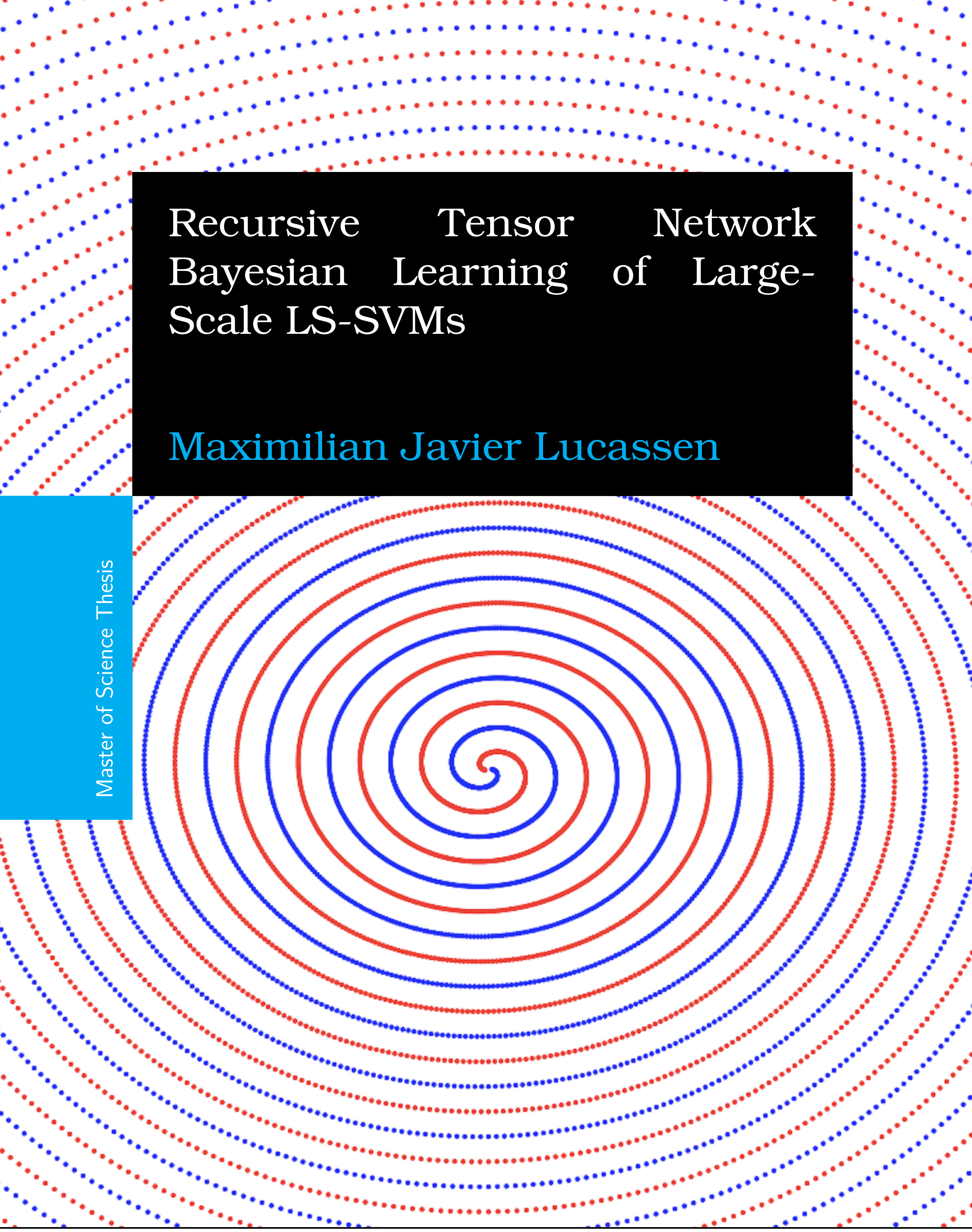

Recursive Tensor Network Bayesian Learning of Large-Scale LS-SVMs

Maximilian Javier Lucassen

Master of Science Thesis

Delft University of Technology

Delft Center for Systems and Control

Recursive Tensor Network Bayesian Learning of Large-Scale LS-SVMs

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Maximilian Javier Lucassen

August 14, 2020

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Recursive Tensor Network Bayesian Learning of Large-Scale LS-SVMs

by

Maximilian Javier Lucassen

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: August 14, 2020

Supervisor(s): Dr.ir. K. Batselier

Reader(s): Dr.ir. M. Mazo

M.E. C. Menzen

Abstract

Least-squares support-vector-machines are a frequently used supervised learning method for nonlinear regression and classification. The method can be implemented by solving either its primal problem or dual problem. In the dual problem a linear system needs to be solved, yet for large-scale problems this can be impractical as current methods suffer from the *curse of dimensionality*. This phenomena causes the computational and memory requirements to exceed the capabilities of standard computers for large datasets. In this thesis, a tensor network Bayesian learning method was developed to avoid these burdensome complexities. The developed method performs competitively with the current state-of-the-art, and unlike other low-rank approximation methods, allows for incorporation of user-knowledge, noise, early stopping, and yields confidence bounds on the obtained model.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my gratitude to my supervisor Kim Batselier for his support and guidance during my thesis. His help and kindness have really allowed me to flourish in the pursuit of my academic goals.

Special thanks go out to my family. Mama, Papa, Benne and Karlijn, thank you for your support, the laughter, and love you give. I would not be where I am today without you.

A shout-out to my friends, who encourage me to challenge myself with a smile on my face.

Delft, University of Technology
August 14, 2020

Maximilian Javier Lucassen

Chapter 1

Introduction

1-1 Context

Least-squares support-vector-machines (LS-SVM) are a commonly used algorithm for nonlinear classification and regression, and are widely applied in the machine learning [37], control [34], and signal processing [35] disciplines. LS-SVM is a kernel method set in a supervised-learning framework, based on the least-squares reformulation [35] of Vapnik's support-vector-machines [38]. One can choose to solve either the LS-SVM primal, presented in Chapter 2, or the dual problem. The dual problem

Dual problem:

$$\begin{bmatrix} 0 & \mathbf{1}^\mathsf{T} \\ \mathbf{1} & \Omega_N + I_N/\gamma \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{y} \end{bmatrix}, \tag{1-1}$$

can be advantageous over the primal problem as it merely requires solving of an unconstrained linear system instead of a quadratic programming (QP) problem. Additionally, often the primal problem is not even possible because an explicit nonlinear feature mapping function is typically unknown to the user. Nevertheless, the advantage of solving the LS-SVM dual problem diminishes for large-scale applications. The non-parametric nature of the dual problem forces a polynomial or even exponential computational and memory scaling with the number of data points N. As a result, direct and iterative methods to solve the linear system quickly become unpractical or infeasible for large data sets, approximately when $N \geq \mathcal{O}(10000)$. Low-rank approximation methods are the next best alternative, in which a degree of accuracy-loss is accepted in order to estimate a solution of the dual problem. These low-rank methods use a subset of the data, but can still suffer from prohibitive scaling and poor accuracy. Thus, a substantial bottleneck of LS-SVM is their application to large-scale problems.

This thesis is concerned with solving large-scale LS-SVM dual problems for nonlinear regression and classification tasks. When the number of data points becomes exponentially large, modeled as $N \to n^d$, the dual problem becomes infeasible. Due to the memory scaling $\mathcal{O}(n^{2d})$, explicit representation of the dual problem in matrix form becomes impossible. Additionally, common methods, such as direct matrix inversion, are too burdensome to implement due to their computational scalings, $\mathcal{O}(n^{2d})$ or $\mathcal{O}(n^{3d})$. The exponential computational and memory complexities demonstrate a phenomena called *the curse of dimensionality*, which is when complexities scale exponentially with the number of dimensions d. This phenomena currently limits the application of LS-SVMs to large-scale problems.

1-2 Problem Statement and Objective

Due to the exponential computational and memory complexities of current state-of-the-art methods to solve large-scale LS-SVM dual problems, the capabilities of standard computers are rapidly exceeded. The main reason for this is because current methods require an explicit representation and work directly on matrices, which can be impractical or infeasible for large-scale problems. The only remaining option is to approximate the dual problem solution with low-rank matrix approximation methods. However, low-rank methods are error-prone and can still suffer from exponential scaling.

The objective of this thesis is to investigate whether large-scale LS-SVM dual problems can be solved with use of tensor networks, a compressed mathematical format that can avoid the burdensome exponential memory and computational complexities by representing $\mathcal{O}(n^d)$ in $\mathcal{O}(nd)$. It is proposed to rewrite the dual problem to a tensor train (TT) format by rewriting vectors to tensor train vectors, and matrices to tensor train matrices. Additionally, within this TT formulation, an alternative method to current low-rank matrix approximation methods is built in context of recursive Bayesian learning. Bayesian learning is a probabilistic framework based on Bayes' rule, in which prior (initial) distributions are updated by witnessing data to form posterior (new) distributions. A recursive Bayesian learning procedure has a number of advantages. First, user-knowledge can be incorporated, such as a prior distribution, measurement noise, and early stopping. Secondly, the probabilistic framework yields confidence bounds on the obtained dual problem solution, informative of the accuracy. Thirdly, in the recursive scheme only individual rows of the dual problem need to be worked with, significantly decreasing the computational and memory complexities. It is desired that the method performs competitively with state-of-the-art low-rank approximation methods and that *the curse of dimensionality* is avoided. To realize this goal, a recursive TT Bayesian learning framework is developed that iteratively updates the solution distribution of the dual problem through a prediction-observation procedure.

1-3 Layout

The thesis is divided into four chapters. First, the background theory is covered in Chapter 2. Concepts relevant to this thesis are described in the background, covering the LS-SVM, low-rank matrix approximation methods, Bayesian learning and tensor networks. In Chapter 3, the approach and main results of this thesis are given, which is the latest version of a journal paper that is planned to be submitted to IEEE later in 2020. In Chapter 4, the approach, results, and proposed method of this thesis are discussed. Lastly, conclusions and future work are presented in Chapter 5.

1-4 Notation and Abbreviations

The notation and abbreviations used in this report is summarized in Table 1-1 and Table 1-2, respectively.

Table 1-1: Used Notation

Scalars	(a,b,...)
Vectors	($\mathbf{a}$,$\mathbf{b}$,...)
Matrices	(A,B,...)
Tensors	($\mathcal{A}$,$\mathcal{B}$,...)
Tensor Train of $\mathcal{A}$	$TT(\mathcal{A})$
Matrix transpose	($\mathbf{a}^\intercal$, $A^\intercal$,...)
Identity of size N by N	I_N
Kernel matrix of size N by N	Ω_N

Table 1-2: Used Abbreviations

FS-LSSVM	Fixed size least-squares support-vector-machines
KKT	Karush-Kuhn-Tucker
LS-SVM	Least-squares support-vector-machines
QP	Quadratic programming
RBF	Radial-basis-function
SVM	Support-vector-machines
SVD	Singular value decomposition
TT	Tensor train
TNKF	Tensor network Kalman filter

Chapter 2

Background

This chapter contains the necessary background for the thesis. First, in Section 2-1 a brief overview is given on why nonlinear modeling is necessary. In Section 2-2, the least-squares support-vector-machine (LS-SVM) theory is given. Thereafter, methods to solve or approximate LS-SVM dual problems are presented in Section 2-3. An introduction to Bayesian learning and this thesis' approach are covered in Section 2-4-1. Lastly, an introduction to tensor networks is given in Section 2-4-2, which is used to reformulate the Bayesian learning procedure to a compressed mathematical format.

2-1 Why Nonlinear Modeling?

In many modeling applications a linear model is desirable because these are easy-to-use and insightful. In practice, however, linearity can be too strong an assumption to sufficiently describe the underlying problem. Many phenomena abide to strong nonlinear behavior. For example, friction, inseparability of data, time-variation, and compound interest [28]. In these cases, nonlinear models can prove useful, and as a consequence nonlinear modeling is necessary and widely used in a variety of fields such as, control [2] [27], signal processing [19][9], and machine learning [26] [21]. Due to the higher-order descriptive power associated with nonlinear modeling, very accurate fits can be achieved. Yet, this comes at a cost. Nonlinear models are not insightful and are generally difficult to work with.

2-2 Least-Squares Support-Vector-Machines

An overview the LS-SVM is given in this section based on [35]. For convenience, all derivations and theory are presented for the regression case. The equations can easily be modified for classification tasks, which is also covered in the paper presented in Chapter 3.

Consider a data set $\{\mathbf{x}_i, y_i\}_{i=1}^N$, with $\mathbf{x}_i \in \mathbb{R}^f$ and $y_i \in \mathbb{R}$ for regression, shown in Figure 2-1. For simplicity and illustrative purposes the input data is assumed to be one-dimensional $f = 1$ here, but for generalization to higher-dimensions the bold vector notation for $\boldsymbol{x}$ is used.

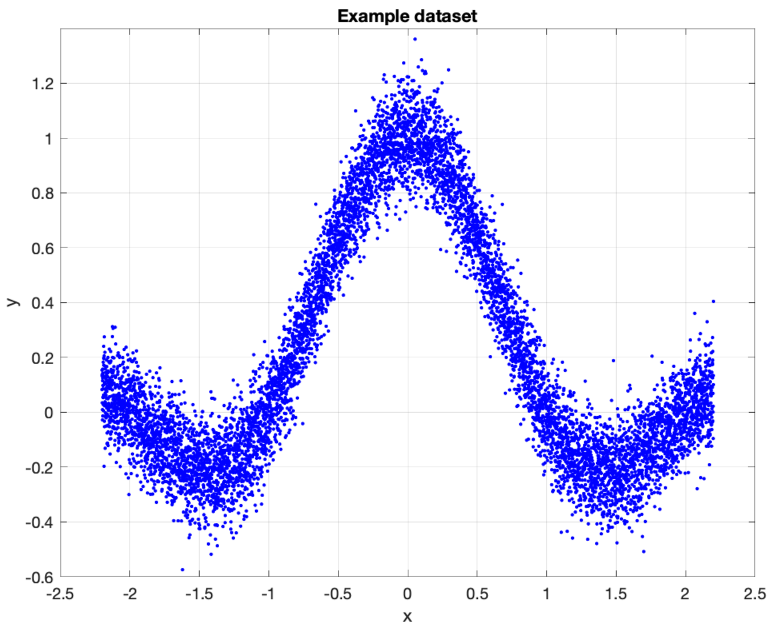

Figure 2-1: An example data set used for a regression task. A linear model would fit the data poorly, which is why a nonlinear model has to be found.

For regression, the task is to fit a function through the shown data set. From the figure it is clear that a linear model would not suffice, and that a nonlinear model is required for an accurate description.

In the LS-SVM formulation, developed by J. Suykens in the early 2000's [35], a regression model in the form of

$$y(\boldsymbol{x}) = \boldsymbol{w}^\intercal \varphi(\boldsymbol{x}) + b \tag{2-1}$$

has to be found to fit the data. In this model the input data is first mapped to a high-dimensional or possibly infinite-dimensional f_h, feature space by a nonlinear mapping function $\varphi(\mathbf{x})$: $\mathbb{R}^f \to \mathbb{R}^{f_h}$. The idea behind this mapping is that in the higher-dimensional feature space linear regression can be performed, accomplished by finding a suitable weight vector $\boldsymbol{w}$ and a bias term b for the linear offset. Visually, the regression model can be understood as illustrated in Figure 2-2. In the input space a highly nonlinear regression model would have to be found, but by mapping the data to a high-dimensional feature space the task is reduced to finding a linear model.

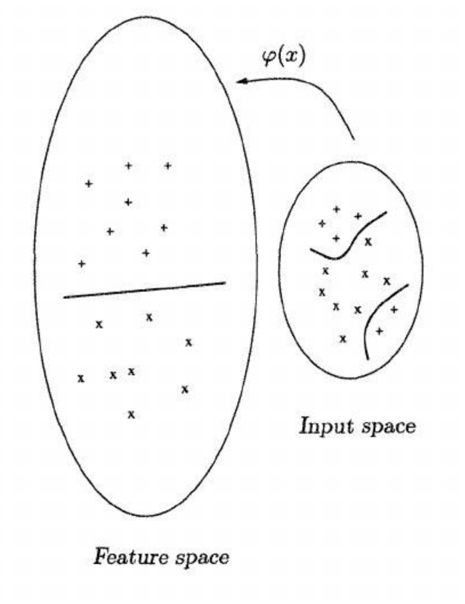

Figure 2-2: The input data is mapped to a higher-dimensional feature space in which linear regression can be performed [35].

With a known nonlinear mapping function, the model weights and bias term can be solved for in a minimization problem known as the primal problem

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{e}} \quad & J_{primal}(\boldsymbol{w},\boldsymbol{e}) = \frac{1}{2}\boldsymbol{w}^\intercal\boldsymbol{w} + \frac{\gamma}{2}\sum_{i=1}^{N} e_i^2 \\ & \text{subject to:} \\ & y_i = \boldsymbol{w}^\intercal\varphi\left(\boldsymbol{x}_i\right) + b + e_i, \quad i = 1,\dots,N. \end{aligned} \tag{2-2}$$

The primal problem is parametric and scales with the number of features of the input data, as $\boldsymbol{w}$ has to weigh each feature of $\varphi\left(\boldsymbol{x}\right)$. Because the data can not be fit perfectly, as this leads to overfitting, modeling errors e are inevitable. In the illustrated data set, it is clear that fitting each data point perfectly would not yield a good overall model, which is why errors have to be accepted. These errors mean that the regression model needs to be modified to prevent overfitting, this is described by the constraint in the primal problem. However, by introducing errors to the regression model there is also a danger of underfitting the data, which occurs if the errors are allowed to be too large. Underfitting the data would lead to a model that barely fits the shape of the plotted data set. To prevent overfitting and underfitting, the weights and errors in the model to be balanced, as given by the quadratic cost function of the primal problem J_{primal}. The γ variable is a regularization parameter that balances the trade-off between over- and underfitting by weighting the relative importance of the sum of squared errors. An important result and advantage of the quadratic cost function is that the minimization is convex, meaning that the nonlinear regression task can be solved with simple quadratic programming (QP) solvers. Also due to convexity a global optimal solution is obtained for the nonlinear modeling problem.

Nevertheless, there are significant drawbacks to the primal problem [35]. First, if an infinite-dimensional feature space is used the primal problem can not be solved, as the weight vector is then also infinite-dimensional. Secondly, the computational cost of QP can make the primal problem too burdensome to solve for high-dimensional input data as it scales with the number of features f. Thirdly, usually an explicit nonlinear mapping function is unknown which makes it impossible to solve the primal problem.

Fortunately, the primal problem can be recast by use of the Lagrangian

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{e}; \boldsymbol{\alpha}) = J_P(\boldsymbol{w}, \boldsymbol{e}) - \sum_{k=1}^{N} \alpha_k \left\{\boldsymbol{w}^\intercal \varphi\left(\boldsymbol{x}_k\right) + b + e_k - y_k\right\} \tag{2-3}$$

and application of the Karush-Kuhn-Tucker (KKT) optimality conditions yield

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial w} = 0 \rightarrow \boldsymbol{w} = \sum_{k=1}^{N} \alpha_k \varphi\left(\boldsymbol{x}_k\right) \\ \frac{\partial \mathcal{L}}{\partial b} = 0 \rightarrow \sum_{k=1}^{N} \alpha_k = 0 \\ \frac{\partial \mathcal{L}}{\partial e_k} = 0 \rightarrow \alpha_k = \gamma e_k, \quad k = 1, \ldots, N \\ \frac{\partial \mathcal{L}}{\partial \alpha_k} = 0 \rightarrow \boldsymbol{w}^\intercal \varphi\left(\boldsymbol{x}_k\right) + b + e_k - y_k = 0, \quad k = 1, \ldots, N. \end{cases} \tag{2-4}$$

By rewriting the equations through elimination of $\boldsymbol{w}$ and $\boldsymbol{e}$ the dual problem is obtained

Dual problem:

$$\begin{bmatrix} 0 & \mathbf{1}^\intercal \\ \mathbf{1} & \Omega_N + I_N/\gamma \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{y} \end{bmatrix} \tag{2-5}$$

which offers an alternative to the LS-SVM primal problem [35]. A linear system needs to be solved instead of a QP problem to obtain a regression model of the data set, which can be much cheaper. The dual weights $\boldsymbol{\alpha}$, which are directly related to each data point's error variable by the KKT condition $\alpha_k = \gamma e_k$, are solved for. As a consequence of this KKT condition, the dual problem scales non-parametrically, meaning that the memory and computational complexities scale directly with the number of data points N. In the derivation of the dual problem, which is not presented entirely here, inner-products of the nonlinear mapping function $\varphi\left(\boldsymbol{x}_k\right)^\intercal \varphi\left(\boldsymbol{x}_k\right)$ are replaced by a user-defined kernel function $k\left(\mathbf{x}, \mathbf{x}_i\right)$. The kernel function implicitly computes the inner-products in the feature space without needing to explicitly map the input data to this space. Replacing the inner-products of the nonlinear mapping function by a kernel function is known as the "kernel-trick". Due to the kernel function, no explicit feature mapping function is needed for the dual problem, and even when the primal is impossible the dual problem can still be solved [35]. In the presented dual problem, Equation 2-5, the kernel matrix is defined as $\Omega_N := k(X, X)$, where X represents the input data matrix containing all $\boldsymbol{x}_i$. By definition the kernel matrix is symmetric positive (semi)-definite, which induces convexity and therefore the dual problem yields a global optimal solution. Common kernel functions include the linear kernel, polynomial kernel, and the radial-basis-function (RBF) kernel. Due to reformulation from the primal to the dual problem, the regression model is also recast. The resulting LS-SVM regression model for the dual problem becomes

$$y(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i k\left(\mathbf{x}, \mathbf{x}_i\right) + b \tag{2-6}$$

$$\tag{2-7}$$

where $\boldsymbol{\alpha}$ and b can be found be solving the linear system, Equation 2-5.

