

THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Efficient training of interpretable, non-linear regression models

Oskar Allerbo



UNIVERSITY OF GOTHENBURG

Division of Applied Mathematics and Statistics
Department of Mathematical Sciences
University of Gothenburg and Chalmers University of Technology
Gothenburg, Sweden 2023

Efficient training of interpretable, non-linear regression models

Oskar Allerbo
Gothenburg 2023
ISBN 978-91-8069-337-0 (TRYCKT)
ISBN 978-91-8069-338-7 (PDF)

© Oskar Allerbo, 2023

Division of Applied Mathematics and Statistics
Department of Mathematical Sciences
University of Gothenburg and Chalmers University of Technology
SE-412 96 Gothenburg
Sweden

Cover image by Selma Allerbo

Typeset with L^AT_EX
Printed by Stema Specialtryck AB, Borås, Sweden, 2023



Efficient training of interpretable, non-linear regression models

Oskar Allerbo

Division of Applied Mathematics and Statistics
Department of Mathematical Sciences
University of Gothenburg and Chalmers University of Technology

Abstract

Regression, the process of estimating functions from data, comes in many flavors. One of the most commonly used regression models is linear regression, which is computationally efficient and easy to interpret, but lacks in flexibility. Non-linear regression methods, such as kernel regression and artificial neural networks, tend to be much more flexible, but also harder to interpret and more difficult, and computationally heavy, to train.

In the five papers of this thesis, different techniques for constructing regression models that combine flexibility with interpretability and computational efficiency, are investigated. In Papers I and II, sparsely regularized neural networks are used to obtain flexible, yet interpretable, models for additive modeling (Paper I) and dimensionality reduction (Paper II). Sparse regression, in the form of the elastic net, is also covered in Paper III, where the focus is on increased computational efficiency by replacing explicit regularization with iterative optimization and early stopping. In Paper IV, inspired by Jacobian regularization, we propose a computationally efficient method for bandwidth selection for kernel regression with the Gaussian kernel. Kernel regression is also the topic of Paper V, where we revisit efficient regularization through early stopping, by solving kernel regression iteratively. Using an iterative algorithm for kernel regression also enables changing the kernel during training, which we use to obtain a more flexible method, resembling the behavior of neural networks.

In all five papers, the results are obtained by carefully selecting either the regularization strength or the bandwidth. Thus, in summary, this work contributes with new statistical methods for combining flexibility with interpretability and computational efficiency based on intelligent hyperparameter selection.

Keywords: sparse regression, kernel regression, neural network regression, early stopping, bandwidth selection

Sammanfattning

Regression är en metod för att skatta funktioner från data och finns i många olika former. En av de vanligaste regressionsmodellerna är linjär regression, vilket är en beräkningsmässigt effektiv metod med resultat som är lätta att tolka, men vars flexibilitet är begränsad. Icke-linjära regressionsmetoder, såsom kärnregression och artificiella neuronnet, är mer flexibla, men också svårare att tolka och svårare, och beräkningsmässigt tyngre, att träna.

I de fem artiklar som ingår i denna avhandling, undersöks olika tekniker för att konstruera regressionsmodeller som kombinerar flexibilitet med tolkningsbarhet och beräkningsmässig effektivitet. I Artikel I och II används gles regulariserade neuronnet för att konstruera flexibla och tolkningsbara metoder för additiva modeller (Artikel I) och dimensionsreduktion (Artikel II). Gles regularisering, i form av elastic net-regularisering, behandlas även i Artikel III, som fokuserar på ökad beräkningsmässig effektivitet genom att ersätta explicit regularisering med iterativ optimering, avbruten före konvergens. I Artikel IV föreslår vi, med inspiration från Jacobiansk regularisering, en beräkningsmässigt effektiv metod för bandbreddsval för kärnregression med en Gaussisk kärna. Även Artikel V behandlar kärnregression, och beräkningsmässigt effektiv regularisering genom avbruten optimering, genom att använda en iterativ algoritm för kärnregression. Att lösa kärnregression iterativt möjliggör också för att ändra kärnan under träningen, något vi använder för att skapa en mer flexibel metod, vars uppträdande liknar neuronnets.

I alla fem artiklar bygger resultaten på noggranna val av antingen regulariseringsstyrka eller bandbredd. Denna avhandling bidrar alltså med ny statistiska metoder för att kombinera flexibilitet med tolkningsbarhet och beräkningsmässig effektivitet genom intelligent val av hyperparametrar.

List of publications

This thesis contains the following papers:

- I. **Allerbo, O.**, Jörnsten, R. (2022). Flexible, Non-parametric Modeling Using Regularized Neural Networks. *Computational Statistics*, 37(4), 2029-2047.
- II. **Allerbo, O.**, Jörnsten, R. (2021). Non-linear, Sparse Dimensionality Reduction via Path Lasso Penalized Autoencoders. *The Journal of Machine Learning Research*, 22(1),12978-13005.
- III. **Allerbo, O.**, Jonasson, J., Jörnsten, R. (2022). Elastic Gradient Descent, an Iterative Optimization Method Approximating the Solution Paths of the Elastic Net. arXiv:2202.02146. *Submitted*.
- IV. **Allerbo, O.**, Jörnsten, R. (2023). Bandwidth Selection for Gaussian Kernel Ridge Regression via Jacobian Control. arXiv:2205.11956. *Submitted*.
- V. **Allerbo, O.**, Jörnsten, R. (2023). Solving Kernel Ridge Regression with Gradient-Based Optimization Methods. *Manuscript*.

