldsymbol{b}_t &= \boldsymbol{\Phi}_t^{\mathrm{T}} y_t \\ &= \boldsymbol{\Phi}_{t-1}^{\mathrm{T}} y_{t-1} + \boldsymbol{\phi}_t y_t \\ &= \boldsymbol{b}_{t-1} + \boldsymbol{\phi}_t y_t \ . \end{aligned}$$

Due to these recursive update rules it is no longer necessary to store previous samples in memory, instead only $\boldsymbol{A}_t$ and $\boldsymbol{b}_t$ are required at each time step. However, the above formulation requires an explicit inversion of $\boldsymbol{A}_t$ at each update, causing the time complexity to be in $\mathcal{O}\big(D^3\big)$. This time complexity can be reduced by directly updating the inverse of $\boldsymbol{A}$, which is known as Recursive Least Squares in the field of signal processing (Sayed, 2008). A straightforward method to perform a rank-1 update of the inverse of a matrix is the Sherman–Morrison formula (Hager, 1989), such that

$$\begin{aligned} \boldsymbol{A}_t^{-1} &= \left(\lambda \boldsymbol{I} + \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1} \\ &= \left(\boldsymbol{A}_{t-1} + \boldsymbol{\phi}_t\boldsymbol{\phi}_t^{\mathrm{T}}\right)^{-1} \\ &= \boldsymbol{A}_{t-1}^{-1} - \frac{\boldsymbol{A}_{t-1}^{-1}\boldsymbol{\phi}\boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{A}_{t-1}^{-1}}{1 + \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{A}_{t-1}^{-1}\boldsymbol{\phi}} \ . \end{aligned}$$

Updating the inverse of the matrix $\boldsymbol{A}$ reduces the time complexity of adding a sample to an existing model to $\mathcal{O}\big(D^2\big)$ and only requires the $D \times D$ inverse matrix $\boldsymbol{A}_t^{-1}$ and D-dimensional vector $\boldsymbol{b}_t$

to be stored in memory. The recursion is complete by setting the initial configuration $\boldsymbol{A}_0^{-1} = \frac{1}{\lambda}\boldsymbol{I}$ and $\boldsymbol{b}_0 = \boldsymbol{0}$.

The simplicity and recursive nature of this update procedure has made it very popular for many applications. The main disadvantage of explicitly updating the inverse matrix is sensitivity to roundoff errors (e.g., Björck, 1996, Section 3.1). This problem can be alleviated to large extent by updating the Cholesky factor of $\boldsymbol{A}$ instead (Golub and Loan, 1996; Björck, 1996), known as the QR algorithm in the field of adaptive filtering (Sayed, 2008, Section 35.2). Let us define the upper triangular Cholesky factor $\boldsymbol{R}$ that satisfies for each time step t

$$\boldsymbol{A}_t = \boldsymbol{R}_t^{\mathrm{T}} \boldsymbol{R}_t \ .$$

Note that $\boldsymbol{R}_t$ is full rank and guaranteed to be unique, since $\boldsymbol{A} \succ 0$ for $\lambda > 0$. Following earlier notation, we are interested in the rank-1 update problem

$$\begin{aligned} \boldsymbol{R}_t^{\mathrm{T}} \boldsymbol{R}_t &= \boldsymbol{A}_t \\ &= \boldsymbol{A}_{t-1} + \boldsymbol{\phi}_t \boldsymbol{\phi}_t^{\mathrm{T}} \\ &= \boldsymbol{R}_{t-1}^{\mathrm{T}} \boldsymbol{R}_{t-1} + \boldsymbol{\phi}_t \boldsymbol{\phi}_t^{\mathrm{T}} \ , \end{aligned}$$

where the initial matrix $\boldsymbol{R}_0 = \sqrt{\lambda}\boldsymbol{I}$. This equates to the problem of finding $\boldsymbol{R}_t$ given the previous Cholesky factor $\boldsymbol{R}_{t-1}$ and the new training sample $\boldsymbol{\phi}_t$. This special form[3] of rank-1 update can be computed by reformulating the problem as

$$\begin{aligned} \boldsymbol{R}_t^{\mathrm{T}} \boldsymbol{R}_t &= \boldsymbol{R}_{t-1}^{\mathrm{T}} \boldsymbol{R}_{t-1} + \boldsymbol{\phi}_t \boldsymbol{\phi}_t^{\mathrm{T}} \\ &= \tilde{\boldsymbol{R}}_{t-1}^{\mathrm{T}} \tilde{\boldsymbol{R}}_{t-1} \ , \end{aligned}$$

where the temporary $(D+1) \times D$ matrix

$$\tilde{\boldsymbol{R}}_{t-1} = \begin{bmatrix} \boldsymbol{R}_{t-1} \\ \boldsymbol{\phi}_t^{\mathrm{T}} \end{bmatrix} \ .$$

Considering that $\boldsymbol{R}_{t-1}$ is upper triangular, the matrix $\tilde{\boldsymbol{R}}_{t-1}$ can be illustrated as

$$\begin{bmatrix} r & r & r & r \\ 0 & r & r & r \\ 0 & 0 & r & r \\ 0 & 0 & 0 & r \\ \phi & \phi & \phi & \phi \end{bmatrix} \ ,$$

[3] A more general problem is updating a QR-decomposition with a rank-1 matrix $\boldsymbol{u}\boldsymbol{v}^{\mathrm{T}}$, for which algorithms can be found in Golub and Loan (1996) or Björck (1996).

where the dimensionality of $D = 4$ is chosen solely for demonstration. It follows that the updated Cholesky factor $\boldsymbol{R}_t$ can be obtained by introducing zeros in the last row of $\tilde{\boldsymbol{R}}_{t-1}$. A standard technique to introduce zeros in a matrix is using Givens rotations (Golub and Loan, 1996; Björck, 1996), which perform a rotation in the plane spanned by two coordinate axes. In this case, a total of D rotations are needed to zero the elements in the last row, where each rotation concerns the plane $(D+1, i)$ for $1 \leq i \leq D$. The last row subsequently contains zeros and can be removed from the matrix $\tilde{\boldsymbol{R}}_{t-1}$, such that the updated $D \times D$ Cholesky factor $\boldsymbol{R}_t$ remains. The weight vector $\boldsymbol{w}_t$ can then be obtained using a back and forward substitution. Although updating the Cholesky factor and subsequently solving for $\boldsymbol{w}$ might seem elaborate, the time and space complexity is identical to rank-1 updates of the inverse matrix. The computational cost in absolute sense is also closely related, such that incremental updates of the Cholesky factor should be preferred without exceptions due to its increased numerical stability.

We may observe that the regularization parameter λ is only taken into account in the initial matrix $\boldsymbol{R}_0$ (or $\boldsymbol{A}_0$) and does not enter in subsequent updates. The effect of regularization therefore decreases with an increasing number of training samples. However, note that the role of regularization is to prevent singularity of the covariance matrix by increasing its eigenvalues (Lo Gerfo et al., 2008). This is primarily needed in the initial stage of the algorithm when $\boldsymbol{\Phi}$ is not full column-rank. Subsequent rank-1 updates increase the smallest eigenvalue of $\boldsymbol{R}$ (known as the interlacing theorem, Horn and Johnson, 1986; Björck, 1996), therefore reducing the need for regularization in later stages. Another perspective is to consider regularization as a measure to prevent overfitting. A more intuitive argument from this perspective is that overfitting is less likely as the number of training samples increases.

4.4 Discussion

The approximate KMs described in this chapter compete directly with earlier methods, some of which were described previously in Section 3. An essential property of the proposed incremental methods is that a linear increase of the kernel expansion is explicitly avoided due to a finite dimensional approximation of the kernel function. This approach is fundamentally different from earlier methods, which attempt to approximate the kernel expansion rather than the kernel function. Denoting the desired "full" expansion as

$$f(\cdot) = \sum_{i}^{m} \alpha_i k(\boldsymbol{x}_i, \cdot) \ ,$$

the sparse approximation of the kernel expansion is given by an alternative active set and coefficients $\tilde{\boldsymbol{\alpha}}$, such that

$$\tilde{f}(\cdot) = \sum_{i}^{\tilde{m}} \tilde{\alpha}_i k(\boldsymbol{x}_i, \cdot) \approx f(\cdot) \ ,$$

where typically $\tilde{m} \ll m$. In contrast, approximation of the kernel function can be described as[4]

$$\tilde{f}(\cdot) = \sum_{i}^{m} \tilde{\alpha}_i \tilde{k}(\boldsymbol{x}_i, \cdot) \approx f(\cdot) \ ,$$

where the kernel $\tilde{k}$ is obtained using randomization. In other words, the kernel function and therefore the optimization problem itself is approximated rather than the solution. Although the difference is subtle, this has significant theoretical and practical consequences. First, the approximate kernel $\tilde{k}$ is a kernel function in its own right and it follows that RFRLS and SSGPR are standard KMs. As such, a wealth of theoretical results for linear and kernel RLS or GPR are directly applicable to the methods described here. These results include generalization bounds (e.g., Cucker and Zhou, 2007; Sun and Wu, 2008), stability bounds (e.g., Bousquet and Elisseeff, 2002; Mohri and Rostamizadeh, 2010), learning rates (e.g., Caponnetto and Vito, 2007), as well as regret bounds (e.g., Cesa-Bianchi and Lugosi, 2006; Kakade et al., 2006; Zhdanov and Kalnishkan, 2010). Second, due to the finite dimensionality of the feature mapping it is no longer necessary to utilize the kernel trick. This has a significant impact on the time and space complexity of the algorithms and enables incremental and open-ended learning without additional control mechanisms (e.g., sample removal).

Although methods that use a random feature mapping are KMs in their own right, their approximation of shift invariant kernels is relevant. In essence, the random feature mapping applies a Johnson-Lindenstrauss-type[5] projection from the kernel induced Reproducing Kernel Hilbert Space (RKHS) into a D-dimensional random Hilbert space, or equivalently $\mathcal{H}_K \to \mathbb{R}^D$ (Johnson and Lindenstrauss, 1984). The Johnson-Lindenstrauss lemma states that a set of points in a high-dimensional space can be embedded into a space of much lower dimension in such a way that distances between the points are nearly preserved. This approximation technique is further justified by the fact that the random Hilbert space $\mathbb{R}^D$ is dense in the RKHS of the kernel (Rahimi and Recht, 2008b). Claim 4.2 already demonstrated that the inner product using the feature mapping converges to the kernel product for each $\boldsymbol{x} \in \mathcal{X}$. Moreover, it can be shown that the maximum deviation between RFRLS converges to KRLS as $\mathcal{O}\left(\frac{1}{\sqrt{D}}\right)$ (Rahimi and Recht, 2008b, Theorem 3.2), as demonstrated empirically in Figure 4.2. Given that the update procedure from Section 4.3 is

[4] Although this form is equivalent (cf. Section 2.3), the alternative form $\tilde{f}(\cdot) = \langle \boldsymbol{w}, \phi(\cdot) \rangle$ is computationally more efficient in the setting considered in this thesis.

[5] The original Johnson-Lindenstrauss lemma concerns finite dimensional Euclidean spaces, although it is possible to extend the theory to infinite dimensional Hilbert spaces (Biau et al., 2008).

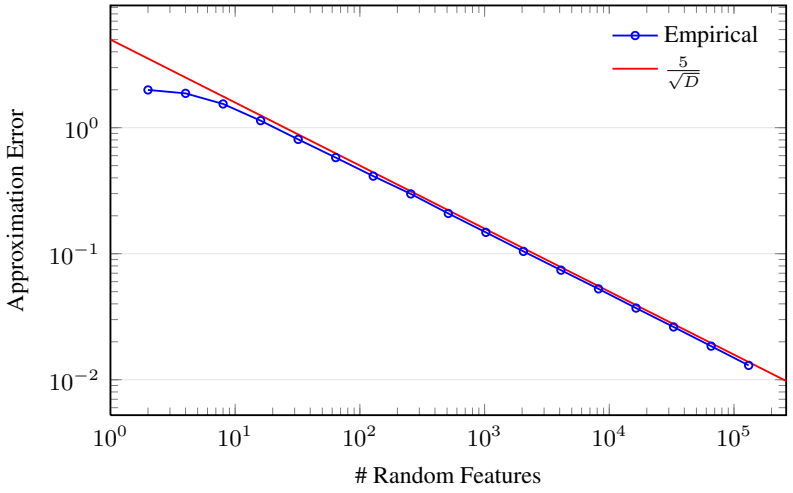


Figure 4.2: Convergence of the maximum error between k and $\langle \phi, \phi \rangle$ over a set of samples with respect to the number of random features D.

exact, it follows that incremental RFRLS produces at each time step a close approximation of batch KRLS trained on all samples observed thus far. Consequently, a single pass over the samples is sufficient and, unlike incremental methods such as stochastic gradient descent, the internal ordering of the samples is irrelevant.

A disadvantage of the described methods is that they are not deterministic due to random initialization of the spectral frequencies. However, randomization is commonly a computationally cheap alternative to optimization (see e.g., Motwani and Raghavan, 1995; Mitzenmacher and Upfal, 2005) and randomized algorithms and projections have been used previously in machine learning in general and KMs in particular (e.g., Achlioptas et al., 2002; Fradkin and Madigan, 2003; Blum, 2006; Balcan et al., 2006; Arriaga and Vempala, 2006). Though the weights $\boldsymbol{w}$ and the spectral frequencies $\boldsymbol{\omega}_i$ for $1 \leq i \leq D$ could be optimized jointly during training, this results in an identical approximation rate at significantly higher computational cost (Rahimi and Recht, 2008b). An alternative perspective along these lines is to consider these methods as a network with a layer of random neurons (or basis functions), to some extent similar to Extreme Learning Machines (Huang et al., 2006) or Reservoir Computing (Lukoševičius and Jaeger, 2009). In our particular case, however, the hidden neurons are restricted to trigonometric activation functions and the sampling distribution for their frequencies is automatically given due to the relation with the kernel function via Bochner's Theorem (cf. Section 4.1). Furthermore, increasing the number of hidden neurons typically causes an increased risk of overfitting in many neural network models. This is not the case in RFRLS and SSGPR, as these methods prevent overfitting by controlling the model complexity either using Tikhonov regularization or a weight prior, respectively. Increasing the number of hidden neurons (i.e., increasing D) therefore improves the quality of the approximation with limited risk of overfitting.

Versatility is another key feature of the proposed methods, as they can be be used both for re-

Algorithm 4.1 Incremental RFRLS with isotropic RBF kernel

Require: $\lambda > 0, \gamma > 0, n > 0, D > 0$

$\boldsymbol{R} \leftarrow \sqrt{\lambda}\boldsymbol{I}_{D\times D}$

$\boldsymbol{w} \leftarrow \mathbf{0}_{D\times 1}$

$\boldsymbol{b} \leftarrow \mathbf{0}_{D\times 1}$

$\boldsymbol{\Omega} \sim \mathcal{N}\left(0, (2\gamma)^2\right)_{D\times n}$

$\boldsymbol{\beta} \sim \mathcal{U}(-\pi, \pi)_{D\times 1}$

for all $(\boldsymbol{x}, y)$ **do**

  $\boldsymbol{\phi} \leftarrow \sqrt{\frac{2}{D}}\cos\left(\boldsymbol{\Omega}\boldsymbol{x} + \boldsymbol{\beta}\right)$ *// Equation* (4.4)

  $\hat{y} \leftarrow \langle \boldsymbol{w}, \boldsymbol{\phi} \rangle$

  $\boldsymbol{b} \leftarrow \boldsymbol{b} + \boldsymbol{\phi} y$

  $\boldsymbol{R} \leftarrow \textsc{CholeskyUpdate}(\boldsymbol{R}, \boldsymbol{\phi})$

  $\boldsymbol{w} \leftarrow \boldsymbol{R} \backslash \left(\boldsymbol{R}^{\mathrm{T}} \backslash \boldsymbol{b}\right)$

  yield $\hat{y}$

end for

gression as well as classification tasks. Though not considered in this thesis, the KRLS algorithm has shown competitive classification performance with respect to state of the art methods such as Support Vector Machine (SVM) (e.g., Rifkin et al., 2003; Rahimi and Recht, 2008a). Furthermore, the algorithm can be trained both in a batch as well as an incremental setting, and initial batch solutions can be subsequently be refined with incremental updates. This aspect is particularly useful for "bootstrapping" an initial model using a batch of training samples, prior to deployment in an incremental setting. The entire training set can be processed in a single time efficient step, contrary to competing methods that require multiple passes over the training samples to obtain a stable model. It is therefore not required to perform multiple passes over an training set. Furthermore, the feature mapping can easily be extended with additional features, for instance to include bias or trend compensation terms.

The pseudocode for incremental RFRLS and SSGPR is demonstrated in Algorithm 4.1 and Algorithm 4.2, respectively. The procedure to update the Cholesky factor was described in Section 4.3. In particular, note the simplicity of these algorithms as compared to algorithms such as LWPR. More so than a mere matter of aesthetics, complicated algorithms are typically harder to comprehend, implement, and apply in practice (for compelling arguments, see Helwig et al., 2010). Arguably, this holds in particular for LWPR, which is often found cumbersome in practical use due to the difficulty of tuning a large number of non-intuitive hyperparameters. In contrast, the methods from this chapter are easy to implement using standard linear algebra routines and require only a small number of intuitive hyperparameters to be configured. Moreover, in case of SSGPR, all hyperparameters (except D, for obvious reasons) can be tuned automatically using log marginal likelihood optimization. In short, minimal effort is required on part of the practitioner to obtain satisfactory generalization performance.

Many competing algorithms erroneously claim to be "real-time", often intending that these algo-

Algorithm 4.2 Incremental SSGPR with anisotropic RBF kernel

Require: $\sigma_n > 0, \sigma_f > 0, \boldsymbol{M} \succ 0, D > 0$

$\boldsymbol{R} \leftarrow \sigma_n \boldsymbol{I}_{2D \times 2D}$
$\boldsymbol{w} \leftarrow \mathbf{0}_{2D \times 1}$
$\boldsymbol{b} \leftarrow \mathbf{0}_{2D \times 1}$
$\boldsymbol{\Omega} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{M})_{D \times n}$
for all $(\boldsymbol{x}, y)$ **do**
  $\boldsymbol{\phi} \leftarrow \frac{\sigma_f}{\sqrt{D}} \left[\cos\left(\boldsymbol{\Omega}\boldsymbol{x}\right)^{\mathrm{T}}, \sin\left(\boldsymbol{\Omega}\boldsymbol{x}\right)^{\mathrm{T}}\right]^{\mathrm{T}}$ // *Equation* (4.6)
  $\hat{y} \leftarrow \langle \boldsymbol{w}, \boldsymbol{\phi} \rangle$
  $\boldsymbol{v} \leftarrow \boldsymbol{R}^{\mathrm{T}} \backslash \boldsymbol{\phi}$
  $s^2 \leftarrow \sigma_n^2 \left(1 + \langle \boldsymbol{v}, \boldsymbol{v} \rangle\right)$
  $\boldsymbol{b} \leftarrow \boldsymbol{b} + \boldsymbol{\phi} y$
  $\boldsymbol{R} \leftarrow \text{CHOLESKYUPDATE}(\boldsymbol{R}, \boldsymbol{\phi})$
  $\boldsymbol{w} \leftarrow \boldsymbol{R} \backslash \left(\boldsymbol{R}^{\mathrm{T}} \backslash \boldsymbol{b}\right)$
  yield $\left(\hat{y}, s^2\right)$
end for

rithms make predictions quickly. This is a common misconception, since more important than being fast, real-time algorithms should be predictable (Buttazzo, 1997). The computational cost of each update in RFRLS and SSGPR, on the other hand, is both bounded (i.e., $\mathcal{O}(1)$) as well as predictable. The number of operations required for each update is quantified in Table 4.1[6]. Note that the number of operations depends primarily on D and remains constant regardless of the number of training samples. In this aspect, the methods proposed here compare favorably to all algorithms described previously in Section 3.2, such as LWPR and Local Gaussian Processes (LGP). The exact computational cost can be controlled directly by varying the number of random features D, which trades computation for approximation accuracy. Tuning this parameter is therefore an effective and direct manner to optimize the generalization performance given domain specific timing constraints. Consequently, the methods are suited for real-time use (in the strict meaning of the term) and could potentially be employed in resource constrained environments (e.g., microcontrollers). In contrast, the maximum computational cost of most other methods (e.g., LWPR or LGP) depends on the distribution of the data and cannot be fixed with certainty within predefined timing constraints[7].

[6] The operation counts do not include overhead due to control routines (e.g., looping) and may vary slightly depending on the particular implementation.

[7] Theoretical bounds will almost surely be too loose for practical use, while empirical estimation is both tedious and inherently inaccurate.

Table 4.1: Typical operation counts for the update procedure of incremental SSGPR as described in Algorithm 4.2.

line #	$*/\div$	$+/-$	$\sin$ / $\cos$	$\sqrt{}$	abs
6	$Dn + 2D$	Dn	$2D$		
7	$2D$	$2D$			
8	$4D^2$	$4D^2 - 2D$			
9	$2D + 1$	$2D + 1$			
10	$2D$	$2D$			
11	$8D^2 + 8D$	$4D^2$		$2D$	$4D$
12	$4D^2 + 2D$	$4D^2 - 2D$			
	$16D^2 + D\,(n + 18) + 1$	$12D^2 + D\,(n + 2) + 1$	$2D$	$2D$	$4D$

5 Generalization and Timing Performance

The proposed incremental learning methods have advantageous theoretical properties over existing methods, such as convergence proofs and a constant time and space complexity for model updates and predictions. Additional benefits with regard to employing the methods in practice should not be underestimated either. In particular, the interpretation of the hyperparameters is straightforward and they can be optimized using a principled procedure that scales well with the number of parameters. In this chapter, these claims are evaluated experimentally in terms of generalization performance, computational requirements, and practicality. The focus in these experiments is the comparison with Locally Weighted Projection Regression (LWPR) and Gaussian Process Regression (GPR). The former method has been used extensively in the robotics community for incremental learning tasks and it is of interest to investigate whether RFRLS and SSGPR are valid alternatives in this application domain. The comparison with GPR, on the other hand, serves as a reference, since both proposed methods – and in particular SSGPR – approximate its generalization performance in the limit. Additionally, this comparison also allows investigation of the advantage of incremental learning over batch solutions such as GPR in realistic learning settings. In such problems, a perfect correlation between training and test data cannot simply be assumed, and incremental learning may therefore have advantages.

A set of three synthetic regression problems is used in Section 5.1 to investigate the behavior of Sparse Spectrum Gaussian Process Regression (SSGPR) with an increasing number of redundant and irrelevant input dimensions. These problems have been used previously for the empirical validation of LWPR and GPR (Vijayakumar et al., 2005). Subsequently, the performance of Random Fourier Regularized Least Squares (RFRLS) is compared in a batch setting with a host of competing methods on three robot dynamics datasets of medium size. The corresponding results, presented in Section 5.2, have been published previously by Gijsberts and Metta (2011). Finally,

in Section 5.3, incremental SSGPR is tested on a number of large dynamics datasets. These experiments demonstrate the generalization performance of this method and the relative advantages of continuous incremental learning over batch models.

5.1 Synthetic Dataset

The first experiment evaluates SSGPR with a varying number of random features on the synthetic Cross datasets, previously used by Vijayakumar et al. (2005) to evaluate the generalization performance and scaling behavior of LWPR. There are three variants of this dataset, where the latter two are constructed by the first with redundant and irrelevant input features. In the base Cross 2D, data is generated according to a two-dimensional function

$$f(\boldsymbol{x}) = \max\left\{e^{-10\boldsymbol{x}_1^2}, e^{-50\boldsymbol{x}_2^2}, 1.25e^{-5\left(\boldsymbol{x}_1^2+\boldsymbol{x}_2^2\right)}\right\} \quad .$$

As seen in Figure 5.1, this function is characterized by a mixture of areas of both low and high curvature. As claimed by Vijayakumar et al., this high variability makes the function interesting for evaluating generalization performance of learning methods. Low complexity algorithms tend to have difficulties to accurately capture the non-linearities in high curvature regions, in contract to high complexity models that overfit on the regions of low curvature. In Cross 10D, eight constant features are added to the input space by applying a random 10-dimensional rotation matrix, effectively causing a high degree of redundancy in the input space. For the third variant (Cross 20D), the input space of Cross 10D is extended with an additional ten irrelevant input features, each of which consists of $\mathcal{N}\left(0, 0.05^2\right)$ random noise. In all three cases, the training data consists of 500 randomly drawn samples, corrupted by $\mathcal{N}\left(0, 0.1^2\right)$ random noise. The 1681 test samples, on the other hand, are noiseless and distributed uniformly on a 41×41 grid in the unit square of the input space.