Whether the primal or dual problem is solved, a regression model for the data is obtained. If suitable regularization and kernel parameters (hyperparameters) are specified the model should fit the data well, without significant over- or under-fitting. Equivalently, a good balance between the local and global descriptions of the data is found to fit a general regression model over the domain. For the previously shown data set, a regression model is illustrated in Figure 2-3.

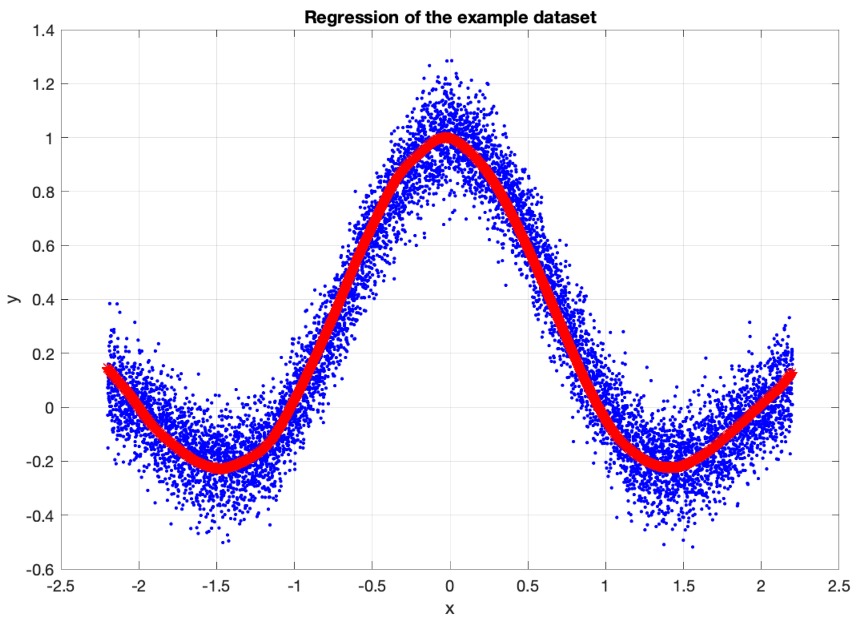

Figure 2-3: An example regression model (red) for the example . The model generalizes the data set well as there is a good balance between local and global fit. (RBF, $\gamma = 1$, $\sigma^2 = 0.5$)

The shown regression model for the data set was designed with the RBF kernel, which is used frequently in this thesis and given by

$$k\left(\boldsymbol{x}, \boldsymbol{x}'\right) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\sigma^2}\right). \tag{2-8}$$

The hyperparameter σ^2 describes how similar the data paints are over the domain, and can be interpreted as a similarity measure that needs to be tuned. Tuning of kernel function hyperparemeters is commonly done with iterative grid searches or higher-level Bayesian learning procedures [35] [10].

In the regression model plot, Figure 2-3, it can be deducted that there is a fair balance between the errors and weights as there is good fit, with no significant signs of overfitting or underfitting of the data. Overfitting and underfitting can be caused by multiple factors, such as the chosen hyperparameters, large outliers in the data, or solver properties such as early stopping. Additionally, the LS-SVM can be sensitive to extreme outliers as the dual weights are proportional to their associated error magnitudes through the KKT condition. Because of the dual error-weight relationship, the LS-SVM dual solution is non-sparse, meaning that every data point error e_k contributes to the regression model.

The popularity of the LS-SVM is due to its linear dual problem. Nonlinear modeling can be performed by merely solving a linear system that yields a global optimal solution. The dual problem is simple to solve for small- and medium-scale problems, for which many methods and solvers available, such as matrix inversion techniques or iterative methods such as conjugate gradient [13]. For large-scale applications, however, the LS-SVM dual problem becomes burdensome or infeasible to solve. The topic of this thesis is how the LS-SVM dual problem can be implemented on large-scale problems. In this thesis, by "large-scale" it is meant that the number of data points N is exponentially large, $N \to n^d \geq \mathcal{O}(10000)$. For such size data sets, solving of the LS-SVM dual problem can exceed capabilities of modern computers as it grows directly with the number of data points. First of all, the memory necessary for the dual problem scales $\mathcal{O}(n^{2d})$. For large data sets it is very difficult or even impossible to explicitly construct or work with the dual problem matrix. Secondly, common solvers are very expensive to implement for such large-scale problems because of their computational scaling $\mathcal{O}(n^{3d})$. Common solver methods require explicit storage and work directly on the dual matrix making them infeasible. For large-scale applications, it is usually only possible to an estimate a solution of the dual problem by employing low-rank approximation methods.

Table 2-1 summarizes the advantages and disadvantages of the LS-SVM dual problem. More on solvers for the LS-SVM dual problem is presented in Section 2-3.

Table 2-1: Advantages and Disadvantages of the LS-SVM Dual Problem [35] [32][40].

Advantages	Disadvantages
• Solve a linear system for nonlinear modeling • Easy to use and many solvers available • Flexibility with kernel choice • Global optimal solution(s)	• Large-scale problems are difficult • Solution is not sparse • Can be sensitive to outliers in the data

2-3 Methods to Solve the LS-SVM Dual Problem

In this section, an overview of methods for solving the LS-SVM dual problem is given. They can be divided three categories, direct methods, iterative methods, and low-rank approximation methods. Most attention is given to low-rank approximation methods, as these are most relevant for large-scale problems. The other methods are briefly covered for context. Direct methods show the expense of computing an inverse of a matrix, which has to be avoided in the thesis' implementation as it is expensive or impossible to compute for large matrices. The iterative methods, have similarities with the thesis' proposed solution, as both are recursive in nature. The presented low-rank approximation methods are the state-of-the-art in solving, or approximating, the solution of large-scale LS-SVM dual problems. The two covered low-rank approximation methods are compared to the thesis' proposed solution.

2-3-1 Preliminary

Before going straight into solvers, which are methods to solve the linear system, some additional background has to be given. Recall the dual problem

$$\begin{bmatrix} 0 & \mathbf{1}^\top \\ \mathbf{1} & \Omega_N + I_N/\gamma \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{y} \end{bmatrix}. \tag{2-9}$$

The applicability of a solver is dependent on the matrix properties. Solvers attempt to utilize matrix properties for computational and/or memory efficiencies. The most important properties of the dual problem matrix are that it is positive (semi)-definite and symmetric. The dual problem can also be recast to a positive definite form to allow implementation of solvers that require this property, as given by, as given by

$$\begin{aligned} &\Psi = (\Omega_N + I_N/\gamma) > 0, \\ &\begin{bmatrix} \mathbf{1}^\top \Psi \mathbf{1} & 0 \\ 0 & \Psi \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} + b\Psi^{-1}\mathbf{1} \end{bmatrix} = \begin{bmatrix} \mathbf{1}^\top \Psi^{-1} \mathbf{y} \\ \mathbf{y} \end{bmatrix}, \end{aligned} \tag{2-10}$$

which can lead to convergence and computational benefits [33] [35]. For large-scale problems however, $N \to n^d$, computing and storing the inverse of the kernel matrix is usually infeasible. For the upcoming methods, it is assumed that the dual matrix is dense, positive (semi)-definite, and symmetric. Only the methods that are most relevant to this thesis are discussed.

2-3-2 Direct Methods

Direct methods are the usual solvers for small-scale problems. For modern computers, these methods are applicable for data set sizes in the low thousands $\mathcal{O}(1000)$ because of their burdensome computational and memory complexities, $\mathcal{O}(n^{3d})$ and $\mathcal{O}(n^{2d})$ [14]. They need explicit storage and work directly on the matrix, thus quickly become impractical to apply. As a result, these can not be used to solve large-scale LS-SVM dual problems. There are three popular methods in this class: direct matrix inversion, Gaussian elimination, and the LU decomposition. In Table 2-2, the approximate costs of direct methods are presented, which scale rapidly with the dimension d.

Table 2-2: Approximate computational and memory complexities direct methods [13] [1] [14].

• Computational scaling	$\mathcal{O}(n^{3d})$ (for solving one linear system)
• Memory requirements	Requires the entire (explicit) dual matrix $\mathcal{O}(n^{2d})$
• Dual solution $\boldsymbol{\alpha}$:	Scales non-parametrically $\mathcal{O}(n^d)$

2-3-3 Iterative Methods

The computational complexity of direct methods can make them impractical to use for medium-scale problems. The next best alternative is to implement iterative methods. Based on an initial guess, a sequence of solutions is generated by updating previous estimates. Because in a single iteration only a row of the considered matrix is needed, iterative methods can have a lower memory cost than direct methods $\mathcal{O}(n^d)$, and can therefore be applied to larger problems. However, the total computational scaling $\mathcal{O}(n^{2d}L)$ to obtain a solution is only significantly advantageous if few iterations L, ($L < n^d$), are needed until convergence (termination) [41] [13]. For large-scale applications, iterative methods also become too burdensome to implement, especially due to their computational complexities.

Two classes of iterative methods exist: stationary and non-stationary iterative methods. In stationary iterative methods only the solution vector is updated throughout the iterations, all other variables are constant ('stationary'). Common methods for this class include the Jacobi method, Gauss-Seidel method, and the Successive Overrelaxation method. These are the simpler but less efficient than their non-stationary counterparts, as their convergence can be up to an order of magnitude slower. In non-stationary methods, more than just the solution vector is changed per iteration. With newly obtained information, variables are adjusted in the equations to update the solution. The most important non-stationary methods belong to a class called Krylov methods, such as the conjugate gradient or minimum residual methods [13] [3]. Krylov methods are known to possess faster convergence rates and are computationally more efficient than stationary methods [36]. The general complexities of iterative methods are presented in Table 2-3.

Table 2-3: Approximate Computational and Memory Complexities of Iterative Methods [41] [13].

• Computational scaling	$\mathcal{O}(n^{2d}L)$
• Memory requirements	Requires the rows of the dual matrix $\mathcal{O}(n^d)$
• Dual solution $\boldsymbol{\alpha}$:	Scales non-parametrically $\mathcal{O}(n^d)$

2-3-4 Low-Rank Approximation Methods

Low-rank approximation methods are necessary when direct or iterative methods are not feasible, as the computational or memory costs become too burdensome, and can even exceed the capabilities of standard computers. Low-rank approximation methods of kernel matrices are especially applicable to this thesis. Typically some estimate of the eigensystem is used to solve or decompose the matrix or estimate a nonlinear mapping function. The underlying idea of low-rank approximation methods is that some accuracy can be sacrificed in order to make a problem feasible, thus inherently there is always a trade-off between complexity and accuracy. Here, two methods are covered, the Nyström method and the fixed size LS-SVM

(FS-LSSVM). These methods are the most applicable to the LS-SVM dual problem and are used to benchmark the performance of this thesis' proposed solution in Chapter 3.

The Nyström method and FS-LSSVM operate on a centered large-scale kernel matrix. The bias can be calculated separately by computing the mean of the output data. The dual problem is therefore reduced to finding an approximate solution to

$$(\Omega_{n^d} + I_{n^d}/\gamma)\,\boldsymbol{\alpha} = \boldsymbol{y}, \tag{2-11}$$

as the bias term is removed from the Lagrangian, Equation 2-3.

The Nyström Method

The Nyström method is probably the most applied low-rank approximation technique for symmetric positive (semi)-definite matrices. The method is a spectral reconstruction technique and is applicable in many nonlinear modeling methods, such as Gaussian processes, LS-SVMs, and principal component analysis for example [35]. With the Nyström method it possible to estimate a solution of the dual problem with low computational and memory costs [35][17][30].

The Nyström method is based on the eigendecomposition of a kernel subset. First, by sampling S input samples $\boldsymbol{x}_i$ a smaller kernel matrix $W_{S\times S}$ and a rectangular kernel matrix $C_{n^d\times S}$, which compares the s samples to the n^d data points, are constructed. Sampling of the data points can be done with random uniform sampling, or according to some user-specified sampling procedure if computationally feasible, such as ridge leverage scores [23], k-means clustering [42], determinant based sampling [18], norm and adaptive sampling procedures [30]. By calculating the eigendecomposition of W, the approximate eigenvectors $\hat{U}_\Omega$ and eigenvalues $\hat{\Lambda}_\Omega$ of the large kernel matrix can be estimated by

$$\hat{U}_\Omega \approx \sqrt{\frac{S}{n^d}}CU_W\Lambda_W^{-1}, \qquad \hat{\Lambda}_\Omega \approx \frac{n^d}{S}\Lambda_W. \tag{2-12}$$

The underlying assumption for the approximation is that the number of samples S is large enough such that the scaled eigensystem of W can accurately represent the eigensystem of Ω_{n^d}. The size of the subset directly impacts the trade-off between the accuracy and the complexity of the Nyström method. Note that in the approximation, the inverse of the eigenvalue matrix is simple to compute due to its diagonal structure. With the obtained estimates of the eigenvectors and eigenvalues of the kernel matrix the dual problem can then be solved, in the approximate sense. An approximation for the dual weights can be calculated by applying the Sherman-Morrison-Woodbury formula

$$\hat{\boldsymbol{\alpha}} = \gamma\left(\mathbf{y} - \hat{U}_\Omega(\frac{I_S}{\gamma} + \hat{\Lambda}_\Omega\hat{U}_\Omega^\intercal\hat{U}_\Omega)^{-1}\hat{\Lambda}_\Omega\hat{U}_\Omega^\intercal\mathbf{y}\right), \tag{2-13}$$

which generally requires less memory and computational cost than applying a direct or iterative method. Nevertheless, the Nyström method can become infeasible if many samples are needed to approximate a solution. For large S, the inverse in the Sherman-Morrison formula becomes the main computational bottleneck. Also, the memory required to explicitly represent the eigenvector matrix U_Ω can exceed the computer's capabilities. The memory and computational complexities of the Nyström method with uniform random sampling are presented in Table 2-4.

Table 2-4: Computational and Memory Complexities of the Nyström Method [14].

• Computational scaling	$\mathcal{O}(S^2 n^d + S^3)$
• Memory requirements	Sample matrix C, $\mathcal{O}(Sn^d)$
• Dual solution $\hat{\boldsymbol{\alpha}}$:	Scales non-parametrically $\mathcal{O}(n^d)$

Fixed Size LS-SVM

FS-LSSVM is a low-rank approximation technique that estimates a nonlinear mapping function φ in the dual space to then solve the primal problem [35][10]. Usually a nonlinear mapping function is unknown for the primal problem, which is why it has to be estimated. FS-LSSVM can be beneficial for large-scale applications because solving the primal problem scales with the number of features f rather than the number of data points n^d. If the number of features of the input data is small, the primal problem can be advantageous. FS-LSSVM is based on the Nyström method and is therefore applicable for symmetric positive (semi)-definite kernel matrices. First a subset of input data points has to be constructed. Usually the Renyi entropy criterion

$$\begin{aligned} H_{Renyi} &= -\log \int p(\mathbf{x})^2 dx \\ \int \hat{p}(\mathbf{x})^2 dx &= \left(\frac{1}{n^{2d}}\right) \mathbf{1}_v^T W \mathbf{1}_v \end{aligned} \tag{2-14}$$

is implemented as a sampling criterion, as is done in this thesis. The criteria attempts to include the S most informative data points in the subset by iteratively minimizing the Renyi entropy H_{Renyi}. The entropy is calculated with the subset's kernel matrix W. In each iteration, a random data point from the subset and training pool are swapped for which the Renyi entropy is evaluated. If the entropy increases the swap is undone, and if the entropy decreases the swap is permanent and the subset updated. Over the iterations, the subset becomes more informative by the swapping procedure, terminating when a stopping condition is satisfied. When a final subset has been found, an eigendecomposition of the small kernel matrix $W_{S,S}$ is computed to obtain the eigenvector matrices, just as in the Nyström method [35]. The nonlinear mapping function can then be approximated with the obtained eigenvectors $\boldsymbol{u}$, as given by

$$\varphi_i(\boldsymbol{x}') = \sqrt{\tilde{\lambda}_i}\hat{\phi}_i(\boldsymbol{x}') = \frac{\sqrt{S}}{\sqrt{\lambda_i}} \sum_{k=1}^{S} \boldsymbol{u}_{ki} K(\boldsymbol{x}_k, \boldsymbol{x}') \tag{2-15}$$

With an estimate of the nonlinear mapping function, the primal can be solved with a QP solver. The implementataion of the FS-LSSVM method with a Renyi entropy sampling procedure is pictured in Figure 2-4, in which intermediate subsets are plotted to demonstrate the increase in informativeness.

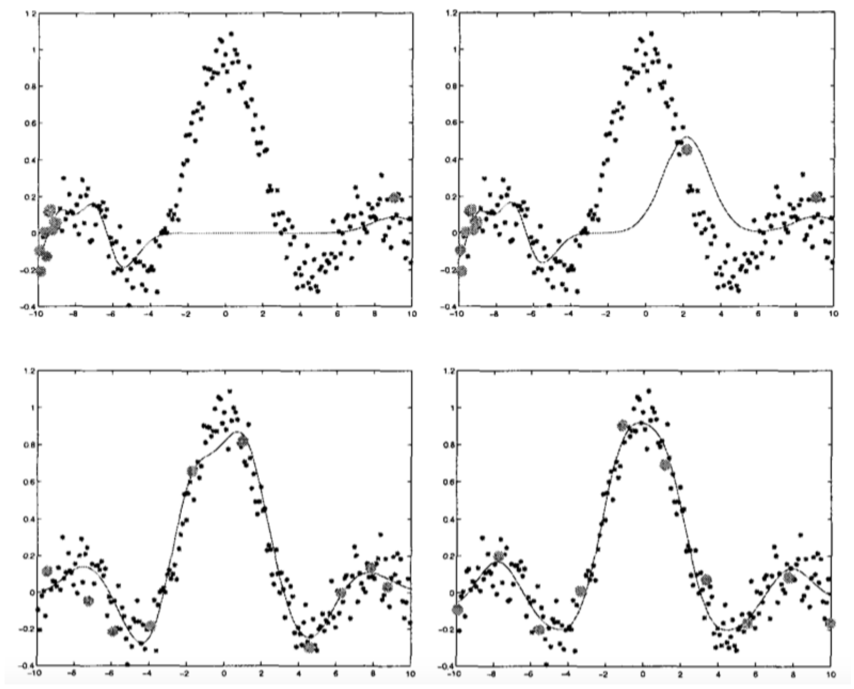

Figure 2-4: The selection of data points with the Renyi entropy sampling criteria. The fat-dots represent the data points in the subset, also called "support-vectors", which are used to find a regression model. These data points are swapped to increase the informativeness of the subset, leading to better model fits of the data [35].