Author contributions

- I. I developed and implemented the method and wrote most of the manuscript.
- II. I developed and implemented the method and wrote the manuscript, with feedback from the coauthor.
- III. I introduced the idea, developed and implemented the method, and wrote the manuscript, with feedback from the coauthors.
- IV. I introduced the idea, developed and implemented the method, and wrote the manuscript, with feedback from the coauthor.
- V. I introduced the idea, developed and implemented the method, and wrote the manuscript, with feedback from the coauthor.

Additional papers not included in this thesis:

- VI. **Allerbo, O.**, Lundström, A., Dimitrievski, K. (2011). Simulations of lipid vesicle rupture induced by an adjacent supported lipid bilayer patch. *Colloids and Surfaces B: Biointerfaces*, 82(2), 632-636.

Acknowledgements

Initially, I would like to thank the Swedish education system, which offers the fifth best economic conditions for Ph.D. students globally¹. In many other countries, it would have larger consequences to quit a job in the industry for starting to pursue a Ph.D.

I would like to thank all my colleagues and friends at Mathematical Sciences and elsewhere at Chalmers and GU, including my supervisors Rebecka and Johan, for making my time with you so pleasant. I also specifically want to thank the organizational policies and culture for all the support I have received. In all organizations, at all levels, there are always people who try to take shortcuts at the expense of others, and since Ph.D. students are, at least nominally, quite low in the hierarchy these shortcuts often go via us. I am very grateful for all the formal and informal procedures that alleviate these tendencies. And I am extremely grateful for the help I have received from various coworkers, both in their professional roles and as fellow human beings. Thanks also for all the interesting conversations in the lunch room, in classrooms, in corridors, in someone's office, in the running tracks, and elsewhere.

I would like to thank freely available information. I have learned so much from free online publications, in the forms of scientific articles, books, source code, blogs, videos, forum discussions, and more. It is hard to imagine acquiring all that knowledge through paid alternatives.

Finally, I would like to thank my family. Thank you for encouraging me to do this, and thank you for supporting and believing in me through these years.

¹Source: <http://isphdforme.com>

Contents

Abstract	iii
List of publications	v
Acknowledgements	vii
Contents	ix
1 Introduction	1
1.1 Aims	3
1.2 Outline	3
2 Regression	5
2.1 Linear (Ridge) Regression	7
2.2 Kernel (Ridge) Regression	8
2.3 Neural Network Regression	10
3 Regularization	13
3.1 Ridge Regularization	14
3.2 Jacobian Regularization	14
3.3 Lasso Regularization	15
3.4 Elastic Net Regularization	15
3.5 Adaptive Lasso Regularization	16

3.6	Group Lasso Regularization	16
3.7	Exclusive Lasso Regularization	17
3.8	Infinity Norm Regularization	17
4	Optimization	19
4.1	Steepest Descent	20
4.2	Accelerating Gradient Descent	20
4.3	Proximal Gradient Descent	21
4.4	Gradient Flow	23
4.5	Early Stopping as a Regularizer	24
4.6	Hyperparameter Optimization	25
5	Summaries of the Included Papers	27
5.1	Paper I	28
5.2	Paper II	29
5.3	Paper III	31
5.4	Paper IV	32
5.5	Paper V	33
6	Conclusions and Future Perspectives	35
	Bibliography	37
	Papers I-V	

1 Introduction

Regression is the process of estimating functions from data. It dates back to work by Legendre [1806] and Gauss [1809], who independently used the least squares model for linear regression. Since then, linear regression has become a standard tool for statistical analyses, with numerous non-linear generalizations. The additional model complexity introduced by non-linear models is, however, double-edged: A more flexible model can represent the data better, thus increasing the model performance. On the other hand, complex models tend to be harder to interpret and more difficult, and computationally expensive, to train. Thanks to its simplicity, linear regression is easy to implement, fast to evaluate, and theoretically accessible. It is, however, limited in terms of flexibility.

The exponential growth in computational power and available data during the last decades have led to a revolution in the usage of non-linear models. A commonly used non-linear regression model is artificial neural networks, or simply neural networks. Although proposed in their first version already by McCulloch and Pitts [1943], and despite their known capabilities of approximating any function arbitrarily well [Cybenko, 1989], their success lingered until the 2000s and the arrival of easily available computation clusters and massive data sets. Compared to linear regression, neural networks possess the flexibility to model much more complex relations, however at the cost of larger computational demands, and reduced interpretability and theoretical understanding.

In some sense, kernel regression (see e.g. Saunders et al. [1998] or Vovk [2013]) can be thought of as a compromise between linear regression and neural networks, where the inferred model is non-linear in the data but linear in the parameters. Similar to linear regression, but in contrast to neural networks, there exists a closed-form solution. This makes kernel regression more theoretically accessible and easier to interpret than neural networks but at the expense of less flexible models. Where neural networks have the capacity to

almost fully adapt to the data, kernel regression relies on stronger assumptions. Additionally, since the closed-form solution includes matrix inversion, it might be computationally heavy, especially for large data sets.