5.1.1 Experimental Setup

The synthetic datasets are used as a benchmark to compare SSGPR with LWPR and GPR. For the former competing method, the initial distance metric *init_D*, learning rate *init_alpha*, and meta learning rate *meta_rate* hyperparameters are optimized greedily using a derivative of the procedure described in the supplementary documentation[1]. Additionally, the activation threshold for adding local models *w_gen* is set to 0.2 and the threshold for adding new projects *add_threshold* is set to 0.9, as per suggestion of Vijayakumar et al. (2005). The measure used to select the optimal configuration is the average one-step-ahead squared error over a total of 200 epochs, each of which consists of a random permutation of the 500 training samples. The final generalization

[1] `http://www.ipab.inf.ed.ac.uk/slmc/software/lwpr/lwpr_doc.pdf`

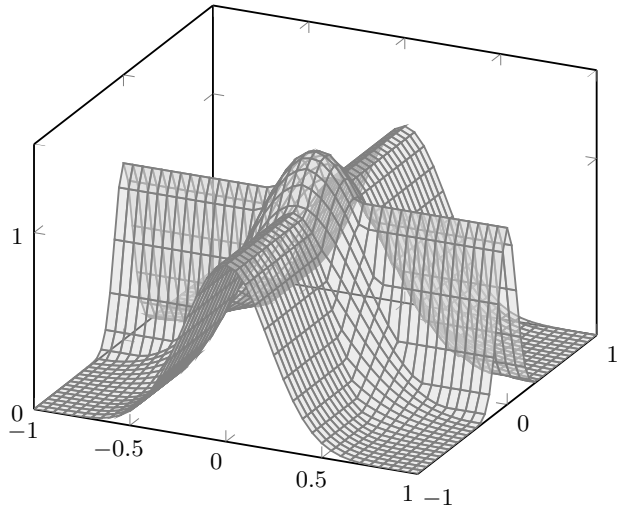


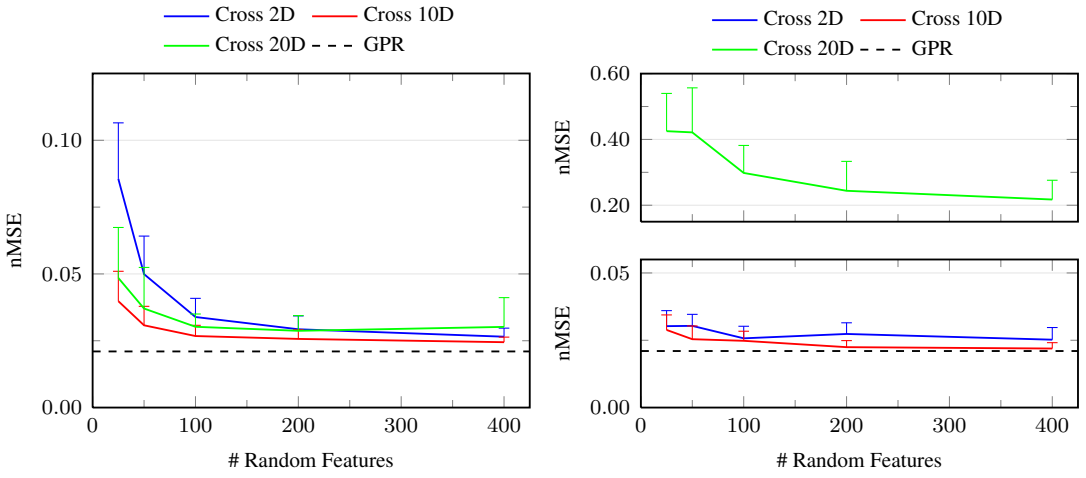
Figure 5.1: Plot of the synthetic Cross 2D problem.

performance instead is measured in terms of the normalized Mean Squared Error (nMSE), defined as the squared error on the test set divided by the variance of the target outputs.

The hyperparameters of GPR are optimized using log marginal likelihood maximization (cf. Section 2.6). Automatic Relevance Detection (ARD) is employed by means of the Asymmetric Squared Exponential (ASE) function, such that the tunable hyperparameters of GPR and the kernel are σ_n, σ_f, and n characteristic length-scales (i.e., one for each of the input features). As initial hyperparameter configuration, σ_n and σ_f are set to 0.25 and 1 times the standard deviation of the target outputs, respectively, and ℓ_i is initialized to twice the range of the i-th input dimension for $1 \leq i \leq n$. The mean of the training outputs is used as offset to ensure a zero-mean Gaussian process. SSGPR is used in a nearly identical setup, the only difference being an additional boolean indicating whether or not the spectral frequencies are optimized. If this is not the case, then the tunable hyperparameters are identical to those for GPR. If the frequencies are instead optimized, however, then the number of tunable parameters increases drastically with an additional $D \times n$ hyperparameters. Parameter D, denoting the number of sparse spectrum features, is configured by the user a priori and not subject to optimization.

5.1.2 Results

The results shown in Figure 5.2 demonstrate good generalization performance for GPR, achieving an nMSE of approximately 0.02 on all three datasets. We can conclude that GPR with ASE kernel remains effective also in case of redundant and irrelevant input features. As expected in case of SSGPR with "fixed" spectral frequencies, increasing the number of sparse spectrum features improves the approximation and therefore the generalization performance in nearly all cases. Interestingly, however, the method requires more random features to obtain satisfactory perfor-



(a) Fixed **(b)** Non-Fixed

Figure 5.2: Test error for SSGPR with respect to the number of sparse spectrum features on three variants of the Cross dataset. For the sparse spectrum features, the results show when (a) the frequencies are fixed and (b) the frequencies are optimized with the other hyperparameters. In both cases, the reported results are the average over 25 runs, with error bars indicating one standard deviation. The test error of GPR is reported as a reference; since this error is nearly identical on all three datasets it is reported only once.

mance with Cross 2D than with the higher dimensional variants. This may be explained by the small number of tunable hyperparameters in the two-dimensional case, which makes it harder to effectively tune the model to the data. Increasing the number of random features, however, is an adequate measure to alleviate this issue.

The generalization performance of SSGPR in case of tuned spectral frequencies is impressive on the Cross 2D and Cross 10D datasets, noting that a mere 25 random features are sufficient to achieve competitive results. However, the opposite is true in case of Cross 20D, for which the performance is significantly inferior with respect to the competing methods. These large test errors in combination with near-zero training errors indicate overfitting of the model to the data, as observed as well by Lázaro-Gredilla et al. (2010) on a selected number of datasets. This is easily understood considering the large number of tunable hyperparameters. In case of 25 random features the number of hyperparameters is already over 500, which is larger than the actual number of training samples. This number increases to over 8000 when $D = 400$. Optimizing such a large number of parameters is error prone, especially given the large fraction of hyperparameters related to irrelevant input features.

The results of LWPR can be seen in Figure 5.3. Although it proved impossible to replicate the results reported by (Vijayakumar et al., 2005)[2], we clearly see that an increasing number of input dimensions has a detrimental effect on the convergence rate and final performance. LWPR ulti-

[2]This may be due to unreported parameter settings by Vijayakumar et al., or due to the fact that here we consider a single random instance of the dataset.

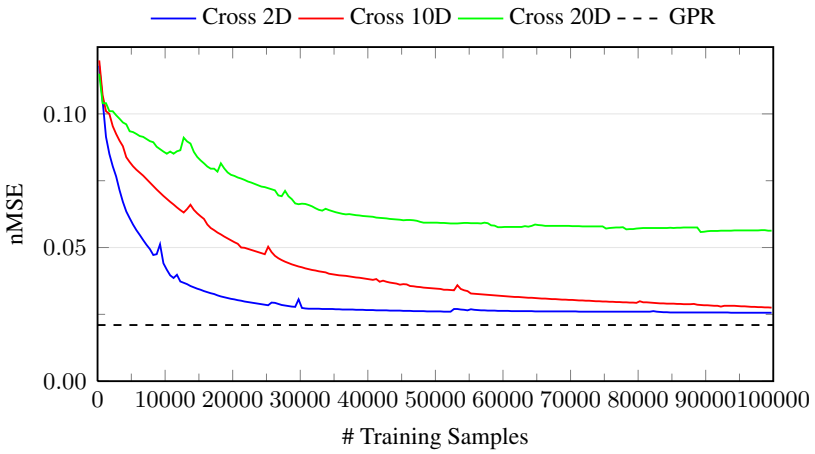


Figure 5.3: Convergence of the test error for LWPR with an increasing number of reiterations of the training samples.

mately attains similar performance to GPR and SSGPR on the Cross 2D and Cross 10D datasets, although it requires a large number of iterations over the training samples. Conversely, the two latter methods require only a single pass on the training data. This is a strong indicator of low sample complexity of LWPR, as will be confirmed in the incremental experiments in Figure 5.3. Additionally, hyperparameter optimization for LWPR is significantly more involved than for GPR and SSGPR, requiring "trial-and-error" tuning of multiple parameters. Moreover, for several of the hyperparameters there is no intuitive understanding of modifications will affect generalization performance. All of these issues have severe implications for the practicality of the algorithm.

5.2 Batch Learning of Inverse Dynamics

Besides performance on synthetic datasets, it is relevant to evaluate performance on realistic robotics datasets. To this extent, several experiments have been conducted to evaluate the performance of RFRLS in a batch setting on the task of learning inverse manipulator dynamics (Gijsberts and Metta, 2011). The three datasets, collected from a Simulated Sarcos arm, a real Sarcos arm, and a Barrett WAM, share an identical 7-DoF manipulator configuration with a torque sensor placed in each individual joint[3]. A compelling reason for using these exact datasets is that they have been used previously in the robotics community, such that RFRLS can directly be compared to a number of state-of-art learning methods (Nguyen-Tuong et al., 2008a, 2009). Additionally, these datasets consist of considerably more samples than the earlier synthetic datasets (cf. Table 5.1), allowing evaluation of the scaling properties of the methods. For all datasets, the manipulator was programmed to perform a periodic motion, during which samples were collected at a rate of approximately 500 Hz. Training and test sets were obtained by subsampling the resulting

[3]The authors are grateful to Nguyen-Tuong et al. for sharing these datasets.

Table 5.1: Datasets used for the batch dynamics experiments.

	#joints	output	#train	#test
Simulated Sarcos	7	$\tau \times 7$	14904	5520
Sarcos	7	$\tau \times 7$	13922	5569
Barrett	7	$\tau \times 7$	13572	5000

dataset at regular intervals.

5.2.1 Experimental Setup

Data preprocessing and hyperparameter selection often has a profound effect on the performance of learning methods. In case of learning manipulator dynamics, it is crucial to properly scale the input features when using an (isotropic) Radial Base Function (RBF) kernel, as to avoid large acceleration features to dominate lower-valued position features. It is therefore desirable to divide all input features by a characteristic length scale $\ell_i > 0$ for $1 \leq i \leq n$, as is done in the ASE kernel. However, the discriminative nature of RFRLS prevents direct optimization of the hyperparameters using the log marginal likelihood. Empirical optimization using grid-search, on the other hand, scales exponentially with the number of hyperparameters and is therefore computationally prohibitive when using a large number of hyperparameters. The regularization tradeoff λ and the characteristic length scales can, however, be optimized indirectly exploiting the similarity between GPR and RFRLS. Log marginal likelihood maximization is used to optimize the hyperparameters of a "proxy" GPR trained on a random subset of 2000 training samples using the ASE kernel (cf. Equation (2.20)) The locally optimal hyperparameters for this "proxy" are subsequently converted to Kernel Regularized Least Squares (KRLS) and RFRLS with *isotropic* RBF kernel by scaling each i-th input feature by ℓ_i^{-1} for $1 \leq i \leq n$, and setting $\lambda = \sigma_n^2/\sigma_f^2$ and $\gamma = \frac{1}{2}$.

The dynamics datasets have multiple outputs and, ideally, a distinct model with corresponding optimal hyperparameters should used for each output dimension. Preliminary experiments, however, showed that all outputs share similar dependencies on hyperparameters. In other words, configurations that perform well for one output are also likely to perform well on the other outputs. This is not surprising, since the outputs are measured in identical physical quantities and units of measurement (i.e., torque in N m). The computational cost of our method can therefore be reduced significantly by sharing a single hyperparameter configuration for all outputs. In this case, we compute a single kernel or covariance matrix, which is subsequently used to solve multiple weight vectors concurrently (i.e., one for each output) at negligible additional cost. For RFRLS, the gain in computation time could be used to improve prediction accuracy by increasing the number of random features. From the optimal hyperparameter configurations for each output, the configuration with the lowest mean error as measured using cross validation on the remaining training samples is selected. Identical configurations were used both for KRLS and RFRLS.

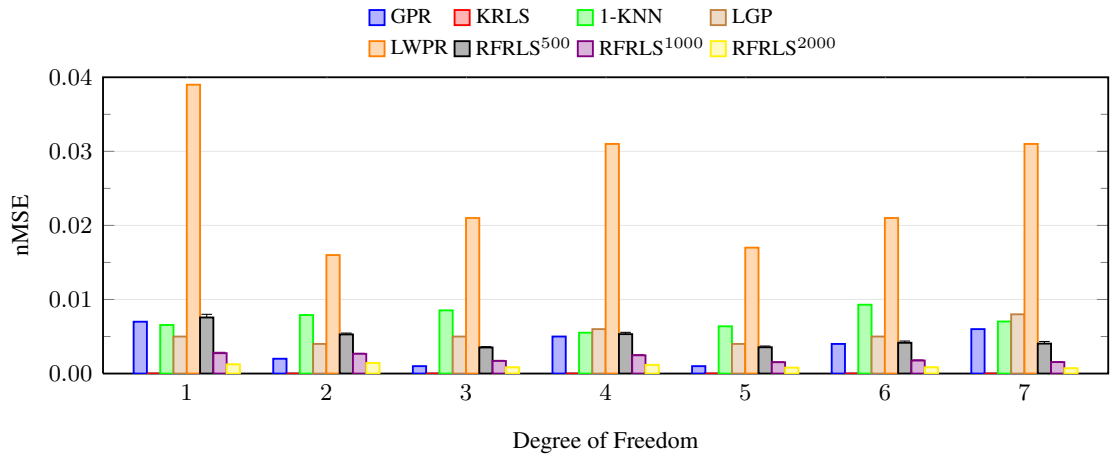
5.2.2 Generalization Performance

The experimental comparison on the Simulated Sarcos, Sarcos, and Barrett datasets is based on earlier published work by Nguyen-Tuong et al. (2009). Their results for LWPR, GPR, and Local Gaussian Processes (LGP) are used as a benchmark[4] and identical experiments were performed with batch RFRLS using 500, 1000, and 2000 random Fourier features (i.e., parameter D). KRLS is included as an additional reference method, since it is an (approximate) upper bound on the generalization performance of RFRLS. A minor deviation from the experimental setup described previously is that for RFRLS^{500} on the Barrett dataset the results using optimal hyperparameter configurations for each distinct output are reported. With this limited number of random features, results using a shared hyperparameter configuration for all outputs were unsatisfactory.

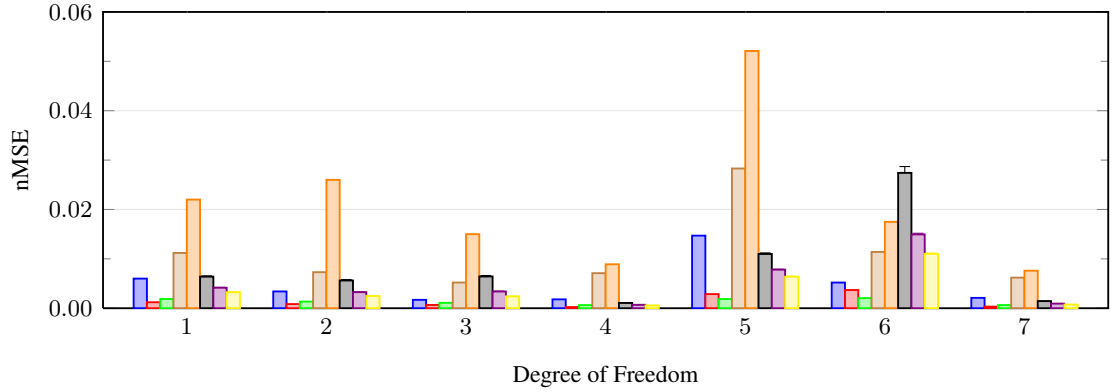
The test nMSEs reported in Figure 5.4 show that, contrary to our expectations, KRLS often outperforms GPR by a significant margin. These two methods are functionally identical and can be expected to show similar performance. These deviations indicate that different hyperparameter configurations were used, which can be explained by different starting configurations combined with the non-convexity of the marginal likelihood optimization problem. Hence, we have to be cautious when interpreting the comparative results on these datasets with respect to generalization performance. The comparison between KRLS and RFRLS, both trained using identical hyperparameters, remains valid and gives an indication of the approximation error of RFRLS. As expected, the performance of RFRLS steadily improves as the number of random features increases for all datasets. Furthermore, RFRLS^{1000} is often sufficient to obtain satisfactory predictions on all datasets. RFRLS^{500}, on the other hand, performs poorly on the Barrett dataset, despite using a distinct hyperparameter configuration for each degree of freedom. In this case, RFRLS^{1000} with a shared hyperparameter configuration is more accurate and requires less overall time for prediction.

The minimal prediction errors of KRLS in general are due to training and test sets being subsampled interleavingly from the same motion, causing a strong correlation between both sets. Combined with noise filtering and dense sampling, it is inevitable that there is a nearly identical training sample for each test sample. This is clearly demonstrated in Figure 5.5, which contains a subset of the union of training and test samples for the Barrett dataset. The high similarity between both subsets and the absence of noise signifies that there is virtually no risk of overfitting for these datasets. To investigate this issue further, additional experiments were performed using Kernel Nearest Neighbor (1-KNN) (Yu et al., 2002). This method is commonly used for classification problems and simply returns the label associated with the most similar training sample as predicted output. Consequently, the performance of 1-KNN is critically dependent on strong similarity between training and test distributions, dense sampling, and noise-free outputs. Also 1-KNN, when used with the same kernel as KRLS, outperforms previously published results in nearly all cases. This confirms that the similarity between training and test samples is very high

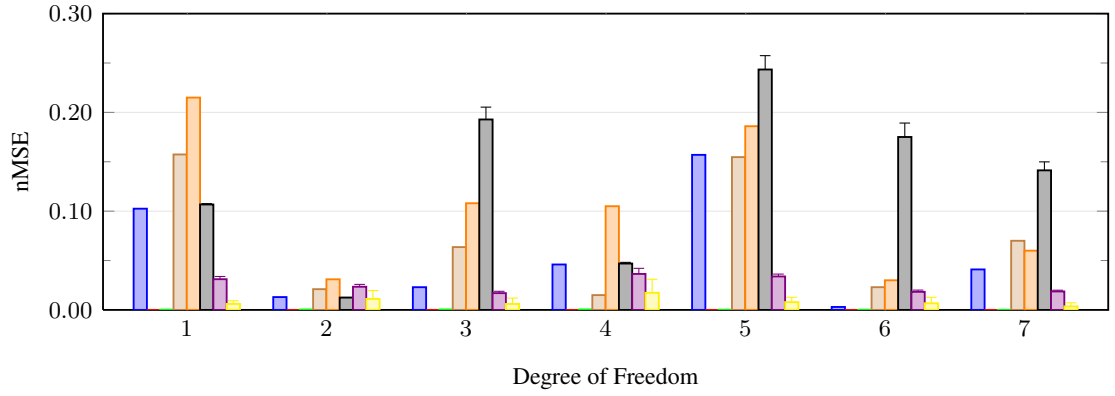
[4] Published results for other methods are comparable to the considered methods and have been left out for conciseness.



(a) Simulated Sarcos

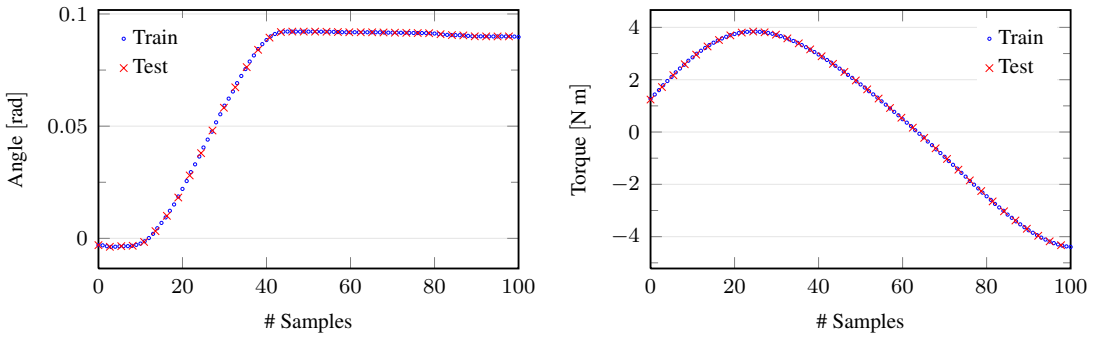


(b) Sarcos



(c) Barrett

Figure 5.4: Prediction error per degree of freedom for the (a) Simulated Sarcos, (b) Sarcos, and (c) Barrett datasets. The results for GPR, LGP, and LWPR are taken from Nguyen-Tuong et al. (2009). The mean error over 25 runs is reported for RFRLS with $D \in \{500, 1000, 2000\}$, whereas error bars mark a distance of one standard deviation. Note that in some cases the prediction errors for KRLS and 1-KNN are very close to zero and therefore barely noticeable.



(a) q_1 **(b)** τ_1

Figure 5.5: Combination of initial training and test samples for the Barrett dataset, after adjustment for different sampling intervals. The figure is restricted to the first joint angle q_1 and joint torque τ_1, since all input features and outputs show identical behavior.

and small predictions errors therefore do not necessarily indicate good generalization performance on these datasets.

5.2.3 Computational Requirements

The second quantitative measure for comparison is the computational performance of all methods. Figure 5.6 shows the prediction times of the various methods when increasing the number of training samples. Note that the timing results for GPR, LWPR, and LGP were measured on different hardware and, consequently, may vary up to a constant factor when compared to the timing results for KRLS, 1-KNN, and RFRLS. Nonetheless, Figure 5.6 shows that the prediction times of the exact kernel methods (i.e., GPR, KRLS, 1-KNN) scale linearly with the number of training samples. Both LWPR and LGP scale sublinearly, though still outperformed by RFRLS due to its constant time complexity. Furthermore, also in absolute sense the prediction times of RFRLS are impressive, measuring from approximately $80\,\mu s$ to $200\,\mu s$ for respectively 500 and 2000 random features. Combined with the prediction results from Figure 5.4, we can confirm that parameter D effectively trades computation for prediction accuracy. This property is particularly useful for real-time control, as it allows to maximize accuracy given strict timing constraints.

5.3 Incremental Learning of Inverse Dynamics

The previous experiments evaluated the performance of SSGPR and RFRLS in a batch learning context. Although this setting is useful to analyze and compare generalization performance, the primary interest in this thesis is their performance for incremental (or online) learning. The advantages of incremental learning over batch learning were detailed in Section 2.8. In short, incremental learning allows scaling to very large numbers of samples as well as adaptation to changing con-

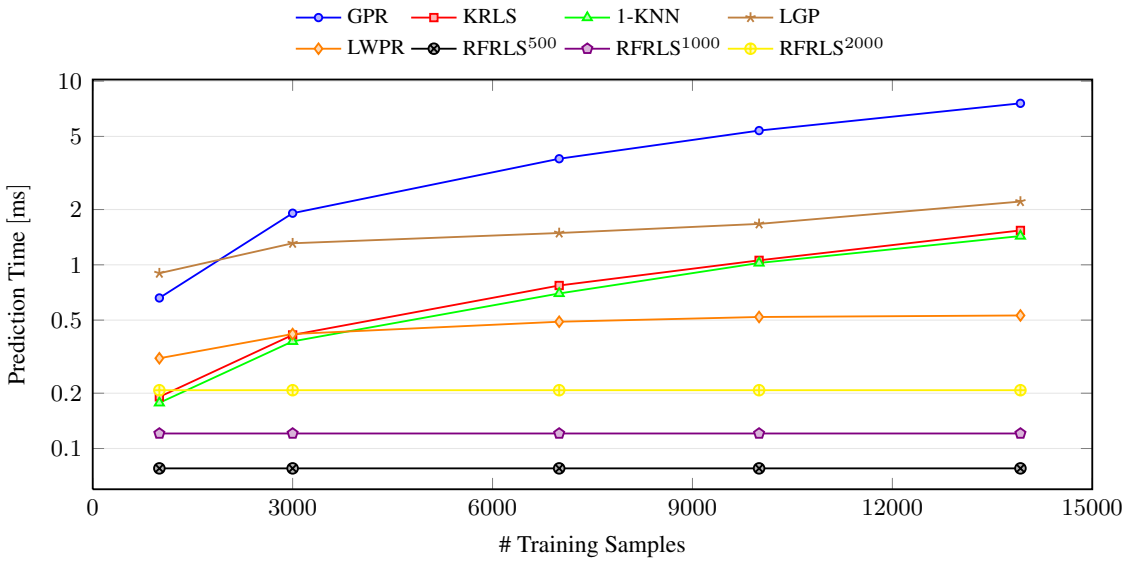


Figure 5.6: Prediction time for all seven output labels with respect to the number of training samples. The results for LWPR, GPR, and LGP are taken from Nguyen-Tuong et al. (2009) and could vary by a constant factor as compared to KRLS, RFRLS and 1-KNN. Note the log scale on the y-axis.

ditions. The methods are evaluated in this incremental scenario using the three dynamics datasets listed in Table 5.2. These datasets are an order of magnitude larger than the previous datasets and describe several minutes up to over an hour of continuous operation of a robot. The first dataset is a larger variant of the Sarcos dataset[5], whereas the second and third datasets consider robot arms of four Degrees of Freedom (DoF)[6]. In this latter configuration, a single force/torque sensor is mounted in the upper arm, positioned just below the shoulder joints, as can be seen in Figure 1.3. These datasets were obtained from the upper torso humanoid James and the full body iCub humanoid (cf. Section 1.1). Fumagalli et al. have previously used the James dataset for a comparison between learning methods and an analytical Rigid Body Dynamics (RBD) model, as described in Section 2.7. For both datasets, the end effector was moved to random Cartesian coordinates in the robot's workspace at constant intervals, while samples were collected at a frequency of roughly 50 Hz. Joint velocities (accelerations) are computed using (double) numerical differentiation of the joint positions.

The objective of these experiments is twofold. The first aim is to demonstrate the performance of SSGPR when applied incrementally on large datasets. In order to have a sufficiently large number of test samples, the traditional subdivision of training and test set of the Sarcos dataset has been reversed. For the James and iCub datasets, the first 15000 samples are reserved for training and hyperparameter tuning, whereas the remaining samples are used to test both generalization performance and computational efficiency. The secondary aim of these experiments is to experimentally verify the benefit of incremental learning in general in a *realistic* learning scenario.

[5]This dataset can be downloaded from `http://www.gaussianprocess.org/gpml/data/`.

[6]We limit ourselves to the shoulder and elbow joints; the actual robot arms have more than four Degrees of Freedom.