The FS-LSSVM method has two advantages due to its shared primal-dual formulation. A much smaller solution is obtained that scales with the number of samples S, and the memory requirement only scales quadratically due to explicit construction of W. Thus if the number of samples can be chosen small with respect to the obtained model's accuracy the FS-LSSVM is advantageous over the Nyström in terms of memory scaling, however does generally require more computational work. In Table 2-5, the computational and memory requirements of FS-LSSVM are summarized.

Table 2-5: Computational and Memory Complexities of the FS-LSSVM Method [20] [10] [14].

• Computational scaling	$\mathcal{O}(S^2 n^d + 2S^3)$
• Memory requirements	Subset kernel W, $\mathcal{O}(S^2)$
• Primal solution $\hat{\boldsymbol{w}}$:	Scales parametrically $\mathcal{O}(S)$

Drawbacks of State-of-the-Art Low-Rank Approximation Methods

Even though the Nyström and FS-LSSVM methods allow large-scale application of LS-SVMs, there are still a number of issues due to which the methods can perform poorly. First, the methods suffer from the *curse of dimensionality*, as can be understood from their respective computational and memory complexities. Even though a subset of the data is used, the methods can still become infeasible if many samples are required. For current low-rank approximation methods the assumption is that $S \ll n^d$ which does not always hold. Secondly, as a consequence of their sampling nature, only statistical sampling-based performance bounds can be established. The accuracy of the methods are entirely dependent on which samples are included in the approximation. Lastly, in the Nyström and FS-LSSVM methods, it is impossible to incorporate user-knowledge of the particular application, such as measurement noise, an initial dual weight solution $\boldsymbol{\alpha}$, and confidence bounds.

2-4 Proposed Solution

To overcome the drawbacks of current low-rank approximation methods it is proposed to develop a Bayesian framework to solve the LS-SVM dual problem. A Bayesian framework has a number of advantages. First, prior knowledge of problem can be incorporated into the learning process, such as measurement noise and an initial distribution. Secondly, the framework generates a solution distribution from which confidence bounds can be obtained, informative of how well the data is described by the mean. Lastly, Bayesian learning is a flexible and easy-to-use framework, which can be easily adjusted to a certain problem format, and modified to suit computational and memory requirements. Unfortunately, the Bayesian framework still suffers from the *curse of dimensionality* as explicit matrices and vectors need to be stored and worked with. To avoid this curse, it is proposed recast the Bayesian framework to a tensor network formulation. With tensor networks no explicit vectors or matrices need to be constructed, and due to their compressed mathematical format reduce exponential complexities to polynomial ones $\mathcal{O}(n^d) \rightarrow \mathcal{O}(dn)$.

In this section the background theory for the proposed solution is covered. First an overview of Bayesian learning is given in Section 2-4-1, presenting the context in which the dual problem is proposed to be solved. Afterwards, a brief introduction to the relevant tensor network theory is presented in Section 2-4-2, used to reformulate and compress the Bayesian learning framework. Additional explanations can be found in the preliminaries and background sections of the paper, Chapter 3.

2-4-1 Bayesian Learning

A Bayesian approach is taken to approximate the large-scale LS-SVM dual problem solution. Bayesian learning is a probabilistic procedure in which observations of the data are used to update a solution distribution. A brief introduction is given to Bayesian learning based on [29], and is presented in the context of Bayesian regression for the LS-SVM dual problem. In this explanation the dual problem is rewritten for convenience

$$C\bar{\boldsymbol{\alpha}} = \bar{\boldsymbol{y}}, \tag{2-16}$$

where C is the dual problem matrix, $\bar{\boldsymbol{\alpha}} = [b; \boldsymbol{\alpha}]$, and $\bar{\boldsymbol{y}} = [0; \boldsymbol{y}]$. The formulation of the Bayesian regression case can easily be modified for Bayesian classification, but this is left to the literature [22].

Bayesian Regression

Consider the case that a linear sample-varying (l) regression model has to be found for the data set

$$\bar{y}_l = \boldsymbol{c}_l \bar{\boldsymbol{\alpha}} + u. \tag{2-17}$$

For the LS-SVM dual problem, "sample-varying" is equivalent to "row-dependent", for this reason the subscript l is used. In the model, y_l is an output data point, $\boldsymbol{c}_l$ is the l-th row of the dual problem, $\bar{\boldsymbol{\alpha}}$ is the dual solution, and $u \sim \mathcal{N}(0, w^2)$ is an I.I.D Gaussian measurement noise. The measurement noise is a robustness variable to prevent overfitting, and weighs how valuable observations of the data are.

In Bayesian learning, it is desired to update the $\bar{\boldsymbol{\alpha}}$ distribution by observing a set of T data points. The update can be understood as generating a new (posterior) distribution that takes into account all previously witnessed data, but also all newly observed data. The learning process is summarized by Bayes' rule

$$\mathcal{P}\left(\bar{\boldsymbol{\alpha}} \mid \bar{y}_{1:T}\right) \propto \mathcal{P}\left(\bar{\boldsymbol{\alpha}} \mid \boldsymbol{m}_0, P_0\right) \prod_{k=1}^{T} \mathcal{P}\left(\bar{y}_k \mid \mathbf{c}_l \bar{\boldsymbol{\alpha}}, u\right), \tag{2-18}$$

given linguistically by

$$\text{posterior distribution } \bar{\boldsymbol{\alpha}} \propto \text{ prior distribution } \bar{\boldsymbol{\alpha}} \times \text{normalized likelihood } \bar{y}_{1:T}. \tag{2-19}$$

The posterior distribution of $\bar{\boldsymbol{\alpha}}$ is calculated by multiplying the old (prior) distribution with the normalized likelihood of the T observed outputs. Bayes' rule can be considered a hypothesis test, an evaluation of how well the observed data is described by the prior distribution. But to apply Bayes' rule, a prior distribution $\mathcal{P}$ needs to be specified for $\bar{\boldsymbol{\alpha}}$. In this thesis', the prior distribution is also specified as a Gaussian, in which it is assumed that the dual weights adhere to such a distribution. Specifying both the prior and measurement noise as Gaussian distributions is advantageous because Bayes' rule simplifies to a product of Gaussians. The posterior and likelihood then become Gaussian distributions too because products of Gaussians yields a Gaussian. Additionally, due to the Gaussian formulation, an analytical solution for the distribution update is available. The posterior distribution mean and covariance after observing T output data points can be calculated in batch (matrix) form by

$$\begin{aligned} \boldsymbol{m}_T &= \left[P_0^{-1} + \frac{1}{w^2} C^\intercal C\right]^{-1} \left[\frac{1}{w^2} C^\intercal \bar{\boldsymbol{y}} + P_0^{-1} \boldsymbol{m}_0\right], \\ P_T &= \left[P_0^{-1} + \frac{1}{w^2} C^\intercal C\right]^{-1}. \end{aligned} \tag{2-20}$$

Even though a posterior distribution for the solution vector is obtained by these batch equations, $\bar{\boldsymbol{\alpha}} \sim \mathcal{N}(\boldsymbol{m}_T, P_T)$, it is required to calculate multiple matrix inverses. Inversions of large matrices are impractical and often impossible because of the burdensome computational complexity $\mathcal{O}(n^{3d})$. Large-scale Bayesian learning with large batch updates (T is too big) is therefore undesirable. Fortunately, Bayesian learning can also be cast into a recursive framework, in which smaller batches or single observations can be used to update the prior distribution. In this thesis, single output observations y_k are used to update the dual weight distribution because this allows for the smallest computational and memory complexities. The linear regression model is modified to

$$\bar{y}_l = \boldsymbol{c}_l \bar{\boldsymbol{\alpha}}_l + u \tag{2-21}$$

to denote its iterative nature and dependence on the rows. By reconsidering Bayes' rule, Equation 2-18, it can be understood that every posterior distribution can serve as prior distribution when a new observation needs to be learned. Bayes' rule is rewritten as

$$\mathcal{P}\left(\bar{\boldsymbol{\alpha}}_l \mid \bar{y}_{1:l}\right) \propto \mathcal{P}\left(\bar{\boldsymbol{\alpha}}_{l-1} \mid \bar{y}_{1:l-1}\right) \mathcal{P}\left(\bar{y}_k \mid \mathbf{c}_l \bar{\boldsymbol{\alpha}}_{l-1}, u\right) \tag{2-22}$$

to represent this recursive framework. An update model of the dual weights and output predictions can be designed in this recursive Bayesian framework as well, such as a state space system. An update model allows incorporation of user-knowledge, such as update dynamics and noise ratios. In this work it is considered that the previous distribution for $\bar{\boldsymbol{\alpha}}_l$ serves as the best approximation of the next solution $\bar{\boldsymbol{\alpha}}_{l+1}$. A update noise term $\boldsymbol{q}$ is introduced, and can used to model a "forgetting-factor" in the updates as done in [15, pg.240]. The output data equation is simply the dual regression model, Equation 2-21, with the measurement noise term used to denote imperfections in the observations. The resulting linear update equations are given by

$$\begin{aligned} &\text{Linear update model:} \\ &\qquad \bar{\boldsymbol{\alpha}}_{l+1} = \bar{\boldsymbol{\alpha}}_l + \boldsymbol{q} \\ &\qquad \bar{y}_l = \mathbf{c}_l \bar{\boldsymbol{\alpha}}_l + u. \end{aligned} \tag{2-23}$$

Leading and directly derived from the recursive Bayesian framework and a user-specified linear update model, the Kalman filter equations are obtained

$$\begin{aligned} &\text{Prediction step:} \\ &\qquad \boldsymbol{m}_l^- = \boldsymbol{m}_{l-1} \\ &\qquad P_l^- = P_{l-1} + Q \end{aligned} \tag{2-24}$$

$$\begin{aligned} &\text{Measurement step:} \\ &\qquad v_l = \bar{y}_l - \boldsymbol{c}_l \boldsymbol{m}_l^- \\ &\qquad s_l = \mathbf{c}_l P_l^- \boldsymbol{c}_l^\intercal + w^2 \\ &\qquad \boldsymbol{k}_l = P_l^- \mathbf{c}_l^\intercal s_l^{-1} \\ &\qquad \boldsymbol{m}_l = \boldsymbol{m}_l^- + \boldsymbol{k}_l v_l \\ &\qquad P_l = P_l^- - \boldsymbol{k}_l s_l \boldsymbol{k}_l^\intercal. \end{aligned} \tag{2-25}$$

The Kalman filter is the optimal closed-form filter to approximate the solution (or state) of a linear Gaussian system [39]. By recursively predicting and measuring the outputs $\bar{y}_l$, the distribution of the dual weights is updated. The filter only requires the inverse of a scalar s_l^{-1}, $\mathcal{O}(1)$, avoiding the expensive matrix inverses from the batch equations, and is therefore computationally much cheaper for large-scale problems. The variables of the Kalman filter are as follows: $\boldsymbol{m}_l \in \mathcal{R}^{n^d}$ and the $P_l \in \mathcal{R}^{n^d \times n^d}$ are the mean vector and covariance matrix of the $\bar{\boldsymbol{\alpha}}$ distribution after iteration l. In iteration l, the solution is given by $\boldsymbol{m}_l$, and the uncertainty in the solution by the covariance P_l. The noise covariance matrix Q, is a weighting term to describe the confidence in the update of covariance matrix. If Q is used as a "forgetting" term an equivalent formulation would be: $P_l^- = \lambda P_{l-1}$, which can induce an (artificial) exponential convergence with $\lambda < 1$ [15, pg.240]. The prediction error is given by $v_l \in \mathcal{R}$. Variable $s_l \in \mathcal{R}$ is the prediction-error variance, which is also dependent on the measurement noise, and determines the confidence in the l-th observation. Lastly, $\boldsymbol{k}_l \in \mathcal{R}^{n^d}$ is the Kalman gain, which influences the degree to which the distribution of $\bar{\boldsymbol{\alpha}}$ is updated. The Kalman filter terminates in iteration L when either all rows of the dual problem have been iterated through, or if user-specified stopping criteria have been satisfied. These stopping criteria can be designed arbitrarily, common specifications are based on the covariance norm or the KKT conditions. The final distribution $\bar{\boldsymbol{\alpha}}_L \sim \mathcal{N}(\boldsymbol{m}_L, P_L)$ contains the solution and confidence for the training stage, and can be used as a basis for regression of new points.

For large-scale applications, however, the Kalman filter is too costly to apply. The filter requires explicit storage and works directly on matrices, therefore rapidly becomes infeasible for large data sets. The memory and computational complexities of the presented Kalman filter equations are $\mathcal{O}(n^{2d})$ and $\mathcal{O}(n^{2d})$, which suffer from the *curse of dimensionality.* To reduce the complexities, a different mathematical format is needed, in which the matrices and vectors can be represented with smaller complexities. Fortunately, tensor networks offers a mathematical framework to achieve these lower complexities, presented in Section 2-4-2.

2-4-2 Tensor Network Reformulation of the Kalman Filter

In order to recast the Kalman filter to a tensor network form for large-scale problems, some basic theory has to be covered first. Thereafter, a high-level introduction to the proposed solution is given, with more elaborate explanations and derivations covered in the paper in Chapter 3.

Tensors Basics and the Diagrammatic Tensor Notation

Tensors are higher-dimensional generalizations of matrices, and are described by the number of dimensions they have ("order"). A d-dimensional tensor has d indices, e.g. $\mathcal{T}(i_1,i_2,...,i_d)$. Working with tensors in mathematical notation is complicated and uninsightful. A diagrammatic framework was developed to make working with tensors easier, as visualized in Figure 2-5. In the diagrammatic framework, tensors are represented by circles ("cores") with d outgoing lines representing the respective indices ("edges") [7]. Often, the specific index is written next to the edge to clarify which index is specified.

With the visual tensor notation covered, the concept of tensor networks can be explained. A tensor network is a higher-order equivalent of a matrix decomposition, represented by a number of cores interconnected by their edges. Interconnected edges denote tensor contractions, loosely speaking higher-order multiplications. An easy and familiar example is the singular

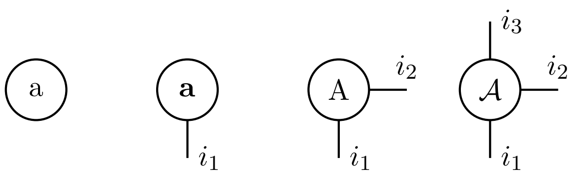

Figure 2-5: Diagrammatic tensor notation of a scalar (a), a vector (**a**), a matrix (A), and a 3-dimensional tensor ($\mathcal{A}$). Each core represents a tensor, with the edges denoting its indices.

value decomposition (SVD), which can be represented in tensor network form as shown in Figure 2-6.

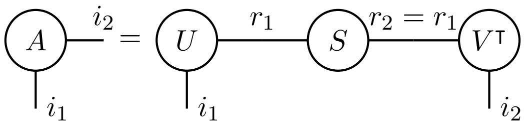

Figure 2-6: Tensor network representation of the singular value decomposition. Each core represents a matrix as they have two edges. The cores are interconnected through the indices of the singular value matrix (S), representing the matrix multiplications. The free edges are the indices of the matrix A, with i_1 denoting the rows and i_2 the columns.

In the SVD tensor network the interconnected edges are simply the dimensions of the singular value matrix, which can be truncated (reduced) to influence the reconstruction/decomposition accuracy. The number of free edges of a tensor network, such as (i_1, i_2) for the SVD, is the order of the resulting tensor if the network is reduced to one core, accomplished by contracting all interconnected edges (r_1, r_2).

The memory cost of explicitly representing a tensor scales exponentially with its dimensions $\mathcal{O}(n^d)$, thus also suffers from the *curse of dimensionality.* For higher-order tensors, tensor networks can resolve the burdensome scaling. A tensor network attempts to reduce the memory complexities, and indirectly computational complexities, by representing a tensor in a compressed and implicit format through a higher-order decomposition. One such decomposition is the easy-to-use and robust tensor train format, which will be used to recast the Kalman filter.

Tensor Train Decomposition

The tensor train (TT) decomposition is implemented to recast the Kalman filter to a compressed mathematical format. The choice of the TT format to recast the Kalman filter is because it is easy-to-use, flexible, and robust [25][24]. Additionally, alternative tensor networks typically suffer from NP-hardness, require expensive optimization procedures, or still suffer from exponential scaling [16][8]. The TT decomposition creates a chain of cores interconnected by shared indices called the "TT-ranks". Two types of TT are used in this thesis, the TT vector and the TT matrix, displayed in Figures 2-7 and 2-8 respectively.

Tensor trains can be understood as coarse graining procedures, in other words, a TT vector can be understood as vectors inside vectors, and a TT matrix, as matrices inside matrices. The resulting representation of a tensor in TT form is dependent on the TT-ranks. Similarly

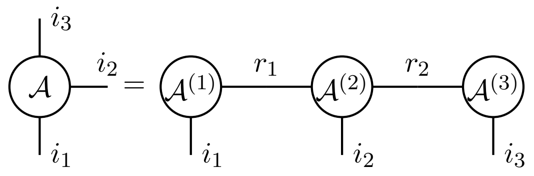

Figure 2-7: Diagrammatic representation of a tensor train vector decomposition of a 3-dimensional tensor. Each core $\mathcal{A}^{(k)}$ can be understood as a vector as they have one free edge. The cores are interconnected through the TT-ranks r_i.

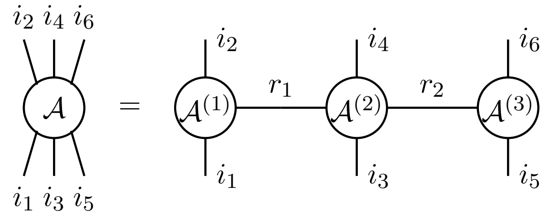

Figure 2-8: Diagrammatic representation of a tensor train matrix decomposition of a 6-dimensional tensor. Each core $\mathcal{A}^{(k)}$ can be understood as a matrix as they have two free edges. The cores are interconnected through the TT-ranks r_i.

to the SVD, the TT-ranks govern the accuracy-complexity trade-off, and can be truncated in order to achieve a more compressed representation. For TT vectors the memory requirement is $\mathcal{O}(ndr^2)$, and for TT matrices $\mathcal{O}(nd^2r^2)$, where r can be understood as the largest TT-rank. If small TT-ranks can be used without significant loss of accuracy, as the ranks determine how accurately the tensor is decomposed/reconstructed, the TT format can allow for much smaller memory complexities relative to the exponential memory complexity $\mathcal{O}(n^d)$ of explicitly representing a tensor.

Without going into technicalities, the TT decomposition is constructed by iteratively applying the SVD for each dimension of a tensor. To apply the SVD in each tensor-dimension, the tensor is first 'unfolded' to a matrix. The leading singular vectors of U are used to construct a core, with the number of vectors determining the size of the TT-rank. More details, such as operations, construction, and rounding of TTs are given in Chapter 3.

The Tensor Network Kalman Filter

The Kalman filter variables are rewritten to a TT format in order to avoid the *curse of dimensionality*, resulting in the tensor network Kalman filter (TNKF) [5]. By working with TTs, no explicit vectors and matrices of the filter have to be explicitly represented or worked with. The approach is as follows, matrices are recast to TT matrices, vectors are recast as TT vectors, and scalars remain scalars. The TNKF variables and their memory complexities are summarized in Table 2-6.

At this point, one might wonder how exactly the Kalman filter variables are recast, and how come the tensor can be applied on vectors and matrices. To answer the question, the concepts of tensorization and super compression have to be explained. A tensorization folds a vector or matrix into a higher-dimensional tensor, after which they can be decomposed with tensor

Table 2-6: Variables of the Tensor Network Kalman Filter

Variables	**Recast to**	**TT variable**	**Memory complexities**
P_l	TT matrix	$TT(P_l)$	$\mathcal{O}(dn^2r^2)$
$\boldsymbol{c}_l$, $\boldsymbol{m}_l$,$\boldsymbol{k}_l$	TT vector	$TT(\boldsymbol{c}_l)$, $TT(\boldsymbol{m}_l)$, $TT(\boldsymbol{k}_l)$	$\mathcal{O}(dnr^2)$
y_l, v_l, s_l, w	Scalar	Unchanged	$\mathcal{O}(1)$

networks, such as the tensor train decomposition. The tensorization process only changes the representation of the data. For example, a vector of length N can be folded to a tensor with indices of size n and dimension d, which only changes the memory complexity notation from $\mathcal{O}(N)$ to $\mathcal{O}(n^d)$ as they still have the same number of elements. The benefit of artificially representing a vector or matrix as a tensor is that tensor networks can then be applied to achieve high compression ratios. Applying the tensor train decomposition after tensorizing a vector or matrix reduces the exponential complexity $\mathcal{O}(n^d)$ to a polynomial one $\mathcal{O}(dn^{\{1,2\}}r^2)$ for example, which is a technique called "super-compression" [7].