For both linear and non-linear regression, it is common to include some sort of regularization, which introduces a second goal, in addition to explaining the observed data. The aim is often to improve model performance on previously unseen data, but can also be used to promote specific model characteristics, such as sparseness with a reduced number of parameters. For linear regression, where the output depends on each covariate through a single parameter, the effects of regularization are straightforward to analyze: When $y = x_j\beta_1 + x_2\beta_2 + \dots + x_p\beta_p + \varepsilon$, the derivative of y with respect to x_j is exactly β_j . Thus, a small value of β_j is the same as a small derivative, which results in a stable function. When noisy data is used to fit the model, shifting the parameters toward a more stable solution tends to promote generalization to new data. Furthermore, if $\beta_j = 0$, x_j is excluded from the model, which results in a sparse model that is easier to interpret. For non-linear regression, the implication of parameter regularization is not as straightforward. For neural networks, inputs and outputs are generally connected through multiple non-linear paths through the network, where changes in some parameters might be canceled by other changes in other parameters, thus keeping the total function constant, in a quite non-comprehensible way. For kernel regression, on the other hand, the parameterization is implicit, which means that the parameters usually cannot be explicitly regularized.

The strength of the regularization is usually selected before fitting the model to data and tuned on a separate data set, and is because of this often referred to as a hyperparameter. For non-linear regression, there are also other hyperparameters. For a neural network, the architecture, which specifies the number of parameters and their mutual connections, can be considered a hyperparameter. For kernel regression, the type of kernel is a hyperparameter and there are also kernel-specific hyperparameters that have to be selected, such as the degree of a polynomial kernel and the bandwidth (length-scale) of a translational-invariant kernel. Appropriately selected hyperparameters can greatly improve the performance of the model. However, since the model has to be reestimated for each considered combination of the hyperparameters, this might come at a high computational cost.

1.1 Aims

Neural networks are extremely flexible but generally difficult to interpret and characterize theoretically. Compared to neural networks, kernel methods are easier to interpret and to analyze, but they are less flexible. For all types of regression, properly selected hyperparameters can greatly improve both performance and interpretability but at a large computational cost.

The aim of this thesis is to construct **flexible, yet interpretable and computationally efficient regression models**.

This is obtained through the following four methods:

- **Sparse neural networks** for flexible and interpretable modeling (Papers I, II).
- **Regularized derivatives** for increased interpretability and for efficient hyperparameter selection (Papers II, IV).
- **Regularization through early stopping** for efficient hyperparameter selection (Papers III, V).
- **Kernel regression with informed bandwidth selection** for efficient hyperparameter selection and for improved flexibility (Papers IV, V).

1.2 Outline

In the following three chapters, the relevant statistical and mathematical concepts and methodology are reviewed. In Chapter 2, the three types of regression used in this thesis are reviewed: linear regression, kernel regression, and neural network regression. In Chapter 3, different types of regularization are reviewed. In Chapter 4, different types of gradient-based, iterative optimization algorithms and their connection to regularization are reviewed together with hyperparameter optimization. In Chapter 5, the five papers included in this thesis are summarized.

2 Regression

Given n paired observations $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, the purpose of regression is to estimate a function, \hat{f} , that maps from \mathbf{x} to y , i.e. $\hat{y} = \hat{f}(\mathbf{x})$.

With $y_i \in \mathbb{R}$ and $\mathbf{x}_i \in \mathbb{R}^p$, the observations can be collected in a design matrix, $\mathbf{X} \in \mathbb{R}^{n \times p}$, and a response vector $\mathbf{y} \in \mathbb{R}^n$. The i -th row of the design matrix contains the transpose of the i -th covariate: $\mathbf{X}_{i,:} = \mathbf{x}_i^\top$. (If the response is multivariate, something that is known as multivariate regression, the response vector generalizes into a matrix.) Often, the data is assumed to be centered, which means that \mathbf{y} has zero mean. If this is not the case, \mathbf{y} can trivially be centered by subtracting the mean. An equivalent option is to add an intercept to the model.

The oldest, simplest, and most well-studied version of regression is linear regression, where \hat{f} is assumed to be a linear function. Although limited in flexibility, linear regression is still a very useful and popular method.

Kernel regression is a generalization of linear regression, where the data is mapped to a pre-defined, non-linear feature expansion and then linear regression is performed on this feature expansion. According to Mercer's Theorem [Mercer, 1909], every feature expansion can equivalently be expressed through its corresponding kernel, which is often a favorable formulation, especially for infinite-dimensional feature expansions. The simplest example of kernel/feature regression is probably polynomial regression in one dimension, where the feature expansion of x is $[1, x, x^2, \dots, x^k]^\top$.

Although kernel regression is non-linear in the data, it is still linear in the parameters, with a closed-form solution similar to that of standard linear regression. However, in contrast to linear regression, where the solution includes inverting a $p \times p$ matrix, where p is the number of parameters, kernel regression requires inverting an $n \times n$ matrix, where n is the number of observations. On one hand, this allows for p going to infinity, but on the other hand, the

matrix inversion might be computationally heavy if n is large. Furthermore, the feature expansion, or equivalently the kernel, has to be chosen before implementing the algorithm, and the performance may be very dependent on the choice of kernel.

Neural networks are generally used for classification, but can also be used for regression. Just as kernel regression, neural networks for regression can also be thought of as linear regression of a feature expansion, but in contrast to kernel regression, the feature expansion is not pre-defined by the user. Instead, it is realized by the parameters of the network and thus depends on the data. One can think of the parameters of the neural network as belonging to one of two categories, the parameters that constitute the feature expansion, and the parameters used for linear regression. However, in practice, all parameters are treated equally.

The number of parameters in neural networks tends to be large, often larger than the number of observations. Thus, there is redundancy among the parameters, which is not necessarily a problem if the main focus is predictive performance. If, however, interpretation is important, this redundancy introduces challenges, something that is often referred to as neural networks being black box models.