Table 5.2: Datasets used for the incremental dynamics experiments.

	#joints	output	#train	#test
Sarcos	7	$\tau \times 7$	4449	44484
James	4	$[F, \tau]_{x,y,z}$	15000	195977
iCub	4	$[F, \tau]_{x,y,z}$	15000	72850

5.3.1 Experimental Setup

The experimental setup resembles the setup for the synthetic datasets as explained in Section 5.1.1 and SSGPR is again compared directly with LWPR and GPR. However, the number of sparse spectrum features is limited to a maximum of $D = 200$ and optimization of the spectral frequencies (i.e., "non-fixed") is only done when the number of random features is less than 100. The sole purpose of these limitations is to contain the computational requirements of these large scale experiments within reasonable margins. Another notable difference in the setup is that hyperparameter optimization for LWPR is based on 5 epochs of the training data, as opposed to 200 for the synthetic datasets. This modification is justified by the vast increase of number of training samples for these experiments. Lastly, the number of training samples for hyperparameter optimization of GPR is limited to a random subset of 5000 samples for the James and iCub datasets. Due to the random subsampling and large number of samples, this restriction is not expected to have a significant effect on the results. After hyperparameter optimization, the GPR is nonetheless trained on the full training set of 15000 samples.

The interpretation of training and test set is slightly different for the incremental methods, as in this setting there is no clear separation between training and test samples. At each time step, a sample is initially used for testing and the method is subsequently allowed to update its model using the target output (i.e., training). The abovementioned training sets are used solely for hyperparameter optimization in case of LWPR and SSGPR, after which the models are reset while retaining the hyperparameter configuration. This methodology allows us to observe the error convergence of the methods on the test set from an initial zero state.

Multiple output models were learned congruently in RFRLS using a shared set of hyperparameters, as explained in Section 5.2.1. This strategy cannot be applied to GPR or SSGPR, in particular with respect to the James and iCub datasets. The reason is that for these datasets the outputs regard two different physical quantities, namely forces (N) and torques (N m). The absolute magnitude and range of the torques is significantly smaller than the forces and, more importantly, their (absolute) level of noise is smaller as well. Noise parameter σ_n relates directly to the magnitude of this (Gaussian) noise and can therefore not be shared between torque and force outputs. Contrary to RFRLS, a realistic approximation of σ_n is crucial for GPR and SSGPR in order to accurately estimate the predictive variance. For these reasons, separate models are trained for outputs with distinct physical quantities. As a result, all 7 outputs of the Sarcos dataset are learned using a

single model, whereas two models are trained for each of the James and iCub datasets (i.e., one for forces and one for torques). During hyperparameter optimization, the log marginal likelihood and its gradient over all outputs are taken as the sum of the same quantities for the individual outputs.

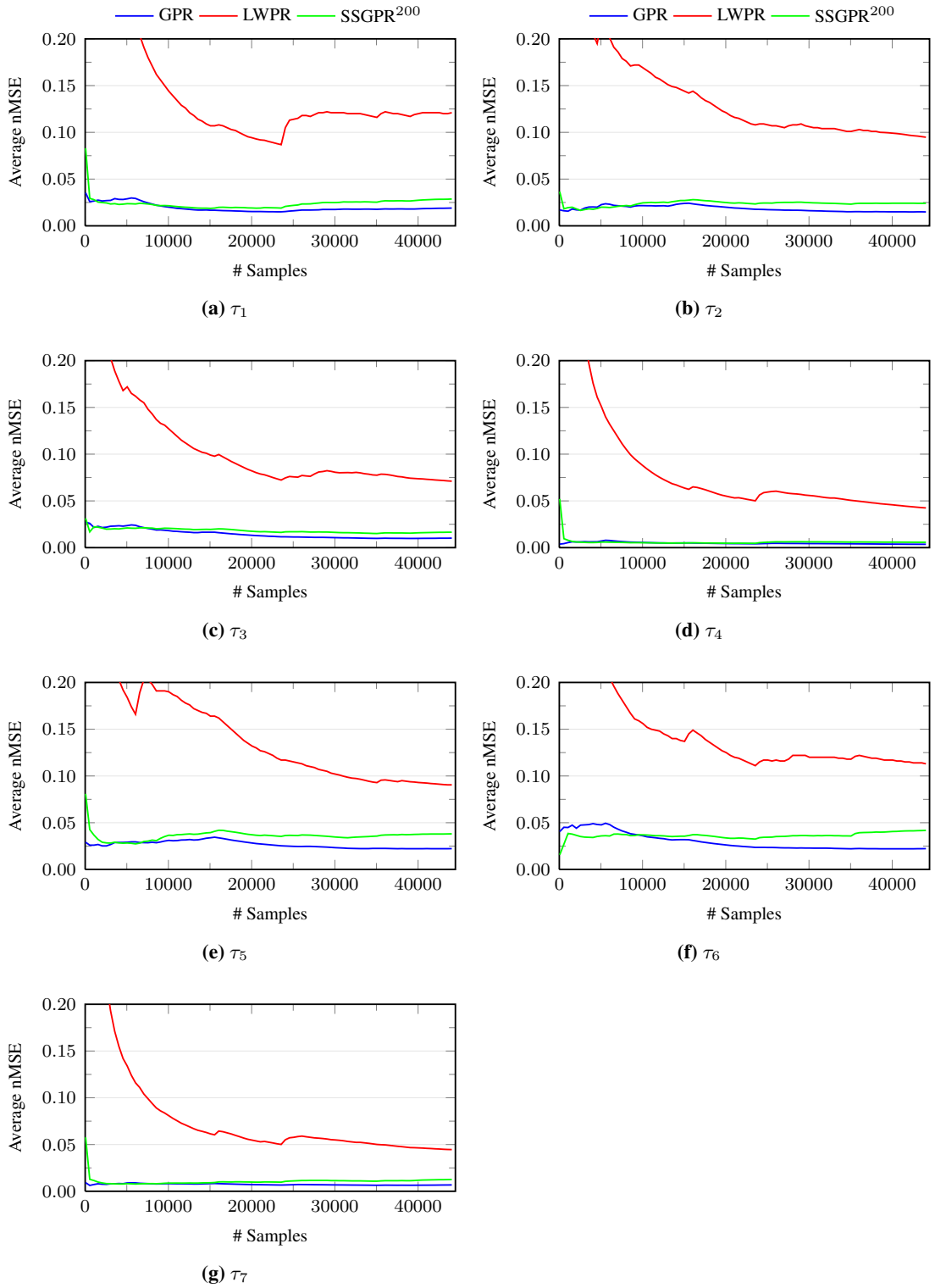
Furthermore, an analytical RBD model is included in the experiments for the iCub dataset as additional reference method[7]. This model uses the Recursive Newton-Euler Algorithm (RNEA) to compute the inverse dynamics given a manipulator configuration (Featherstone, 2007; Siciliano and Khatib, 2008). Inclusion of this analytical model allows us not only to compare the learning methods with respect to each other, but also to compare learned models with respect to analytical solutions in terms of predictive accuracy and computational requirements. Several reasons why learned models may be preferred over analytical ones were mentioned previously in Section 2.8. Although no learning is required for this model, the initial 15000 training samples are used to calibrate the offset of the sensor.

5.3.2 Generalization Performance

The generalization performance for these experiments is quantified by the nMSE calculated over the one-step-ahead (or recursive) predictions. Figure 5.7 shows the convergence of the average error with respect to an increasing number of test samples for the two reference methods and SSGPR with 200 sparse spectrum features. This instance has been selected for comparison as it showed significant improvement over other instances with fewer random features. This can be verified in Table 5.3, which lists the average error over all test samples for all considered instances of SSGPR and the competing reference methods. The results in Figure 5.7 show that SSGPR converges within only hundreds of samples and maintains generalization performance that is similar to GPR. Furthermore, SSGPR^{50} with optimized frequencies improves further on these results on a subset of the outputs, although it is outperformed by SSGPR^{200} with fixed frequencies on the remaining output dimensions. In any case, these results show that a relatively limited number of random features are sufficient to obtain satisfactory generalization performance on this dataset. LWPR, on the other hand, is characterized by slow convergence and requires a large amount of training samples. After the entire test set of nearly 45000 samples its performance lacks significantly with respect to GPR and SSGPR.

The behavior of LWPR on the James dataset is similar to the Sarcos dataset. Error convergence is again slow with respect to the other methods and the final performance is significantly worse. The comparison between GPR and SSGPR, however, is significantly different from the Sarcos dataset. SSGPR^{200} outperforms GPR after less than 1000 test samples and obtains significantly better final performance for all outputs. The different results on these two datasets can be contributed to the different sampling strategies. As mentioned previously, the Sarcos dataset is split in training and test sets by subsampling the total dataset at different intervals. Consequently, there is a strong

[7] The analytical model for the iCub dynamics is implemented in the *iDyn* software library, for more information see `http://eris.liralab.it/wiki/IDyn`.



(a) τ_1 **(b)** τ_2

(c) τ_3 **(d)** τ_4

(e) τ_5 **(f)** τ_6

(g) τ_7

Figure 5.7: Convergence of the average one-step-ahead prediction error of GPR, LWPR, and SSGPR^{200} methods on the Sarcos dataset. The results for SSGPR^{200} are the average error over 25 randomized runs. The standard deviation over the various runs is negligible and error bars are therefore omitted for clarity.

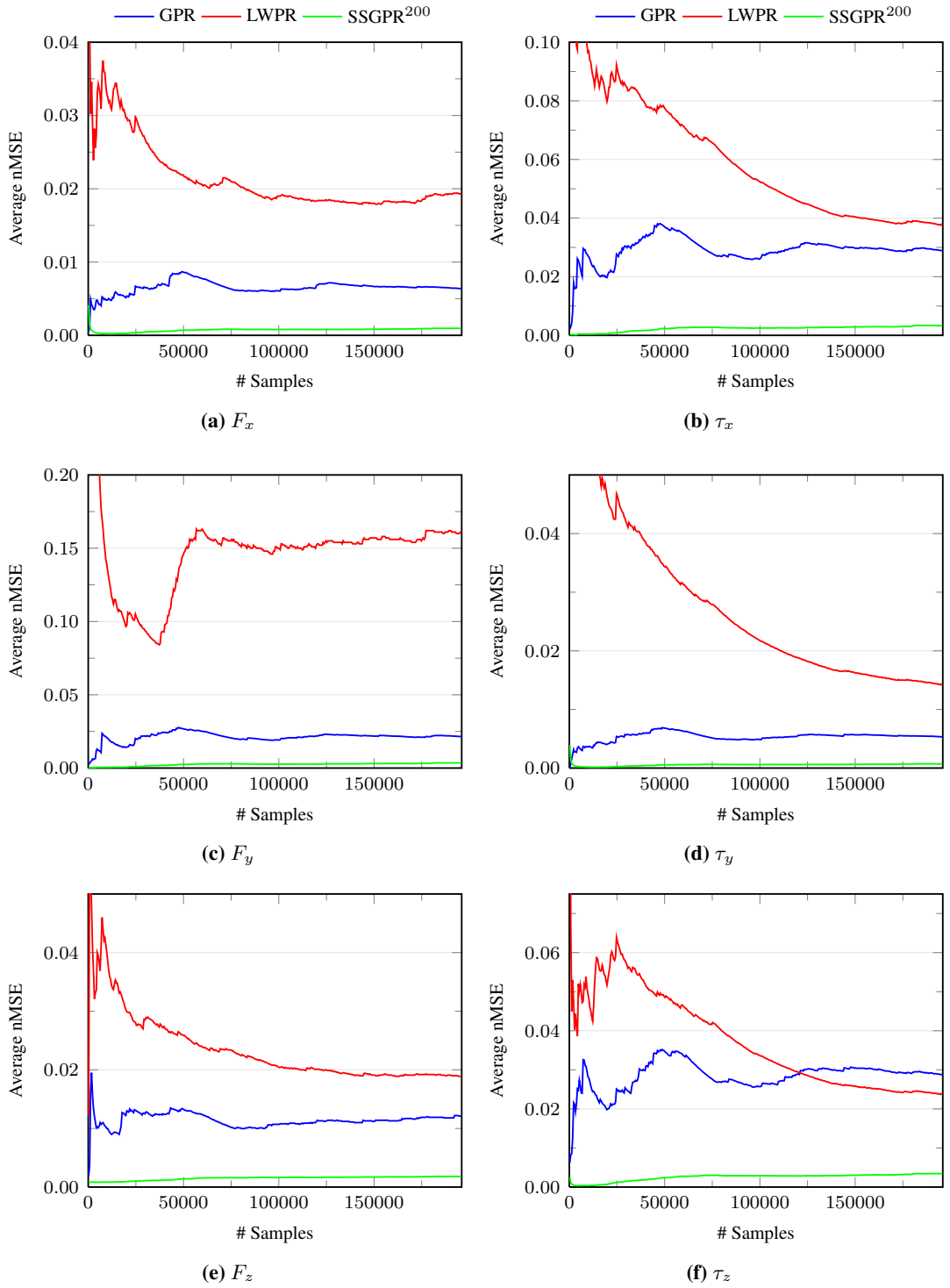
correspondence between both subsets. For the James dataset, on the other hand, the training set consists of the first 15000 samples of the total dataset. The presence of temporal dependencies will cause a divergence in the probability distribution of training and test set, such that training data is no longer completely representative for subsequent test data (cf. Section 2.8). This is evident from the fact that the average error of GPR actually increases over time. Incremental methods are much less affected by this divergence, since by nature any test data subsequently becomes training data as well. This ability to adapt to changes significantly improves online generalization performance on this dataset, demonstrated by the fact that even LWPR outperforms GPR in one case (cf. Figure 5.8f).

Figure 5.9 shows the generalization performance on the iCub dataset, which are similar to the results on the James dataset. This is not surprising, given the fact that James and iCub share the same manipulator configuration and both datasets were collected and subsampled in identical manner. There are some notable differences, however. Firstly, the force and torque measurements for James are filtered on the sensor, whereas these measurements are unfiltered for iCub. The desired outputs in case of the former robot are therefore inherently less noisy, evidenced by nMSEs that are nearly an order of magnitude smaller than for iCub. Secondly, 50 random features are sufficient to obtain satisfactory performance on the iCub dataset using SSGPR. Increasing the number of random features further only results in a marginal improvement of the prediction error (cf. Table 5.3). Figure 5.10 shows an extract of the three torque measurements and the corresponding predictive mean and variance of SSGPR^{50}. We can verify visually that, despite the low number of sparse spectrum features, the method accurately predicts the sensor measurements and that the predictive variance is a reasonable estimate of the true data variance.

The analytical RBD model demonstrates acceptable performance with respect to the learned models. It consistently outperforms LWPR, although LWPR has not yet reached stable performance at the end of the dataset. Interestingly, RBD demonstrates better predictive performance than GPR for two of the six outputs (i.e., F_2 and F_3). An advantage of the analytical model is that it incorporates knowledge of the entire input domain, whereas the learned GPR model is restricted to information present in the training data. SSGPR, however, utilizes all available samples for training and shows superior performance over all competing methods.

5.3.3 Compensation for Sensor Drift

An interesting observation in Figure 5.9 is that the force predictions of the non-adaptive GPR and RBD models deteriorate over time. In case of GPR, one might argue that this is due to a discrepancy between training and test data. However, this does not explain why torque predictions do not suffer from the same problem, nor does it account for the fact that we observe identical behavior for the analytical RBD model. A more reasonable explanation is sensor drift for the force measurements due to unmodeled variables (e.g., temperature or electromagnetic interference). Further insight in this problem is given by Figure 5.11, which shows the average one-step-ahead (or re-

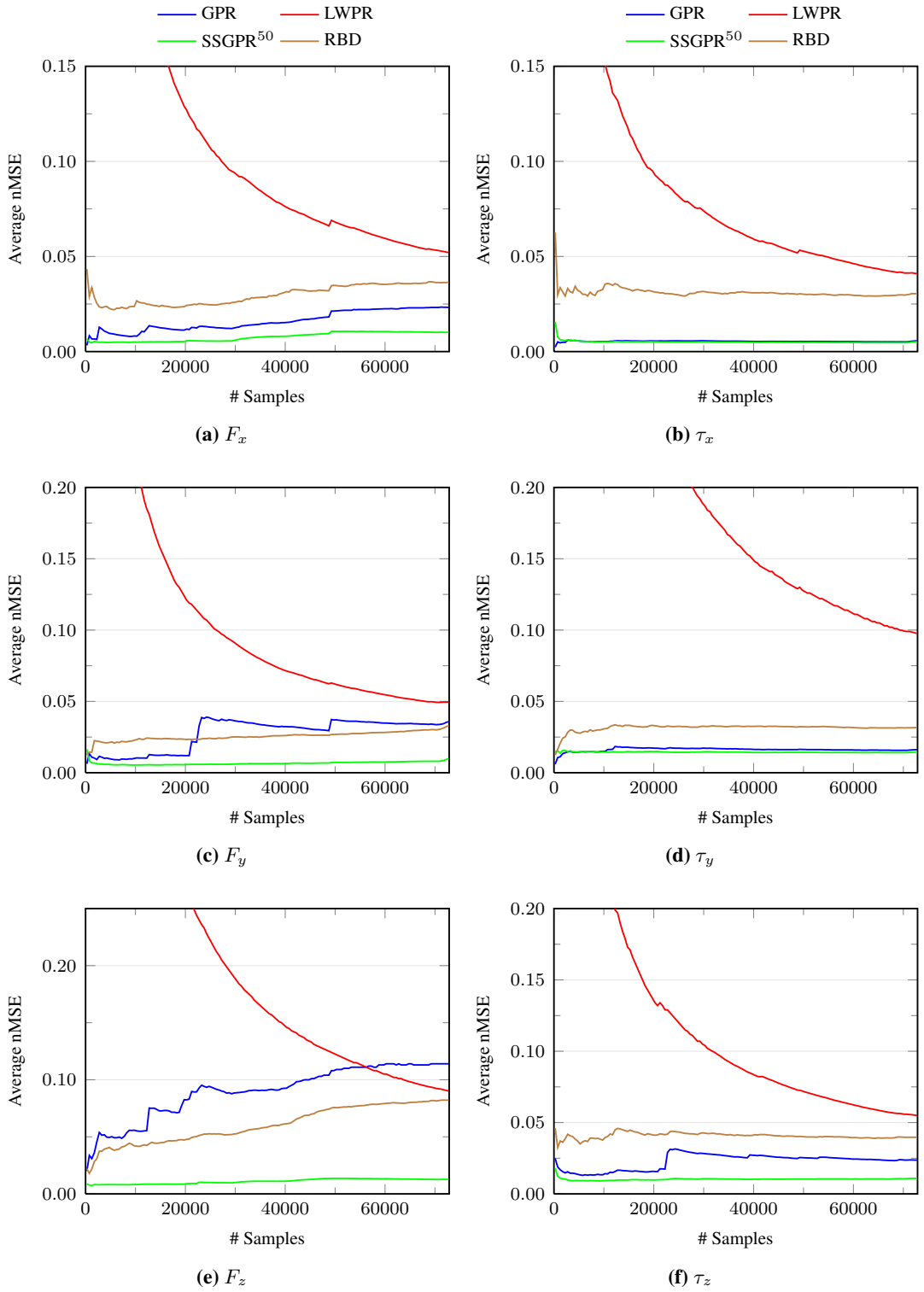


(a) F_x **(b)** τ_x

(c) F_y **(d)** τ_y

(e) F_z **(f)** τ_z

Figure 5.8: Convergence of the average one-step-ahead prediction error of GPR, LWPR, and SSGPR^{200} methods on the James dataset. The results for SSGPR^{200} are the average error over 25 randomized runs. The standard deviation over the various runs is negligible and error bars are therefore omitted for clarity.



(a) F_x **(b)** τ_x

(c) F_y **(d)** τ_y

(e) F_z **(f)** τ_z

Figure 5.9: Convergence of the average one-step-ahead prediction error of GPR, LWPR, and SSGPR^{50} methods on the iCub dataset. The results for SSGPR^{50} are the average error over 25 randomized runs. The standard deviation over the various runs is negligible and error bars are therefore omitted for clarity.

Table 5.3: One-step-ahead prediction errors in terms of %nMSE.

		SSGPR^{25}		SSGPR^{50}		SSGPR^{100}	SSGPR^{200}	GPR	LWPR	RBD
		fixed	non-fixed	fixed	non-fixed	fixed	fixed			
Sarcos	τ_1	5.055 ($\pm$ 0.170)	2.876 ($\pm$ 0.053)	4.225 ($\pm$ 0.082)	2.391 ($\pm$ 0.043)	3.493 ($\pm$ 0.056)	2.878 ($\pm$ 0.042)	1.910	12.064	
	τ_2	5.158 ($\pm$ 0.311)	2.650 ($\pm$ 0.056)	3.869 ($\pm$ 0.166)	2.075 ($\pm$ 0.042)	3.057 ($\pm$ 0.070)	2.406 ($\pm$ 0.041)	1.493	9.443	
	τ_3	4.386 ($\pm$ 0.469)	2.141 ($\pm$ 0.092)	2.863 ($\pm$ 0.221)	1.638 ($\pm$ 0.045)	2.148 ($\pm$ 0.065)	1.659 ($\pm$ 0.035)	1.025	7.092	
	τ_4	1.777 ($\pm$ 0.214)	0.903 ($\pm$ 0.040)	1.155 ($\pm$ 0.061)	0.653 ($\pm$ 0.021)	0.804 ($\pm$ 0.030)	0.573 ($\pm$ 0.016)	0.356	4.219	
	τ_5	12.995 ($\pm$ 2.229)	12.298 ($\pm$ 1.167)	8.335 ($\pm$ 1.193)	8.157 ($\pm$ 0.467)	5.379 ($\pm$ 0.467)	3.822 ($\pm$ 0.402)	2.234	9.031	
	τ_6	15.413 ($\pm$ 2.341)	9.811 ($\pm$ 0.706)	10.166 ($\pm$ 2.185)	6.587 ($\pm$ 0.369)	5.948 ($\pm$ 0.349)	4.195 ($\pm$ 0.165)	2.238	11.282	
	τ_7	3.538 ($\pm$ 0.196)	2.690 ($\pm$ 0.104)	2.695 ($\pm$ 0.182)	1.958 ($\pm$ 0.059)	1.806 ($\pm$ 0.086)	1.265 ($\pm$ 0.062)	0.683	4.437	
James	F_x	0.220 ($\pm$ 0.013)	0.182 ($\pm$ 0.013)	0.147 ($\pm$ 0.006)	0.173 ($\pm$ 0.010)	0.114 ($\pm$ 0.004)	0.095 ($\pm$ 0.002)	0.637	1.935	
	F_y	0.904 ($\pm$ 0.095)	0.684 ($\pm$ 0.067)	0.519 ($\pm$ 0.032)	0.702 ($\pm$ 0.059)	0.418 ($\pm$ 0.018)	0.348 ($\pm$ 0.012)	2.151	16.077	
	F_z	0.285 ($\pm$ 0.036)	0.253 ($\pm$ 0.017)	0.217 ($\pm$ 0.009)	0.270 ($\pm$ 0.014)	0.198 ($\pm$ 0.005)	0.182 ($\pm$ 0.003)	1.212	1.884	
	τ_x	1.022 ($\pm$ 0.148)	0.723 ($\pm$ 0.117)	0.586 ($\pm$ 0.055)	0.744 ($\pm$ 0.055)	0.411 ($\pm$ 0.029)	0.330 ($\pm$ 0.014)	2.887	3.776	
	τ_y	0.233 ($\pm$ 0.025)	0.175 ($\pm$ 0.018)	0.134 ($\pm$ 0.007)	0.159 ($\pm$ 0.014)	0.092 ($\pm$ 0.006)	0.071 ($\pm$ 0.002)	0.535	1.423	
	τ_z	1.285 ($\pm$ 0.169)	0.787 ($\pm$ 0.073)	0.717 ($\pm$ 0.060)	0.693 ($\pm$ 0.042)	0.444 ($\pm$ 0.017)	0.345 ($\pm$ 0.010)	2.874	2.384	
iCub	F_x	1.063 ($\pm$ 0.011)	1.139 ($\pm$ 0.038)	1.010 ($\pm$ 0.013)	1.125 ($\pm$ 0.033)	0.961 ($\pm$ 0.012)	0.915 ($\pm$ 0.011)	2.317	5.204	3.652
	F_y	1.020 ($\pm$ 0.010)	1.051 ($\pm$ 0.025)	0.977 ($\pm$ 0.012)	1.080 ($\pm$ 0.036)	0.947 ($\pm$ 0.011)	0.904 ($\pm$ 0.008)	3.599	4.952	3.310
	F_z	1.306 ($\pm$ 0.009)	1.684 ($\pm$ 0.123)	1.264 ($\pm$ 0.009)	1.882 ($\pm$ 0.190)	1.239 ($\pm$ 0.009)	1.215 ($\pm$ 0.008)	11.375	9.003	8.231
	τ_x	0.575 ($\pm$ 0.015)	0.559 ($\pm$ 0.011)	0.528 ($\pm$ 0.003)	0.588 ($\pm$ 0.019)	0.518 ($\pm$ 0.002)	0.512 ($\pm$ 0.001)	0.576	4.120	3.082
	τ_y	1.551 ($\pm$ 0.031)	1.496 ($\pm$ 0.019)	1.452 ($\pm$ 0.007)	1.564 ($\pm$ 0.046)	1.435 ($\pm$ 0.002)	1.427 ($\pm$ 0.002)	1.619	9.752	3.166
	τ_z	1.236 ($\pm$ 0.043)	1.258 ($\pm$ 0.043)	1.095 ($\pm$ 0.007)	1.261 ($\pm$ 0.060)	1.059 ($\pm$ 0.004)	1.034 ($\pm$ 0.004)	2.366	5.501	3.977
iCub + trend	F_x	0.797 ($\pm$ 0.010)	0.859 ($\pm$ 0.024)	0.761 ($\pm$ 0.009)	0.873 ($\pm$ 0.033)	0.719 ($\pm$ 0.006)	0.682 ($\pm$ 0.007)			
	F_y	0.837 ($\pm$ 0.008)	0.868 ($\pm$ 0.026)	0.796 ($\pm$ 0.011)	0.884 ($\pm$ 0.033)	0.755 ($\pm$ 0.008)	0.709 ($\pm$ 0.008)			
	F_z	1.049 ($\pm$ 0.011)	1.052 ($\pm$ 0.015)	1.015 ($\pm$ 0.011)	1.122 ($\pm$ 0.032)	0.996 ($\pm$ 0.007)	0.978 ($\pm$ 0.004)			
	τ_x	0.575 ($\pm$ 0.016)	0.561 ($\pm$ 0.011)	0.528 ($\pm$ 0.002)	0.595 ($\pm$ 0.019)	0.518 ($\pm$ 0.002)	0.512 ($\pm$ 0.001)			
	τ_y	1.550 ($\pm$ 0.031)	1.494 ($\pm$ 0.017)	1.451 ($\pm$ 0.006)	1.566 ($\pm$ 0.045)	1.435 ($\pm$ 0.002)	1.426 ($\pm$ 0.002)			
	τ_z	1.236 ($\pm$ 0.045)	1.260 ($\pm$ 0.036)	1.094 ($\pm$ 0.005)	1.291 ($\pm$ 0.048)	1.058 ($\pm$ 0.003)	1.031 ($\pm$ 0.006)			