In context of solving the LS-SVM dual problem with the TNKF, only the (explicit) dual problem rows have to be tensorized in each iteration. The rows are calculated in row-form because of the kernel function, after which they have to be recast to TT vector. In the process of recasting the dual problem row, all kernel-data is used. No sampling procedure is required as its iterative and tensor network formulation allows it to assess all kernel-data in compressed form, unlike current low-rank approximation methods. All other TNKF variables, can be initialized and worked with in TT form during the iterations. Noteworthy is that never in the TNKF is an explicit covariance matrix constructed, limiting the memory complexities that of the tensor trains and dual problem rows.

The TNKF includes the advantages of Bayesian learning and tensor networks. Due to its recursive tensor network Bayesian formulation, prior knowledge can specified, confidence bounds can be obtained, *the curse of dimensionality* is avoided, and all kernel-information of the large-scale problem is utilized. With the mentioned properties, the TNKF overcomes and exceeds the deficiencies of current low-rank approximation methods. In Chapter 3 the TNKF is presented in more detail, its performance evaluated, and compared to the Nyström and FS-LSSVM methods.

Chapter 3

Approach and Results

In this chapter the approach and results of my thesis are given. Currently, I am collaborating with K. Batselier and J.A. Suykens, the inventor of the LS-SVM method, to submit a journal paper to *IEEE: Neural Networks and Learning Systems* later in 2020. Here, the current draft of the paper is presented, covering the outcomes of this thesis.

In the paper, more background theory is given, especially with respect to the implementation of tensor networks. Additionally, the tensor network Kalman filter (TNKF) is analyzed and derived in more detail. The TNKF is applied on four benchmark datasets, two regression and two classification problems, and compared to the performances of the Nyström and FS-LSSVM methods. A short discussion and conclusion of the results are also given, with further details covered in Chapters 4 and 5.

Recursive Tensor Network Bayesian Learning of Large-Scale LS-SVMs

Maximilian Lucassen[†], Kim Batselier[†]

Abstract*—Least squares support vector machines are a commonly used supervised learning method for nonlinear regression and classification. They can be implemented in either their primal or dual form. The latter requires solving of a linear system, which can be advantageous as an explicit mapping of the data to a possibly infinite-dimensional feature space is avoided. However, for large-scale applications, current low-rank approximation methods can perform inadequately for a number of reasons. Their sampling nature, poor approximation power due to the low-ranks, or because they become infeasible due to the curse of dimensionality. In this paper, a recursive Bayesian learning framework based on tensor networks and the Kalman filter is presented to alleviate the demanding memory and computational complexities associated with solving large-scale dual problems. The proposed method is iterative, does not require explicit storage of the kernel matrix, and allows the formulation of early stopping conditions. Additionally, the framework allows incorporation of prior knowledge, measurement noise, truncation of variables, and yields confidence estimates of obtained models, unlike its alternatives. The performance is tested on two regression and two classification experiments, and compared to the Nyström and fixed size LS-SVM methods. Results show that it can achieve high performance and is particularly useful in cases of high nonlinearity or when alternative methods are deficient or impractical.

***Index Terms*—Supervised learning, Bayesian learning, large-scale, ls-svm, fixed size ls-svm, Nyström, dual problem, tensor network, Kalman filter, recursive least-squares, regression, classification, kernel method.**

I. Introduction

Kernel methods are a class of nonlinear modeling methods based on mapping the nonlinear input data to a high-dimensional feature space. In this high-dimensional feature space, the inner product of the data is implicitly computed by a user-defined kernel function, correlating the data points through their similarity. Due to their formulations, the nonlinear data can be modeled with the simplicity of linear algorithms [1] [2] [3]. As a result, kernel methods are an attractive modeling class because of the insight and ease they provide, and are widely applied in numerous disciplines such as control [4], machine learning [5] and signal processing [6].

Support vector machines (SVM) is a popular supervised learning kernel method. The SVM theory was initially proposed by Vapnik [1], in which a cost function is minimized by solving either the parametric primal problem or the non-parametric dual problem. Either case, the modeling is solved with quadratic programming, generating a unique global solution. The least-squares SVM (LS-SVM) theory reformulates the original SVM minimization problem [7] [8]. By changing the cost function to a least squares form, and altering the inequality constraints to one equality constraint, the dual problem is recast to a linear system. This makes the LS-SVM dual problem much easier to solve compared to its SVM counterpart.

Solving of the LS-SVM dual problem, (”dual problem”), scales with the number of data points (N). For small and medium-scale problems, general solvers from the direct and iterative methods classes are applicable [9] [10]. However, for large-scale problems, these can not be used because the memory and computational requirements are too demanding as the number of data points can become exponentially large $N \to n^d$. This demonstrates a phenomena called the ”curse of dimensionality”, in which the exponential scaling with d, the dimension, makes a problem too difficult to solve. One approach of solving such large dual problems is with low-rank approximation methods, such as the Nyström method [11], and fixed-size LS-SVM (FS-LSSVM) [12]. These aim to work around the prohibitive scaling by training on a subset of the data and by employing the primal-dual relationship of LS-SVM. Numerous drawbacks to current low-rank approximation methods exist. Firstly, no confidence bounds of the approximate solution are obtained, only theoretical statistical guarantees can be given. Secondly, it is not possible to include prior knowledge and noise into the solution for more robust modeling. Lastly, the size of the subset significantly impacts the performance, in accuracy, but also the memory and computational complexity. Other approaches to deal with large-scale linear systems commonly focus on parallel computation. For example, in [13] a parallel GPU implementation is used for kernel machines. The linear scaling associated with increasing batch sizes is extended with adaptive kernels to speed up training times and more efficient parallelization.

Bayesian learning is a probabilistic framework in which prediction-observation procedure is used to gradually learn a model [14]. An initial distribution is updated by witnessing instances of the data, yielding posterior distributions that incorporate the observations of the data through a mean and variance. This framework has some advantages. First, user knowledge can be incorporated into the algorithms, such as measurement noise or in the prior distribution. Secondly, the resulting model is also a distribution, from which confidence bounds can be established. Lastly, the algorithms are generally easy to implement and can be designed in a flexible manner. For example, they can be used as a direct method, or alternatively, recursive schemes can be created. Numerous methods exist in this framework, applicable dependent on the

[†] Delft Center for Sytems and Control, Delft University of Technology, Mekelweg 2 2628 CD, Delft, The Netherlands

task. There are filtering algorithms, such as Kalman filters and particle filters, which iteratively update a solution vector of a model [15] [16]. Comparably, there are also kernel Bayesian learning methods. One popular of such is Gaussian processes [17], in which a multivariate normal distribution is found over the set of functions that describe the input-output data. Additionally, common kernel methods can be expanded to Bayesian learning, as has been done with the LS-SVM in [7].

In this paper, the curse of dimensionality associated with solving the large-scale dual problem, and deficiencies of current low-rank approximation methods are addressed. The dual problem is cast to a tensor train (TT) form, in which substantial compression ratios can be achieved. In this form, the dual matrix never has to be constructed explicitly, nor any other matrix. A recursive Bayesian procedure is developed to solve the large-scale dual problem in TT form by implementation of a tensor network Kalman filter (TNKF). The contributions of this work are:

- Circumventing the curse of dimensionality of solving large-scale dual problems by using a low-rank TT representation.
- A non-sampling based algorithm, that evaluates the entire dual matrix to construct low-rank approximations of it, and offers an alternative or supplement to current low-rank approximation methods.
- Implementation of a recursive Bayesian filtering procedure in which, unlike current low-rank approximation methods, prior knowledge and measurement noise can be specified, and prediction confidence bounds are generated.

This paper is organized as follows. In Section II the basics of LS-SVMs and tensor networks are covered, including required operations and the TT decomposition for the TNKF algorithm. In Section III, the Kalman filter is introduced in the context of recursive Bayesian learning. The final TNKF algorithm is presented in Section IV and Section V how the compute its performance on the four datasets in Section VI. Lastly, conclusions and further work are discussed in Section VII.

The notation and abbreviations used in this paper are given in Table I and Table II, respectively.

TABLE I: Used Notation

Scalars	(a,b,...)
Vectors	(**a**,**b**,...)
Matrices	(A,B,...)
Tensors	($\mathcal{A}$,$\mathcal{B}$,...)
Tensor Train of $\mathcal{A}$	$TT(\mathcal{A})$
Matrix transpose	($\mathbf{a}^\intercal$,$A^\intercal$,...)
Identity matrix of size N	I_N
Kernel matrix of size N by N	Ω_N

TABLE II: Used Abbreviations

TT	Tensor train
SVM	Support-vector-machines
LS-SVM	Least-squares support-vector-machines
FS-LSSVM	Fixed size least-squares support-vector-machines
KKT	Karush-Kuhn-Tucker
SVD	Singular value decomposition
TNKF	Tensor network Kalman filter

II. Preliminaries

A. Least-Squares Support Vector Machines

In a supervised learning setting, the input and output data are known, from which a model can be derived. Consider the training set $\{\mathbf{x}_i, y_i\}_{i=1}^N$, with $\mathbf{x}_i \in \mathbb{R}^f$, $y_i \in \mathbb{R}$ for regression, and $y_i \in \{-1, 1\}$ for classification. The primal formulations of the LS-SVM are given in Equation 1 and Equation 2, which can be solved with quadratic programming.

$$\begin{aligned} &\min_{\mathbf{w},b,\mathbf{e}} \quad J_{primal}(\mathbf{w},\mathbf{e}) = \frac{1}{2}\mathbf{w}^\intercal\mathbf{w} + \frac{\gamma}{2}\sum_{i=1}^N e_i^2 \\ &\qquad\text{subject to:} \\ &\qquad y_i = \mathbf{w}^\intercal\varphi(\mathbf{x}_i) + b + e_i, \quad i = 1,\dots,N \\ &\qquad\text{Primal regression model:} \\ &\qquad y(\mathbf{x}) = \mathbf{w}^\intercal\varphi(\mathbf{x}) + b \end{aligned} \tag{1}$$

$$\begin{aligned} &\min_{\mathbf{w},b,\mathbf{e}} \quad J_{primal}(\mathbf{w},\mathbf{e}) = \frac{1}{2}\mathbf{w}^\intercal\mathbf{w} + \frac{\gamma}{2}\sum_{i=1}^N e_i^2 \\ &\qquad\text{subject to:} \\ &\qquad y_i\left(\mathbf{w}^\intercal\varphi(\mathbf{x}_i) + b\right) = 1 - e_i, \quad i = 1,\dots,N \\ &\qquad\text{Primal classifier model:} \\ &\qquad y(\mathbf{x}) = \text{sign}\left(\mathbf{w}^\intercal\varphi(\mathbf{x}) + b\right) \end{aligned} \tag{2}$$

In the primal models, $\varphi(\mathbf{x})$ is a nonlinear mapping function from the f-dimensional input space to the f_h-dimensional feature space, $\varphi(\mathbf{x})$: $\mathbb{R}^f \to \mathbb{R}^{f_h}$, $\mathbf{w}$ is the primal weight vector, γ is a regularization parameter, e_i are the model error terms, and b the bias term.

The primal problems can be advantageous to solve for large-scale problems if the number of features is small, since it is parametric. However, the primal problem is often impossible because a suitable nonlinear mapping function is usually unknown, difficult to determine, and possibly infinite-dimensional. Fortunately, the dual problem can be solved alternatively, derived by rewriting the inner-products of the nonlinear functions in the feature space by a user-defined kernel function $k(\mathbf{x}', \mathbf{x})$. This is known as the "kernel trick", which implicitly computes the inner-product in the feature space without ever needing an explicit nonlinear mapping function [7]. In Equation 3 and Equation 4 the dual problems for regression and classification are shown, which require solving of a linear system instead of quadratic programming. In the dual form, the $\boldsymbol{\alpha}$ weights are related to the error variables through $\alpha_k = \gamma e_k$, and are solved for to obtain a regression or classification model.

$$\begin{aligned} &\begin{bmatrix} 0 & \mathbf{1}^\intercal \\ \mathbf{1} & \Omega_N + I_N/\gamma \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{y} \end{bmatrix} \\ &\text{Dual regression model:} \\ &y(\mathbf{x}) = \sum_{i=1}^N \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b \end{aligned} \tag{3}$$

$$\begin{bmatrix} 0 & \mathbf{y}^\intercal \\ \mathbf{y} & \Omega_N + I_N/\gamma \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{1} \end{bmatrix}$$

Dual classification model: (4)

$$y(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{N} y_i \alpha_i k\left(\mathbf{x}, \mathbf{x}_i\right) + b\right)$$

B. The Large-Scale Dual Problem and Low-Rank Approximation Methods

For small problems, the dual is simple to solve. Many direct and iterative methods are applicable, such as the LU decomposition and conjugate gradient [10] [18]. Large-scale problems become difficult or infeasible because these methods need explicit storage of - and directly operate on the kernel matrix. Due to the $\mathcal{O}(n^{2d})$ memory requirement and computational complexities, $\mathcal{O}(n^{3d})$ for direct methods and $\mathcal{O}(n^{2d}l)$ for iterative methods (l iterations), the curse of dimensionality limits their application. A common way to deal with this difficulty is to employ low-rank approximation methods. Utilizing a subset of the data by sampling S columns, it is possible to approximate the solution. A trade-off between the obtainable accuracy, and the associated computational and memory requirements is made for the problem to be feasible. Two well-known low-rank approximation techniques for LS-SVMs are the Nyström method and FS-LSSVM [11] [12]. Both are sampling-based methods that implicitly rely on the eigendecomposition of the kernel matrix. The Nyström method solves the LS-SVM in the dual with computational and memory complexities of $\mathcal{O}(S^2 n^d + S^3)$ and $\mathcal{O}(Sn^d)$. FS-LSSVM estimates a nonlinear mapping function in the dual, then solves the primal problem. The computational and memory costs of FS-LSSVM are $\mathcal{O}(S^2 n^d + 2S^3)$ and $\mathcal{O}(S^2)$ [9]. For both methods, the sampling procedure has a significant impact on the performance. Many procedures exist for this, such as uniform (random) sampling, or if computationally achievable, more advanced adaptive techniques can be used, such as sparse greedy matrix approximation [19], leverage-scores [20], column-norm sampling [21], and Renyi-entropy sampling [7].

C. Tensor Network Basics

Tensors are multidimensional extensions of vectors and matrices to higher dimensions, relevant in many fields: signal processing, machine learning, and quantum physics [22] [23]. In this article, we define a tensor $\mathcal{T} \in \mathcal{R}^{n_1 \times n_2 \cdots n_d}$ as having an order d and dimensions $n_1, \ldots, n_d$. The elements of a tensor are given by its indices, (i_1,i_2,...,i_d), for which the MATLAB notation is used: $1 \leq i_k \leq n_k$. Even though tensors can be worked with in their mathematical form, it is easier to use a visual representation, as shown in Figure 1. A tensor is illustrated as a core (circle) with a number of outgoing edges (lines) equal to its order.

Tensor networks are factorizations of tensors, analogous to matrix factorizations of matrices. In fact, it is a generalization of matrix decompositions to higher orders. An easy introduction is given through the singular value decomposition (SVD) of a matrix A into two orthogonal matrices

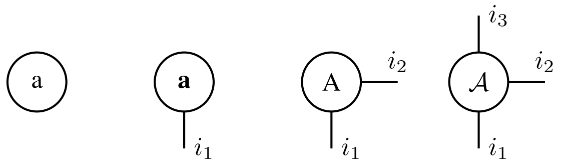

Fig. 1: Diagrammatic tensor notation of a scalar (a), a vector (**a**), a matrix (A), and a 3-dimensional tensor ($\mathcal{A}$).

(U,V) and a diagonal matrix (Σ). It is one specific type of matrix factorization, and can therefore also be represented as a tensor network. Visually, this is shown in Figure 2. The interconnected edges represent multiplications between the SVD matrices. The sizes of the shared edges (r_1,r_2) are determined by the number of singular values and are the dimensions of matrix Σ. Generalized for higher orders, the shared edges between tensors represent tensor contractions, which are summations over shared indices as defined in Definition 1. Tensor contractions can be understood as a higher order generalization of matrix multiplications.

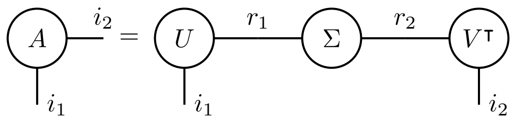

Fig. 2: Tensor network representation of the SVD.

Definition 1 (Tensor contraction [24]). *Without loss of generality, a tensor contraction of two tensors over two shared indices i_2 and i_3 is mathematically defined as:*

$$\mathcal{C}(i_1, i_4, i_5, i_6) = \sum_{i_2=1}^{n_2} \sum_{i_3=1}^{n_3} \mathcal{A}(i_1, i_2, i_3) \mathcal{B}(i_2, i_3, i_4, i_5, i_6) \quad (5)$$

In Figure 3, the visual representation of Definition 1 is given, in which tensors $\mathcal{A}$ and $\mathcal{B}$ are contracted over two indices. As can be seen, more than one edge can be interconnected and contracted. The number of free edges of a tensor network determines the order of the resulting tensor. For this reason, the contraction results in a 4-way tensor.

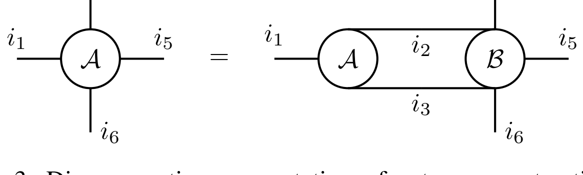

Fig. 3: Diagrammatic representation of a tensor contraction over the shared indices i_2 and i_3.

An important concept in tensor learning is reshaping. Tensors can be folded and unfolded to be represented in different dimensions. It is useful for concepts like matricization and tensorization. A matricization unfolds a tensor into a matrix, and is commonly implemented to allow straightforward application of concepts from linear algebra. A tensorization folds a vector or matrix into a higher-dimensional tensor, after which it can be decomposed with tensor networks. These operations are shown in Figure 4. The right arrow displays a matricization operation, in which the indices are joined together to form a new artificial combined index. The left arrow denotes the tensorization operation, in which indices are split to form more producing a higher-dimensional tensor. Both operations are accomplished through the reshape operation presented in Definition 2.

Definition 2 (Reshape operation [25]). *We adopt the MATLAB reshape operation, which reshapes a d-dimensional tensor into a tensor with dimensions $n_1 \times n_2 \times ... \times n_d$ by "reshape($\mathcal{A}$, $[n_1, n_2, n_3, ...]$)". The total number of elements of the reshaped tensor must be the same as the initial one $n_1 \times n_2 \times ... \times n_d$.*

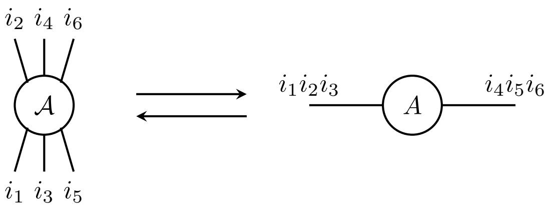

Fig. 4: Arrow to the right: matricization of a 6th-order tensor over its index i_3. Arrow to the left: tensorization of the matrix to a 6th-order tensor.

D. Tensor Trains

In their high-dimensional form, tensors can be difficult to work with. Firstly, the computational and memory complexities are burdensome because of the exponential scaling with the order d, $\mathcal{O}(n^d)$. Secondly, concepts from linear algebra are not directly applicable, making analysis difficult. Tensor networks give more flexibility, as the tensor is factorized into a number of lower-dimensional tensor network components, also called cores. These factorizations can alleviate the adverse scaling and offer more insight. The CP and Tucker decompositions are two well-known tensor network structures, but have significant drawbacks, such as NP-hardness and poor scaling [24]. The tensor train (TT) decomposition avoids these issues and can therefore be a more easy-to-use and robust format [26] [27]. As the name suggests, a TT decomposition factorizes a tensor into a chain of cores, which are linked through their "TT-ranks" (r_k, $r_k \leq r$) as shown in Figure 5 and 6. These ranks signify the amount of correlation between the dimensions of a tensor, and determine the complexities of the representation. In this paper, two types of tensor trains are used: tensor train vectors, illustrated in Figure 5, and tensor train matrices in Figure 6. Intuitively, these can be thought of as a coarse graining procedure. TT vectors can be understood as vectors inside vectors, and TT matrices as matrices inside matrices. For small TT-ranks, this format yields significant compression ratios as the memory complexities scale with the ranks. The storage cost scales $\mathcal{O}(dnr^2)$ for a TT vector, and $\mathcal{O}(dn^2r^2)$ for a TT matrix.