The data-dependent feature expansion of neural networks provides two advantages compared to the manually selected feature expansions of kernel regression. First, while for kernel regression the optimal kernel has to be assessed through testing different kernels to see which performs best, a neural network, ideally, automatically identifies a suitable feature expansion/kernel. Furthermore, while the available kernels in kernel regression are limited by the imagination of the user, the feature expansions of neural networks have no such limitations. For instance, when the data is heterogeneous, it may be difficult to manually select a suitable kernel with different local behavior. Letting a neural network automatically select the feature expansion, however, circumvents this problem.

The network architecture specifies the number of parameters, how different parameters interact with each other, and how and where non-linearities are introduced in the model, and is up to the user to specify. Even if some architecture choices can be obtained by optimizing the parameters, choosing a suitable architecture is still crucial for good performance. Thus, architecture selection is an important hyperparameter. Furthermore, in general, there is no closed-form solution to neural network regression, and iterative optimization methods have to be used, which introduces further hyperparameters related to optimization, including parameter initialization and choice of the optimization algorithm.

Since the optimization problem is generally non-convex, these choices may have a large impact on which local optimum the algorithm converges to.

2.1 Linear (Ridge) Regression

Assuming that the function that maps from x to y is linear, $f(x_i) = x_i^\top \beta$, we obtain

$$y_i = x_i^\top \beta + \varepsilon_i, \quad i = 1, 2, \dots, n$$

or in matrix form

$$\mathbf{y} = \mathbf{X}\beta + \varepsilon,$$

where $\beta \in \mathbb{R}^p$ is referred to as the parameter vector, and $\varepsilon \in \mathbb{R}^n$ denotes observation noise. The objective is to find the β that minimizes the noise, i.e. to minimize $L(\mathbf{y} - \mathbf{X}\beta)$ with respect to β , for some loss function $L(\cdot)$ that quantifies the discrepancy between the true and modeled values. According to the Gauss-Markov Theorem [Gauss, 1823, Markov, 1899], if the errors are uncorrelated with equal variances and expected value zero, then the best unbiased estimator is obtained if the ℓ_2 -norm is used as loss function. Thus, the objective function of linear regression can be written as

$$\hat{\beta} = \operatorname{argmin}_{\beta \in \mathbb{R}^p} \|\mathbf{y} - \mathbf{X}\beta\|_2 = \operatorname{argmin}_{\beta \in \mathbb{R}^p} \frac{1}{2} \|\mathbf{y} - \mathbf{X}\beta\|_2^2. \quad (2.1)$$

The gradient with respect to β is $-\mathbf{X}^\top \mathbf{y} + \mathbf{X}^\top \mathbf{X}\beta$. Setting it to zero we obtain

$$\hat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y},$$

with predictions on new data given by

$$\hat{f}(x^*) = x^{*\top} \hat{\beta} = x^{*\top} (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}.$$

Equation 2.1 can be generalized by adding a penalty on the norm of β ,

$$\hat{\beta} = \operatorname{argmin}_{\beta \in \mathbb{R}^p} \frac{1}{2} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \frac{\lambda}{2} \|\beta\|_2^2, \quad (2.2)$$

for $\lambda \geq 0$, something that is known as ridge regularization. In Equation 2.2, there are two competing objectives, balanced by the magnitude of λ . For a very small value of λ , the first term dominates and the inferred $\hat{\beta}$ will minimize the reconstruction error, $\frac{1}{2} \|\mathbf{y} - \mathbf{X}\beta\|_2^2$. For a very large value of λ , the second

term dominates and the inferred function will basically predict $\mathbf{0}$. An ideally tuned λ will result in an inferred function that is more stable than if optimizing only for the reconstruction error, but not so stable that it only predicts $\mathbf{0}$, which tends to result in better predictive performance on new data. Regularization is discussed in deeper detail in Chapter 3.

The closed form solution of Equation 2.2 is

$$\hat{\beta} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_p)^{-1} \mathbf{X}^\top \mathbf{y} = \mathbf{X}^\top (\mathbf{X} \mathbf{X}^\top + \lambda \mathbf{I}_n)^{-1} \mathbf{y}, \quad (2.3)$$

where $\mathbf{I}_p \in \mathbb{R}^{p \times p}$ and $\mathbf{I}_n \in \mathbb{R}^{n \times n}$ denote identity matrices, and where the second equality can be proved using the matrix inversion lemma,

$$(\mathbf{D} - \mathbf{C} \mathbf{A}^{-1} \mathbf{B})^{-1} \mathbf{C} \mathbf{A}^{-1} = \mathbf{D}^{-1} \mathbf{C} (\mathbf{A} - \mathbf{B} \mathbf{D}^{-1} \mathbf{C})^{-1},$$

with $\mathbf{A} = \mathbf{I}_n$, $\mathbf{B} = -\mathbf{X}$, $\mathbf{C} = \mathbf{X}^\top$ and $\mathbf{D} = \lambda \mathbf{I}_p$.