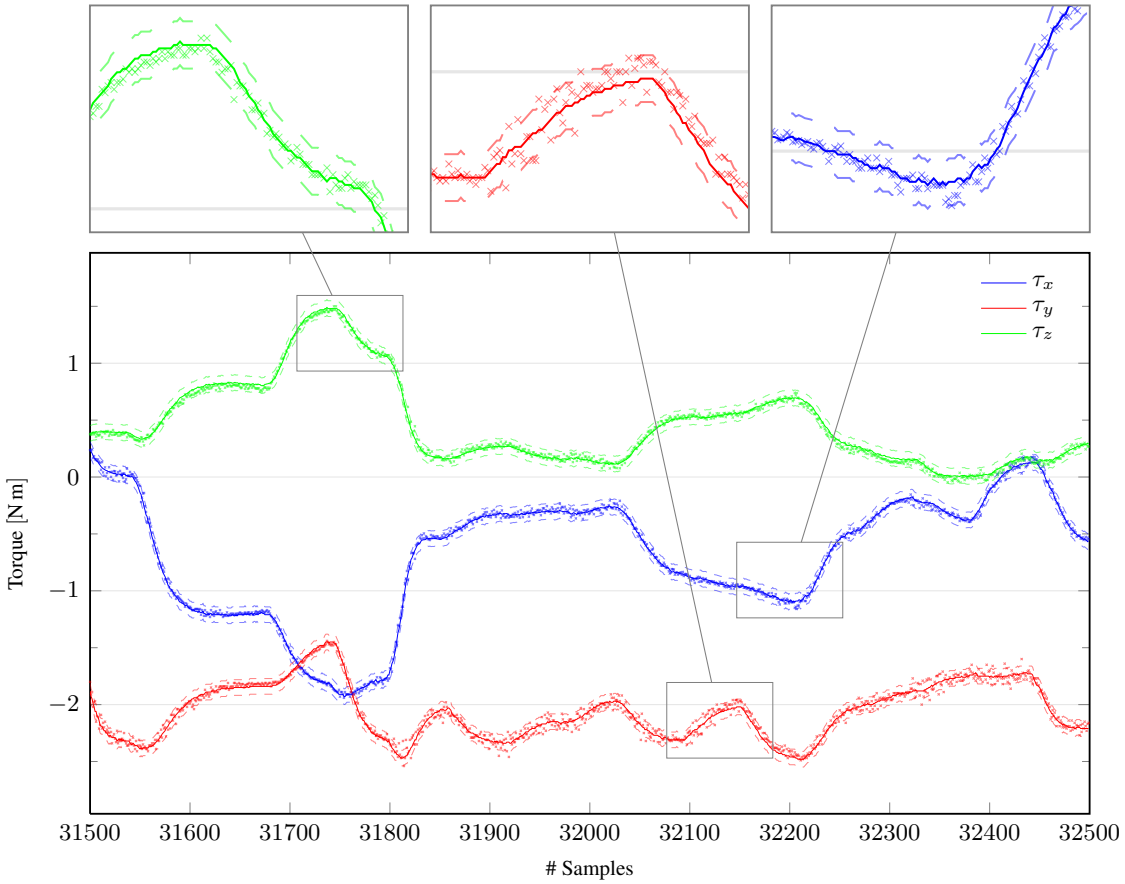
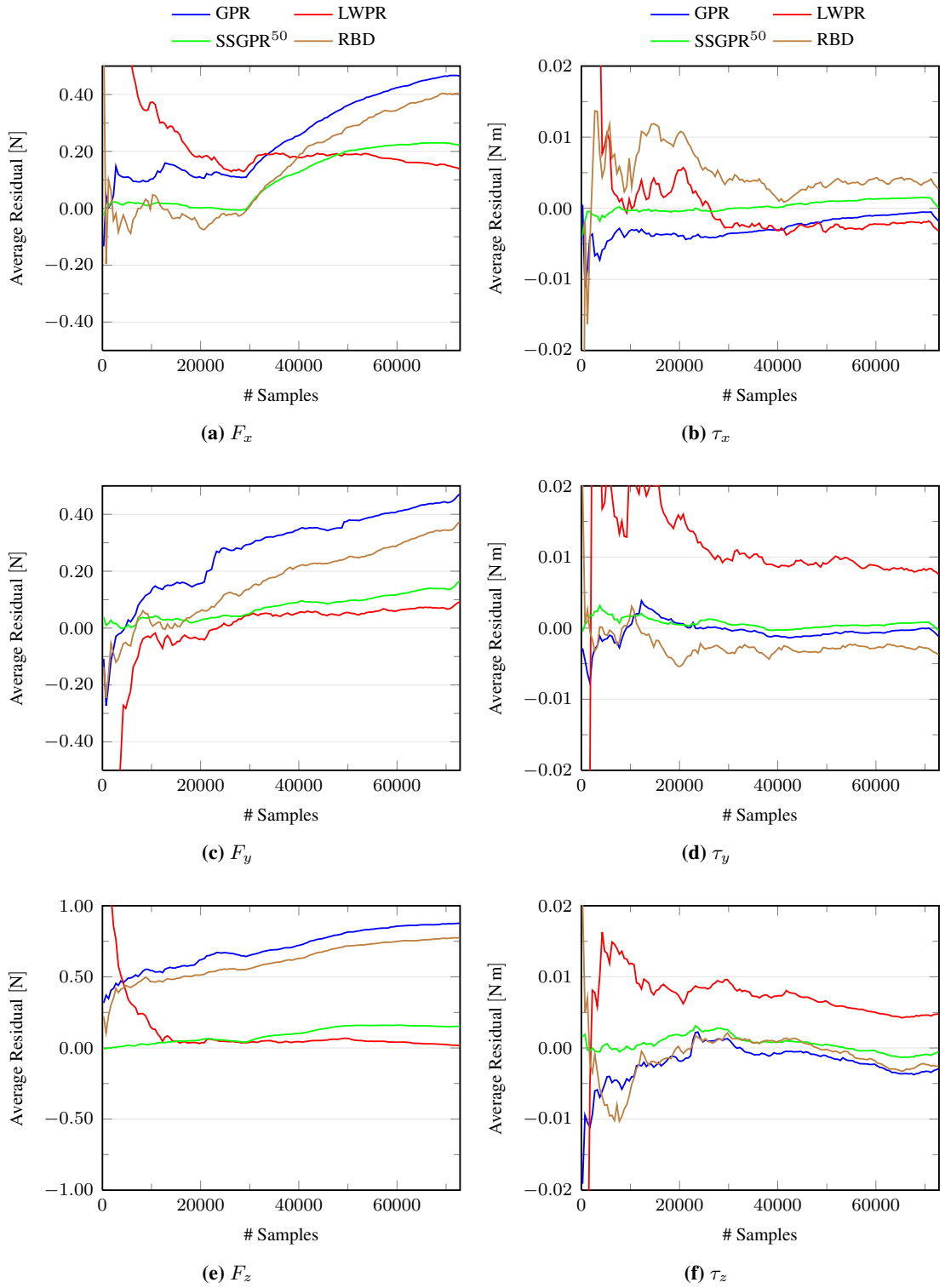


Figure 5.10: Extract of the three torque measurements (denoted with crosses) and the predictive mean and variance of SSGPR^{50} on the iCub dataset. The dashed lines denote a confidence interval of twice the standard deviation.

cursive) residuals for the considered methods. The sensor drift eventually causes GPR and RBD to underestimate forces up to an extent of approximately 1 N. A similar effect is observable for SSGPR^{50}, although much less profound due to continuous model adjustments. We can see in Figure 5.12 that the residuals approximately follow a normal distribution with non-zero mean for force predictions, while the distribution of residuals for torque predictions is centered around zero.

This constant offset of the residuals biases any direct comparison between the dynamics model and sensor measurements, and is therefore not desirable. Although it is arguable that the sensor is at fault rather than the models, the sensor measurements often form the ground truth in comparisons. It thus necessary that the models make predictions in the reference frame of the (inaccurate) sensor, rather than in terms of accurate but unmeasurable physical quantities. In regression methods, (sensor) drift can be compensated for using a trend model. Given the nature of the drift as observed in Figure 5.11, it seems appropriate to model the drift using a linear factor. Extension of SSGPR with an additional linear term is straightforward by extending the feature map $\phi(\cdot)$ from



(a) F_x **(b)** τ_x

(c) F_y **(d)** τ_y

(e) F_z **(f)** τ_z

Figure 5.11: Average one-step-ahead residual with respect to the number of samples of GPR, LWPR, and SSGPR^{50} on the iCub dataset. The results for SSGPR^{50} are the average error over 25 randomized runs. Note the different limits of the y-axis for the various plots.

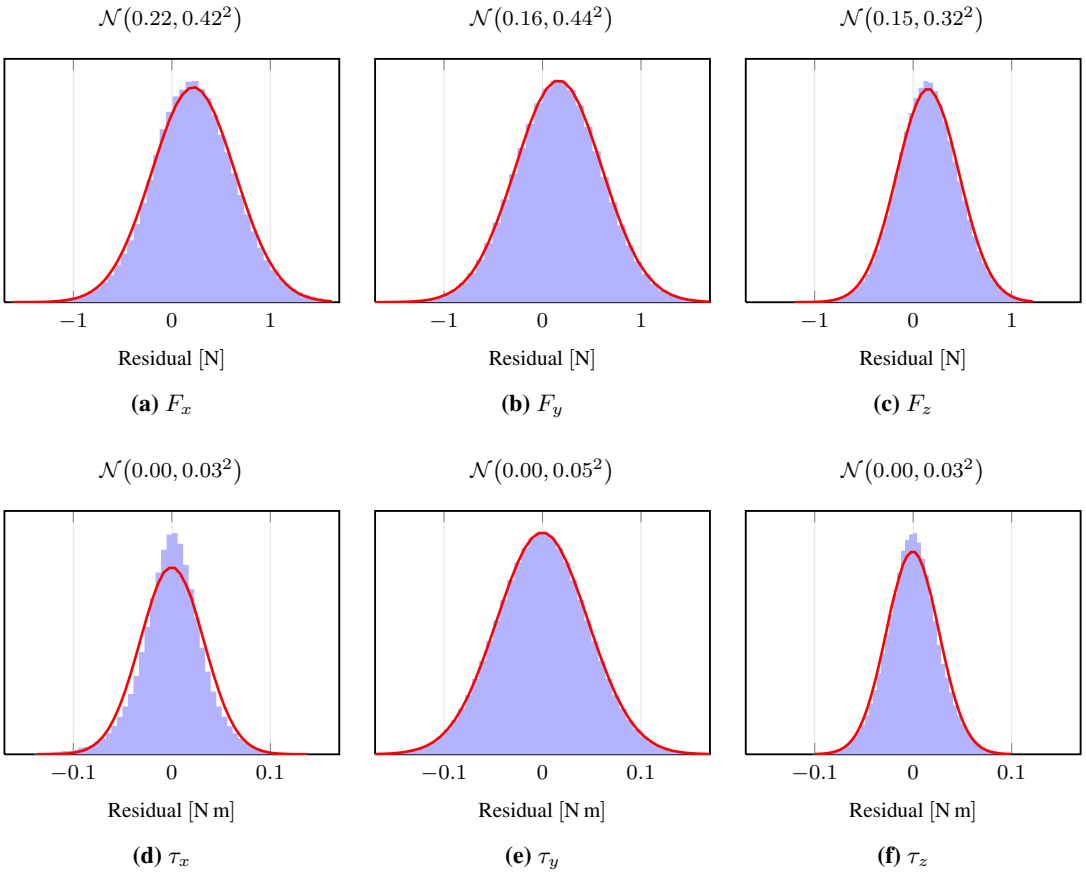
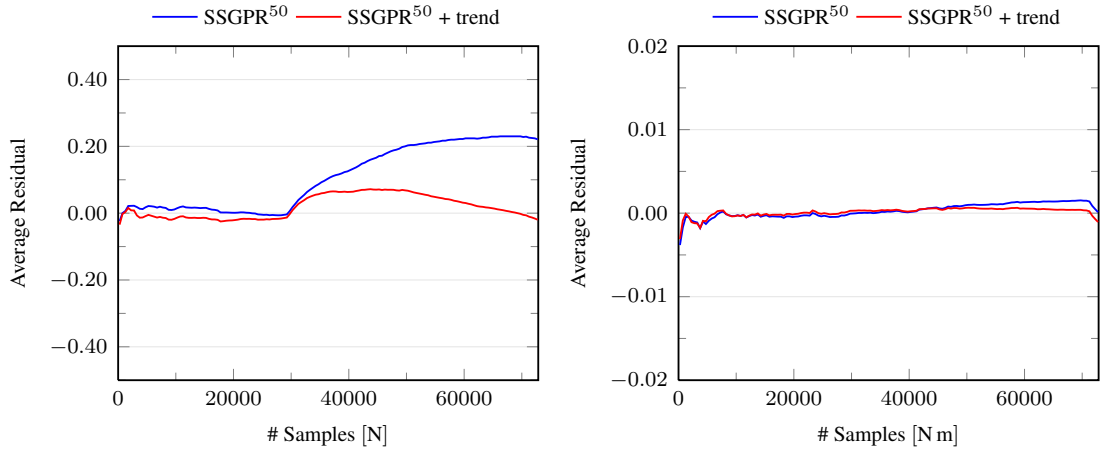


Figure 5.12: Histogram of the residuals for SSGPR^{50} on the iCub dataset. The red line indicates a Gaussian distribution with the specified mean and variance.

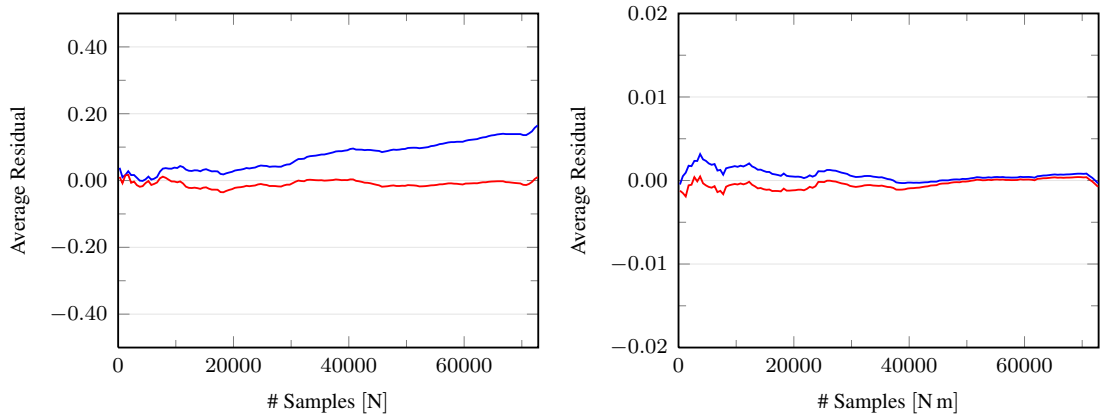
Equation (4.6) with an additional sample index t, such that

$$\phi(\boldsymbol{x}) = \frac{\sigma_f}{\sqrt{D}} \left[\sin\left(\langle \boldsymbol{\omega}_1^{\mathrm{T}}, \boldsymbol{x}\rangle\right), \cos\left(\langle \boldsymbol{\omega}_1^{\mathrm{T}}, \boldsymbol{x}\rangle\right), \cdots, \sin\left(\langle \boldsymbol{\omega}_D^{\mathrm{T}}, \boldsymbol{x}\rangle\right), \cos\left(\langle \boldsymbol{\omega}_D^{\mathrm{T}}, \boldsymbol{x}\rangle\right), t\right]^{\mathrm{T}} .$$

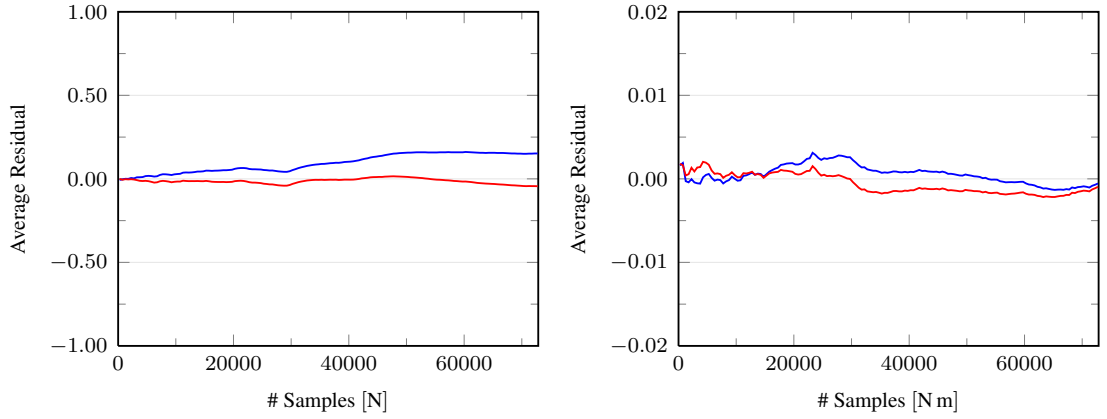
The average residuals of SSGPR^{50} with the linear trend model are shown in Figure 5.13. As we can see, the method is effective at compensating for most of the sensor drift, keeping the average residual close to zero over the whole dataset. The effect of drift compensation is also noticeable in terms of nMSE, in case of SSGPR^{50} the prediction errors are reduced by 20% to 25% (cf. Table 5.1). The linearity assumption on the drift may seem as a very strong assumption, however, SSGPR continuously adapts the corresponding regression coefficient in order to adjust for non-linear drift behavior. In absence of drift the regression coefficient is driven to zero and trend compensation will effectively be disabled. One disadvantage, however, is the requirement to continuously adjust the corresponding regression coefficient. If the model is not updated regularly, then small deviations in the estimation of this coefficient will cause increasingly large prediction errors due to the increasing time index t.



(a) F_x **(b)** τ_x



(c) F_y **(d)** τ_y



(e) F_z **(f)** τ_z

Figure 5.13: Average one-step-ahead residual with respect to the number of samples of SSGPR^{50} on the iCub dataset when using a linear trend for drift compensation. The presented results are the average error over 25 randomized runs. Note the different limits of the y-axis for the various plots.

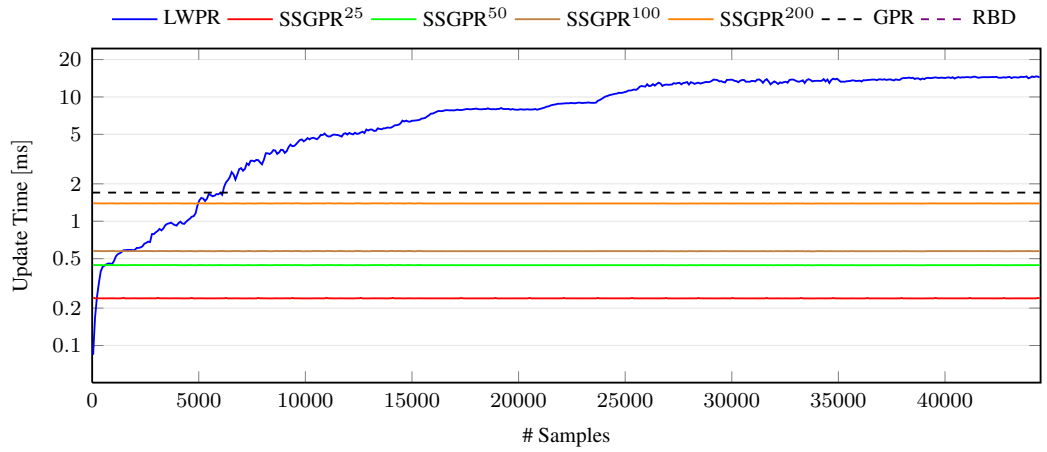
5.3.4 Computational Requirements

Applications utilizing predicted robot dynamics are commonly characterized by (hard) real-time constraints. Model predictions do not only have to be accurate, but should also be computed within a timely fashion to be useful. For example, in case of James and iCub humanoids, the F/T sensor is measured at an interval of around $50\,\text{Hz}$. Prediction times over $20\,\text{ms}$ are therefore unacceptable, as these are based on outdated information. For incremental methods, the time spent for a single sample not only includes a the prediction time, but also the subsequent model update. Figure 5.14 shows the per-sample timing for LWPR and SSGPR with respect to the number of test samples on all three datasets. The constant prediction times of GPR and RBD have been included as reference.

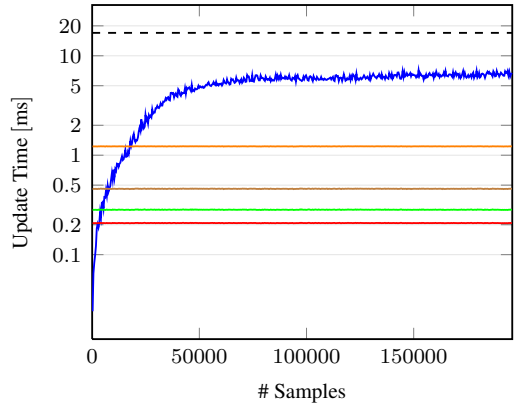
As described in Section 4.4, the timing of SSGPR remains perfectly stable over time, making this method suitable for use in real-time control loops. Furthermore, the time spent on each sample ranges from approximately $250\,\mu\text{s}$ for 25 to $1.5\,\text{ms}$ for 200 sparse spectrum features, respectively. We can therefore conclude that the computational requirements of incremental SSGPR are minimal and well within the target limit of $20\,\text{ms}$. A common argument for analytical models is their apparent efficiency. However, the RBD model is only marginally faster than SSGPR^{25} on the iCub dataset[8]. Combined with the superior predictive accuracy of SSGPR^{25} (cf. Table 5.2), there is little reason to prefer the analytical RBD model over a learned SSGPR solution.

Although efficient with a small number of samples, the per-sample timings of LWPR are characterized by a steep increase. Not only do these exceed $20\,\text{ms}$ in some cases, the unpredictability of the computational requirements make LWPR unsuitable for real-time usage. In Figure 5.14c, for instance, the prediction times seem to have stabilized after approximately 80000 samples, only to start increasing again around 150000 samples. In Figure 5.15, we observe an $\mathcal{O}\left(x^{0.76}\right)$ scaling behavior of LWPR on the entire iCub dataset as calculated using an empirical estimate (Goldsmith et al., 2007). The timing behavior of LWPR could be controlled by tuning the thresholds for creation and removal of receptive fields. In an extreme case, for instance, the model could guarantee $\mathcal{O}(1)$ scaling by disabling the creation of new receptive fields. However, this tuning is cumbersome and will likely have a negative effect on generalization performance. Moreover, the exact balance between creation and removal of receptive fields is data-dependent and further assumptions on the data will be necessary to make formal guarantees. SSGPR, on the other hand, has $\mathcal{O}(1)$ scaling "by design" and scales as $\mathcal{O}\left(n^2\right)$ with respect to the number of inputs (controlled directly by the hyperparameter D). The computational requirements are therefore predictable for any given parameter configuration.

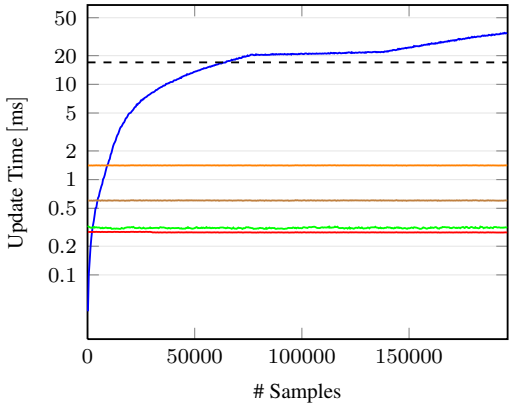
[8] This comparison is not entirely fair: the RBD model is implemented entirely in C++, while the SSGPR model is partially implemented in Python.