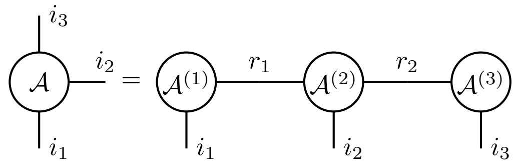

Fig. 5: Diagrammatic representation of a tensor train vector decomposition of a 3-dimensional tensor. Each core ($\mathcal{A}^{(k)}$) can be understood as a vector as they have one free edge.

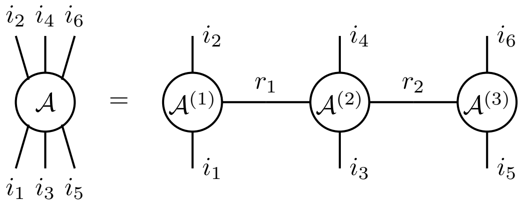

Fig. 6: Diagrammatic representation of a tensor train matrix decomposition of a 6-dimensional tensor. Each core ($\mathcal{A}^{(k)}$) can be understood as a matrix as they have two free edges.

Essential operations performed with the TTs in this paper and summarized as following. A contraction between two TTs is accomplished by contracting the k-th core of one TT with the k-th core of the other TT, and denoted by "Contract($TT(\mathcal{A}), TT(\mathcal{B})$)". In Figure 7 this is visualized for a contraction between two TT vectors, which in this case is equivalent to an inner-product as there are no free edges. Similarly, the outer product between two TTs can be computed by taking the outer product between individual cores, given by "Out-Prod($TT(\mathcal{A}), TT(\mathcal{B})$)", shown in Figure 8. The Hadamard product is calculated by applying the Khatri-Rao product between respective individual cores. For these operations, the order in which the TTs are written matters, as only adjacently written TTs have shared indices. Addition and subtraction in TT form is done by a concatenation procedure between respective cores. Multiplication between a scalar and a TT is executed by multiplying the elements of one of the TT cores by the scalar. These operations are denoted by their regular mathematical notation. For more detailed explanations see [24] [26] [27].

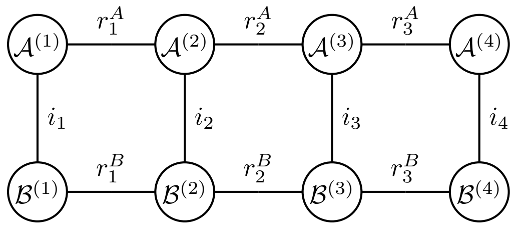

Fig. 7: Diagrammatic representation of an inner product between two tensor train vectors. As there are no free edges, the result is a scalar.

In this paper, the "TT-SVD" and "TT-rounding" algorithms from [26] are used for the construction and rounding of the TTs. In the TT-SVD algorithm, a δ-truncated SVD is

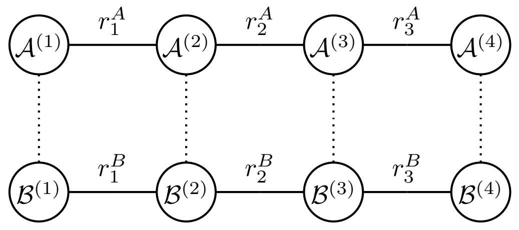

Fig. 8: Diagrammatic representation of an outer product between two tensor train vectors. Each dotted line represents an outer product between the respective cores. In this particular case, the result is a TT matrix.

iteratively applied on matricizations according to each tensor dimension to form a TT. According to a specified maximum rank and/or norm (ϵ, r), the TT "truncation parameters", the smallest singular values are removed. The remaining singular values determine the respective TT-ranks between the k-th and (k+1)-th cores, and therefore also the computational and memory requirements of the resulting TT. The TT-SVD constructs a TT as seen in Figure 5 and Figure 6 with a computational cost of $\mathcal{O}(dnr^3)$. The TT-rounding algorithm is necessary to limit the memory and computational complexities, when the TT-ranks grow too large. Rank growth is caused by mathematical operations, such as contractions and additions, and can cause the TT representation to become burdensome. The algorithm first computes QR decompositions of the TT cores, then performs core contractions, and lastly employs the TT-SVD algorithm to reconstruct the TT. Its computational cost is $\mathcal{O}(dnr^2 + dr^4)$. Even though TT-rounding can limit rank growth, it can also be a computational bottleneck due to the quartic scaling with the ranks. Fortunately, with quantization, it is also possible to restrain the TT-ranks. Quantization is the process of reshaping a large-dimensions-low-order tensor into a small-dimensions-large-order tensor $\mathcal{O}(n_{\text{large}}{}^{d_{\text{small}}}) \rightarrow \mathcal{O}(n_{\text{small}}{}^{d_{\text{large}}})$. Computing a TT after quantization generally requires much smaller TT-ranks in its representation. Very high compression ratios can be achieved this way, making it possible to store and work with extremely large datasets [28].

III. Bayesian Learning and the Kalman Filter

A Bayesian learning approach is adopted to solve the LS-SVM dual problem, which has numerous benefits. First, prior knowledge can be incorporated, such as measurement noise for imperfect observations, and the specification of an initial distribution for the solution. Secondly, the learning process yields confidence bounds, informative of how well the data is described by the mean. Lastly, Bayesian learning is a flexible and easy-to-use framework, and can readily be adopted to solve the LS-SVM dual problem.

A regression task is considered in this derivation, but the formulation can easily be modified for a classification case. Here, the dual problem is simplified to $C\bar{\boldsymbol{\alpha}} = \bar{\boldsymbol{y}}$ for convenience. C is the dual problem matrix, $\bar{\boldsymbol{\alpha}} = [b; \boldsymbol{\alpha}]$, and $\bar{\boldsymbol{y}} = [0; \boldsymbol{y}]$. It is desired to obtain a linear model to fit the kernelized data by finding a suitable $\bar{\boldsymbol{\alpha}}$, as presented in Equation 6. A zero-mean Gaussian variable $u \sim \mathcal{N}(0, w^2)$ is introduced to model observation noise, and serves as a robustness measure to prevent overfitting. Each output value is associated with a row of the dual problem $\boldsymbol{c}_l$, as given by the subscript l.

$$\bar{y}_l = \boldsymbol{c}_l \bar{\boldsymbol{\alpha}} + u \tag{6}$$

The Bayesian learning process is summarized by Bayes' rule Equation 7 [15].

$$\mathcal{P}\left(\bar{\boldsymbol{\alpha}} \mid y_{1:T}\right) \propto \mathcal{P}\left(\bar{\boldsymbol{\alpha}} \mid \boldsymbol{m}_0, P_0\right) \prod_{k=1}^{T} \mathcal{P}\left(\bar{y}_k \mid \mathbf{c}_l \bar{\boldsymbol{\alpha}}, u\right) \tag{7}$$

By witnessing T observations of the data, a posterior distribution can be computed based on updating the prior. In this paper, it is assumed that the dual weights adhere to a Gaussian distribution such that the prior can be specified as $\bar{\boldsymbol{\alpha}} \sim \mathcal{N}(\boldsymbol{m}_0, P_0)$. Because the prior and measurement noise are Gaussian, the likelihood and posterior distributions are also Gaussian. As a consequence, from Bayes' rule an analytic solution can be derived to calculate the posterior batch distribution $\bar{\boldsymbol{\alpha}} \sim \mathcal{N}(\boldsymbol{m}_T, P_T)$, given in Equation 8.

$$\begin{aligned} \boldsymbol{m}_T &= \left[P_0^{-1} + \frac{1}{w^2} C^\intercal C\right]^{-1} \left[\frac{1}{w^2} C^\intercal \bar{\boldsymbol{y}} + P_0^{-1} \boldsymbol{m}_0\right] \\ P_T &= \left[P_0^{-1} + \frac{1}{w^2} C^\intercal C\right]^{-1} \end{aligned} \tag{8}$$

The matrix inverse computations in the batch equations pose a significant problem for large-scale Bayesian learning, as they scale $\mathcal{O}(n^{3d})$. By realizing that posterior distributions can also serve as prior distribution for each succeeding observation, a recursive framework can be developed that avoids matrix inversions. In this paper, single observations are used to update the $\bar{\boldsymbol{\alpha}}$ distribution, which is equivalent to solving the dual problem row-by-row. Linear update equations can be designed for this iterative procedure, that can incorporate user-knowledge of the update-dynamics, as shown Equation 9. For the classification case, the iterative model is modified by changing $\bar{\boldsymbol{y}}_l$ to $[0; \mathbf{1}] = \bar{\mathbf{1}}$, presented in Equation 10.

Regression model:

$$\begin{aligned} \bar{\boldsymbol{\alpha}}_{l+1} &= \bar{\boldsymbol{\alpha}}_l + \boldsymbol{q} \\ \bar{y}_l &= \mathbf{c}_l \bar{\boldsymbol{\alpha}}_l + u \end{aligned} \tag{9}$$

Classification model:

$$\begin{aligned} \bar{\boldsymbol{\alpha}}_{l+1} &= \bar{\boldsymbol{\alpha}}_l + \boldsymbol{q} \\ \bar{1}_l &= \mathbf{c}_l \bar{\boldsymbol{\alpha}}_l + u \end{aligned} \tag{10}$$

In the update equations of the dual weights, a Gaussian noise term $\boldsymbol{q} \sim \mathcal{N}(\mathbf{0}, Q)$ is introduced to model uncertainties in the updates. The recursive Bayesian learning procedure is written in Equation 11.

$$\mathcal{P}\left(\bar{\boldsymbol{\alpha}}_l \mid \bar{y}_{1:l}\right) \propto \mathcal{P}\left(\bar{\boldsymbol{\alpha}}_{l-1} \mid \bar{y}_{1:l-1}\right) \mathcal{P}\left(\bar{y}_k \mid \mathbf{c}_l \bar{\boldsymbol{\alpha}}_{l-1}, u\right) \tag{11}$$

The designed recursive Bayesian framework directly leads to the Kalman filter equations, an optimal-closed-form algorithm for linear stochastic models [15] [16]. By recursively predicting and measuring the output values, or labels, the $\bar{\boldsymbol{\alpha}}_{l-1}$ distribution is updated for each row. The Kalman filter equations are presented in Equation 12 and 13, split into a prediction and update (observation) step. The Kalman filter requires a number of assumptions. First, all distributions are Gaussian, and the observation noise u is I.I.D with $w^2 > 0$. Secondly, linear iterative prediction and update models are required, which can be modeled differently than in this paper if desired [15].

Prediction step:

$$\begin{aligned} \boldsymbol{m}_l^- &= \boldsymbol{m}_{l-1} \\ P_l^- &= P_{l-1} + Q \end{aligned} \tag{12}$$

Update step:

$$\begin{aligned} v_l &= \bar{y}_l - \boldsymbol{c}_l \boldsymbol{m}_l^- \quad \text{(Regression)} \\ v_l &= \bar{1}_l - \boldsymbol{c}_l \boldsymbol{m}_l^- \quad \text{(Classification)} \\ s_l &= \mathbf{c}_l P_l^- \boldsymbol{c}_l^\intercal + w^2 \\ \boldsymbol{k}_l &= P_l^- \mathbf{c}_l^\intercal s_l^{-1} \\ \boldsymbol{m}_l &= \boldsymbol{m}_l^- + \boldsymbol{k}_l v_l \\ P_l &= P_l^- - \boldsymbol{k}_l s_l \boldsymbol{k}_l^\intercal \end{aligned} \tag{13}$$

The Kalman filter is summarized as follows: $\boldsymbol{m}_l \in \mathcal{R}^N$ and the $P_l \in \mathcal{R}^{N\times N}$ are the mean vector and covariance matrix of $\bar{\boldsymbol{\alpha}}$ after iteration l, $\bar{\boldsymbol{\alpha}}_l \sim \mathcal{N}(\boldsymbol{m}_l, P_l)$. The covariance matrix can be understood as a measure of uncertainty in $\bar{\boldsymbol{\alpha}}$, which theoretically converges to zero over the iterations. The noise covariance matrix Q is a weighting term to describe the confidence in the update of covariance matrix. If it set to zero, or used as a "forgetting" term ($P_l^- = \lambda P_{l-1}$), the recursive least squares algorithm is obtained for the specific iterative models from Equation 9 and 10. A forgetting term can induce an exponential convergence in of the covariance [29, pg.240]. The prediction error, dependent on whether classification or regression is performed, is given by $v_l \in \mathcal{R}$. Variable $s_l \in \mathcal{R}$, is the prediction-error variance, which is partially dependent on the variance of the observation noise. Lastly, $\boldsymbol{k}_l \in \mathcal{R}^N$ is the Kalman gain, which influences the degree to which observations are used to update the distribution of $\bar{\boldsymbol{\alpha}}_l$. Termination ($L$) occurs when all rows of the dual problem have been iterated over, or if user-specified stopping conditions have been satisfied. Stopping criteria can be designed arbitrarily, and can help prevent overfitting and save training time. Common specifications are based on the covariance norm, but for application on LS-SVM dual problems, the Karush-Kuhn-Tucker (KKT) conditions can also be used. The final distribution ($\boldsymbol{\alpha}_L \sim \mathcal{N}(\boldsymbol{m}_L, P_L)$) is the obtained solution (posterior) for the training stage. The final mean $\boldsymbol{m}_L$ is the obtained model for regression or classification, for which confidence bounds can be calculated with its covariance (P_L), similarly to Gaussian processes [4][17]. New points can be learned with the Kalman filter in similar fashion, by using the obtained solution as the new prior distribution.

IV. Tensor Network Kalman Filter

The Kalman filter gives a simple iterative method for solving the dual problem, but the covariance matrix limits its feasibility for large-scale problems. Its explicit representation demands an $\mathcal{O}(n^{2d})$ complexity. Fortunately, recasting the Kalman filter to a tensor network Kalman filter (TNKF) form avoids this problem, as has been done similarly in [30] and [31]. The reformulation is simple, vectors are represented as TT vectors, matrices as TT matrices, and scalars remain scalars. To generate the TTs, the Kalman filter variables are quantized and reshaped to tensors by a user-defined n and d. After choosing a quantization, the variables can then be given in their TT form, resulting in the compressed representation in Table III.

TABLE III: Variables of the TNKF

Variables	Recast to	TT variable	Memory Scaling
P_l	TT matrix	$TT(P_l)$	$\mathcal{O}(dn^2r^2)$
$\boldsymbol{c}_l$, $\boldsymbol{m}_l$, $\boldsymbol{k}_l$	TT vector	$TT(\boldsymbol{c}_l)$, $TT(\boldsymbol{m}_l)$, $TT(\boldsymbol{k}_l)$	$\mathcal{O}(dnr^2)$
y_l, v_l, s_l, w	Scalar	Unchanged	$\mathcal{O}(1)$

The operations of the Kalman filter also easily translate to the TT format. As described in Subsection II-D, these are simply done by performing sequences of contractions or concatenations. The TNKF algorithm is summarized in Algorithm 1.

Algorithm 1 Tensor Network Kalman Filter

```
Require: $TT(\boldsymbol{m}_0)$, $TT(P_0)$, $w^2$, $k(\boldsymbol{x}', \boldsymbol{x})$, $\{\boldsymbol{x}_i, y_i\}_{i=1}^N$, $\gamma$, $\lambda$, $n$, $d$, $(\boldsymbol{\epsilon}, \boldsymbol{r})$.
  while Termination conditions not met do
    1. $TT(\boldsymbol{m}_l^-) = TT(\boldsymbol{m}_{l-1})$
    2. $TT(P_l^-) = TT(P_{l-1}) + Q$
    3. Construct $l$-th dual row $\mathbf{c}_l$, with $k(\boldsymbol{x}_l, \boldsymbol{x}_i)$ and $\gamma$
    4. Reshape $\mathbf{c}_l$ to a $n^d$-tensor, $\mathcal{C}_l$.
    5. $TT(\boldsymbol{c}_l)$ = TT-SVD$(\mathcal{C}_l, \epsilon_{\boldsymbol{c}}, r_{\boldsymbol{c}})$.
    6. $v_l = y_l -$Contract$\big(TT(\mathcal{C}_l), TT(\boldsymbol{m}_l^-)\big)$ or
       $v_l = 1-$Contract$\big(TT(\mathcal{C}_l), TT(\boldsymbol{m}_l^-)\big)$
    7. $s_l$ = Contract$\big(TT(\boldsymbol{c}_l), TT(P_l^-), TT(\boldsymbol{c}_l)^\intercal\big) + w^2$
       with intermediate TT-rounding , $(\epsilon_s, r_s)$
    8. $TT(\boldsymbol{k}_l)$ = Contract$\big(TT(P_l^-), TT(\boldsymbol{c}_l)\big)\ s_l^{-1}$
    9. $TT(\boldsymbol{k}_l)$ = TT-rounding$(TT(\boldsymbol{k}_l), \epsilon_{\boldsymbol{k}}, r_{\boldsymbol{k}})$
    10. $TT(\boldsymbol{m}_l) = TT(\boldsymbol{m}_l^-) + TT(\boldsymbol{k}_l) \cdot v_l$
    11. $TT(\boldsymbol{m}_l)$ = TT-rounding$(TT(\boldsymbol{m}_l), \epsilon_{\boldsymbol{m}}, r_{\boldsymbol{m}})$
    12. $TT(P_l) = TT(P_l^-)+$Out-Prod$(TT(\boldsymbol{k}_l), TT(\boldsymbol{k}_l))s_l^{-1}$
    13. $TT(P_l)$ = TT-rounding$(TT(P_l), \epsilon_P, r_P)$
  end while
```

The initial distribution of $\bar{\boldsymbol{\alpha}}$ is supplied in TT form as the explicit construction of the covariance matrix can be impractical or even impossible for large datasets. In [30], a more detailed explanation is given on how a TT-rank-1 initial distribution can be supplied for the TNKF. In each iteration, a row of the dual is constructed with the training data and kernel function. The dual row is then reshaped to a tensor, after which it is decomposed to a low-rank TT vector. The necessity of storing more than one row of the dual matrix is thereby avoided, and permits application of very large datasets. During the iterations, the ranks of the TTs have to be truncated after operations because their growth quickly diminishes the computational and memory advantages of this format. For each TT variable, truncation parameters can be defined, for convenience, a vector notation is used to refer to all of them ($\boldsymbol{\epsilon}$, $\boldsymbol{r}$). These govern the size of the ranks, and therefore determine trade-off between accuracy and algorithm complexity. In Table IV, the complexity of each step is presented. Two assumptions are made in this, Q is designed as a forgetting factor ($\lambda \neq 1$) with cost $\mathcal{O}(n^2 r_P)$, and that kernel functions need at least $\mathcal{O}(n^{2d})$ operations per row to compare the data points. From the overall TNKF complexity, which includes the construction of the l-th dual row, it is clear that the ranks heavily influence scaling. Especially the ranks of $TT(\boldsymbol{c}_l)$ and $TT(P_l)$ determine much of the algorithm's computational cost, as they are required in many of the steps, and impact the ranks of the other variables. It is also evident that construction of the dual rows can be a dominating factor for large datasets. Neglecting the complexity of this, and of lower order terms, training with the TNKF scales approximately $\mathcal{O}(ldnr^6)$, where r can be considered the largest achieved rank in the algorithm.