2.2 Kernel (Ridge) Regression

Kernel regression is a non-linear generalization of linear regression, using a feature expansion of the data. A feature expansion is a mapping from \mathbf{x} to $\varphi(\mathbf{x}) = [\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), \dots, \varphi_q(\mathbf{x})]^\top$, where $\{\varphi_j\}_{j=1}^q$ is a set of user defined functions, e.g. polynomials of increasing order for polynomial regression. We let $\Phi = \Phi(\mathbf{X}) \in \mathbb{R}^{n \times q}$ denote the feature expansion of the design matrix, such that $\Phi_{i,:} = \varphi(\mathbf{x}_i)^\top$. Then, linear ridge regression in feature space is a simple generalization of Equations 2.2 and 2.3:

$$\begin{aligned} \hat{\beta} &= \operatorname{argmin}_{\beta \in \mathbb{R}^q} \frac{1}{2} \|\mathbf{y} - \Phi \beta\|_2^2 + \frac{\lambda}{2} \|\beta\|_2^2 \\ &= (\Phi^\top \Phi + \lambda \mathbf{I}_q)^{-1} \Phi^\top \mathbf{y} = \Phi^\top (\Phi \Phi^\top + \lambda \mathbf{I}_n)^{-1} \mathbf{y}, \end{aligned} \quad (2.4)$$

where predictions on new data are given by

$$\hat{f}(\mathbf{x}^*) = \varphi(\mathbf{x}^*)^\top \hat{\beta} = \varphi(\mathbf{x}^*)^\top \Phi^\top (\Phi \Phi^\top + \lambda \mathbf{I}_n)^{-1} \mathbf{y}. \quad (2.5)$$

When the dimensionality of the feature space, and hence of β , is large (or even infinite), the expressions above become unpractical (or impossible) to handle. However, Equations 2.4 and 2.5 can be reformulated using kernels, eliminating the need to explicitly evaluate $\varphi(\mathbf{x}^*)$ and Φ .

Table 2.1: Examples of kernels.

Kernel Name	Kernel Function
Linear	$\mathbf{x}^\top \mathbf{x}'$
Polynomial	$(\mathbf{x}^\top \mathbf{x}' + c)^d$, for $c \in \mathbb{R}$, $d \in \mathbb{N}$
Gaussian	$e^{-\frac{\ \mathbf{x} - \mathbf{x}'\ _2^2}{2\sigma^2}}$, for $\sigma > 0$
Laplace	$e^{-\frac{\ \mathbf{x} - \mathbf{x}'\ _2}{\sigma}}$, for $\sigma > 0$
Cauchy	$\left(1 + \frac{\ \mathbf{x} - \mathbf{x}'\ _2^2}{\sigma^2}\right)^{-1}$, for $\sigma > 0$

A kernel function, $k(\cdot, \cdot)$, is a two-argument, symmetric, positive semi-definite function. That k is symmetric means that $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$, and that k is positive semi-definite means that $\mathbf{c}^\top \mathbf{K} \mathbf{c} \geq 0$, for any vector $\mathbf{c} \in \mathbb{R}^n$, and for any set of observations, $\{\mathbf{x}_i\}_{i=1}^n$, where $\mathbf{K}_{ij} = \mathbf{K}(\mathbf{X}, \mathbf{X})_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. The connection between kernels and feature expansions is given by Mercer's Theorem, which states that every kernel can be written as the inner product of a feature expansion, i.e. $k(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}(\mathbf{x}_i)^\top \boldsymbol{\varphi}(\mathbf{x}_j)$. Using Mercer's Theorem on Equation 2.5, we can write $\boldsymbol{\Phi} \boldsymbol{\Phi}^\top = \mathbf{K} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{\varphi}(\mathbf{x}^*)^\top \boldsymbol{\Phi}^\top = \mathbf{k}(\mathbf{x}^*, \mathbf{X})^\top$, where $\mathbf{k}(\mathbf{x}^*, \mathbf{X}) \in \mathbb{R}^n$. (Note that $\boldsymbol{\Phi}$ contains the vectors $\boldsymbol{\varphi}(\mathbf{x}_i)$ in transposed form, which might cause some confusion about the transposes.) We then obtain

$$\hat{f}(\mathbf{x}^*) = \mathbf{k}(\mathbf{x}^*, \mathbf{X})^\top (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}. \quad (2.6)$$

Furthermore, with $\boldsymbol{\alpha} \in \mathbb{R}^n$ implicitly defined through $\boldsymbol{\beta} = \boldsymbol{\Phi}^\top \boldsymbol{\alpha}$, Equation 2.4 can be rewritten as

$$\begin{aligned} \hat{\boldsymbol{\alpha}} &= \operatorname{argmin}_{\boldsymbol{\alpha} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{y} - \mathbf{K} \boldsymbol{\alpha}\|_2^2 + \frac{\lambda}{2} \boldsymbol{\alpha}^\top \mathbf{K} \boldsymbol{\alpha} = \operatorname{argmin}_{\boldsymbol{\alpha} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{y} - \mathbf{K} \boldsymbol{\alpha}\|_2^2 + \frac{\lambda}{2} \|\boldsymbol{\alpha}\|_{\mathbf{K}}^2 \\ &= (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y} \end{aligned}$$

with predictions on new data given by

$$\hat{f}(\mathbf{x}^*) = \mathbf{k}(\mathbf{x}^*, \mathbf{X})^\top \hat{\boldsymbol{\alpha}} = \mathbf{k}(\mathbf{x}^*, \mathbf{X})^\top (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}.$$

In Table 2.1 some commonly used kernels are presented. Even though it is not obvious at first glance, according to Mercer's Theorem, these kernels can all be written as inner products of feature expansion. In Table 2.2 we present the feature expansions for three kernels when the data is one-dimensional. It is straightforward, but slightly tedious, to verify that $\boldsymbol{\varphi}(x)^\top \boldsymbol{\varphi}(x') = k(x, x')$ in all

Table 2.2: Examples of kernels for one-dimensional data, and corresponding feature expansions.

Kernel Name	Kernel Function	Feature Expansion
Linear	$x \cdot x'$	$[x]$
Quadratic	$(x \cdot x' + c)^2$	$[x^2 \quad \sqrt{2c}x \quad \alpha]^\top$
Gaussian	$e^{-\frac{(x-x')^2}{2\sigma^2}}$	$e^{-\frac{x^2}{2\sigma^2}} \left[1 \quad \frac{x^1}{\sigma^1 \sqrt{1!}} \quad \dots \quad \frac{x^k}{\sigma^k \sqrt{k!}} \quad \dots \right]^\top$

three cases.