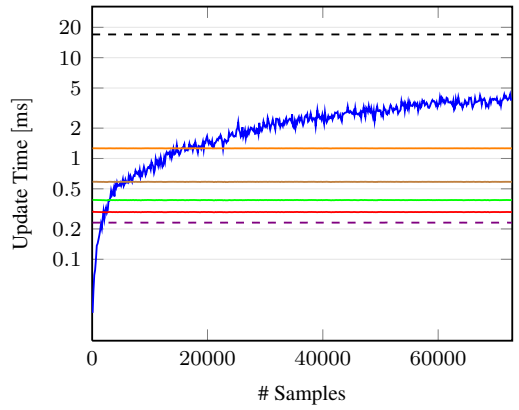
(a) Sarcos Torques



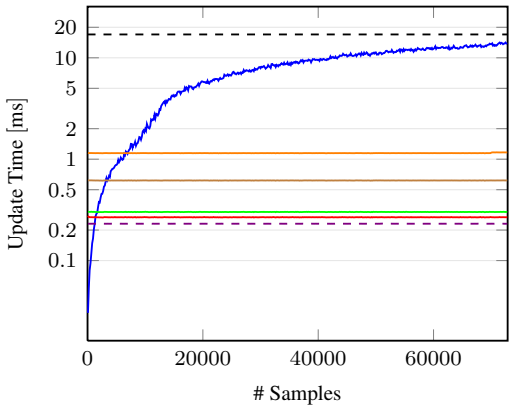
(b) James Forces



(c) James Torques



(d) iCub Forces



(e) iCub Torques

Figure 5.14: Sample time with respect to the number of training samples. The results for GPR and RBD only include the prediction time, whereas the timing for LWPR and SSGPR also includes the model update. Note that the y-axis is in log scale and that data for RBD is only available for the iCub dataset.

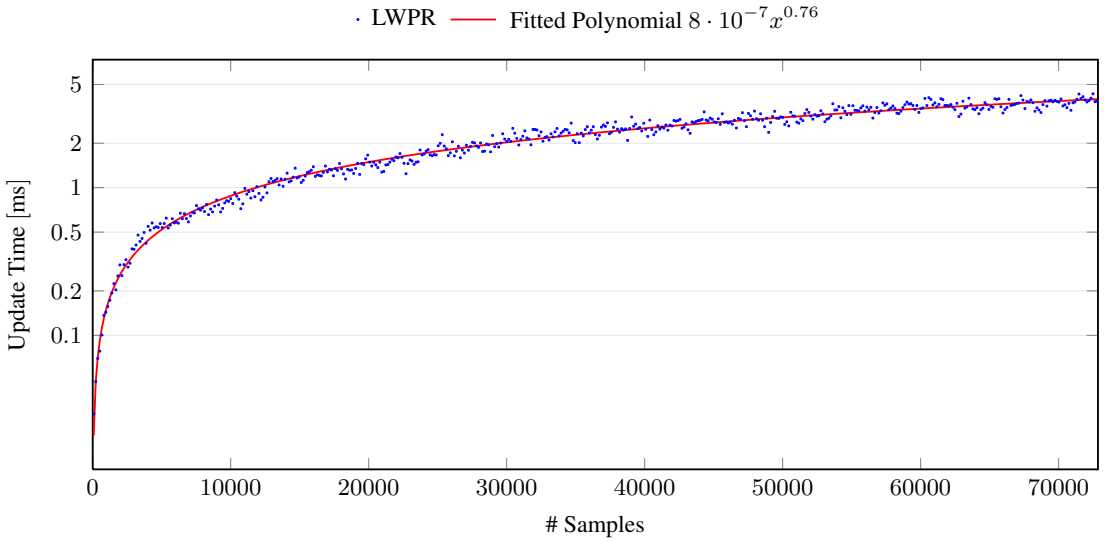


Figure 5.15: Per-sample update time of LWPR with respect to the number of samples for the force components of the iCub dataset. The red line indicates the empirical estimate of the complexity obtained by fitting an exponential curve to the measurements. Note the log scale on the y-axis.

5.4 Incremental Learning of the Visual Location of the Hand

The last experiment concerns learning the association between head and arm posture, and the corresponding visual response. This particular sensorimotor mapping is useful for a number of robotic tasks (see Natale et al., 2007, and references therein), such as gaze control or visually guided reaching and grasping. The forward model of this association can be defined as

$$f : \{\boldsymbol{q}_{\text{arm}}, \boldsymbol{q}_{\text{head}}\} \mapsto \mathcal{I} \ ,$$

where $\boldsymbol{q}_{\text{arm}} \in \mathbb{R}^7$ contains joint angles for all 7 DoF of the arm and $\boldsymbol{q}_{\text{head}} \in \mathbb{R}^6$ describes both the neck (i.e., pitch, roll, and yaw) and eye configuration (i.e., common tilt, common version, and vergence). Furthermore, the image representation $\mathcal{I} = \{u_{\text{left}}, v_{\text{left}}, u_{\text{right}}, v_{\text{right}}\} \in \mathbb{N}^4$ contains the (approximate) horizontal and vertical coordinates of the center of the hand in the left and right image planes.

Obtaining the inverse model $f^{-1} : \mathcal{I} \mapsto \{\boldsymbol{q}_{\text{arm}}, \boldsymbol{q}_{\text{head}}\}$ is typically more interesting from a practical perspective, since it computes the action required to drive the robot to a desired image representation $\mathcal{I}$. However, there are multiple joint configurations that result in the same image representation, which is due to redundancy in the DoF in advanced humanoid robots. It follows that the mapping function f is not bijective and thus not invertible, which in return signifies that constructing the inverse mapping f^{-1} is an ill-posed learning problem. Although well-posedness can be enforced by imposing additional constraints on the solution (e.g., see Lopes and Santos-Victor, 2006), the primary interest in this work is to compare learning methods and the forward learning

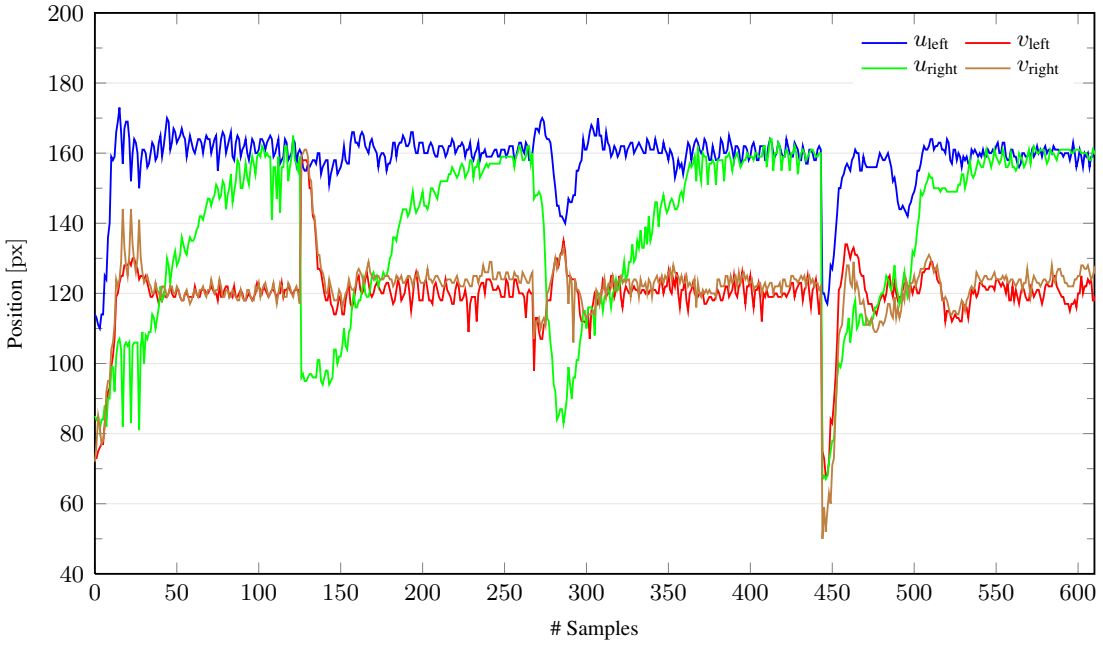


Figure 5.16: Extract of the visual coordinates of the iCub hand during the first four epochs of the data collection procedure. Note that the robot is controlled to center the hand in the visual frames of both eyes, explaining convergence to 160 and 120 pixels for the horizontal and vertical axes, respectively.

problem is therefore considered in the following.

A total of 37476 samples were collected from the iCub humanoid, subdivided over 429 epochs. In each epoch, the hand is first moved to a random Cartesian position, after which the head and eyes are controlled to center the hand in both left and right image planes of 320×240 pixels each. During this procedure, one of the wrist joints is activated to repeatedly shake the hand from left to right. These relatively small movements allow visual localization of the hand using independent motion detection. Figure 5.16 demonstrates the convergence of the u and v coordinates in the first four epochs. We observe that consecutive samples are strongly correlated due to the described data collection methodology. Furthermore, imperfect localization of the hand in the image frames causes a relatively high level of noise and in some occasions erroneous output labels (cf. samples 20 to 30 for u_{right} and v_{right}). These complications are interesting from a machine learning perspective, since it allows evaluation of learning methods under particularly difficult conditions.

5.4.1 Experimental Setup

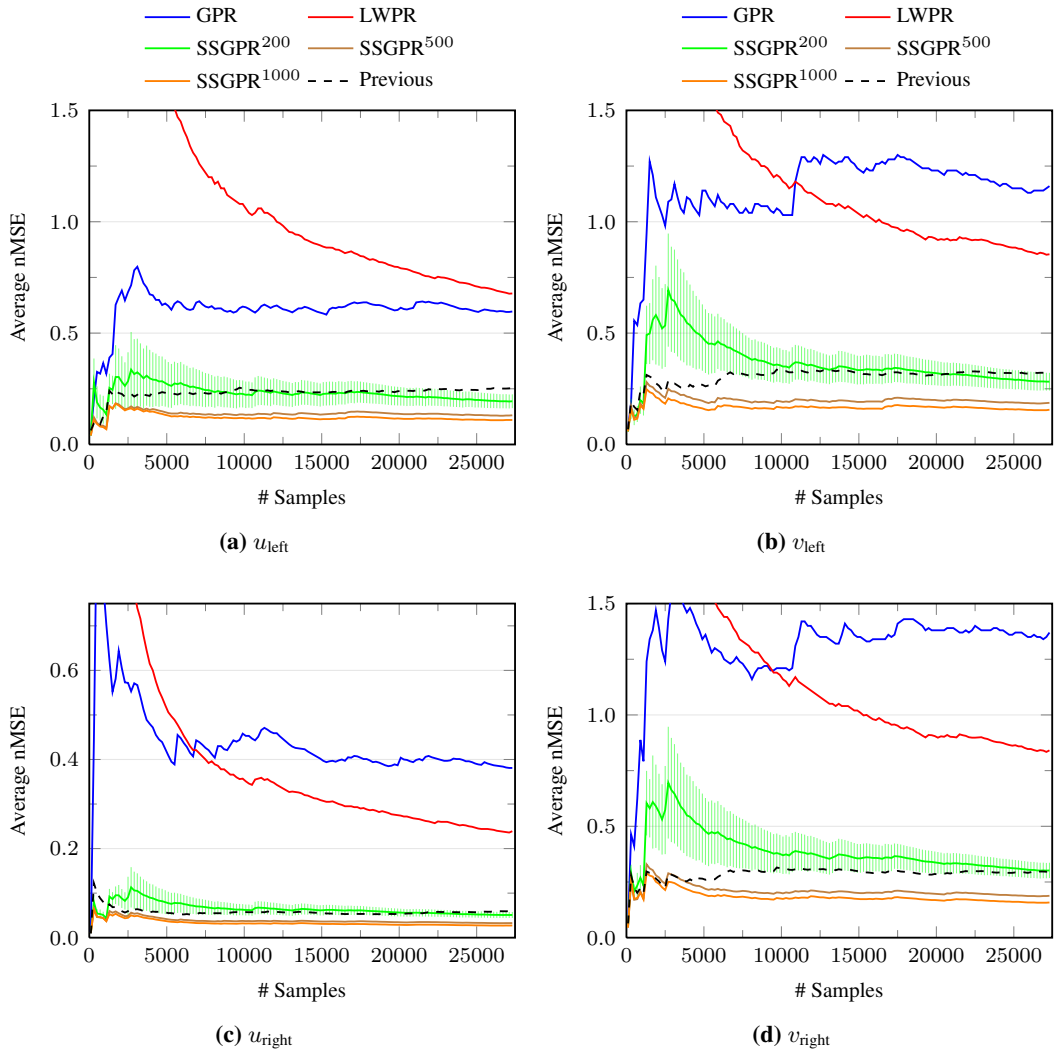
Similar to the previous experiments, the aim of these experiments is to compare the performance of SSGPR and LWPR when learning the forward sensorimotor model incrementally. The experimental setup is therefore identical to the the one described in Section 5.3.1. The first 10000 samples are designated as the training set, while the remaining 27476 samples are used for testing. As previously, the incremental learning methods use the training samples only for hyperparameter

optimization, while batch GPR predicts the test set after being trained on the training set. Based on performance results in preliminary experiments, the considered number of sparse spectrum features is chosen as $D \in \{200, 500, 1000\}$. Furthermore, the trivial function $f(\boldsymbol{x}_t) = \boldsymbol{y}_{t-1}$, which simply returns as prediction the revealed output of the previous sample, is used as an additional reference method. Given the similarity of successive samples, this sequential function constitutes a more realistic benchmark than predicting the mean of the training outputs (i.e., an nMSE of approximately 1).

5.4.2 Results

Figure 5.17 demonstrates the generalization performance of the considered methods for the four output coordinates. These results clearly confirm the inherent difficulty for learning methods on this dataset. In particular, we observe that GPR performs poorly in all cases with a final nMSE that ranges approximately from 0.4 to 1.5. This leads to the conclusion that the 10000 training samples contain highly redundant information and are not sufficient to construct a satisfactory model. Although LWPR outperforms GPR for three out of four outputs, its performance is still significantly worse than the reference method that predicts the previous outputs. This inability to compete with a trivial predictor ascertains that using LWPR to learn this dataset is not justified by any means.

The results are more positive for incremental SSGPR, which demonstrates stable and superior performance for all outputs. In the most economic configuration of 200 sparse spectrum features, its performance is roughly similar to the reference method. Increasing the number of features to $D = 500$ or $D = 1000$, however, causes a significant improvement in learning performance. In these cases, SSGPR unequivocally outperforms the reference method, demonstrating that learning is advantageous on this dataset. Moreover, this result confirms that the incremental SSGPR learning method can be used successfully in the challenging setting of highly dependent and noisy data.



(a) u_{left} **(b)** v_{left}

(c) u_{right} **(d)** v_{right}

Figure 5.17: Convergence of the average one-step-ahead prediction error of the considered methods when predicting the visual location of the hand. The results for SSGPR are the average error over 25 randomized runs and the corresponding standard deviation is indicated with error bars for SSGPR^{200}. In case of SSGPR^{500} and SSGPR^{1000}, the standard deviation is negligible and error bars are therefore omitted for clarity.

6 CONTACT DETECTION

One useful application of inverse dynamics models is to detect contact situations. Given a prediction of the internal dynamics, the robot can infer a contact situation by comparing this prediction with the current sensor measurement. The difference between these two quantities is ideally zero if there is no external force acting on the manipulator. Conversely, any deviation indicates an external force exerted on the robot by the environment. In practice, the relationship between residuals and external forces is complicated by the presence of prediction errors. Inaccuracies of both the sensor measurement and the regression model are likely to cause non-zero residuals even in absence of external forces. The primary difficulty of contact detection is therefore the ability to distinguish between prediction errors and true changes in the dynamics model due to contact situations. Note that in the following we strictly consider the problem of classifying contact situations; exact quantification or localization of external forces is not within the scope of this thesis.

Statistical tests for change detection can be used to make informed decisions whether to classify non-zero residuals as true contact situations. Detection of changes in the underlying model can be approximated efficiently using reasonable assumptions on the distribution of the residuals and a fixed window of comparison. A contact situations is predicted when a likelihood ratio exceeds a predefined threshold, at which point updates to the model are suspended until the statistic drops below the threshold again. The exact procedure is explained in detail in Section 6.1. Subsequently, the procedure is demonstrated on a variant of the synthetic Cross 2D dataset, which has been extended to four different output regimes. These regimes are alternately activated by means of a probabilistic finite-state automaton. This chapter is concluded in Section 6.3, which presents the experimental validation of this procedure on the task of detecting external forces exerted on the right arm of the iCub humanoid.

6.1 Window-Limited Generalized Likelihood Ratio

The online change detection problem is to quickly detect abrupt changes in a stream of observed random variables. At an unknown change time t_c, a density parameter θ of these variables changes from θ_0 to θ_1. The goal is to promptly detect this transition, while keeping the number of false alarms at a minimum. Two well-known statistical tests that can be used for this particular problem are the cumulative sum control chart (CUSUM) (Page, 1954; Basseville and Nikiforov, 1993) and Generalized Likelihood Ratio (GLR) tests (Lorden, 1971; Basseville and Nikiforov, 1993). Both these procedures are based on the likelihood ratio between the null hypothesis $H_0 : \theta = \theta_0$ and change hypothesis $H_1 : \theta = \theta_1$, the primary difference being the availability of prior information on θ_1. In contrast to GLR, the CUSUM procedure assumes that the parameter θ_1 after change is known a priori. Given that external forces on a robot manipulator cannot be quantified a priori, in the following we concentrate on the GLR procedure as described by Basseville and Nikiforov (1993).

At the core of the GLR procedure is the log-likelihood ratio, for a variable z defined as

$$s(z) = \ln \frac{p_{\theta_1}(z)}{p_{\theta_0}(z)} \ .$$

Noting that $\mathbb{E}_{\theta_0}[s] < 0$ and $\mathbb{E}_{\theta_1}[s] > 0$, it follows that a change in the parameter θ is reflected as a change in the sign of the mean value of s. For a finite sequence of observations z_i for $j \geq i \geq t$, we can define analogously

$$S_{j:t} = \sum_{i=j}^{t} s(z_i) \ .$$

The unknown change time t_c can subsequently be approximated by finding the index j at which this ratio is at a maximum. However, this ratio cannot be evaluated without further information, since the parameter after change θ_1 is unknown as well. The standard statistical approach is to replace θ_1 with a maximum likelihood estimate, yielding the double optimization problem

$$g_t = \max_{1 \leq j \leq t} \sup_{\theta_1} S_{j:t} \ .$$

This formulation allows the definition of a decision function indicating whether a change has occurred, i.e.,

$$d_t = \begin{cases} g_t & \text{if } g_t \geq h \\ 0 & \text{otherwise} \end{cases} \ , \tag{6.1}$$

where h is a predefined activation threshold. The corresponding predicted change time (i.e, the

alarm time) can then easily be defined as

$$t_a = \min \{t : d_t > 0\} \quad .$$

The maximum likelihood estimate of θ_1 is carried out for each possible change time $1 \leq t_c \leq t$ and cannot be computed recursively, causing the computational requirements of the GLR test to grow with respect to the number of observations. This growth can be alleviated by restricting the procedure to a moving window of length M (i.e., the M most recent observations), which is known as Window-Limited Generalized Likelihood Ratio (WGLR) (Appel and Brandt, 1983; Capizzi, 2001). Incorporating this restriction in the maximization of the change time yields

$$g_t = \max_{t-M<j\leq t} \sup_{\theta_1} S_{j:t} \quad .$$

The underlying idea is that older observations are less relevant, since earlier changes are likely to have already been detected.

For our particular application of contact detection using a multi-variate regression model, the sensor measurements can be described as a combination of the model predictions $\hat{\boldsymbol{y}} = \mathbb{E}[f(\boldsymbol{x})]$, the Gaussian measurement noise ϵ, and a possible external disturbance Υ. Therefore, for any given time step t

$$\boldsymbol{y}_t = \hat{\boldsymbol{y}}_t + \epsilon_t + \Upsilon(t, t_c) \quad ,$$

where the p-dimensional $\Upsilon(t, t_c)$ is an unknown additive change. The contact problem can thus be formalized as testing for the alternative hypothesis $H_1 : \Upsilon_1(t, t_c) \neq \mathbf{0}$ for $t \geq t_c$, as opposed to the no-contact situation $H_0 : \Upsilon_0(t, t_c) = \mathbf{0}$ for $t < t_c$. The alternative hypothesis cannot be tested directly, since both the magnitude and direction of Υ are unknown. However, assuming that the model predictions are accurate under the no-change condition, the additive change Υ is reflected directly in the recursive residuals $\boldsymbol{e}_t = \boldsymbol{y}_t - \hat{\boldsymbol{y}}_t$. Let us consider the recursive residuals standardized by the predicted standard deviation (Brown et al., 1975)

$$\boldsymbol{u} = \left[\frac{e_1}{s_1}, \ldots, \frac{e_p}{s_p}\right] \quad ,$$

and assume these to be independent and distributed according to $\boldsymbol{u} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ (recall the standardization). The relevant test thus becomes to detect a deviation from zero in the standardized residuals $\boldsymbol{u}$, indicating consistent prediction errors by the model and therefore a likely contact situation. In practice, however, it is convenient to not strictly require the residuals to be zero, but instead to allow for a margin on the conditions of the null hypothesis H_0 and alternative hypothesis H_1. Incorporating an upper bound a for the no-change condition $\boldsymbol{\theta}_0$ and a lower bound b for the

magnitude of change, the detection problem becomes

$$\boldsymbol{\theta}(t) = \begin{cases} \boldsymbol{\theta} : (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^{\mathrm{T}} (\boldsymbol{\theta} - \boldsymbol{\theta}_0) = \|\boldsymbol{\theta}\|^2 \leq a^2 & \text{when } t < t_c \\ \boldsymbol{\theta} : (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^{\mathrm{T}} (\boldsymbol{\theta} - \boldsymbol{\theta}_0) = \|\boldsymbol{\theta}\|^2 \geq b^2 & \text{when } t \geq t_c \end{cases} ,$$

where $a < b$ and $\boldsymbol{\theta}_0 = \mathbf{0}$. The corresponding log-likelihood ratio of the GLR test is then

$$S_{j:t} = \ln \frac{\sup_{\|\boldsymbol{\theta}\| \geq b} \prod_{j=1}^{t} p_{\boldsymbol{\theta}}(\boldsymbol{u}_j)}{\sup_{\|\boldsymbol{\theta}\| \leq a} \prod_{j=1}^{t} p_{\boldsymbol{\theta}}(\boldsymbol{u}_j)} .$$

This can be rewritten as (Basseville and Nikiforov, 1993, Section 7.2.1.7)

$$\frac{2}{t-j+1} S_{j:t} = \begin{cases} -\left(\|\bar{\boldsymbol{u}}_{j:t}\| - b\right)^2 & \text{when} \quad \|\bar{\boldsymbol{u}}_{j:t}\| < a \\ -\left(\|\bar{\boldsymbol{u}}_{j:t}\| - b\right)^2 + \left(\|\bar{\boldsymbol{u}}_{j:t}\| - a\right)^2 & \text{when} \quad a \leq \|\bar{\boldsymbol{u}}_{j:t}\| \leq b \\ +\left(\|\bar{\boldsymbol{u}}_{j:t}\| - a\right)^2 & \text{when} \quad \|\bar{\boldsymbol{u}}_{j:t}\| > b \end{cases} ,$$

where $\bar{\boldsymbol{u}}_{j:t}$ denotes a p-dimensional vector of the average standardized residuals from time steps j to t.

It is important to note that the problem under consideration is actually slightly more involved than the standard change detection problem. In case of the latter, is suffices to raise an alarm as soon as a change is detected. The behavior after change is unspecified and depends strongly on the desired task. For instance, change detection can be used to detect faulty sensors, such that an alarm signifies the necessity to replace the sensor. In the problem domain considered in this thesis, an alarm indicates a deviation from the base model and thus the (predicted) presence of an external force. However, the GLR test is repeated continuously for each individual time step (i.e., sample) regardless of the state, in order to detect not only the presence but also the duration of a contact situation. Given the presumption that small deviations cannot be detected instantaneously, this prediction strategy only works if the change persists over multiple time steps. In the following, the adversarial setting in which *individual* samples are generated according to different models is therefore explicitly excluded.

Restricting the log-likelihood ratio to a window of fixed length will reduce detectability of small external forces, and the assumption of an additive change with a static profile may not be entirely representative for contact situations. Consequently, the described procedure will not be statistically optimal in this problem domain. However, as we will see in the following sections, the procedure is computationally efficient and produces satisfactory results on both the synthetic and manipulator contact datasets.

6.2 Synthetic Dataset

The purpose of an initial experiment using a synthetic dataset is to evaluate the method in a controlled setting, in which the labeled data is unambiguously available. This dataset, subsequently referred to as Change-Point Cross 2D, extends the synthetic Cross 2D dataset from Section 5.1 with three additional output functions (i.e., *regimes*). These alternative regimes are variations of the "base" regime, obtained respectively by means of an additive change, a different configuration of the function constants, and a rotation of the input space. Furthermore, in Change-Point Cross 2D, the input samples are no longer chosen at random from a uniform distribution. Instead, a dependent sequence of 2-dimensional inputs is generated using the inverse Fourier transform of an increasing sequence disturbed by Gaussian noise. As a result, this sequence describes a smooth trajectory within the input space and is therefore representative of a typical trajectory in joint space of a robotic manipulator.