TABLE IV: Complexities of the TNKF

Step	**Computational complexities**
1.	$\mathcal{O}(1)$
2.	$\mathcal{O}(r_P n^2)$
3.	$\mathcal{O}(\geq n^{2d}) = (\mathcal{O}(\geq N^2))$
4.	$\mathcal{O}(1)$
5.	$\mathcal{O}(dnr_{\boldsymbol{c}}^3)$
6.	$\mathcal{O}(dnr_{\boldsymbol{c}}^2 r_{\boldsymbol{m}}^2)$
7.	$\mathcal{O}(dnr_{\boldsymbol{c}}^4 r_P^2)$
8.	$\mathcal{O}(dn^2 r_{\boldsymbol{c}}^2 r_P^2)$
9.	$\mathcal{O}(dnr_{\boldsymbol{k}}^2 + dr_{\boldsymbol{k}}^4)$
10	$\mathcal{O}(nr_{\boldsymbol{k}})$
11.	$\mathcal{O}(dnr_{\boldsymbol{m}}^2 + dr_{\boldsymbol{m}}^4)$
12.	$\mathcal{O}(dn^2 r_{\boldsymbol{k}}^4)$
13.	$\mathcal{O}(dnr_p^2 + dr_P^4)$
Overall	$\approx \mathcal{O}(l(n^{2d} + dn^2\boldsymbol{r}^4 + dnr_{\boldsymbol{c}}^4 r_P^2))$
Only TNKF	$\approx \mathcal{O}(ldnr^6)$

V. Computation of Confidence Bounds

The output of Algorithm 1 is the posterior distribution of the dual weights and bias, $\mathcal{P}(\bar{\boldsymbol{\alpha}}|\{y_i\}_{i=1}^L) \sim \mathcal{N}(TT(\boldsymbol{m}_L), TT(P_L))$, which can then be used for validation and testing. The notation for test data and variables is a superscript t. The dual models, from Equation 3 and 4, can be implemented in TT form to keep working with lower computational and memory complexities. Consider N^t test points, which can also be exponentially large ($N^t \to n^{d_t}$), $\{\mathbf{x}_i^t\}_{i=1}^{N^t}$ for which $\{y_i^t\}_{i=1}^{N^t}$ are to be predicted. Similarly to Algorithm 1, dual rows based on the training and test data are constructed and transformed to a TT vector $TT(\boldsymbol{c}_l^t)$, with indices of size n and d cores. Then, depending on the task, are eventually contracted with $TT(\boldsymbol{m}_L)$, yielding the predictions. To obtain the confidence bounds of the predictions, step 7. from Algorithm 1 is repeated with $TT(\boldsymbol{c}_l^t)$. $TT(P_L)$ is used to construct the $\pm 3\sigma$ (99.73%) confidence bounds by incoporating the standard deviations into the model. The bounds describe how well the kernel regression or classification models fit the data, thus are still very dependent on the hyperparameters. The TT test regression and classification procedures are given in Algorithm 2 and 3, respectively.

Algorithm 2 TNKF Regression - Prediction and computation of confidence bounds of new inputs.

```
Require: $TT(\boldsymbol{m}_L)$, $TT(P_L)$, $w^2$, $k(\boldsymbol{x}', \boldsymbol{x})$, $\{\boldsymbol{x}_i\}_{i=1}^N$, $\{\boldsymbol{x}_i^t\}_{i=1}^{N^t}$, $\gamma$, $n$, $d$, $(\boldsymbol{\epsilon}, \boldsymbol{r})$
  for $j$ = $1 : (N^t + 1)$ do
    1. Construct $j$-th dual row $\boldsymbol{c}_j^t$, with $k\left(\boldsymbol{x}_j^t, \boldsymbol{x}_i\right)$ and $\gamma$
    2. Reshape $\boldsymbol{c}_j^t$ to a $n^d$-tensor, $\mathcal{C}_j^t$.
    3. $TT(\boldsymbol{c}_j^t)$ = TT-SVD($\mathcal{C}_j^t, \epsilon_{\boldsymbol{c}}, r_{\boldsymbol{c}}$).
    4. $\bar{y}_j^t$ = Contract($TT(\boldsymbol{m}_L)$, $TT(\boldsymbol{c}_j^t)$)
    5. $\sigma_j^2$ = Contract$\big(TT(\boldsymbol{c}_j^t), TT(P_L), TT(\boldsymbol{c}_j^t)\big) + w^2$ with intermediate TT-rounding ($\epsilon_{var}$, $r_{var}$)
  end for
  $\mathbf{y}^t = \bar{\boldsymbol{y}}^t$
  $\mathbf{y}_{99\%}^t = \bar{\boldsymbol{y}}^t \pm 3\boldsymbol{\sigma}$
```

Algorithm 3 TNKF - Classification and computation of confidence bounds of new samples.

```
Require: $TT(\boldsymbol{m}_L)$, $TT(P_L)$, $w^2$, $k(\boldsymbol{x}', \boldsymbol{x})$, $\{\boldsymbol{x}_i, y_i\}_{i=1}^N$, $\{\boldsymbol{x}_i^t\}_{i=1}^{N^t}$, $\gamma$, $n$, $d$, $(\boldsymbol{\epsilon}, \boldsymbol{r})$
  1. Reshape $\boldsymbol{y}$ to a $n^d$-tensor.
  2. $TT(\boldsymbol{y})$ = TT-SVD($\mathbf{y}, \epsilon_{\boldsymbol{y}}, r_{\boldsymbol{y}}$).
  3. Compute $TT(\boldsymbol{\beta})$ = Hadamard($TT(\boldsymbol{y}), TT(\boldsymbol{m}_L)$),
  for $j$ = $1 : (N^t + 1)$ do
    4. Construct $j$-th dual row, $\boldsymbol{c}_j^t$, with $k\left(\mathbf{x}_j^t, \mathbf{x}_i\right)$ and $\gamma$.
    5. Reshape $\boldsymbol{c}_j^t$ to a $n^d$-tensor, $\mathcal{C}_j^t$.
    6. $TT(\boldsymbol{c}_j^t)$ = TT-SVD($\mathcal{C}_j^t, \epsilon_{\boldsymbol{c}}, r_{\boldsymbol{c}}$).
    7. $\bar{y}_j^t$ = Contract($TT(\boldsymbol{\beta}), TT(\boldsymbol{c}_j^t)$)
    8. $\sigma_j^2$ = Contract$\big(TT(\boldsymbol{c}_j^t), TT(P), TT(\boldsymbol{c}_l^t)\big)\big) + w^2$ with intermediate TT-rounding ($\epsilon_{var}$, $r_{var}$)
  end for
  $\mathbf{y}^t$ = sign $(\bar{\boldsymbol{y}}^t)$
  $\mathbf{y}_{3\sigma}^t$ = sign $(\bar{\boldsymbol{y}}^t \pm 3\boldsymbol{\sigma})$
```

In comparison to the other considered low-rank approximation methods, Nyström and FS-LSSVM, the TNKF can be advantageous if the data admits to a low-rank TT structure. Without considering the computational cost of sampling procedures and construction of dual matrices or rows, the TNKF is favorable when the alternatives require many samples for the problem, as given in Table V. If this is the case, the computational and memory benefits of the TNKF can be significant because the Nyström and FS-LSSVM methods require storage of and operate on matrices. Additionally, for small TT-ranks, the final dual model complexity produced by the TNKF can be significantly smaller because of the TT format. If the number of needed support vectors is small, then the Nyström and FS-LSSVM are favorable. This is simply because the subset is small and informative enough to compensate for evaluating the entire dual matrix, like TNKF does. On the other hand, because the TNKF evaluates the entire dual to construct a δ-

truncated TT, it incorporates the most essential information and is deterministic, unlike the other methods.

TABLE V: Complexities of Low-rank Approximation Methods [9].

Method	Computational complexity	Memory complexity	Model complexity
Nyström	$\mathcal{O}(S^2n^d + S^3)$	$\mathcal{O}(Sn^d)$	$\mathcal{O}(n^d)$
FS-LSSVM	$\mathcal{O}(S^2n^d + 2S^3)$	$\mathcal{O}(S^2)$	$\mathcal{O}(f)$
TNKF	$\mathcal{O}((J+L)dnr^6)$	$\mathcal{O}(n^d)$ or $\mathcal{O}(dn^2r^2)$	$\mathcal{O}(dn^2r^2)$

* Here, N is written as n^d, f is the number of features of $\boldsymbol{x}$, and J the number of iterations in Algorithm 2 or 3.

VI. Experiments

In this section, the TNKF algorithm is applied to a number of large-scale regression and classification problems and compared to the FS-LSSVM and Nyström methods. In the first regression problem a sinc function corrupted with Gaussian noise needs to be estimated, as done in [7]. In the second regression problem, the F16 nonlinear benchmark dataset, the vibration dynamics of a dummy-load equipped wing needs to be modeled from a sine-sweep input signal [32]. A large-scale version of the two-spiral problem is the first classification problem, as considered in [7], but here with over 90 turns around the center. Lastly, the Adult classification problem [33] is used to determine whether a person's annual income is greater, or less than, $50k per year. Table VI gives an overview of the datasets.

TABLE VI: Considered datasets in this paper

Datasets	# Training and Quantization	# Test and Quantization	# Features
Noisy sinc	16384, (2^{14})	8096, (2^{13})	1
F16 Benchmark	16384, (2^{14})	16384, (2^{14})	13
Two-spiral	65536, (2^{16})	65536, (2^{16})	2
Adult	19683, (3^9)	2187, (3^7)	$16 \rightarrow 99$

Because of the differences between the considered low-rank approximation methods, comparison is not straightforward. The following steps and assumptions are made to simplify the process.

- The Nyström method uses a uniform random sampling procedure and the FS-LSSVM uses the Renyi-entropy criterion.
- The number of Renyi sampling iterations for FS-LSSVM is chosen equal to the number of iterations of the TNKF.
- Multiple performance measures are shown for each method, including different truncation parameters and/or samples (S).
- The same hyperparameters are used to compare the methods. These are found by grid searches and iterative tuning of the TNKF.
- For regression, the root-mean-square error (RMSE) and fit percentage based on the normalized root-mean-square error (NMRSE) are used as performance measures, shown in Equations 14 and 15.

$$\text{RMSE} = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{14}$$

$$\text{NMRSE} = 100 \cdot \left(1 - \frac{\|y - \hat{y}\|}{\|y - \text{mean}(y)\|}\right)\% \tag{15}$$

- For classification, the performance is given through the percentage of correctly assigned labels. The confidence (%) is based on the total number of misclassifications by considering the (3σ) confidence bounds as decision models.

In the upcoming subsections, the following conventions are used: All datasets are centered, effectively removing the bias term and first column of the dual problem. This means that only the dual weights are solved for ($\boldsymbol{\alpha}$). The bias term can simply be computed by calculation of the mean and added to the final model if desired. To visualize the performance of the TNKF, for the purpose of clarity, local snapshots of the centered or normalized large-scale problem are used. In all figures, the data points are printed blue, the TNKF mean is printed red, and the confidence bounds in green. In the performance tables, "NA" stands for "not applicable". This is when the algorithm is impractical or infeasible because of computational or memory requirements. The truncation parameters for the TNKF are given below the performance tables. The number of samples for Nyström and FS-LSSVM is given through "# S" after their names.

All experiments were conducted in MATLAB - version R2019b, and performed on a 1aptop with an Intel 6-Core i7 processor running at 2.6 GHz and 16GB of RAM. The Matlab code can be downloaded from https://github.com/mlucatud/LS-SVM_Dual_TTKF. For the FS-LSSVM and Nyström methods the "ls-svmlab" toolbox was used, freely available at https://www.esat.kuleuven.be/sista/lssvmlab/.

A. Noisy Sinc Function

To find an approximation of $\boldsymbol{\alpha}$, the TNKF needs to be initialized. The sinc function is corrupted with noise distributed as $\mathcal{N}(0, 0.1^2)$. It is assumed that this is known, and therefore u is equal to this noise. The initial distribution of $\boldsymbol{\alpha}$ is $\mathcal{N}(\boldsymbol{0}, diag[5...5])$, supplied in TT form, to emphasize little confidence uncertainty in the initial solution. The RBF kernel function is used and hyperparameter values chosen as: ($\gamma = 0.005$, $\sigma^2_{RBF} = 0.005$). Lastly, the output data is sorted in ascending order according to the input values to help achieve small TT-ranks. It should be noted that sorting the data can prohibit implementation of early stopping in the TNKF, as consecutive observations are typically much more local.

The performance of the TNKF is dependent on its ranks, which are affected by the truncations and selected kernel hyperparameters. If both are appropriately specified, the TNKF can obtain near-optimal RMSE and fit values, that are determined by the noise's standard deviation (0.1). The performance is poor when the truncations are specified too large, as expected, proven by TNKF^a.

TABLE VII: Test performance of the approximation methods on the noisy sinc function (RBF, $\gamma = 0.005$, $\sigma^2_{RBF} = 0.005$)

(a) Performance values

Method	**Training RMSE**	**Test RMSE**	**Fit %**
TNKF [a]	0.1994	0.1998	65.1
TNKF [b]	0.1042	0.1044	74.8
FS-LSSVM 20 S	0.2605	0.2600	25.5
FS-LSSVM 100 S	0.0998	0.1019	75.3
FS-LSSVM 500 S	**0.0991**	**0.0994**	**76.0**
Nyström 10 S	0.7877	0.7862	31.6
Nyström 50 S	0.1463	0.1482	60.2
Nyström 250 S	0.1397	0.1402	56.0

(b) Truncation values for the TNKF

	$\epsilon_{\boldsymbol{m}}$	$\epsilon_{\boldsymbol{C}}$	ϵ_{P}	$\epsilon_{\boldsymbol{k}}$	ϵ_{s}	$\epsilon_{\boldsymbol{var}}$
$a.$	0.005	0.1	0.02	0.005	0.1	0.001
$b.$	0	0.003	0.015	0.0075	0	0.015

For this experiment the number of samples does not need to be very large for the FS-LSSVM and Nyström methods. These methods may therefore be preferable over the TNKF in terms of complexity. FS-LSSVM also manages to realize near-optimal results. Like the TNKF, FS-LSSVM is also less sensitive to the chosen hyperparameters. With the Renyi-sampling criterion, this method can select the most informative columns with sampling to acquire a nonlinear mapping function. The Nyström method is generally more sensitive for the hyperparameter values. The selected values are not entirely suitable as they lead to a degree of overfitting. For this reason, the Nyström method has a moderate performance. If the hyperparameters were tuned per method, however, the Nyström method would perform better and be the easiest computationally.

One advantage the TNKF has is that the truncation-parameters allow it to be less sensitive to overfitting. This is espically the case when the kernel variance is chosen small, as is the case here, as the truncations act as generalization measures. Truncations are easiest and most beneficial when the hyperparameters are small because this produces a rapidly decaying singular value spectrum for which low TT-ranks can be achieved in the TT-SVD. However, the selection of truncation and kernel parameters is interdependent. A balance needs to be found between them in order to fit the data well, with reasonable complexities. A corollary of performing truncations, or generalizations, is that the confidence region becomes wider, allowing for more uncertainties in the problem. The regression of TNKF^b exemplifies this, shown in Figure 9. The mean (red) describes the data well, but as can be seen from the peaks of the confidence bounds (green), local variances can still be large. This means that the covariance has not completely converged to zero, due to truncations, and that local confidence can be smaller.

The TNKF performance is also impacted by the specified noise variance w^2 and the initial distribution. If w^2 is chosen differently than its true value the measurements are considered less, or more, informative. This influences the balance between generalization and overfitting, affecting both the width of the confidence bounds and the accuracy of the mean. In the supplied initial distribution, the mean does not impact regression. The first row of the dual problem is a KKT optimality condition ($\sum \alpha_i = 0$) that resets the mean to zero. The initial covariance does impact the regression, but only if significant truncations are performed or early stopping is used in the TNKF, such that local variances (var_j) do not converge to w^2. Therefore the initial covariance influences the width of the covariance bounds. In Figure 10, the initial distributions of $\boldsymbol{\alpha}$ in TNKF^b are set equal to the noise distribution. This results in smoother confidence bounds because the local variance peaks from Figure 9 are iteratively set equal to w^2.

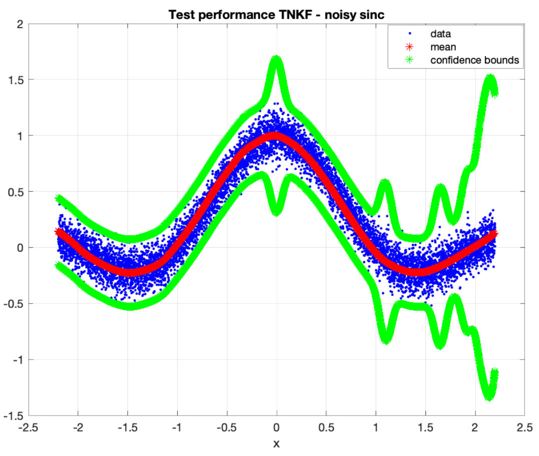

Fig. 9: Regression performance of TNKF^b. Local confidence can differ depending on truncations and initialization, and denotes the accuracy of the approximation.

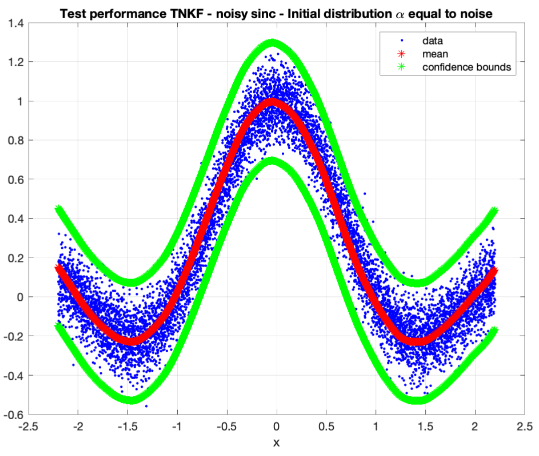

Fig. 10: Regression performance of TNKF^b, but with the initial distribution set to equal to the noise, $u \sim \mathcal{N}(0, 0.1^2)$

B. F-16 Ground Vibration Test

In this paper, the sine-sweep force data is considered as the input signal, containing a frequency range [15-2] Hz with a rate of -0.05 Hz/s. It is generated from a shaker located underneath the wing of an F-16 fighter jet. The acceleration of a dummy payload at the edge of the wing is used as output data. Two separate sine-sweep tests are used, one serving as a training set, the other as a test set. For each, a linear subset is used starting from the 5000-th sample, $x \in$ [5000:5000+2^{14}]. A linear kernel is implemented with regularization ($\gamma = 0.05$), and based on an auto-regressive structure that includes the previous input, no input delays, and the 12 last output values. In the dataset the noise is estimated from the first 1600-samples, giving $u \sim \mathcal{N}(0,\ 3 \cdot 10^{-05})$. The initial distribution of $\boldsymbol{\alpha}$ is $\mathcal{N}(\mathbf{0}, diag[1...1])$. The performances are given in Table VIII.

TABLE VIII: Test performances of the approximation methods on the F16 Ground Vibration Test (Linear, $\gamma = 0.05$)

(a) Performance values

Method	Training RMSE	Validation RMSE	Fit %
TNKF[a]	0.0981	0.0976	90.7
TNKF[b]	0.0327	0.0322	94.3
FS-LSSVM 5 S	0.0190	0.0177	98.2
FS-LSSVM 10 S	0.0142	0.0130	98.7
FS-LSSVM 20 S	**0.0099**	**0.0083**	**99.2**
Nyström 3 S	0.0819	0.0743	86.2
Nyström 5 S	0.0180	0.0141	97.5
Nyström 10 S	0.0141	0.0119	97.9

(b) Truncation values for the TNKF performances

	ϵ_m	r_c	ϵ_P	r_k	ϵ_s	ϵ_{var}
a.	0.001	6	0.025	6	0.005	0.001
b.	0.001	3	0.05	3	0.05	10^{-6}

This dataset is difficult to learn with the TNKF. Even though the frequencies are learned perfectly, which can be accomplished with very low TT-ranks due to the auto-regressive structure, the amplitudes are not. To correctly learn the amplitudes, the TNKF requires large TT-ranks, which are computationally expensive. Therefore truncations had to be performed at a cost of significant descriptive power.

The Nyström and FS-LSSVM methods have the benefit of approximating the kernel with its eigensystem. Due to the similarities, the kernel matrix is easily described in a low-rank form with few eigencomponents. Therefore these methods can applied with very few samples required.