2.3 Neural Network Regression

Neural networks model data according to $\hat{f}(x) = \hat{f}(x; \hat{\theta})$, where the function \hat{f} is parameterized by θ . To emphasize that we have left the linear regime, the parameter vector is denoted with θ , using the notation β exclusively for the linear case.

Just as previously, the objective is to minimize the discrepancy between observations and model predictions, with respect to the parameter vector θ ,

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} L(f(\mathbf{X}; \theta), \mathbf{y}), \quad (2.7)$$

for some loss function $L(\cdot)$, such as the ℓ_2 -norm of the difference which is commonly used for regression, in which case

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{2} \|\mathbf{y} - \mathbf{f}(\mathbf{X}; \theta)\|_2^2.$$

In general, there is no closed-form solution to Equation 2.7. However, there exist several iterative methods for reaching a (local) minimum of the objective function. Some of these are reviewed in Chapter 4.

A commonly used neural network architecture is the feed-forward neural network, which consists of an input layer x , an output layer $\hat{y} = \hat{f}(x)$, and arbitrarily many intermediary layers, often referred to as hidden layers. The input of layer l , for $l \in \{1, \dots, l_M\}$, is a vector $\mathbf{i}_l \in \mathbb{R}^{p_l}$ that is a linear combination of the output of layer $l-1$, $\mathbf{o}_{l-1} \in \mathbb{R}^{p_{l-1}}$: $\mathbf{i}_l = \mathbf{W}_l \mathbf{o}_{l-1} + \mathbf{b}_l$, where $\mathbf{W}_l \in \mathbb{R}^{p_l \times p_{l-1}}$ is referred to as a weight matrix and $\mathbf{b}_l \in \mathbb{R}^{p_l}$ is referred to as a

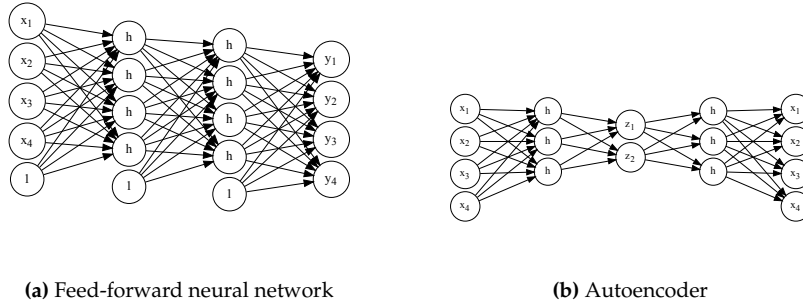


Figure 2.1: Left: Sketch of a feed-forward neural network with two hidden layers and four nodes in each layer. Of course, any number of layers with any number of nodes can be used. Each arrow in the graph corresponds to a parameter θ_i .

Right: Sketch of an autoencoder mapping from 4 to 2 dimensions. Original dimensions are denoted by x and latent dimensions by z . Bias vectors are omitted to enhance readability.

bias vector. The output of the 0-th layer is the input data, $\mathbf{o}_0 = \mathbf{x}$. To make the network non-linear, in each hidden layer a non-linear activation function, $h_k(\cdot)$, is applied element-wise to the input: $\mathbf{o}_k = \mathbf{h}_k(\mathbf{i}_k)$. Examples of activation functions include the hyperbolic tangent, $h(x) = \tanh(x)$; the rectified linear unit, $h(x) = \max(x, 0)$; and the linear function, $h(x) = x$. Hence, a feed-forward neural network can be expressed as

$$\hat{f}(\mathbf{x}; \{(\mathbf{W}_l, \mathbf{b}_l)\}_{l=1}^{l_M}) = \mathbf{h}_{l_M}(\mathbf{W}_{l_M} \mathbf{h}_{l_M-1}(\dots \mathbf{h}_1(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \dots + \mathbf{b}_{l_M-1}) + \mathbf{b}_{l_M}),$$

where the parameter vector θ consists of all the elements in the weight matrices and bias vectors. Using the chain rule, it is tedious, but straightforward, to express the gradient of the loss function with respect to $(\mathbf{W}_l, \mathbf{b}_l)_{l=1}^{l_M}$, something that is often referred to as backpropagation [Werbos, 1974].

In Figure 2.1a, a sketch of a feed-forward neural network is displayed. Each arrow in the graph corresponds to a parameter θ_i . The number of layers, and the number of nodes in each layer, are parts of the network design and are up to the user to choose.

With the autoencoder architecture [Kramer, 1991], feed-forward neural networks can be used for non-linear dimensionality reduction. In an autoencoder, the same data, \mathbf{X} , is used both as the input and the response, so the goal is simply to reconstruct the original data, \mathbf{X} , but via a low dimensional representation, \mathbf{Z} . In Figure 2.1b we show a simple autoencoder (omitting the bias

vectors in the figure to increase readability). Here,

$$z = \mathbf{W}_2 h(\mathbf{W}_1 x + \mathbf{b}_1) + \mathbf{b}_2$$

and

$$\hat{x} = \mathbf{W}_4 h(\mathbf{W}_3 z + \mathbf{b}_3) + \mathbf{b}_4,$$

3 Regularization

Regularization is the process of enforcing a certain behavior on the inferred function. Thus, the objective of estimating a function that fits the observed data is balanced by a second objective that shifts the solution toward a predefined property, sometimes referred to as a prior. The most commonly used regularization methods for linear regularization are probably ridge regression (which was briefly covered in Chapter 2) [Hoerl and Kennard, 1970], lasso [Tibshirani, 1996], and the elastic net [Zou and Hastie, 2005].