In more detail, the variations for the alternative regimes are given by a parameterization of the output function f. Given a tuple of parameters

$$\zeta = (\boldsymbol{a}, \boldsymbol{R}, c)\,,$$

where $\boldsymbol{a}$ is a vector of four coefficients, $\boldsymbol{R}$ is a rotation matrix, and c is an additive bias term. The corresponding output function given ζ is defined as

$$f(\boldsymbol{x}; \zeta) = \max\left\{ e^{a_1(\boldsymbol{R}\boldsymbol{x})_1^2}, e^{a_2(\boldsymbol{R}\boldsymbol{x})_2^2}, a_3 e^{a_4\left((\boldsymbol{R}\boldsymbol{x})_1^2 + (\boldsymbol{R}\boldsymbol{x})_2^2\right)} \right\} + c \ .$$

Note that the different regimes are characterized by a change in the conditional output distribution $P_\zeta(y|\boldsymbol{x})$, whereas the prior distribution $P(\boldsymbol{x})$ is constant over all regimes. The transitions between these regimes are governed by the Probabilistic Finite-State Automaton (Paz, 1971) shown in Figure 6.1. Note that alternative regimes are always preceded and succeeded by the base regime; direct transitions from one alternative regime to another are not considered. Additionally, all alternative regimes are equally likely to be selected and transition probabilities are such that a regime typically lasts for multiple time steps. The parameter tuples corresponding to the base regime q_0 and the alternative regimes q_1, q_2, and q_3 are

$$\begin{aligned}
q_0 &: \zeta = \left([-10, -50, 1.25, -5.0]^{\mathrm{T}}, \boldsymbol{I}, 0\right) \\
q_1 &: \zeta = \left([-10, -50, 1.25, -5.0]^{\mathrm{T}}, \boldsymbol{I}, 0.1\right) \\
q_2 &: \zeta = \left([-10, -50, 1.25, -1.5]^{\mathrm{T}}, \boldsymbol{I}, 0\right) \\
q_3 &: \zeta = \left([-10, -50, 1.25, -5.0]^{\mathrm{T}}, \begin{bmatrix} \cos\frac{\pi}{6} & -\sin\frac{\pi}{6} \\ \sin\frac{\pi}{6} & \cos\frac{\pi}{6} \end{bmatrix}, 0\right) \ .
\end{aligned}$$

The first alternative regime (i.e., q_1, as shown in Figure 6.2b), differs from the base regime by a

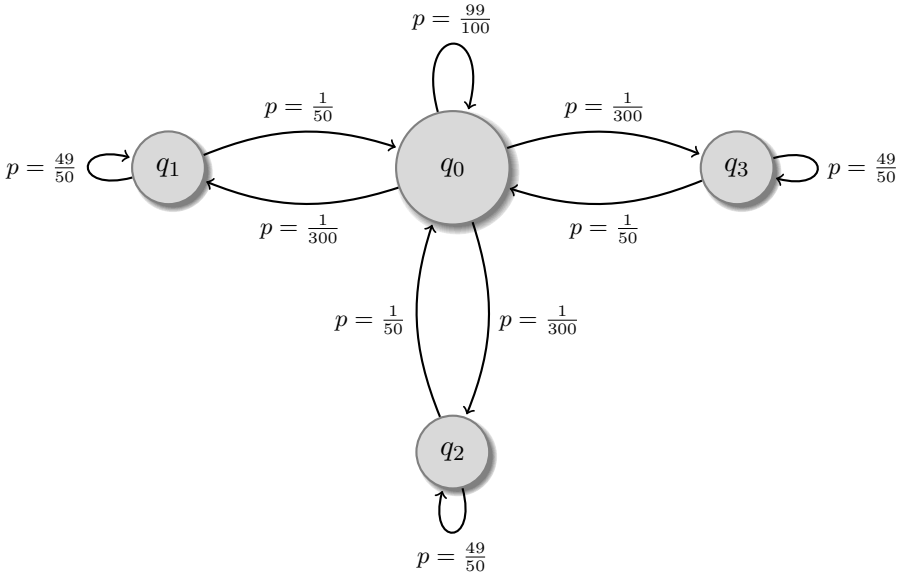


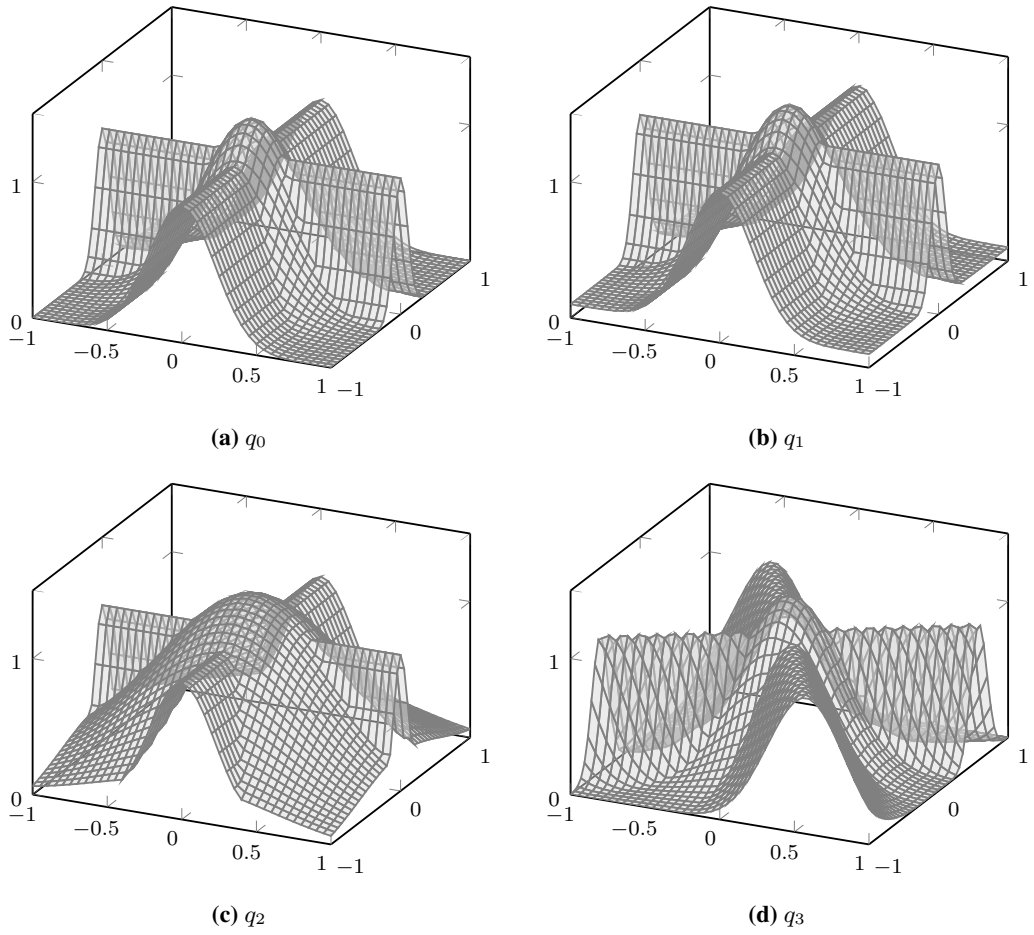
Figure 6.1: Overview of the Probabilistic Finite-State Automaton used to create the synthetic dataset.

Table 6.1: Number of occurrences and samples for the four regimes in the considered instance of the Change-Point Cross 2D dataset.

regime	#occurrences	#samples	%samples
q_0	32	3570	71.40%
q_1	11	454	9.08%
q_2	8	365	7.30%
q_3	12	611	12.22%
	63	5000	100.00%

constant additive term of 0.1. Consequently, the effect of the regime change on the output value is invariant with respect to the input space. In Figure 6.2c-d, we can observe that this is not the case for the second and third alternatives. These regimes are obtained by a modification of a coefficient and a counter clockwise rotation of the input space by $30°$, respectively. The difference between these regimes and the base regime varies with respect to the input space, such that regime changes may not be reflected consistently in the output distribution.

As shown in Figure 6.3, the Change-Point Cross 2D dataset consists of in total 6000 samples, all of which are corrupted by $\mathcal{N}\left(0, 0.1^2\right)$ random noise. Of these 6000 samples, an initial 1000 are fixed to the base regime q_0 to allow the learning algorithm to obtain a reliable base model prior to possible regime changes. Subsequently, the remaining 5000 samples are generated using the automaton from Figure 6.1. As we can see in Table 6.1, approximately 70% of these are generated according to the base regime.



(a) q_0 **(b)** q_1

(c) q_2 **(d)** q_3

Figure 6.2: The four output regimes considered for the Change-Point Cross 2D problem.

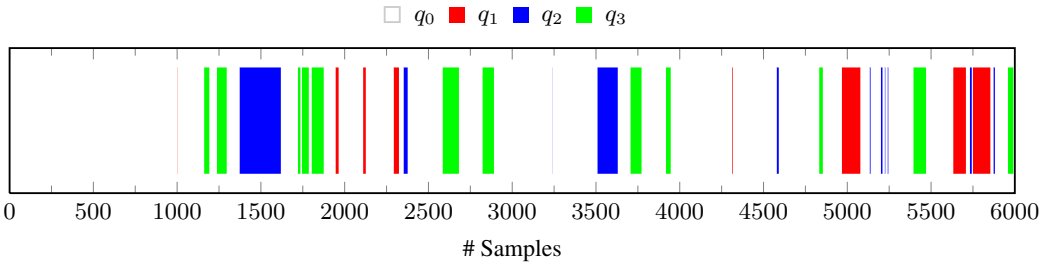


Figure 6.3: Sequence of the selected regimes for the Change-Point Cross 2D dataset. The first 1000 samples are fixed to the base regime q_0.

6.2.1 Experimental Setup

The experiments are formulated as an online sequential change detection problem. The learning method is incremental SSGPR[50] with non-fixed spectral points, which demonstrated an excellent trade-off between computational requirements and generalization performance on the Cross 2D dataset in Section 5.1.2. Hyperparameter optimization is performed on the initial 1000 training samples, using a setup that is identical to the experiments in Chapter 5. A model of the base regime is obtained by (incrementally) training Sparse Spectrum Gaussian Process Regression (SSGPR) on the initial 1000 training samples, after which SSGPR continues updating on the remaining 5000 test samples. Though the learning method receives all samples (i.e., both training and test) in a single pass, subsequent analysis for the change detection problem is restricted to the latter 5000 test samples.

Change detection is performed using WGLR as explained in Section 6.1. The problem of interest is a binary classification problem, namely, whether or not the base regime is active. Although the regime state is known exactly throughout the entire dataset, explicit annotation of the regimes by the algorithm (e.g., discriminating q_1 versus q_2) is not considered. The binary predictions of WGLR are therefore limited to the decision function d_t in Equation (6.1), where the final measure used to calculate classification errors is given by $d_t > 0$. Furthermore, in order to minimize contamination of the base model, only samples that are classified as belonging to the base regime are used for model updates of SSGPR.

A number of configurations is used to investigate the sensitivity of WGLR to parameter configurations. Firstly, the bounds on θ_0 and θ_1 are chosen such that $a \in \{0, 0.25, 0.5\}$ and $b \in \{0.5, 1\}$, where $a < b$. The effect of the window length on the predictive accuracy is tested using four different window lengths, namely $M \in \{10, 20, 50, 100\}$. Lastly, the threshold h is chosen from a wide range of values $h \in \{0.5, 1, 2, 5, 10, 20, 50\}$. The subset formed by the first 2000 test samples, henceforth referred to as the validation set, is used to prevent overfitting of the parameters to the test data when comparing the classification errors of different configurations.

6.2.2 Results

The predictions of non-base regimes using different configurations of WGLR are primarily analyzed at the hand of receiver operating characteristic (ROC)-curves. These curves capture the transition from exclusively predicting negative values to exclusively predicting positive values. In the former case, the rate of false positives will necessarily be 0, at the cost of a true positive rate of 0 as well (i.e., no positive sample is predicted as such). Conversely, in the latter extreme, both the false and true positive rate will necessarily be 1. With respect to the problem setting under consideration, the output labels are chosen positive when an alternative regime is active, whereas a negative sample equates to the base regime being active. The rate of false positive and negatives is regulated by varying the threshold h.

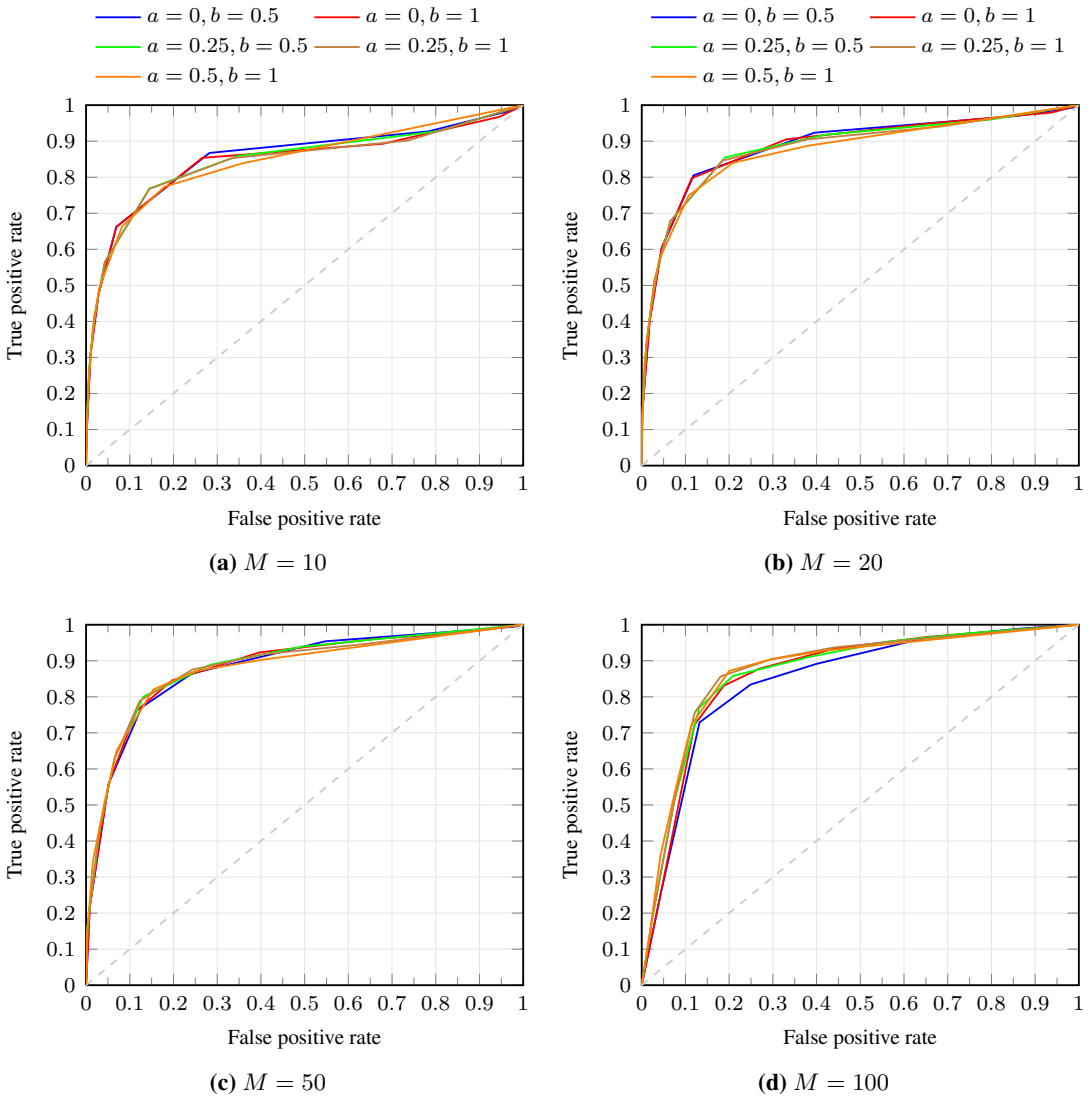
Figure 6.4 shows four ROC-curves corresponding to the different window lengths, each of which contains the results for five different configurations of a and b. It is clearly observable that varying a and b only has a minor effect on the predictions. This property is desirable in practice, as it indicates that the predictive accuracy is not sensitive to (minor) variations of these parameters. Further insight is given by the fact that varying the parameter bounds has a similar effect as varying the threshold h, since both affect the discrimination threshold. With regard to the window length, we observe that the effect of increasing the window length is twofold. Firstly, an increased window length initially causes a slightly higher rate of false positives. Consider $M = 100$: in this case a true positive rate of, for instance, 0.5 causes more false positives with respect to smaller window lengths. The reason is that older regime changes will continue to "resonate" longer when using larger windows. However, this observation is eventually reversed, as is evident from the fact that $M = 100$ suffers less false positives for a desired true positive rate of 0.9 and higher. The optimal window length will therefore depend on the respective cost of false positives and negatives given the application domain.

Assuming equal cost for false positives and false negatives, the optimal configuration as measured on the validation subset obtains an average classification error of $14.05\% \pm 0.77\%$ on the total test set. Although this error may not seem impressive at first sight, we have to consider that regime changes are detected indirectly through the residuals of SSGPR over a dependent sequence of samples. Additionally, regime changes can often only be detected reliably after multiple samples, causing WGLR to make prediction errors in the mean while. The output corresponding to the decision function d_t for the optimal parameter configuration is shown in Figure 6.5. As can be verified, WGLR is generally successful at detecting the alternative regimes, particularly if the regime lasts for a larger number of time steps.

6.3 Online Manipulator Contact Detection

The primary interest of this chapter is to investigate the performance of WGLR on the problem of contact detection of a robotic manipulator using a learned dynamics model. To this extent, a dynamics dataset has been collected from the iCub humanoid in similar manner to the earlier iCub dataset from Section 5.3. However, while the original iCub dataset was obtained without any external disturbances, in the Disturbed iCub dataset a number of external forces of increasing magnitude is applied to the manipulator over a total runtime of 100000 samples. The purpose of these disturbances is to simulate prolonged contact situations, such as grasping and holding an object. The first 10000 samples are not subject to any disturbances and therefore reserved for hyperparameter optimization and initial training of the base regime (i.e., without disturbance).

A total of six different disturbances are considered, as demonstrated in Table 6.2. The first type is a manually applied disturbance, characterized by a highly dynamic profile and large forces and torques. The five remaining disturbances are obtained by (manually) attaching calibrated weights



(a) $M = 10$ **(b)** $M = 20$

(c) $M = 50$ **(d)** $M = 100$

Figure 6.4: ROC-Curves for WGLR on the Change-Point Cross 2D dataset using five different configurations of a and b and window lengths $M \in \{10, 20, 50, 100\}$. The curves are obtained by varying the threshold h and represent the mean of 25 runs of SSGPR^{50} with non-fixed spectral points. Points close to the origin are exemplary for high thresholds, whereas points close to the right-top corner typically indicate a low threshold.

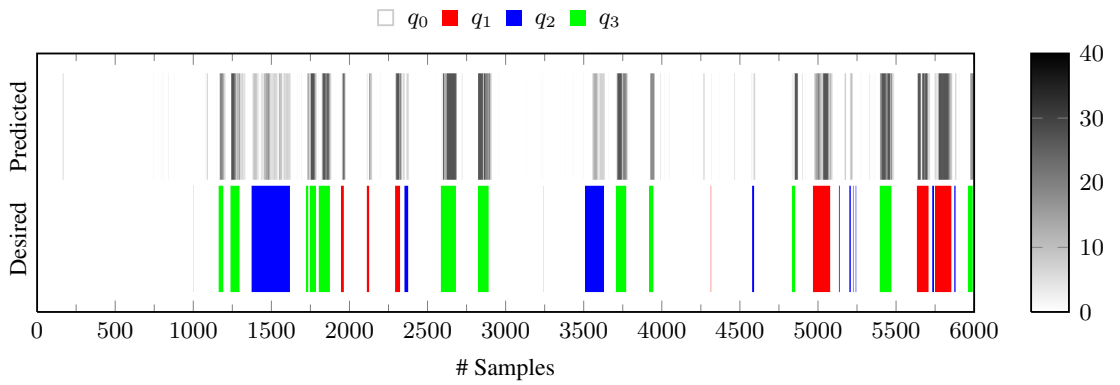


Figure 6.5: Non-base regimes of the Change-Point Cross 2D dataset as predicted using WGLR with configuration $a = 0$, $b = 0.5$, $M = 20$, and $h = 5$, which resulted in the lowest binary classification error on the validation subset. Note that the output values of d_t are capped at 40 for visualization purposes.

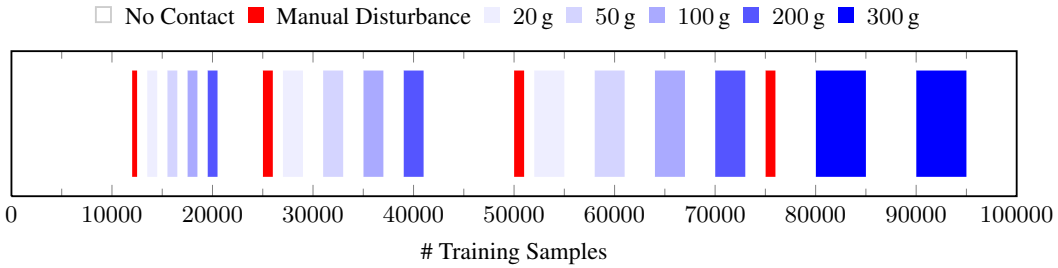


Figure 6.6: Sequence of disturbances for the Disturbed iCub dataset. The first 10000 samples are free of any kind of disturbances.

to the end effector, where the weights are chosen from 20 g, 50 g, 100 g, 200 g, and 300 g. This range of weights is chosen such that the lightest is likely to be undetectable, while the heaviest have a profound impact on the dynamics model and can therefore be detected easily. Note that adding 20 g to the end effector equates to a small additional force of approximately 0.2 N. The corresponding torque varies with the arm configuration, although it can easily be upper bounded at 0.1 N m given the limited arm length of the iCub humanoid. Each disturbance lasts between 10 s and 100 s, or respectively 500 and 5000 samples. The *exact* starting time and length of each regime is unknown, due to the obvious difficulty of manually attaching a weight to a moving manipulator within a 20 ms time frame. Figure 6.6 shows the various regimes applied within the dataset.

6.3.1 Experimental Setup

The setup used for the Disturbed iCub dataset is similar to the setup as described in Section 6.2.1. However, a wider range of bound parameters and thresholds is considered, with $a \in \{0, 0.5, 1, 2\}$ and $b \in \{0.5, 1, 2, 4, 8\}$, where $a < b$, and $h \in \{1, 2, 5, 10, 20, 50, 100, 200, 500\}$. The considered window lengths are identical, namely $M \in \{10, 20, 50, 100\}$. The validation subset is

Table 6.2: Number of occurrences and samples for the seven regimes in the Disturbed iCub dataset.

regime	#occurrences	#samples	%samples
No contact	19	52500	58.33%
Manual disturbance	4	3500	3.89%
20 g	3	6000	6.67%
50 g	3	6000	6.67%
100 g	3	6000	6.67%
200 g	3	6000	6.67%
300 g	2	10000	11.11%
	37	90000	100.00%

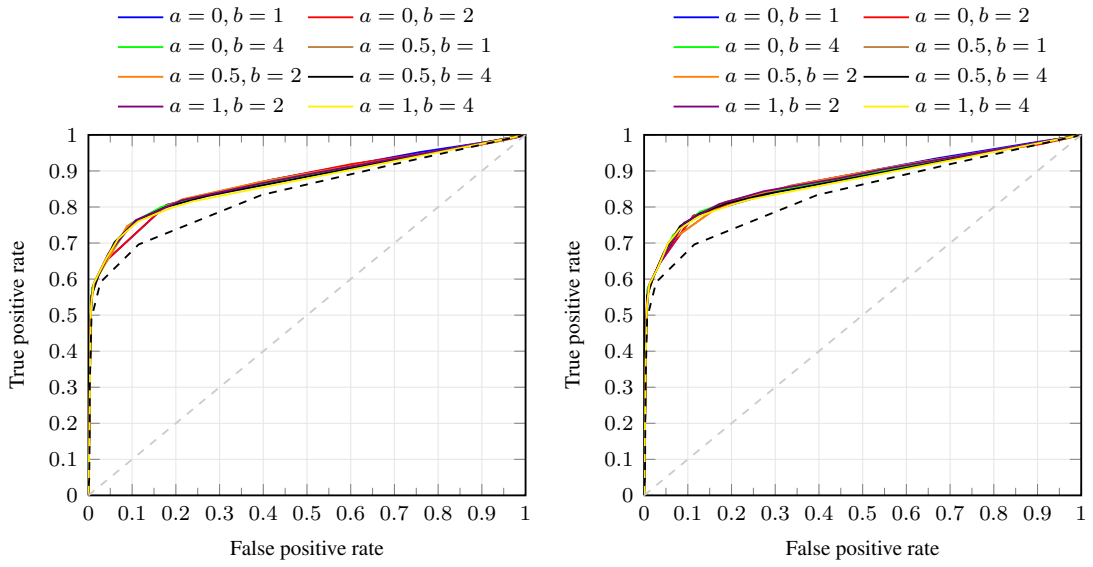
composed of the first 12500 test samples, which include one iteration of the first five disturbances (cf. Figure 6.6).

The learning method in this case is SSGPR^{50} with fixed spectral points, which has shown good performance on the original iCub dataset (cf. Section 5.3.2). Ideally, change detection would be based on the force measurements, as these are invariant to the arm configuration. However, as we observed in Section 5.3.3, the force measurements are subject to drift and the force residuals are therefore a much less reliable indicator for change. The drift compensation presented in Section 5.3.3 cannot be applied to alleviate this problem, since model updates are temporarily suspended for the duration of (detected) contact situations. Nonetheless, preliminary experiments confirm that residuals of the torque predictions show superior performance for change detection. In the following we will therefore limit our attention to the torque residuals.

6.3.2 Results

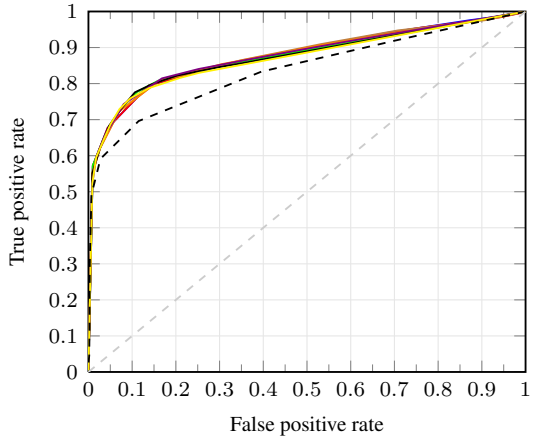
The ROC-curves for a subset of the considered parameter configurations are shown in Figure 6.4. We observe a similar pattern as for the synthetic Change-Point Cross 2D dataset, namely that the performance is not very sensitive to the configuration of bounds a and b, and that increasing the window size does not significantly improve the results. Furthermore, WGLR has difficulties to obtain true positive rates higher than approximately 80% for all window lengths, the primary reason being that the 20 g disturbance is very difficult to detect. Given that this disturbance accounts for 16% of all non-base samples, it is clear that increasing the true positive rate above 84% will consequently result in a large number of false positives. The same effect is present, though to a lesser extent, for the 50 g disturbance. The difficulty of correctly detecting these two minor disturbances explains the apparent "crossover" point around 75% to 80% true positive rate.