C. Two-spiral Classification Problem

A large-scale version of the two-spiral problem is considered, a well-known and difficult machine learning problem. The task is to classify two identical, but phase shifted, spirals from each other. The data is separable, but a highly nonlinear decision boundary has to be found [7]. No noise is considered to act on the data, but has to be chosen greater than zero due to the Kalman filter assumptions. It is therefore initialized with distribution $u \sim \mathcal{N}(0, 10^{-6})$. The RBF kernel is used, and the hyperparameters are: ($\gamma = 0.0045$, $\sigma^2_{RBF} = 0.005$). The initial distribution for the weights is chosen as, $\boldsymbol{\alpha} \sim \mathcal{N}(\mathbf{0}, diag[1...1])$. The data is sorted according to the labels, which allows for very small ranks, even TT-rank-1 structures.

TABLE IX: Test performance of the approximation methods on the large-scale two-spiral problem (RBF, $\gamma = 0.0045$, $\sigma^2_{RBF} = 0.005$)

(a) Performance values

Method	Correctly Labeled %	Confidence %
TNKF[a]	100	100
TNKF[b]	**100**	**100**
FS-LSSVM 50 S	0.9	NA
FS-LSSVM 500 S	9.4	NA
FS-LSSVM 800 S	15.25	NA
Nyström	NA	-

(b) Truncation values for the TNKF

	ϵ_m/r_m	ϵ_c/r_c	ϵ_P	ϵ_k/r_k	ϵ_s	ϵ_{var}
a.	0.001	0.005	0.01	0.005	0.005	0.01
b.	r_m=1	r_c=1	0.001	r_k=1	0	0.05

The TNKF manages to distinguish the two spirals perfectly, as displayed in Figure 11. Additionally, confidence bounds of the decision function are also found. An example of decision confidence bonds is shown for a small scale problem with sorted data, Figure 12. Because of the small TT-ranks the TNKF only requires about 20 minutes for the classification, and does this without the explicit storage of a matrix. The Nyström method could not be used, because the memory complexities needed to construct the matrices and solve for $\boldsymbol{\alpha}$ exceeded the RAM capabilities. Computing a nonlinear mapping function in the FS-LSSVM is also difficult for this particular problem, due to the many turns. Many support vectors have to be used in order to estimate an accurate function, but the number quickly makes the method infeasible, approximately the case when $S >$1000. This problem demonstrates that the TNKF is especially suitable in cases where the Nyström and FS-LSSVM methods can't be used or perform poorly.

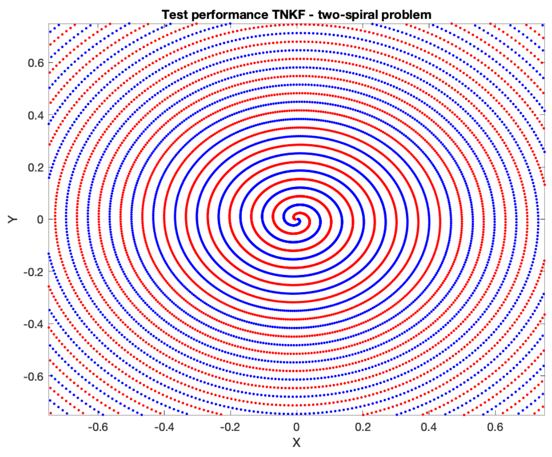

Fig. 11: The two classified spirals classified with the TNKF[a]. A sub-domain is shown for clarity.

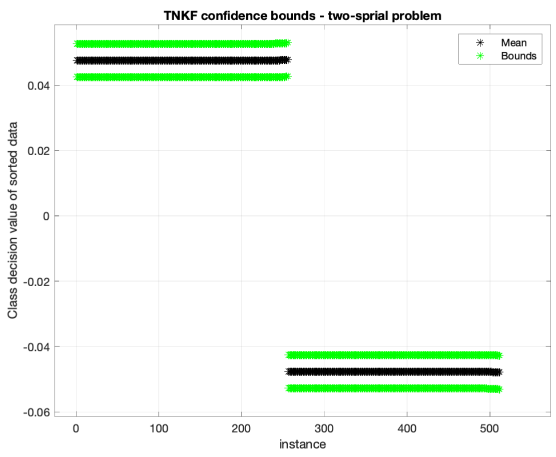

Fig. 12: Confidence bounds for a two-spiral decision function. A small-scale example is used for clarity.

D. Adult Dataset

The classification problem is to predict whether a person makes over $50k a year. Fourteen features, numeric and categorical, are available. In this paper, all variables are utilized. Categorical variables are converted to numeric by use of dummy variables, therefore the total number of features artificially increases to 99. The RBF kernel is used, and the chosen hyperparameters are: ($\gamma = 0.0015$, $\sigma^2_{RBF} = 0.5$). TNKF^a and TNKF^b are trained on data sorted according to their labels. TNKF^c is trained on unsorted data to demonstrate early stopping, which needs observations of both classes before termination. The TNKFs are implemented with initial distributions, $\boldsymbol{\alpha} \sim \mathcal{N}(\mathbf{0}, diag[1...1])$ It is assumed that the noise is irrelevant to the consensus data, therefore it is chosen very small $u \sim \mathcal{N}(0, 10^{-6})$.

TABLE X: Test performance of the approximation methods on the Adult dataset ($\gamma = 0.0015$, $\sigma^2_{RBF} = 0.5$)

(a) Performance values

Method	**Correctly Labeled %**	**Confidence %**
TNKF^a	83	98.2
TNKF^b	**84.5**	**99.2**
TNKF^c	81.4	96
FS-LSSVM 50 S	44.3	NA
FS-LSSVM 500 S	73.2	NA
FS-LSSVM 800 S	76.2	NA
Nyström 5 S	84.4	NA
Nyström 25 S	84.4	NA
Nyström 100 S	84.4	NA

(b) Truncation values for the TNKF

	$\epsilon_{\boldsymbol{m}}$	$\epsilon_{\boldsymbol{c}}$ / $r_{\boldsymbol{c}}$	ϵ_P	$r_{\boldsymbol{k}}, \epsilon_{\boldsymbol{k}}$	ϵ_s	$\epsilon_{\boldsymbol{var}}$
$a.$	0.001	r_c=6	0.005	0.005	0.001	0.001
$b.$	0.001	ϵ_c=0.01	0.003	0.003	0.001	0.001
$c.$	0.001	r_c=30	0.001	r_k=6	0.001	0.001

The TNKF is capable of achieving performances equal to that found in the literature for SVM's and LS-SVMs [7] [34]. The sorted data adheres to small TT-ranks, therefore severe truncations are unnecessary. TNKF^a and TNKF^b show this. TNKF^c, which employed early stopping and a forgetting factor, managed to reach decent labeling power. For the chosen termination conditions and forgetting factor, described in Appendix A, only 2200 rows had to be iterated over. This reduced training time by around 90%, but yields a lower classification accuracy. The chosen kernel hyperparameter (σ^2_{RBF}) generates a kernel matrix that can easily be approximated with a low-rank matrix structure. The Nyström method is particularly suitable for this, and achieves very high performance values with few sampled columns. The nonlinear mapping function, however, is difficult to approximate. FS-LSSVM requires many support vectors and is computationally the most demanding in this problem.

VII. Conclusion

This paper presents a recursive Bayesian learning framework with the tensor network Kalman filter to solve large-scale LS-SVM dual problems. To the best of our knowledge, it is the first non-sampling based algorithm for large-scale LS-SVM dual problems. The recursive Bayesian framework allows incorporation of prior knowledge, measurement noise, early stopping and yields confidence bounds on the predicted dual functions. Current low-rank approximation methods lack such properties. Also, the curse of dimensionality is avoided in this work because the method is iterative and has a tensor train representation. The TNKF is especially advantageous in situations with high nonlinearities, when the kernel data conforms to a low-rank TT structure, or when many samples are required for alternative methods. It attains high accuracies on all considered problems and matches performances of other low-rank approximation and kernel methods.

A. Future Work

The TNKF computational complexities can be further reduced. The algorithm can easily be extended to a parallel implementation and a batch framework to reduce training times. Additionally, directly constructing the dual rows in TT form can save a significant portion of the computational complexity for extremely large problems and problems with many features. A big obstacle in the use of the TNKF is the selection of the truncation parameters and kernel hyperparameters, which are interdependent. Inference methods can be developed to determine these parameters and would save a lot of tuning work, currently done by exhaustive grid searches. Lastly, the TNKF can be readily adopted to other linear kernel methods and fields involved with solving large-scale linear systems.

Appendix A
Early stopping Adult dataset

The forgetting factor and early stopping conditions implemented for TNKF^a are described below. These were designed arbitrarily and in consideration of the classification performance. Termination occurred when all stopping conditions were satisfied.

- Forgetting factor $\lambda = 0.9975$
- Norm of $TT(P)$ had to be smaller than $1 \cdot 10^{-20}$
- The change in $TT(P)$-norm had to at least -$3 \cdot 10^{-5}$ for at least 2 iterations.
- The KKT condition (for classification) has to be smaller than one, $\left(\sum_{k=1}^{N} \alpha_k y_k\right) < 1$

Acknowledgment

The authors would like to thank...

References

[1] V. Vapnik, *The Nature of Statistical Learning Theory*. Berlin, Heidelberg: Springer-Verlag, 1995.
[2] B. Scholkopf and A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.
[3] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
[4] J. Kocijan, *Modelling and Control of Dynamic Systems Using Gaussian Process Models*. Springer International Publishing, January 2016.
[5] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. The MIT Press, 2012.
[6] J. Rojo-Alvarez, M. Martinez-Ramon, J. Munoz-Mari, and G. Camps-Valls, *Digital Signal Processing with Kernel Methods*, 1st ed. Wiley-IEEE Press, 2018.
[7] J. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle, *Least Squares Support Vector Machines*. World Scientific, 2002.
[8] J. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, pp. 293–300, June 1999.
[9] B. Hamers, "Kernel Models for Large Scale Applications," Ph.D. dissertation, Katholieke Universiteit Leuven, ESAT, Katholieke Universiteit Leuven, Belgium, Kasteelpark Arenberg 10, 3001 Leuven, June 2004.
[10] G. Golub and C. Van Loan, *Matrix Computations*, 4th ed. The Johns Hopkins University Press, 2013.
[11] C. Williams and M. Seeger, "Using the Nyström Method to Speed Up Kernel Machines," in *Advances in Neural Information Processing Systems 13*. MIT Press, 2001, pp. 682–688.
[12] K. De Brabanter, J. De Brabanter, J. A. K. Suykens, and B. De Moor, "Optimized Fixed-size Kernel Models for Large Data Sets," *Comput. Stat. Data Anal.*, vol. 54, no. 6, pp. 1484–1504, June 2010.
[13] S. Ma and M. Belkin, "Kernel machines that adapt to gpus for effective large batch training," 2018.
[14] A. Smola and B. Schölkopf, "Bayesian kernel methods," in *Advanced Lectures on Machine Learning*, January 2002, pp. 65–117.
[15] S. Särkkä, *Bayesian Filtering and Smoothing*, ser. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2013.
[16] M. Verhaegen and V. Verdult, *Filtering and System Identification: A Least Squares Approach*. Cambridge University Press, 2007.
[17] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, ser. Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press, January 2006.
[18] B. Hamers, "A Comparison of Iterative Methods for Least Squares Support Vector Machine Classifiers," Katholieke Universiteit Leuven, Belgium, Tech. Rep., 2001.
[19] A. Smola and B. Schökopf, "Sparse greedy matrix approximation for machine learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ser. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, p. 911–918.
[20] A. Rudi, D. Calandriello, L. Carratino, and L. Rosasco, "On fast leverage score sampling and optimal learning," 2018.
[21] P. Drineas and M. Mahoney, "On the nyström method for approximating a gram matrix for improved kernel-based learning," *Journal of Machine Learning Research*, vol. 6, 2005.
[22] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. PHAN, "Tensor Decompositions for Signal Processing Applications: From Two-way to Multiway Component Analysis," *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 145–163, March 2015.
[23] A. Avella and F. Mancini, *Strongly Correlated Systems: Numerical Methods*. Springer Berlin Heidelberg, January 2013.
[24] T. Kolda and B. Bader, "Tensor Decompositions and Applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, September 2009.
[25] K. Batselier, Z. Chen, and N. Wong, "Tensor network alternating linear scheme for mimo volterra system identification," *Automatica*, vol. 84, July 2016.
[26] I. Oseledets, "Tensor-Train Decomposition," *SIAM J. Scientific Computing*, vol. 33, pp. 2295–2317, January 2011.
[27] I. Oseledets, "Approximation of $2^d \times 2^d$ Matrices Using Tensor Decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, January 2010.
[28] A. Cichocki, "Tensor networks for big data analytics and large-scale optimization problems," *CoRR*, vol. abs/1407.3124, 2014.
[29] S. Haykin, *Kalman Filtering and Neural Networks*. USA: John Wiley & Sons, Inc., 2001.
[30] K. Batselier, Z. Chen, and N. Wong, "A Tensor Network Kalman filter with an application in recursive MIMO Volterra system identification," *Automatica*, vol. 84, October 2016.
[31] D. Gedon, P. Piscaer, K. Batselier, C. Smith, and M. Verhaegen, "Tensor Network Kalman Filter for LTI Systems," in *2019 27th European Signal Processing Conference (EUSIPCO)*, September 2019, pp. 1–5.
[32] J. Noël and M. Schoukens, "F-16 aircraft benchmark based on ground vibration test data," 2017 *Workshop on Nonlinear System Identification Benchmarks*, pp. 19-23, Brussels, Belgium, April 24-26, 2017.
[33] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[34] A. Pal and S. Pal, *Pattern Recognition and Big Data*. World Scientific, 2017.

PLACE
PHOTO
HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.

Chapter 4

Discussion

In this Chapter, the results and properties of the tensor network Kalman filter (TNKF) are further discussed. In Section 4-1, the results and contribution of the thesis work are covered. Additionally, some elaboration on the TNKF properties is given too. Thereafter, in Section 4-2, improvements on how the TNKF was implemented in this thesis are presented.

4-1 Results and Properties of the Tensor Network Kalman Filter

In this section the thesis work and the TNKF discussed in more depth. First the outcomes of the thesis are presented, then progressively more in depth material of the TNKF is covered, such as tuning and filter stability.

4-1-1 Avoiding the Curse of Dimensionality

The primary goal of this thesis was to develop a low-rank approximation method that avoids the *curse of dimensionality* associated with solving least-squares support-vector-machine (LS-SVM) dual problems. Current low-rank approximation methods suffer from burdensome computational and memory scalings due to which they can become impractical or infeasible to apply for large-scale applications. The developed TNKF can circumvent the prohibitive scalings if the data set adheres to small tensor train (TT)-ranks. The main objective of the thesis has therefore been realized for low TT-rank data sets. Because the TNKF complexity is data-dependent through the TT-ranks $\mathcal{O}(ldnr^6)$, its application is really dependent on the data. For example, for the F16 benchmark data set the TNKF required very large ranks making it computationally impractical, yet for the two-spiral problem TT-rank-one structures could be used with very little accuracy-loss. The F16 sine-sweep data is very uncorrelated and requires large ranks for the TNKF variables, which had to be truncated for computational feasibility. In such cases the TNKF can be a poor choice, and is solved much more easily with other low-rank approximation methods. On the other hand, if the TT-ranks can be small, the TNKF can be far more advantageous than the alternative methods. Frequently, low TT-ranks are possible when the data is very correlated, these have a rapid decay in the singular value spectrum for which the truncation-error is generally much smaller. The large-scale two-spiral problem is the ideal example where the *curse of dimensionality* is avoided by the TNKF, in which the FS-LSSVM and the Nyström are not even applicable due to their required computational and memory costs.

4-1-2 Including User-Knowledge

The second goal of this thesis was to develop a low-rank approximation method that can include user-knowledge of the application, such as measurement noise, an initial solution, and confidence terms. In current low-rank approximation methods specification of user-knowledge is not possible, additionally, they do not yield confidence bounds on the obtained solution. The TNKF has a distinct advantage due its Bayesian formulation, due to which user-knowledge can be incorporated and confidence bounds on the solution are found. Thereby, the second objective of thesis is also realized. For many problems where noise corrupts the data, it can be advantageous to explicitly specify a noise variable, such as for the noisy sinc problem, in which it effectively acts as a generalization measure to prevent overfitting. An initial distribution on the dual weights can be used to include prior knowledge of a dual solution, useful for when new data points become available for example. The Bayesian formulation is also advantageous, especially with regards to other low-rank approximation methods, because deterministic confidence bounds are obtained. Other low-rank approximation methods use sampling-procedures, and can only give statistical sampling-based error estimates of the approximation. The TNKF evaluates all kernel data points to obtain a solution, due to which it obtains deterministic confidence bounds that include the effects of performed TT-rank truncations, hyperparameter choices, and early stopping. In other words, the TNKF approximates a solution to the dual problem, and can inform the user of how informative the approximation is (locally and globally), unlike other low-rank approximation methods.

4-1-3 Applicability of the Tensor Network Kalman Filter

The TNKF is particularly suitable in applications in which other, or current, low-rank approximation methods do not work well. These applications are focused towards large-scale and highly nonlinear problems, distinguishing itself from the state-of-the-art low-rank approximation methods. The two-spiral problem is one example is where the TNKF demonstrates its capabilities, in which a large-scale and highly nonlinear problem is learned with ease, avoiding the *curse of dimensionality* that makes the alternative low-rank methods infeasible. Due to the tensor network representation much larger data sets can be employed for the approximation, especially if the data sets conform to low TT-ranks. For highly nonlinear problems, large subsets are usually needed to be informative enough to capture all the behavior of the underlying application. Sampling-based methods are then disadvantageous as the complexities become too burdensome, however the TNKF has no such issues due to its compressed and iterative formulation, and because it evaluates all data points to construct the most descriptive TT-structures. Another situation where the TNKF is desirable is if user-knowledge wants to be specified and/or confidence bounds need be found. The recursive Bayesian framework makes the TNKF a particularly attractive low-rank approximation methods when dealing with uncertainties in the data points.

There are also scenarios where the TNKF can be disadvantageous to apply. The first situation is when the data set is small. The TT format is only beneficial for a data set size beyond some threshold. Other low-rank approximation methods are likely to be less computationally and memory costly up to that threshold. The reason for this is predominantly because the TNKF algorithm needs to perform additional computations to cast the data into a TT format, which is only efficient when the data set is large enough. The threshold size is difficult to determine as it problem-specific, and also depends on the performance of other low-rank approximation methods. Another situation where the TNKF is less applicable is when the kernel matrix is low-rank or has a rapidly decaying eigenvalue spectrum. Low-rank rank approximation methods, as the name suggest, attempt to estimate a matrix with a low(er) rank matrix. If few ranks are required for the estimation, few samples are typically needed, and the reconstruction error is usually small. For example, in the Adult data set problem, the Nyström method only needs five samples to approximate the eigenspace accurately, as the designed kernel matrix has a rapidly decaying eigenvalue spectrum that can be approximated by the first five eigencomponents. For such cases, it is preferable to implement a low-rank approximation method rather than the TNKF which are then computationally and memory-wise much cheaper. Many "real" data sets conform to a low-rank kernel matrix approximation, therefore it is advisable to first attempt a simpler low-rank approximation method if possible, instead of directly applying the TNKF. Lastly, the TNKF requires much more work to implement from scratch than its alternatives. Therefore it is again advisable, unless a software packet becomes available, to first try using the simpler alternative low-rank methods.

4-1-4 Accuracy-Complexity Trade-Off

A major advantage of the TNKF is that the computational and memory requirements can be directly controlled with the TT-rank truncations. The TT-truncations, which influence the size of the TT-ranks, are the most influential factor in the computational and memory scaling of the TNKF. With the truncations, the data is approximated by the best fitting low-rank TT structure. All TT variables can in principle be truncated, however, the TT-ranks that impact scaling the most are of the variables $TT(\boldsymbol{c}_l)$, $TT(P_l)$, and $TT(\boldsymbol{k}_l)$. The TT-ranks of these variables either quickly grow burdensome over the iterations, or can require very large ranks to describe a slowly decaying singular value spectra. Because all the kernel data is evaluated in the TNKF, a deterministic trade-off between the complexities and accuracy exists, which is governed by the TT-ranks. Another possibility to influence the complexity-accuracy trade-off is to limit the total computational cost of the TNKF by implementing early stopping. The per iteration cost remains the same, but because only a fraction of the rows are iterated over, the total cost goes down. For the adult data set early stopping is demonstrated, in which only 10% of the rows are iterated over to generate a decent approximation, saving roughly 90% of work for a accuracy decrease of about 3%. On the contrary, the complexities of current state-of-the-art methods are dominated by how large the sampled subsets are. Due to the sampling dependency, the accuracy-complexity trade-off for these methods are probabilistic. Therefore, current low-rank approximation methods can only impact the accuracy-complexity trade-off indirectly, by specifying how many samples are employed. The TNKF is a breakthrough for low-rank approximation methods, as it eliminates the need of a probabilistic sampling procedure, and allows for direct influence of the accuracy-complexity trade-off through the TT-ranks.