Ridge regression shifts all model parameters toward zero. For linear regression this is equivalent to constraining the derivatives of the inferred function, thus rendering a more stable solution, which usually improves generalization. For non-linear models, when the derivatives do not coincide with the parameter values, the term Jacobian regularization [Jakubovitz and Giryes, 2018] is usually used when regularizing derivatives. Lasso regularization also shifts model parameters toward zero, but has the additional capability of setting some parameters exactly to zero, thus eliminating them from the inferred model. For linear regression, when the model depends on each covariate through a single parameter, this is equivalent to eliminating the corresponding covariate from the model, thus rendering a more interpretable model. This is especially useful if the data is high-dimensional. The elastic net is a convex combination of ridge regression and lasso, combining the benefits of the two regularization schemes.

Lasso regularization comes in several flavors, some of which are reviewed below. The adaptive lasso [Zou, 2006] uses an individual regularization strength for each parameter, thus reducing bias. In group lasso [Yuan and Lin, 2006] and exclusive lasso [Zhou et al., 2010], groups of parameters are regularized together. Infinity norm regularization can be seen as the opposite of lasso. Instead of setting some parameters exactly to zero, some parameters obtain exactly equal values.

The different regularization methods are reviewed in more detail below. In

all cases except Jacobian regularization, the methods are presented for linear regression.

3.1 Ridge Regularization

As stated earlier, ridge regression is obtained by adding a penalty on the ℓ_2 -norm of the parameter vector to the reconstruction error:

$$\hat{\boldsymbol{\beta}} = \frac{1}{2} \operatorname{argmin}_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \frac{\lambda}{2} \|\boldsymbol{\beta}\|_2^2 = \operatorname{argmin}_{\boldsymbol{\beta}} \sum_{i=1}^n (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \frac{\lambda}{2} \sum_{j=1}^p \beta_j^2,$$

with closed-form solution

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y},$$

The hyperparameter $\lambda \geq 0$ is known as the penalty, or regularization strength, and shifts the solution toward a smaller $\hat{\boldsymbol{\beta}}$. The term ridge can be attributed to the "ridge" added to $\mathbf{X}^\top \mathbf{X}$ by shifting its diagonal elements by λ . Since $\hat{f}(\mathbf{x}) = \mathbf{x}^\top \hat{\boldsymbol{\beta}}$,

$$\frac{d\hat{f}(\mathbf{x})}{d\mathbf{x}} = \frac{d\mathbf{x}^\top \hat{\boldsymbol{\beta}}}{d\mathbf{x}} = \hat{\boldsymbol{\beta}}.$$

Thus, a solution with smaller $\hat{\beta}_j$'s is a solution with smaller derivatives, i.e. a more stable function, which tends to promote generalization.

3.2 Jacobian Regularization

Jacobian regularization can be seen as a non-linear generalization of ridge regularization and has been used successfully for neural networks. The Jacobian penalty is simply the Frobenius norm of the Jacobian of the inferred function:

$$\hat{\boldsymbol{\theta}} = \operatorname{argmin}_{\boldsymbol{\theta}} \frac{1}{2n} \sum_{i=1}^n \left(\|\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta})\|_2^2 + \lambda \left\| \frac{\partial \mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta})}{\partial \mathbf{x}_i} \right\|_F^2 \right). \quad (3.1)$$

In Equation 3.1, we allow for a multivariate response. Thus, for the n observations, we have $\mathbf{y}_i \in \mathbb{R}^q$, $\mathbf{f}(\mathbf{x}_i, \boldsymbol{\theta}) \in \mathbb{R}^q$, and $\mathbf{x}_i \in \mathbb{R}^p$. For linear regression, with

$f(\mathbf{x}) = \boldsymbol{\beta}^\top \mathbf{x}$ the Jacobian penalty reduces to the ridge penalty:

$$\frac{1}{n} \sum_{i=1}^n \left\| \frac{\partial f(\mathbf{x}_i)}{\partial \mathbf{x}_i} \right\|_F^2 = \frac{1}{n} \sum_{i=1}^n \left\| \frac{\partial \mathbf{x}_i^\top \boldsymbol{\beta}}{\partial \mathbf{x}_i} \right\|_F^2 = \|\boldsymbol{\beta}\|_2^2.$$

3.3 Lasso Regularization

The purpose of lasso regularization is to select a sparse parameter vector $\hat{\boldsymbol{\beta}}$ and thus make it easier to interpret the inferred model. In lasso, the squared ℓ_2 -norm penalty on $\boldsymbol{\beta}$ of ridge regularization is replaced by an ℓ_1 -norm penalty:

$$\hat{\boldsymbol{\beta}} = \frac{1}{2} \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 = \frac{1}{2} \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^n (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \lambda \sum_{j=1}^p |\beta_j|,$$

Just as for ridge regression, small parameter values are encouraged but, as a consequence of the non-differentiability of the absolute value at zero, lasso has the capability to set some components of $\hat{\boldsymbol{\beta}}$ exactly to zero, thus eliminating them from the model. Hence, lasso can be seen as a combination of regularization and model selection. In contrast to ridge regularization, no closed-form solution exists for lasso, unless the covariates are uncorrelated and $\mathbf{X}^\top \mathbf{X} = \mathbf{I}$. Instead, the solution can be obtained through proximal gradient descent, a method discussed in Chapter 4.