A simple per-sample threshold approach is included as reference. This method predicts changes using the residuals on an individual sample, and is emulated using WGLR by setting $M = 1$ and $a = b = 0$. It is clear from Figure 6.4 that WGLR approaches using a larger window length show

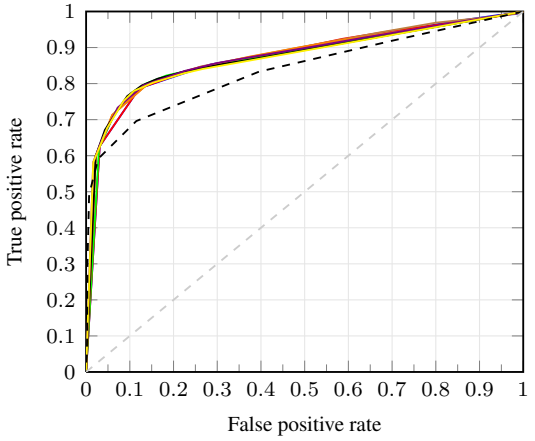


(a) $M = 10$

(b) $M = 20$



(c) $M = 50$



(d) $M = 100$

Figure 6.7: ROC-Curves for WGLR on the Disturbed iCub dataset using five different configurations of a and b and window lengths $M \in \{10, 20, 50, 100\}$. The curves are obtained by varying the threshold h. The per-sample threshold method is included as benchmark and marked with dashed black lines. Points close to the origin are exemplary for high thresholds, whereas points close to the right-top corner typically indicate a low threshold.

Table 6.3: Detection rates per regime for three different parameter configurations on 90000 test samples of the Disturbed iCub dataset. The configuration described in column (a) is $a = 0, b = 4, M = 100, h = 100$, whereas column (b) corresponds to a per-sample threshold approach emulated with parameters $a = 0, b = 0, M = 1, h = 5$. Lastly, column (c) is obtained using parameters $a = 0.5, b = 8, M = 10, h = 20$ and maximizes detection rates while restricting the false negative rate to 1%. The bottom line indicates the overall accuracy for each configuration. The reported results are averaged over 25 runs and indicate a variation of one standard deviation.

	detection rate (%)		
regime	(a)	(b)	(c)
No contact	89.56 (± 0.73)	88.51 (± 0.52)	99.02 (± 0.26)
Manual disturbance	96.05 (± 0.03)	94.17 (± 0.37)	93.40 (± 0.61)
20 g	4.27 (± 1.95)	10.55 (± 1.31)	1.28 (± 0.91)
50 g	72.40 (± 3.03)	35.00 (± 0.94)	1.05 (± 0.26)
100 g	88.68 (± 2.91)	73.21 (± 1.45)	49.25 (± 1.19)
200 g	98.46 (± 0.13)	97.60 (± 0.59)	95.55 (± 1.48)
300 g	98.49 (± 0.13)	98.26 (± 0.16)	97.62 (± 0.39)
	84.51 (± 0.61)	80.64 (± 0.32)	82.05 (± 0.23)

superior detection rates. In order to investigate the exact differences further, we limit our attention to three exemplary configurations. The first is given by $a = 0$, $b = 4$, $M = 100$, and $h = 100$, which resulted in the lowest classification error on the validation set. The second configuration is a per-sample threshold approach given by parameters $a = 0$, $b = 0$, $M = 1$, and $h = 5$. In this case, the threshold h is chosen such that the false positive rate is close to the same rate of the first configuration. The third and last configuration uses parameters $a = 0.5$,$b = 8$, $M = 10$, and $h = 20$, and maximizes change detection rates, while respecting an upper limit of 1% on the false positive rate. The individual detection rates on the 90000 test samples for all three configurations are shown in Table 6.3, respectively. The results in columns (a) and (b) confirm that WGLR shows superior performance with respect to the simple threshold approach. Although the false positive rate is similar in both cases, the detection rates of nearly all types of disturbances are much higher in case of WGLR (cf. Table 6.3a). This is particularly evident for the 50 g disturbance, for which WGLR attains a detection rate more than twice as high as the per-sample threshold approach.

Unsurprisingly, the results in column (c) demonstrate that reduction of the false negative rate leads to inferior detection rates of all disturbances. Nonetheless, the manual, 200 g, and 300 g disturbances can still be detected reliably. In contrast, only half of the samples that were disturbed with an additional 100 g weight are still detected as such, while 20 g and 50 g weights are proven undetectable. The task dependent balance between false positives and negatives is primarily regulated using lower bound b and threshold h. Changing these parameters has an immediate impact in WGLR, therefore allowing changes to be made during runtime to obtain the desired tradeoff.

An interesting observation in Table 6.3a is that the recognition rate of the 20 g disturbance is actually lower than the false positive rate (i.e., 8.79%). This is a strong indication that the samples

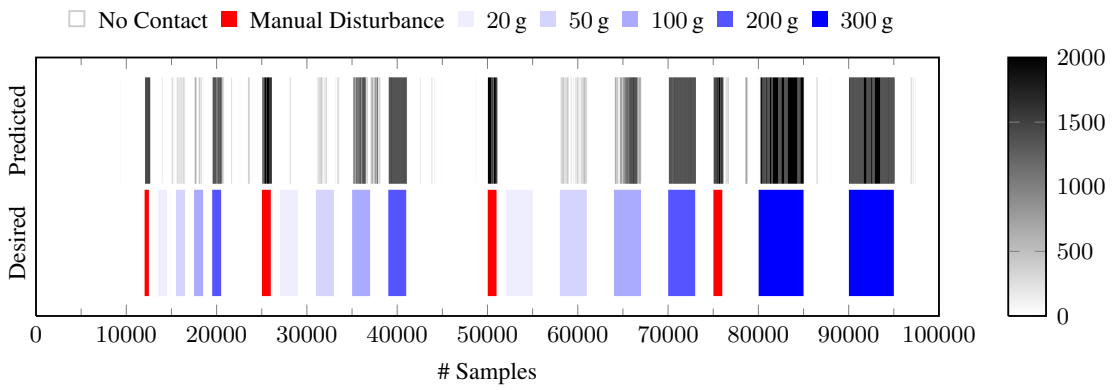


Figure 6.8: Non-base regimes of the Disturbed iCub dataset as predicted using WGLR with configuration $a = 0$, $b = 4$, $M = 100$, and $h = 100$, which resulted in the lowest binary classification error on the validation subset. Note that the output values of d_t are capped at 2000 for visualization purposes.

that result in a false positive error (i.e., samples that are erroneously detected as disturbance) are not evenly spread throughout the dataset. The predictions obtained using the corresponding configuration, shown in Figure 6.8, confirm that false positives indeed tend to occur in batches rather than as single instances. Consider for instance $t \approx 15000$, $t \approx 79500$, or $t \approx 97000$; around these time steps, a series of samples is erroneously classified as non-base regime. The exact reason for this behavior is unknown, although it seems that the cause is rather external (e.g., faulty sensor measurements) than true prediction errors of the contact detection procedure. Regardless, the predictions in Figure 6.8 generally seem to give a reliable indication of the manual disturbance and additional weights of more than $50\,\mathrm{g}$. The case of a disturbance of $20\,\mathrm{g}$, on the other hand, is proven to be undetectable, while $50\,\mathrm{g}$ is around the crossover point.

7 CONCLUSIONS

Sensorimotor anticipation is important for safe and efficient operation of robots in unstructured and non-stationary human environments. This thesis presented incremental variants of the Random Fourier Regularized Least Squares (RFRLS) and Sparse Spectrum Gaussian Process Regression (SSGPR) regression methods that allow sensorimotor associations to be learned during robot operation. The design objectives of these incremental learning methods were (1) a theoretical foundation, (2) computational efficiency, and (3) practical convenience. Rather than developing a novel algorithm from the ground up, the methods are based on the thoroughly studied Regularized Least Squares (RLS) and Gaussian Process Regression (GPR) algorithms, therefore ensuring a solid theoretical foundation. Extension of these linear methods with a kernel approximation is essential for generalization performance on non-linear problems as well as computational complexity. The typical linear dependency of standard kernel methods (KMs) on the number of training samples is avoided by explicitly performing a finite-dimensional feature mapping. Consequently, efficient and exact incremental update routines can be used and, contrary to related work, no additional mechanisms are required to constrain computational requirements. This bounded update complexity facilitates open-ended learning as well as use in a (hard) real-time setting. Employment in resource constrained environments, such as embedded systems, is further facilitated by the algorithmic simplicity of the approach. As a consequence, the methods are easily understood, implemented, and deployed in practice. Other features that contribute to practical convenience are a limited number of hyperparameters and, in case of incremental SSGPR, a principled methodology (i.e., likelihood optimization) to optimize these hyperparameters automatically. Furthermore, a single hyperparameter controls the dimensionality of the feature mapping and therefore the trade-off between computational cost and predictive accuracy, such that generalization performance can easily be maximized while conforming to domain specific timing constraints.

A number of synthetic and real robot dynamics datasets were used to evaluate the empirical performance of the proposed methods with respect to current state of the art. An initial set of experiments demonstrates that RFRLS is competitive in terms of generalization performance in a batch learning setting, while the computational efficiency is significantly superior in absolute sense as well as in terms of scaling behavior. The most important experimental contribution of this thesis is the comparison of incremental SSGPR with Locally Weighted Projection Regression (LWPR) and batch GPR on a series of large scale manipulator dynamics data sets. These experiments demonstrate that incremental SSGPR significantly outperforms LWPR in terms of predictive accuracy as well as computational requirements. In addition, it outperforms an analytical Rigid Body Dynamics (RBD) model by several factors in terms of predictive accuracy, while having comparable computational requirements (less than 1 ms per update). This demonstrates both the benefit of learned sensorimotor models over analytical solutions as well as the computational efficiency of the proposed method. Furthermore, batch GPR was inferior in this realistic learning setting, in which an initial subset of the samples was used for training and the remaining samples were used for testing.

The relevant interpretation of this result is that incremental learning poses significant advantages when samples cannot be assumed independent and identically distributed (i.i.d.). In the considered dynamics learning setting, subsequent samples are almost surely dependent and possibly subject to concept drift. Incremental learning methods have significant advantages in this setting, as they use a maximum amount of training data and adapt continuously to changes in the environment. Further experimentation on the problem of learning the mapping from head and arm posture to the visual localization of the hand confirms these findings. The samples for this problem are highly dependent due to the data collection methodology. Both GPR and LWPR perform poorly on this problem and are unable to compete with a trivial reference method. Incremental SSGPR, on the other hand, handles this adverse learning condition adequately, evidenced by the fact that it outperforms the reference method by a significant margin.

In the final part of this thesis, a dynamics model obtained using learning was used for contact detection on the iCub humanoid robot. External forces exerted on the manipulator result in deviations between measured dynamics and those predicted by the model. Consequently, contact situations are characterized by a change in the predictive accuracy of the learning method. The proposed approach uses incremental SSGPR to learn the dynamics model and a Window-Limited Generalized Likelihood Ratio (WGLR) method to detect changes in the distribution of the residuals. Empirical validation using real-life data from the iCub humanoid suggests that weights of 100 g mounted at the end effector can reliably be detected using this approach.

7.1 Future Work

The emphasis in this thesis was the development of an incremental learning method for use in robotic learning problems. There are a number of interesting directions for future work. Within the application domain of robotics, it would be interesting to further investigate the use of these methods in the context of model-based reinforcement learning (Sutton and Barto, 1998). Incremental SSGPR could be particularly useful in this setting, due to the availability of a predictive variance. This measure of uncertainty in the predictions can effectively be used as guidance for active learning and the tradeoff between exploration versus exploitation, as demonstrated previously using related learning methods (Strehl and Littman, 2008; Li and Littman, 2010). With respect to the study of autonomously acting and developing robots, this is closely related to the concept of intrinsic motivation (Hart and Grupen, 2010; Oudeyer et al., 2010). Regardless, although robotics applications have been emphasized in this thesis, the proposed methods are by no means limited to this application domain. In future work, it would be interesting to investigate their performance in other application domains that are incremental by nature, such as time series modeling (Hamilton, 1994; Fan and Yao, 2003) and adaptive filtering (Haykin, 2001; Sayed, 2008; Liu et al., 2010).

There are several standing challenges for future work regarding technical properties of the algorithms. Though hyperparameters can be tuned in SSGPR using log marginal likelihood, this requires collection of an initial batch of training samples. An interesting and useful extension would therefore be to investigate whether hyperparameter optimization can be performed "online", that is during operation of the algorithm. This is not as straightforward as it may initially seem, since adapting the hyperparameters invalidates all knowledge learned thus far. It is therefore required to either change the configuration gradually (and accepting a limited detrimental effect of changes) or to retrain completely after changes using previous samples stored in memory. Furthermore, it would be interesting to see whether recent extensions of KMs in general and GPR in particular could be integrated in incremental SSGPR, while respecting its original design objectives. These extensions include dependent multi-output learning[1] (Micchelli and Pontil, 2005; Evgeniou et al., 2005; Boyle and Frean, 2005; Bonilla et al., 2008; Alvarez and Lawrence, 2009), heteroskedastic noise models (Le et al., 2005; Kersting et al., 2007), and Bayesian change point detection (Turner et al., 2009; Saatçi et al., 2010). A primary difficulty in the integration of these extensions is the requirement of incremental operation with bounded time and space complexity.

[1] In this thesis, multiple outputs were learned concurrently by assuming independence among them.

BIBLIOGRAPHY

Achlioptas, D., McSherry, F., & Schölkopf, B. (2002). Sampling techniques for kernel methods. In: *Advances in Neural Information Processing Systems 14*, T. G. Dietterich, S. Becker, & Z. Ghahramani, ed., pages 335–342. MIT Press.

Aizerman, M. A., Braverman, E. M., & Rozoner, L. I. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837.

Alvarez, M. & Lawrence, N. D. (2009). Sparse convolved gaussian processes for multi-output regression. In: *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, & L. Bottou, ed., pages 57–64. MIT Press.

Anderson, E., Bai, Z., Bischof, C. H., Blackford, S., Demmel, J., Dongarra, J. J., Croz, J. D., Hammarling, S., Greenbaum, A., McKenney, A., & Sorensen, D. C. (1999). *LAPACK Users' Guide*, (third ed.). Society for Industrial and Applied Mathematics.

Appel, U. & Brandt, A. V. (1983). Adaptive sequential segmentation of piecewise stationary time series. *Information Sciences*, 29(1):27–56. Institute of Electrical and Electronics Engineers Workshop 'Applied Time Series Analysis'.

Arriaga, R. I. & Vempala, S. (2006). An algorithmic theory of learning: Robust concepts and random projection. *Machine Learning*, 63:161–182.

Bach, F. R. & Jordan, M. I. (2005). Predictive low-rank decomposition for kernel methods. In: *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, pages 33–40. ACM.

Balcan, M.-F., Blum, A., & Vempala, S. (2006). Kernels as features: On kernels, margins, and low-dimensional mappings. *Machine Learning*, 65:79–94.

Basseville, M. & Nikiforov, I. V. (1993). *Detection of abrupt changes: theory and application*. Prentice-Hall, Inc.

Biau, G., Devroye, L., & Lugosi, G. (2008). On the performance of clustering in hilbert spaces. *IEEE Transactions on Information Theory*, 54(2):781–790.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc.

Björck, Å. (1996). *Numerical Methods for Least Squares Problems*. SIAM.

Blum, A. (2006). Random projection, margins, kernels, and feature-selection. In: *Workshop on Subspace, Latent Structure and Feature Selection (SLSFS 2005)*, C. Saunders, M. Grobelnik, S. R. Gunn, & J. Shawe-Taylor, ed., volume 3940, pages 52–68. Springer.

Bochner, S. (1933). Monotone funktionen, stieltjessche integrale und harmonische analyse. *Mathematische Annalen*, 108:378–410. 10.1007/BF01452844.

Bonilla, E., Chai, K. M., & Williams, C. (2008). Multi-task gaussian process prediction. In: *Advances in Neural Information Processing Systems 20*, J. Platt, D. Koller, Y. Singer, & S. Roweis, ed., pages 153–160. MIT Press.

Bordes, A., Ertekin, S., Weston, J., & Bottou, L. (2005). Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In: *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, pages 144–152. ACM.

Bousquet, O. & Elisseeff, A. (2002). Stability and generalization. *Journal of Machine Learning Research*, 2:499–526.

Boyle, P. & Frean, M. (2005). Dependent gaussian processes. In: *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, & L. Bottou, ed., pages 217–224. MIT Press.

Brown, R. L., Durbin, J., & Evans, J. M. (1975). Techniques for testing the constancy of regression relationships over time. *Journal of the Royal Statistical Society. Series B (Methodological)*, 37(2):149–192.

Bădoiu, M. & Clarkson, K. L. (2008). Optimal core-sets for balls. *Computational Geometry: Theory and Applications*, 40(1):14–22.

Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167.

Buttazzo, G. C. (1997). *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Kluwer Academic Publishers.

Capizzi, G. (2001). Design of change detection algorithms based on the generalized likelihood ratio test. *Environmetrics*, 12(8):749–756.

Caponnetto, A. & Vito, E. D. (2007). Optimal rates for the regularized least-squares algorithm. *Foundations of Computational Mathematics*, 7:331–368.

Cauwenberghs, G. & Poggio, T. (2001). Incremental and decremental support vector machine learning. In: *Advances in Neural Information Processing Systems 13*, pages 409–415. MIT Press.

Cavallanti, G., Cesa-Bianchi, N., & Gentile, C. (2007). Tracking the best hyperplane with a simple budget perceptron. *Machine Learning*, 69:143–167.

Cederborg, T., Li, M., Baranes, A., & Oudeyer, P.-Y. (2010). Incremental local online gaussian mixture regression for imitation learning of multiple tasks. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 267–274.

Cesa-Bianchi, N., Conconi, A., & Gentile, C. (2004). On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057.

Cesa-Bianchi, N. & Gentile, C. (2008). Improved risk tail bounds for on-line algorithms. *IEEE Transactions on Information Theory*, 54(1):386–390.

Cesa-Bianchi, N. & Lugosi, G. (2006). *Prediction, Learning, and Games*. Cambridge University Press.

Chapelle, O., Vapnik, V. N., Bousquet, O., & Mukherjee, S. (2002). Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1–3):131–159.

Comaniciu, D., Ramesh, V., & Meer, P. (2003). Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25:564–575.

Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20:273–297.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., & Singer, Y. (2006). Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585.

Crammer, K., Kandola, J. S., & Singer, Y. (2004). Online classification on a budget. In: *Advances in Neural Information Processing Systems 16*, S. Thrun, L. Saul, & B. Schölkopf, ed. MIT Press.

Cristianini, N., Campbell, C., & Shawe-Taylor, J. (1999). Dynamically adapting kernels in support vector machines. In: *Advances in Neural Information Processing Systems 11*, M. J. Kearns, S. A. Solla, & D. A. Cohn, ed., pages 204–210. MIT Press.

Cristianini, N. & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

Csató, L. & Opper, M. (2002). Sparse on-line gaussian processes. *Neural Computation*, 14:641–668.

Cucker, F. & Zhou, D. X. (2007). *Learning Theory: An Approximation Theory Viewpoint (Cambridge Monographs on Applied & Computational Mathematics)*. Cambridge University Press.

Dekel, O., Shalev-Shwartz, S., & Singer, Y. (2008). The forgetron: A kernel-based perceptron on a budget. *SIAM Journal on Computing*, 37(5):1342–1372.

Dongarra, J. J., Bunch, J. R., Moler, C. B., & Stewart, G. W. (1979). *LINPACK users' guide*. Society for Industrial and Applied Mathematics.

Downs, T., Gates, K. E., & Masters, A. (2002). Exact simplification of support vector solutions. *Journal of Machine Learning Research*, 2:293–297.

Drineas, P. & Mahoney, M. W. (2005). On the nyström method for approximating a gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2153–2175.

Drucker, H., Burges, C. J. C., Kaufman, L., Smola, A. J., & Vapnik, V. (1996). Support vector regression machines. In: *Advances in Neural Information Processing Systems 9*, M. Mozer, M. I. Jordan, & T. Petsche, ed., pages 155–161. The MIT Press.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern Classification*. Wiley-Interscience.

Engel, Y., Mannor, S., & Meir, R. (2002). Sparse online greedy support vector regression. In: *ECML '02: Proceedings of the 13th European Conference on Machine Learning*, pages 84–96. Springer-Verlag.

Engel, Y., Mannor, S., & Meir, R. (2004). The kernel recursive least squares algorithm. *IEEE Transactions on Signal Processing*, 52:2275–2285.

Escalante, H. J., Montes, M., & Sucar, L. E. (2009). Particle swarm model selection. *Journal of Machine Learning Research*, 10:405–440.

Evgeniou, T., Micchelli, C. A., & Pontil, M. (2005). Learning multiple tasks with kernel methods. *Journal of Machine Learning Research*, 6:615–637.

Evgeniou, T., Pontil, M., & Poggio, T. (2000). Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13:1–50.

Fan, J. & Yao, Q. (2003). *Nonlinear Time Series: Nonparametric and Parametric Methods*. Springer Series in Statistics. Springer.

Featherstone, R. (2007). *Rigid Body Dynamics Algorithms*. Springer-Verlag New York, Inc.

Fine, S. & Scheinberg, K. (2002). Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264.

Fradkin, D. & Madigan, D. (2003). Experiments with random projections for machine learning. In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 517–522. ACM.

Freund, Y. & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, 37:277–296.

Friedrichs, F. & Igel, C. (2004). Evolutionary tuning of multiple svm parameters. In: *ESANN 2004: Proceedings of the 12th European Symposium on Artificial Neural Networks*, pages 519–524.

Fu, W. J. (1998). Penalized regressions: The bridge versus the lasso. *Journal of Computational and Graphical Statistics*, 7(3):397–416.

Fumagalli, M., Gijsberts, A., Ivaldi, S., Jamone, L., Metta, G., Natale, L., Nori, F., & Sandini, G. (2010). Learning to exploit proximal force sensing: A comparison approach. In: *From Motor Learning to Interaction Learning in Robots*, O. Sigaud & J. Peters, ed., pages 149–167. Springer.

Fung, G. & Mangasarian, O. L. (2001). Proximal support vector machine classifiers. In: *Proceedings of the seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 77–86. ACM.

Gaber, M. M., Zaslavsky, A., & Krishnaswamy, S. (2005). Mining data streams: a review. *ACM SIGMOD Record*, 34:18–26.

Gates, B. (2007). A robot in every home. *Scientific American Magazine*, 296(1):58–65.

Ghahramani, Z. & Jordan, M. I. (1994). Supervised learning from incomplete data via an EM approach. In: *Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. T. Tesauro, & J. Alspector, ed., pages 120–127. Morgan Kaufmann.

Gijsberts, A. & Metta, G. (2011). Incremental learning of robot dynamics using random features (accepted). In: *IEEE International Conference on Robotics and Automation*.

Gijsberts, A., Metta, G., & Rothkrantz, L. (2010a). Evolutionary optimization of least-squares support vector machines. In: *Data Mining*, R. Sharda, S. Voß, R. Stahlbock, S. F. Crone, & S. Lessmann, ed., volume 8 of *Annals of Information Systems*, pages 277–297. Springer US.

Gijsberts, A., Tommasi, T., Metta, G., & Caputo, B. (2010b). Object recognition using visuo-affordance maps. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1572–1578.

Girosi, F., Jones, M., & Poggio, T. (1993). Priors stabilizers and basis functions: From regularization to radial, tensor and additive splines. Technical report, Massachusetts Institute of Technology.

Goldsmith, S. F., Aiken, A. S., & Wilkerson, D. S. (2007). Measuring empirical computational complexity. In: *Proceedings of the the 6th joint meeting of the European software engineering*

conference and the ACM SIGSOFT symposium on The foundations of software engineering, ESEC-FSE '07, pages 395–404. ACM.

Golub, G. H. & Loan, C. F. V. (1996). *Matrix computations (3rd ed.)*. Johns Hopkins University Press.

Hagan, M. T. & Menhaj, M. B. (1994). Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks*, 5:989–993.

Hager, W. W. (1989). Updating the inverse of a matrix. *SIAM Rev.*, 31:221–239.

Hamilton, J. D. (1994). *Time Series Analysis*, (1 ed.). Princeton University Press.

Hart, S. & Grupen, R. (2010). Learning generalizable control programs (submitted). *IEEE Transactions on Autonomous Mental Development*.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, (2 ed.). Springer.

Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR.

Haykin, S. (2001). *Adaptive Filter Theory (4th Edition)*. Prentice Hall.

Helwig, S., Hüffner, F., Rössling, I., & Weinard, M. (2010). Selected design issues. In: *Algorithm Engineering: Bridging the Gap between Algorithm Theory and Practice*, M. Müller-Hannemann & S. Schirra, ed., volume 5971 of *Lecture Notes in Computer Science*, pages 58–126. Springer.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, 40:185–234.

Hoerl, A. E. & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67.

Horn, R. A. & Johnson, C. R. (1986). *Matrix Analysis*. Cambridge University Press.

Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2006). Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1-3):489–501. Neural Networks - Selected Papers from the 7th Brazilian Symposium on Neural Networks (SBRN '04), 7th Brazilian Symposium on Neural Networks.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3:79–87.

Jamone, L., Nori, F., Metta, G., & Sandini, G. (2006). James: A humanoid robot acting over an unstructured world. In: *International Conference on Humanoid Robots*, pages 143–150.

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In: *Proceedings of the 10th European Conference on Machine Learning*, pages 137–142. Springer-Verlag.