One drawback, inherent to all kernel methods, is that kernel data needs to be constructed. The computation of a kernel matrix is intensive, whether it is done apriori or only the rows are constructed during the iterations. For the TNKF, construction of the kernel rows, which are used to construct the dual problem rows, gives a computational lower bound of $\mathcal{O}(N^2)$, or equivalently $\mathcal{O}(n^{2d})$. For problems with many features or extremely large data points, kernel row construction is commonly a big source of computational cost. An advantage of the TNKF however is that due to its iterative nature, if early stopping is employed, not all of the kernel rows have to be constructed. Other low-rank approximation methods typically have to construct an entire kernel matrix (or matrices) based on the subset. Depending on the specific problem, the TNKF memory complexity can be smaller or greater than working with explicit subset kernel matrices, but computationally, significant advantages are achieved by only using the kernel rows.

4-1-5 Tuning of the Hyperparameters and TT-ranks

It is impossible to know suitable truncation- and hyper-parameters for the TNKF and dual problem matrix. The truncation- and hyper-parameters are closely related, and cooperatively determine the accuracy of the low-rank approximation. Here it is assumed the RBF kernel function is used, therefore the hyperparameters (γ, σ^2) directly determine the singular value decay of the dual problem matrix. The singular value spectra are important for the construction and rounding of the TTs. More specifically, the rate at which the spectrum decays has direct influence over the ranks, as the TT-SVD and TT-rounding truncate the the singular values only after some specified norm or (ordered) index. The TT-ranks are determined by the truncations parameters of the singular value spectra, which usually can not be left untruncated due to memory and computational complexities. Therefore, it is nearly always the case that a balance needs to be found between the hyperparameters and the truncation parameters to influence the accuracy-complexity trade-off. Tuning can therefore be a strenuous task, especially for large data sets. Currently, tuning is performed iterative grid searches, as no tuning method exists yet that takes into account the duality of the hyperparameters and truncation parameters. A common observation is that a rapidly declining spectrum is desirable for the TNKF, as this makes it easier to construct low-rank TT variables. TNKF truncations can, to a degree, compensate overfitting (generalization) caused by small hyperparameters. However, care needs to be taken in such an approach, because too much truncation can lead to instability of the TNKF. Truncations are generally most easily performed by based on the respective norm as it keeps the TT-ranks more flexible. Also, some variables have more influence in the accuracy-complexity trade-off than others. Most important is the truncation of the dual row $TT(\boldsymbol{c}_l)$, covariance TT $TT(P_l)$, and Kalman gain $TT(\boldsymbol{k}_l)$, since these variables are used in many steps of the filter, and tend to dominate the complexities and a significant portion of the accuracy.

4-1-6 Stability of the Tensor Network Kalman Filter

The stability of the Kalman filter is a difficult matter. First of all, the iterative state space models are time-varying aspect due to the row-dependent nature of the output equation ($\boldsymbol{c}_l$), making analysis complex. Secondly, the tensor train form does not allow for easy analysis of the eigenvalues, which would have to be found with expensive optimization procedures.

The only pragmatic way of checking stability is to analyze the covariance. Because the dual problem matrix is positive semi-definite it is known that a global solution exists, a result of convexity. If convergence is attained after training, the norm of the covariance matrix should approximately be zero or at a global minimum (depending on truncations/early stopping/noise). In practical terms, after witnessing all the rows, the uncertainty (covariance) in the solution updates should decrease to zero or a minimum. Additionally, during the iterations the norm should monotonically decrease as more of the data is witnessed, a consequence of orthogonality in Kalman gain updates (dual matrix rows are independent) and positive semi-definiteness of the covariance. These conditions can be implemented to verify the stability whilst using the algorithm. Fortunately, the covariance norm is easily computed or logged in TT form during the iterations because of the singular value decomposition (SVD) in the TT-SVD algorithm, or alternatively by computation of the trace. An analogous example for the control community would be to consider the covariance as a Lyapunov function/matrix. Note that the analysis of stability is dependent on the designed iterative models, therefore can be different for other update equations. It is interesting to note, that if convergence is attained, that covariance matrix is a scaled copy of the kernel matrix. The reason for this phenomena is still an open question, but it is believed that this is derived from reproducing kernel Hilbert space property. Another issue of stability is how truncations affect the properties of the covariance.

Truncations of the TT-rank heavily influence the stability of the Kalman filter. There is no way to guarantee positive definiteness and symmetry for the covariance TT for example, unless it has a TT-rank-one structure. However, if a TT-rank-one structure is chosen, usually too much information is lost and poor dual solutions are yielded. Experience informs that the best way to truncate the covariance TT is with norm-based truncations, which are more flexible for TT-rank sizes even though no stability guarantees can be given. Similarly for other variables, if truncations are performed too aggressively, the Kalman filter becomes unstable due to the which the covariance diverges. The measurement noise also affects the stability in the aspect of convergence of the covariance TT, the noise measurement noise weighs how much confidence can be placed in the observations. If the noise is specified too large, the Kalman filter will updates will be insignificant and cause underfitting to occur. On the other hand, if the noise is specified to small, overfitting can occur.

4-2 Improvements

There are also a numerous areas in which this thesis work an be improved. The complexity of the implemented TNKF can be further reduced. Also, the particular TNKF formulation can be modified to improve the method's performance.

4-2-1 Computational Improvements

Computationally, there are numerous of improvements that can be made. First of all, it is vital to mention that the ranks of the TTs play a crucial role in determining feasibility of the TNKF. As witnessed for the F16 nonlinear benchmark data set, if the ranks need to be too large, accuracy-loss has to be incurred to make the problem feasible. Ordering the rows can be a smart "work-around", to create low-rank TT structures but in some cases this might

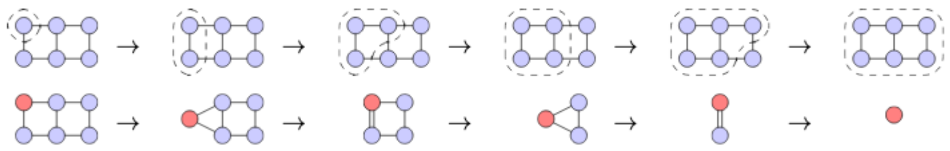

Figure 4-1: "Zip-lock" contraction procedure between two TT vectors. The contractions are performed sequentially leading to a lower computational complexity. [6].

not be possible, such as for early stopping. To create a more favorable trade-off between the accuracy and complexity, the computational complexity of the algorithm and routines should be investigated. Multiple approaches can be taken to decrease the computational complexity. The construction of the kernel, or dual problem row, can be attempted to be done directly in TT form, as is discussed in future work. The frequency at which the TT-rounding algorithm is called can be decreased, as right now it is implemented after nearly each TT-contraction. TT-rounding is a costly routine to run, but often necessary to keep the TT-ranks small, thus a balance needs to be found in how often it is implemented. The TT-contraction procedure can be adjusted to either a "zip-lock" CPU or parallel GPU implementation. Currently, the contractions are performed non-optimally. Fortunately, TT-contractions appear to have only a minor cost with respect to the TT-rounding and the construction of $TT(\boldsymbol{c}_l$. Nevertheless, by modifying the contraction procedure, the time for contractions itself can be optimized. In [11], it was reported that a parallel implementation can be an order of magnitude faster. On the other hand, if a CPU implementation is wanted, the "zip-lock" contraction scheme as shown in Figure 4-1 is more advantageous.

The "zip-lock" procedure effectively reduces the operations from matrix-matrix multiplications to matrix-vector multiplications, thus has a lower complexity. Lastly, there is also the matter of batch size. In this thesis, individual rows were iterated over, which is probably not optimal in terms of runtime. As in many learning methods, it is common to learn in batches. Here that would translate to using multiple rows in one iteration, and could save time in steps such as kernel construction, or the classification/regression algorithms. Also, in batch learning less iterations are then needed meaning that the number of implementations of common routines, such as contractions and the TT-rounding, is much lower. On the downside, a matrix or TT inverse would have to be computed for the TNKF which can be computationally difficult for large batch sizes.

4-2-2 Improvements in the Formulation

There are also some modifications that can be done in the formulation of the TNKF. The iterative state space models can be redesigned for example. In this work, the formulation was based on the thought/assumption that the previous solution makes the best estimate of the next solution. One simple idea could be to take a weighted average of a number previous solutions, or to choose one of the previous estimates corresponding to the biggest covariance decrease. Another point that can be addressed is the early stopping criteria, which were chosen to be dependent on the covariance norm and the sum of errors Karush-Kuhn-Tucker (KKT) condition in the paper, but can actually be designed arbitrarily. One idea could be to include the prediction error v_l in the list of criteria. Early stopping is especially attractive if one knows that a threshold test performance value has been achieved. Therefore it could also be an idea to validate the test performance every i iterations with the $l - th$ solution. Lastly, there is also the topic of sparsity of the dual solution, which is lost in the least-squares formulation of the support-vector-machine. It can be investigated whether TT-rounding of the final approximation, $TT(\boldsymbol{m}_l), TT(P_l)$, can lead to sparser or simplified models. By application of the TT-Rounding algorithm, the most important information is found with the remainder truncated, which could improve generalization performance for test data sets.

Chapter 5

Conclusion and Future Work

Conclusion

In this thesis, a recursive Bayesian learning framework with the tensor network Kalman filter was presented for large-scale LS-SVM dual problems. The framework is flexible, can easily be modified, and is readily adoptable to other linear kernel methods. The TNKF performs competitively with current low-rank approximation methods as was tested on two regression and two classification problems, on which it achieves high accuracies. The method is especially useful in situations where the alternative methods perform inadequately due to to high-nonlinearities, require many samples and become too burdensome to compute, and/or when the data conforms to a low-rank tensor train structure. Due to the Bayesian and tensor network formulation, the TNKF generates confidence bounds, circumvents the *curse of dimensionality*, and allows incorporation of prior knowledge, measurement noise, and early stopping.

Future Work

One big obstacle of implementing large-scale kernel methods is finding suitable hyperparameters. Current methods for hyperparameter-tuning scale poorly with the size of the data set, usually completed with some cross-validation scheme or iterative grid searches, which are very expensive [31] [35]. For this thesis, iterative tuning was used to find the hyperparameters and truncation parameters, a lot of tedious work. A useful development for future implementations would be extending the framework to include some hyperparameter selection method. By taking into account the tensor train (TT) structure the respective truncation parameters, the poor computational and memory complexities can be reduced for a hyperparameter search or tuning procedure. Additionally, the interdependence between the truncation and hyperparameters can be investigated further to see how tuning or parameter selection can be done more effectively.

The lower bound of the computational complexity is determined by the construction of the kernel row elements, costing at least $\mathcal{O}(N^2)$. All kernel methods suffer from the need to construct a kernel row or matrix, which is very costly. For the TNKF, it can be researched if kernel or dual row construction can be done directly in TT form, eliminating the need recast a kernel row to a TT form.

The TNKF can be also be expanded and implemented for other linear kernel methods, or even as a general linear system solver. In this thesis, the application was limited to the LS-SVM dual problem. However, there are close connections to other methods. These similarities can be exploited to expand the TNKF framework and its applicability. Because of its TT formulation, it can be very useful for solving large-scale data sets that do need not be represented explicitly. The TNKF has already been applied to solve a Volterra series [4] [5] and used for linear-time-invariant state space filtering [12].

Bibliography

[1] G. Allaire and S.M. Kaber. *Numerical Linear Algebra*, volume 55. Springer, 1 edition, 2008.

[2] A. Astolfi, D. Karagiannis, and R. Ortega. *Nonlinear and Adaptive Control with Applications*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[3] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[4] K. Batselier, Z. Chen, and N. Wong. A Tensor Network Kalman filter with an application in recursive MIMO Volterra system identification. *Automatica*, 84, October 2016.

[5] K. Batselier and N. Wong. Matrix output extension of the tensor network Kalman filter with an application in MIMO Volterra system identification. *Automatica*, 95:413 – 418, 2018.

[6] J.C. Bridgeman and C.T. Chubb. Hand-waving and interpretive dance: an introductory course on tensor networks. *Journal of Physics A: Mathematical and Theoretical*, 50(22):223001, 2017.

[7] A. Cichocki. Era of Big Data Processing: A New Approach via Tensor Networks and Tensor Decompositions. *CoRR*, abs/1403.2048, 2014.

[8] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. PHAN. Tensor Decompositions for Signal Processing Applications: From Two-way to Multiway Component Analysis. *IEEE Signal Processing Magazine*, 32(2):145–163, March 2015.

[9] D. Comminiello and J. Principe. *Adaptive Learning Methods for Nonlinear System Modeling*. Butterworth-Heinemann, Elsevier, June 2018.

[10] K. De Brabanter, J. De Brabanter, J. A. K. Suykens, and B. De Moor. Optimized Fixed-size Kernel Models for Large Data Sets. *Comput. Stat. Data Anal.*, 54(6):1484–1504, June 2010.

[11] S. Efthymiou, J. Hidary, and S. Leichenauer. Tensornetwork for machine learning. *CoRR*, abs/1906.06329, 2019.

[12] D. Gedon, P. Piscaer, K. Batselier, C. Smith, and M. Verhaegen. Tensor Network Kalman Filter for LTI Systems. In *2019 27th European Signal Processing Conference (EUSIPCO)*, pages 1–5, September 2019.

[13] G.H. Golub and C.F Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 4 edition, 2013.

[14] B. Hamers. *Kernel Models for Large Scale Applications*. PhD thesis, Katholieke Universiteit Leuven, ESAT, Katholieke Universiteit Leuven, Belgium, Kasteelpark Arenberg 10, 3001 Leuven, June 2004.

[15] S.S. Haykin. *Kalman Filtering and Neural Networks*. John Wiley & Sons, Inc., USA, 2001.

[16] T.G Kolda and B.W Bader. Tensor Decompositions and Applications. *SIAM Review*, 51(3):455–500, September 2009.

[17] S. Kumar, M. Mohri, and A. Talwalkar. Sampling Methods for the Nyström Method. *Journal of Machine Learning Research*, 13(1):981–1006, April 2012.

[18] C. Li, S. Jegelka, and S. Sra. Fast DPP Sampling for Nyström with Application to Kernel Methods. *CoRR*, abs/1603.06052, 2016.

[19] W. Liu, J. C. Principe, and S. Haykin. *Kernel Adaptive Filtering: A Comprehensive Introduction*. Wiley Publishing, 1st edition, 2010.

[20] R. Mall and J. A. K. Suykens. Very Sparse LSSVM Reductions for Large-Scale Data. *IEEE Transactions on Neural Networks and Learning Systems*, 26(5):1086–1097, May 2015.

[21] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

[22] M.N. Murty and V.S. Devi. *Bayes Classifier*, pages 86–102. Springer London, London, 2011.

[23] C. Musco and C. Musco. Provably Useful Kernel Matrix Approximation in Linear Time. *CoRR*, abs/1605.07583, 2016.

[24] I. Oseledets. Approximation of $2^d \times 2^d$ Matrices Using Tensor Decomposition. *SIAM Journal on Matrix Analysis and Applications*, 31, January 2010.

[25] I. Oseledets. Tensor-Train Decomposition. *SIAM J. Scientific Computing*, 33:2295–2317, January 2011.

[26] CE. Rasmussen and CKI. Williams. *Gaussian Processes for Machine Learning.* Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, January 2006.

[27] W.J. Rugh. *Nonlinear system theory.* Johns Hopkins University Press Baltimore, MD, 1981.

[28] J. Schoukens and L. Ljung. Nonlinear system identification: A user-oriented roadmap. *CoRR*, abs/1902.00683, 2019.

[29] S. Srkk. *Bayesian Filtering and Smoothing.* Cambridge University Press, New York, NY, USA, 2013.

[30] S. Sun, J. Zhao, and J. Zhu. A review of Nyström methods for large-scale machine learning. *Information Fusion*, 26:36 – 48, 2015.

[31] J. Suykens, J. De Brabanter, K. De Brabanter, K. Pelckmans, et al. LS-SVMlab. https://www.esat.kuleuven.be/sista/lssvmlab/.

[32] J. Suykens, J. De Brabanter, L. Lukas, and J. Vandewalle. Weighted least squares support vector machines: robustness and sparse approximation. *Neurocomputing*, 48:85–105, 10 2002.

[33] J. Suykens, L. Lukas, P. Van Dooren, B. De Moor, and J. Vandewalle. Least Squares Support Vector Machine Classifiers: a Large Scale Algorithm. *Euro Conf Circ Theory Design (ECCTD'99)*, June 2000.

[34] J. Suykens, J. Vandewalle, and B. De Moor. Optimal control by least squares support vector machines. *Neural networks : the official journal of the International Neural Network Society*, 14:23–35, February 2001.

[35] J.A.K Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines.* World Scientific, 2002.

[36] H.A van der Vorst. Efficient and reliable iterative methods for linear systems. *Journal of Computational and Applied Mathematics*, 149(1):251 – 265, 2002. Scientific and Engineering Computations for the 21st Century - Methodologies and Applications Proceedings of the 15th Toyota Conference.

[37] T. Van Gestel, J. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. De Moor, and J. Vandewalle. Benchmarking least squares support vector machine classifiers. *Machine Learning*, 54, June 2002.

[38] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, Berlin, Heidelberg, 1995.

[39] M. Verhaegen and V. Verdult. *Filtering and System Identification: A Least Squares Approach.* Cambridge University Press, 2007.

[40] H. Wang and D. Hu. Comparison of SVM and LS-SVM for Regression. In *2005 International Conference on Neural Networks and Brain*, volume 1, pages 279–283, October 2005.

[41] H. Wendland. *Iterative Methods for Solving Linear Systems*, page 101–131. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2017.

[42] K. Zhang, I. W. Tsang, and J. T. Kwok. Improved Nyström Low-rank Approximation and Error Analysis. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1232–1239, New York, NY, USA, 2008. ACM.

Glossary

List of Acronyms

FS-LSSVM	Fixed size least-squares support-vector-machines
KKT	Karush-Kuhn-Tucker
LS-SVM	Least-squares support-vector-machines
QP	Quadratic programming
RBF	Radial-basis-function
SVM	support-vector-machines
SVD	Singular value decomposition
TT	Tensor train
TNKF	Tensor Network Kalman Filter

List of Symbols

Abbreviations

$\boldsymbol{\alpha}$	Dual weights
$\boldsymbol{m}$	Mean of the normal distribution.
$\boldsymbol{w}$	Primal weights.
$\boldsymbol{x}$	Input data vector of dimension f.
γ	Regularization parameter
λ or Λ	Eigenvalue(s), can be in scalar or matrix form
$\mathbf{u}$ or U	Eigenvector(s), can be in vector or matrix form.
$\mathcal{J}_{primal}$	Primal cost function.
$\mathcal{N}$	Normal distribution.
$\mathcal{O}$	Big O notation for complexities.
$\mathcal{P}$	Probability distribution.
Ω_N	Kernel matrix of size N.
Ψ	Dual problem matrix.
σ	Standard deviation, (w.r.t RBF kernel function or Kalman filter variance)
$\varphi()$	The nonlinear feature mapping function.
b	Bias term.
C	Dual matrix.
d	Dimension.
e	Error variable.
f	Number of features (dimension) of the input data.
i	Index of a tensor.
I_N	Identity matrix of size N.
J	Total number of iterations for regression/classification of new samples.
j	Iteration count for regression/classification of new samples.
$k()$	Kernel function.
L	Total number of iterations.
l	Iteration count.
N	Number of data points.
n	Index sizes after tensorization.
P	Covariance matrix of normal distribution.
r	TT-rank.
S	Number of samples from a sampling procedure.
T	Number of observations of the output data or labels.
$TT(\mathcal{A})$	Tensor train of tensor $\mathcal{A}$.
W	Small sampled kernel matrix (subset).
y	Output data for a regression task, or a label for classification task.