Johnson, W. & Lindenstrauss, J. (1984). Extensions of lipschitz mappings into a hilbert space. In: *Conference in Modern Analysis and Probability*, volume 26 of *Contemporary Mathematics*, pages 189–206. American Mathematical Society.

Jones, E., Oliphant, T., Peterson, P., et al. (2001). SciPy: Open source scientific tools for Python.

Kakade, S., Seeger, M., & Foster, D. (2006). Worst-case bounds for gaussian process models. In: *Advances in Neural Information Processing Systems 18*, Y. Weiss, B. Schölkopf, & J. Platt, ed., pages 619–626. MIT Press.

Kakade, S. M. & Tewari, A. (2009). On the generalization ability of online strongly convex programming algorithms. In: *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, & L. Bottou, ed., pages 57–64. MIT Press.

Keerthi, S. S., Chapelle, O., & DeCoste, D. (2006). Building support vector machines with reduced classifier complexity. *Journal of Machine Learning Research*, 7:1493–1515.

Keerthi, S. S., Sindhwani, V., & Chapelle, O. (2007). An efficient method for gradient-based adaptation of hyperparameters in svm models. In: *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. Platt, & T. Hoffman, ed., pages 673–680. MIT Press.

Kersting, K., Plagemann, C., Pfaff, P., & Burgard, W. (2007). Most likely heteroscedastic gaussian process regression. In: *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 393–400. ACM.

Kivinen, J., Smola, A. J., & Williamson, R. C. (2004). Online learning with kernels. *IEEE Transactions on Signal Processing*, 52(8):2165–2176.

Klanke, S., Vijayakumar, S., & Schaal, S. (2008). A library for locally weighted projection regression. *Journal of Machine Learning Research*, 9:623–626.

Lampert, C. H. (2009). *Kernel Methods in Computer Vision*. Now Publishers Inc.

Lanckriet, G. R. G., Cristianini, N., Bartlett, P., Ghaoui, L. E., & Jordan, M. I. (2004). Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72.

Lázaro-Gredilla, M., Quiñonero-Candela, J., & Figueiras-Vidal, A. R. (2007). Sparse spectral sampling gaussian processes. Technical Report MSR-TR-2007-152, Microsoft Research.

Lázaro-Gredilla, M., Quiñonero-Candela, J., Rasmussen, C. E., & Figueiras-Vidal, A. R. (2010). Sparse spectrum gaussian process regression. *Journal of Machine Learning Research*, 11:1865–1881.

Le, Q. V., Smola, A. J., & Canu, S. (2005). Heteroscedastic gaussian process regression. In: *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, pages 489–496. ACM.

Lee, Y.-J. & Mangasarian, O. L. (2001). RSVM: Reduced support vector machines. In: *Proceedings of the SIAM International Conference on Data Mining*. SIAM.

Lessmann, S., Stahlbock, R., & Crone, S. F. (2006). Genetic algorithms for support vector machine model selection. In: *IJCNN '06: International Joint Conference on Neural Networks*, pages 3063–3069. IEEE Press.

Li, L. & Littman, M. L. (2010). Reducing reinforcement learning to kwik online regression. *Annals of Mathematics and Artificial Intelligence*, 58:217–237.

Liang, Z. & Li, Y. (2009). Incremental support vector machine learning in the primal and applications. *Neurocomputing*, 72:2249–2258.

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318.

Littlestone, N. (1989). From on-line to batch learning. In: *Proceedings of the Second Annual Workshop on Computational learning theory*, pages 269–284. Morgan Kaufmann Publishers Inc.

Liu, W., Principe, J. C., & Haykin, S. (2010). *Kernel Adaptive Filtering: A Comprehensive Introduction*, (1st ed.). Wiley Publishing.

Lo Gerfo, L., Rosasco, L., Odone, F., Vito, E. D., & Verri, A. (2008). Spectral algorithms for supervised learning. *Neural Computation*, 20:1873–1897.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444.

Lopes, M. & Santos-Victor, J. (2006). Learning sensory-motor maps for redundant robots. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2670–2676.

Lorden, G. (1971). Procedures for reacting to a change in distribution. *The Annals of Mathematical Statistics*, 42(6):1897–1908.

Lukoševičius, M. & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149.

Lungarella, M., Metta, G., Pfeifer, R., & Sandini, G. (2003). Developmental robotics: a survey. *Connection Science*, 15(4):151–190.

Ma, J., Theiler, J., & Perkins, S. (2003). Accurate on-line support vector regression. *Neural Computation*, 15:2683–2703.

MacKay, D. J. C. (1992). Bayesian interpolation. *Neural Computation*, 4:415–447.

MacKay, D. J. C. (1995). Probable networks and plausible predictions - a review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6:469–505.

MacKay, D. J. C. (1999). Comparison of approximate methods for handling hyperparameters. *Neural Computation*, 11:1035–1068.

MacKay, D. J. C. (2002). *Information Theory, Inference & Learning Algorithms*. Cambridge University Press.

Martin, M. (2002). On-line support vector machine regression. In: *Proceedings of the 13th European Conference on Machine Learning*, ECML '02, pages 282–294. Springer-Verlag.

Mercer, J. (1909). Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society, London*, 209:415–446.

Metta, G., Natale, L., Nori, F., Sandini, G., Vernon, D., Fadiga, L., von Hofsten, C., Rosander, K., Lopes, M., Santos-Victor, J., Bernardino, A., & Montesano, L. (2010). The icub humanoid robot: An open-systems platform for research in cognitive development. *Neural Networks*, 23:1125–1134.

Metta, G., Sandini, G., Vernon, D., Caldwell, D., Tsagarakis, N., Beira, R., Santos-Victor, J., Ijspeert, A., Righetti, L., Cappiello, G., Stellin, G., & Becchi, F. (2006). The robotcub roject - an open framework for research in embodied cognition. In: *Humanoids Workshop, Proceedings of the IEEE–RAS International Conference on Humanoid Robots*.

Micchelli, C. A. & Pontil, M. (2005). Kernels for multi-task learning. In: *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, & L. Bottou, ed., pages 921–928. MIT Press.

Mitzenmacher, M. & Upfal, E. (2005). *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

Mohri, M. & Rostamizadeh, A. (2010). Stability bounds for stationary ϕ-mixing and β-mixing processes. *Journal of Machine Learning Research*, 11:789–814.

Mol, C. D., Vito, E. D., & Rosasco, L. (2009). Elastic-net regularization in learning theory. *Journal of Complexity*, 25:201–230.

Momma, M. & Bennett, K. P. (2002). A pattern search method for model selection of support vector regression. In: *Proceedings of the Second SIAM International Conference on Data Mining*. SIAM.

Motwani, R. & Raghavan, P. (1995). *Randomized Algorithms*. Cambridge University Press.

Natale, L., Nori, F., Sandini, G., & Metta, G. (2007). Learning precise 3D reaching in a humanoid robot. In: *IEEE International Conference of Development and Learning*, pages 324–329.

Nguyen-Tuong, D. & Peters, J. (2010). Incremental sparsification for real-time online model learning. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, M. T. Teh, Y. W., ed., pages 557–564.

Nguyen-Tuong, D. & Peters, J. (2011). Incremental online sparsification for model learning in real-time robot control (accepted). *Neurocomputing*.

Nguyen-Tuong, D., Peters, J., & Seeger, M. (2008a). Computed torque control with nonparametric regression models. In: *2008 American Control Conference*, pages 212–217. IEEE Service Center.

Nguyen-Tuong, D., Peters, J., Seeger, M., & Schölkopf, B. (2008b). Learning inverse dynamics: A comparison. In: *16th European Symposium on Artificial Neural Networks*, pages 13–18. d-side.

Nguyen-Tuong, D., Seeger, M. W., & Peters, J. (2009). Model learning with local gaussian process regression. *Advanced Robotics*, 23(15):2015–2034.

Oliphant, T. E. (2006). *Guide to NumPy*.

Orabona, F., Jie, L., & Caputo, B. (2010). Online-batch strongly convex multi kernel learning. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 0:787–794.

Orabona, F., Keshet, J., & Caputo, B. (2009). Bounded kernel-based online learning. *Journal of Machine Learning Research*, 10:2643–2666.

Oudeyer, P.-Y., Baranes, A., & Kaplan, F. (2010). Intrinsically motivated exploration for developmental and active sensorimotor learning. In: *From Motor Learning to Interaction Learning in Robots*, O. Sigaud & J. Peters, ed., pages 107–146. Springer.

Page, E. S. (1954). Continuous inspection schemes. *Biometrika*, 41(1/2):100–115.

Pan, Z.-W. & Xiao, Q.-W. (2009). Least-square regularized regression with non-iid sampling. *Journal of Statistical Planning and Inference*, 139(10):3579–3587.

Paz, A. (1971). *Introduction to Probabilistic Automata*. Academic Press, Inc.

Peterson, P. (2009). F2PY: a tool for connecting fortran and python programs. *International Journal of Computational Science and Engineering*, 4(4):296–305.

Piaget, J. (1952). *The Origins of Intelligence in Children*. International University Press.

Platt, J. C. (1999). *Fast Training of Support Vector Machines using Sequential Minimal Optimization*, pages 185–208. MIT Press.

Poggio, T. & Girosi, F. (1990). Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497.

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

Rahimi, A. & Recht, B. (2008a). Random features for large-scale kernel machines. In: *Advances in Neural Information Processing Systems 20*, J. Platt, D. Koller, Y. Singer, & S. Roweis, ed., pages 1177–1184. MIT Press.

Rahimi, A. & Recht, B. (2008b). Uniform approximation of functions with random bases. In: *Allerton Conference on Communication Control and Computing (Allerton08)*, pages 555–561.

Ranganathan, A. & Yang, M.-H. (2008). Online sparse matrix gaussian process regression and vision applications. In: *Proceedings of the 10th European Conference on Computer Vision: Part I*, ECCV '08, pages 468–482. Springer-Verlag.

Rani, P., Liu, C., Sarkar, N., & Vanman, E. (2006). An empirical study of machine learning techniques for affect recognition in human-robot interaction. *Pattern Analysis & Applications*, 9:58–69.

Rasmussen, C. E. & Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Rifkin, R., Yeo, G., & Poggio, T. (2003). Regularized least squares classification. In: *Advances in Learning Theory: Methods, Model and Applications*, volume 190, pages 131–154. VIOS Press.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408.

Roth, V. (2004). The generalized lasso. *IEEE Transactions on Neural Networks*, 15(1):16–28.

Saatçi, Y., Turner, R., & Rasmussen, C. E. (2010). Gaussian process change point models. In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, J. Fürnkranz & T. Joachims, ed., pages 927–934. Omnipress.

Saigo, H., Vert, J.-P., Ueda, N., & Akutsu, T. (2004). Protein homology detection using string alignment kernels. *Bioinformatics*, 20:1682–1689.

Saunders, C., Gammerman, A., & Vovk, V. (1998). Ridge regression learning algorithm in dual variables. In: *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 515–521. Morgan Kaufmann Publishers Inc.

Sayed, A. H. (2008). *Adaptive Filters*. Wiley-IEEE Press.

Schaal, S. & Atkeson, C. G. (1998). Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084.

Schneider, M. & Ertel, W. (2010). Robot learning by demonstration with local gaussian process regression. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 255–260.

Schölkopf, B., Herbrich, R., & Smola, A. J. (2001). A generalized representer theorem. In: *Proceedings of the 14th Annual Conference on Computational Learning Theory and and 5th European Conference on Computational Learning Theory*, COLT '01/EuroCOLT '01, pages 416–426. Springer-Verlag.

Schölkopf, B. & Smola, A. J. (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press.

Schölkopf, B., Tsuda, K., & Vert, J.-P. (2004). *Kernel Methods in Computational Biology*. Computational Molecular Biology. The MIT Press.

Seeger, M., Williams, C. K. I., & Lawrence, N. D. (2003). Fast forward selection to speed up sparse gaussian process regression. In: *Artificial Intelligence and Statistics*, volume 9.

Shalev-Shwartz, S. & Singer, Y. (2005). A new perspective on an old perceptron algorithm. In: *Proceedings of the 18th Annual Conference on Learning Theory*, P. Auer & R. Meir, ed., volume 3559 of *Lecture Notes in Computer Science*, pages 264–278. Springer.

Shawe-Taylor, J. & Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Siciliano, B. & Khatib, O. (2008). *Springer Handbook of Robotics*. Springer.

Silva, C. & Ribeiro, B. (2009). *Inductive Inference for Large Scale Text Classification: Kernel Approaches and Techniques*, (1st ed.). Springer Publishing Company, Incorporated.

Smale, S., Rosasco, L., Bouvrie, J. V., Caponnetto, A., & Poggio, T. (2010). Mathematics of the neural response. *Foundations of Computational Mathematics*, 10(1):67–91.

Smale, S. & Yao, Y. (2006). Online learning algorithms. *Foundations of Computational Mathematics*, 6(2):145–170.

Smola, A. J. & Bartlett, P. L. (2001). Sparse greedy gaussian process regression. In: *Advances in Neural Information Processing Systems 13*, T. K. Leen, T. G. Dietterich, & V. Tresp, ed., pages 619–625. MIT Press.

Smola, A. J. & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222.

Smola, A. J., Schölkopf, B., & Müller, K.-R. (1998). The connection between regularization operators and support vector kernels. *Neural Networks*, 11:637–649.

Snelson, E. & Ghahramani, Z. (2006). Sparse gaussian processes using pseudo-inputs. In: *Advances in Neural Information Processing Systems 18*, Y. Weiss, B. Schölkopf, & J. Platt, ed.

Sonnenburg, S., Rätsch, G., & Rieck, K. (2007). Large scale learning with string kernels. In: *Large Scale Kernel Machines*, L. Bottou, O. Chapelle, D. DeCoste, & J. Weston, ed., pages 73–103. MIT Press.

Sonnenburg, S., Rätsch, G., Schäfer, C., & Schölkopf, B. (2006). Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7:1531–1565.

Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2006). *Robot Modeling and Control*. Wiley.

Steinwart, I. (2003). Sparseness of support vector machines. *Journal of Machine Learning Research*, 4:1071–1105.

Steinwart, I., Hush, D., & Scovel, C. (2009). Learning from dependent observations. *Journal of Multivariate Analysis*, 100:175–194.

Strehl, A. & Littman, M. (2008). Online linear regression and its application to model-based reinforcement learning. In: *Advances in Neural Information Processing Systems 20*, J. Platt, D. Koller, Y. Singer, & S. Roweis, ed., pages 1417–1424. MIT Press.

Sun, H. & Wu, Q. (2008). Regularized least square regression with dependent samples. *Advances in Computational Mathematics*, 32:175–189. 10.1007/s10444-008-9099-y.

Sutton, R. S. & Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press.

Suykens, J. A. K., Lukas, L., & Vandewalle, J. (2002a). Sparse approximation using least squares support vector machines. In: *Proceedings of the 2000 IEEE International Symposium on Circuits and Systems*, volume 2, pages 757–760.

Suykens, J. A. K., van Gestel, T., de Brabanter, J., de Moor, B., & Vandewalle, J. (2002b). *Least Squares Support Vector Machines*. World Scientific Publishing Co., Pte, Ltd.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, 58:267–288.

Tikhonov, A. N. & Arsenin, V. Y. (1977). *Solution of Ill-Posed Problems*. V.H. Winston and Sons, Washington D.C.

Titsias, M. (2009). Variational learning of inducing variables in sparse gaussian processes. In: *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 5.

Tsagarakis, N. G., Metta, G., Sandini, G., Vernon, D., Beira, R., Becchi, F., Righetti, L., Santos-Victor, J., Ijspeert, A. J., Carrozza, M. C., & Caldwell, D. G. (2007). iCub: The design and realization of an open humanoid platform for cognitive and neuroscience research. *Advanced Robotics*, 21(10):1151–1175.

Tsang, I. W., Kocsor, A., & Kwok, J. T. (2007). Simpler core vector machines with enclosing balls. In: *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 911–918. ACM.

Tsang, I. W., Kwok, J. T., & Cheung, P.-M. (2005a). Core vector machines: Fast svm training on very large data sets. *Journal for Machine Learning Research*, 6:363–392.

Tsang, I. W., Kwok, J. T., & Lai, K. T. (2005b). Core vector regression for very large regression problems. In: *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 912–919. ACM Press.

Tsang, I. W., Kwok, J. T., & Zurada, J. M. (2006). Generalized core vector machines. *IEEE Transactions on Neural Networks*, 17(5):1126–1140.

Turner, R., Saatçi, Y., & Rasmussen, C. E. (2009). Adaptive sequential bayesian change point detection. In: *Temporal Segmentation Workshop at NIPS 2009*, Z. Harchaoui, ed.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27:1134–1142.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc.

Vijayakumar, S., D'souza, A., & Schaal, S. (2005). Incremental online learning in high dimensions. *Neural Computation*, 17(12):2602–2634.

Vijayakumar, S. & Wu, S. (1999). Sequential support vector classifiers and regression. In: *Proceedings of the Third ICSC Symposia on Intelligent Industrial Automation (IIA'99) and Soft Computing (SOCO'99)*, pages 610–619. ICSC Academic Press.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., & Borgwardt, K. M. (2010). Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242.

Vishwanathan, S. V. N., Schraudolph, N. N., & Smola, A. J. (2006). Step size adaptation in reproducing kernel hilbert space. *Journal of Machine Learning Research*, 7:1107–1133.

Vovk, V., Gammerman, A., & Shafer, G. (2005). *Algorithmic Learning in a Random World*. Springer-Verlag New York, Inc.

Wahba, G. (1990). *Spline Models for Observational Data*, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM).

Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M., & Thelen, E. (2001). Autonomous mental development by robots and animals. *Science*, 291(5504):599–600.

Weston, J., Bordes, A., & Bottou, L. (2005). Online (and offline) on an even tighter budget. In: *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, R. G. Cowell & Z. Ghahramani, ed., pages 413–420. Society for Artificial Intelligence and Statistics.

Whaley, R. C. & Petitet, A. (2005). Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software: Practice and Experience*, 35(2):101–121.

Williams, C. K. I. & Seeger, M. (2001). Using the nyström method to speed up kernel machines. In: *Advances in Neural Information Processing Systems 13*, T. K. Leen, T. G. Dietterich, & V. Tresp, ed., pages 682–688. MIT Press.

Wold, S., Sjöström, M., & Eriksson, L. (2001). Pls-regression: a basic tool of chemometrics. *Chemometrics and Intelligent Laboratory Systems*, 58(2):109–130.

Yu, K., Ji, L., & Zhang, X. (2002). Kernel nearest-neighbor algorithm. *Neural Processing Letters*, 15:147–156.

Zhang, K., Tsang, I. W., & Kwok, J. T. (2008). Improved nyström low-rank approximation and error analysis. In: *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1232–1239. ACM.

Zhang, T. (2005). Data dependent concentration bounds for sequential prediction algorithms. In: *Proceedings of the 18th Annual Conference on Learning Theory*, P. Auer & R. Meir, ed., volume 3559 of *Lecture Notes in Computer Science*, pages 173–187. Springer.

Zhang, W., Tang, X., & Yoshida, T. (2007). Text classification with support vector machine and back propagation neural network. In: *ICCS 2007: Proceedings of the 7th International Conference on Computational Science*, volume 4490 of *Lecture Notes in Computer Science*, pages 150–157. Springer.

Zhdanov, F. & Kalnishkan, Y. (2010). An identity for kernel ridge regression. In: *Proceedings of the 21st international conference on Algorithmic learning theory*, ALT'10, pages 405–419. Springer-Verlag.

Zou, H. & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society B*, 67:301–320.

IMPLEMENTATION A

The majority of the results presented in this thesis have been obtained using a custom unified Python framework. The primary motivation for this framework is to provide an environment that allows

1. rapid prototyping and implementation of algorithms;
2. flexible design of experiments; and
3. high computational performance and efficiency on large scale experiments.

Python is an appropriate language to addressed these requirements, being an advanced and modern programming language supported by a large and comprehensive standard library (i.e., "batteries included"). Furthermore, the NumPy and SciPy packages add support for linear algebra and generic scientific computing (Oliphant, 2006; Jones et al., 2001, respectively). Overall efficiency can be obtained by implementing critical components in efficient C or Fortran extensions, as is common in NumPy and SciPy. A detailed treatment of the design of this framework is outside the scope of this thesis. Nonetheless, the following sections present technical details that are relevant for the experimental results as presented in Chapter 5.

A.1 Technical Details

A.1.1 Gaussian Process Regression and Kernel Regularized Least Squares

The framework contains implementations of Gaussian Process Regression (GPR) and Kernel Regularized Least Squares (KRLS) using the Cholesky factor of the kernel matrix (for details, see

Algorithm 2.1 in Rasmussen and Williams, 2005). Although these methods are relatively easy to implement (in contrast to, e.g., Support Vector Machines), there are a number of important technical details to ensure high performance. Linear algebra operations, such as matrix-vector products or computing the Cholesky factor, are performed efficiently by linking NumPy to the ATLAS library for linear algebra (Whaley and Petitet, 2005). Furthermore, the significant looping overhead of Python when performing a large number of kernel evaluations (e.g., to construct the kernel matrix) is avoided by means of a custom (threaded) C extension for all kernel related operations. Significant savings in memory usage are achieved by performing linear algebra routines "in-place" when appropriate.

A.1.2 Incremental Random Fourier Regularized Least Squares and Sparse Spectrum Gaussian Process Regression

The pseudocode for incremental Random Fourier Regularized Least Squares (RFRLS) and Sparse Spectrum Gaussian Process Regression (SSGPR) can be translated nearly one-to-one to Python code, as demonstrated in Algorithm A.1 in case of SSGPR. The `dgetrs` and `dpotrs` routines are implemented in LAPACK (Anderson et al., 1999) and accessible in Python through the `scipy.linalg.flapack` wrapper module in SciPy[1]. Conversely, the `dchud` routine (implementing rank-1 Cholesky updates) is not implemented in LAPACK or SciPy. LINPACK, the predecessor of LAPACK, however, contains an implementation of `dchud` in Fortran (Dongarra et al., 1979). This implementation has been made accessible from Python using the F2PY Fortran to Python interface builder (Peterson, 2009).

A.1.3 Locally Weighted Projection Regression

Implementing the complicated Locally Weighted Projection Regression (LWPR) algorithm is tedious and error-prone, hence the reference implementation has been used for all presented experiments (Klanke et al., 2008). This implementation is distributed with Python extensions and can therefore easily be integrated in the above-mentioned framework. Furthermore, the library is accessed through an additional wrapper layer that implements the common interface used in the Python framework.

A.2 Hardware and Library Configuration

Unless indicated otherwise, all experiments were performed in double precision using the Gentoo GNU/Linux operating system running on a standard Intel® Core™2 Quad Q9550 processor (cf. 2.83 GHz) with 4 GiB of installed memory. The installed versions of NumPy of SciPy were 1.5.1

[1] Alternatively, one could use `scipy.linalg.lu_solve` and `scipy.linalg.cho_solve`.

Algorithm A.1 Algorithm 4.2 annotated with corresponding Python code.

Require: $\sigma_n > 0, \sigma_f > 0, \boldsymbol{\ell} > \mathbf{0}, D > 0, \operatorname{diag}\left(\boldsymbol{M}^{-\frac{1}{2}}\right) = \boldsymbol{l}^2$

1:	$\boldsymbol{R} \leftarrow \sigma_n \boldsymbol{I}_{2D\times 2D}$	`R = sigma_n * identity(2*D)`
2:	$\boldsymbol{w} \leftarrow \mathbf{0}_{2D\times 1}$	`w = zeros(2*D)`
3:	$\boldsymbol{b} \leftarrow \mathbf{0}_{2D\times 1}$	`b = zeros(2*D)`
4:	$\boldsymbol{\Omega} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{M})_{D\times n}$	`O = randn(D, n) / ell`
5:	**for all** $(\boldsymbol{x}, y)$ **do**	`for x, y in fetch_sample():`

6:	$\boldsymbol{\phi} \leftarrow \frac{\sigma_f}{\sqrt{D}}\left[\cos\left(\boldsymbol{\Omega x}\right)^{\mathrm{T}}, \sin\left(\boldsymbol{\Omega x}\right)^{\mathrm{T}}\right]^{\mathrm{T}}$	`nf = sigma_f / sqrt(D)`
		`Ox = dot(O, x)`
		`phi = nf * vstack((sin(Ox), cos(Ox)))`

7:	$\hat{y} \leftarrow \langle \boldsymbol{w}, \boldsymbol{\phi} \rangle$	`y_hat = dot(phi, w)`
8:	$\boldsymbol{v} \leftarrow \boldsymbol{R}^{\mathrm{T}} \backslash \boldsymbol{\phi}$	`v = dgetrs(R, arange(2*D), phi, 1, 0)`
9:	$s^2 \leftarrow \sigma_n^2\left(1 + \langle \boldsymbol{v}, \boldsymbol{v} \rangle\right)$	`s2 = sigma_n**2 * (1 + dot(v,v))`
10:	$\boldsymbol{b} \leftarrow \boldsymbol{b} + \boldsymbol{\phi} y$	`b += phi * y`
11:	$\boldsymbol{R} \leftarrow \textsc{CholeskyUpdate}(\boldsymbol{R}, \boldsymbol{\phi})$	`R = dchud(R, phi)`
12:	$\boldsymbol{w} \leftarrow \boldsymbol{R} \backslash \left(\boldsymbol{R}^{\mathrm{T}} \backslash \boldsymbol{b}\right)$	`w = dpotrs(R, b)`
13:	**yield** $\left(\hat{y}, s^2\right)$	
14:	**end for**	

and 0.9.0, respectively. For computational efficiency, these libraries were linked against threaded variants of version 3.9.23 of the ATLAS library. The use of threading is primarily beneficial in case of batch GPR and KRLS. In case of incremental RFRLS and SSGPR, the overhead of parallel execution outweighs the computational gains. It is therefore safe to assume that these methods utilize only a single core in practice.