3.4 Elastic Net Regularization

The elastic net is a convex combination of lasso and ridge regression:

$$\hat{\boldsymbol{\beta}} = \frac{1}{2} \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \left(\alpha \|\boldsymbol{\beta}\|_1 + \frac{1-\alpha}{2} \|\boldsymbol{\beta}\|_2^2 \right),$$

for $\alpha \in (0, 1)$. The elastic net still selects a sparse model, but alleviates two shortcomings of lasso: First, in the high-dimensional setting, when $p > n$, lasso can select at most n variables. Second, if two or more variables are highly correlated, lasso tends to include only one of these in the model, and be quite indifferent as to which, while the elastic net tends to include correlated variables together.

The elastic net is also used in sparse principal component analysis [Zou et al.,

2006], a linear dimensionality reduction technique where each original dimension is represented only in a subset of the latent dimensions, and vice versa.

3.5 Adaptive Lasso Regularization

Compared to the non-regularized solution for $\hat{\beta}$, lasso shifts all parameters toward zero, and if the shift is large enough for the parameter to "hit" zero, it is removed from the model. Since the magnitude of the shift is the same for all parameters, only parameters with small enough non-regularized values will have a chance to be excluded from the model. The remaining parameters will remain non-zero but with a shift. To reduce the bias introduced by shifting the parameters kept in the inferred model, it is sometimes desirable to penalize small parameters more than large ones. In the adaptive lasso, this is obtained by using an individual, adaptive penalty for each parameter, β_j :

$$\hat{\beta} = \frac{1}{2} \operatorname{argmin}_{\beta} \sum_{i=1}^n (y_i - \mathbf{x}_i^\top \beta)^2 + \lambda \sum_{j=1}^p \frac{|\beta_j|}{|\hat{\beta}_j^R|^\gamma},$$

where $\hat{\beta}_j^R$ is the ridge estimate of β_j and where $\gamma > 0$. Hence, each parameter is assigned an individual penalty, and parameters with ridge estimates close to zero are penalized harder than parameters with larger ridge estimates. Thus, parameters far away from zero are only marginally affected by the regularization.

3.6 Group Lasso Regularization

The purpose of the group lasso is to penalize predefined groups of parameters together, so that either all parameters in the group are included in the model, or none are. This is obtained by the objective function

$$\frac{1}{2} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \sum_{k=1}^G \|\beta_{\mathbf{g}_k}\|_2 = \frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{x}_i^\top \beta)^2 + \lambda \sum_{k=1}^G \sqrt{\sum_{j \in \mathbf{g}_k} \beta_j^2},$$

where each parameter is assigned to one of G groups, \mathbf{g}_k contains the indices of group k and $\beta_{\mathbf{g}_k}$ denotes a sub-vector of β , including only the elements that are represented in \mathbf{g}_k . Since $\sqrt{\sum_{j \in \mathbf{g}_k} \beta_j^2} = 0$ only if all parameters in group

k are zero, this is a way of eliminating entire groups together. If every group consists only of a single parameter, or equivalently if each parameter belongs to a unique group, then the group lasso simplifies to standard lasso.

3.7 Exclusive Lasso Regularization

The exclusive lasso can be seen as the opposite of the group lasso. Again, the parameters are split into pre-defined groups, but now the goal is instead to impose a similar number of non-zero parameters in every group. With the same notation as for group lasso, the objective function is

$$\frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{k=1}^G \|\boldsymbol{\beta}_k\|_1^2 = \frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \lambda \sum_{k=1}^G \left(\sum_{j \in \mathbf{g}_k} |\beta_j| \right)^2.$$

Since the number of mixed terms in the squared sum grows with the number of elements in the sum, the total number of mixed terms over all the groups is minimized when the non-zero elements are evenly distributed among the groups.

3.8 Infinity Norm Regularization

The objective function of infinity norm regularization is

$$\hat{\boldsymbol{\beta}} = \frac{1}{2} \operatorname{argmin}_{\boldsymbol{\beta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_\infty = \frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{x}_i^\top \boldsymbol{\beta})^2 + \lambda \cdot \max_j |\beta_j|.$$

In a similar way to how lasso enforces a solution with zeros in $\hat{\boldsymbol{\beta}}$, infinity norm regularization enforces a solution where the absolute values of the components in $\hat{\boldsymbol{\beta}}$ exactly coincide. This is often not desired for linear regression but can be advantageous in kernel regression since it encourages observations to contribute equally to the solution.

4 Optimization

The goal of every minimization problem is to find the global minimum of the loss function. If the problem is convex, there exists a global minimum which in some cases can be accessed analytically by selecting parameters such that the gradient is zero. However, for many optimization problems, even if convex, no closed-form solutions exist and iterative optimization methods have to be used. For the optimization objectives stated in Chapters 2 and 3, closed-form solutions exist for only linear and kernel ridge regression. Neural networks are intrinsically non-linear and almost by construction non-convex, without closed-form solutions. For the different forms of regularization reviewed in Chapter 3, all but ridge and Jacobian regularization contain discontinuities in the gradient of the loss functions. Therefore, they not only lack closed-form solutions but they also cannot be solved using standard gradient-based optimization methods, which assume continuous gradients. Instead more computationally heavy proximal gradient methods have to be used [Rockafellar, 1976].

The idea behind gradient-based optimization methods is to iteratively move downhill in the optimization landscape until a (local) minimum is reached. In each iteration, the gradient is calculated and used for finding suitable downhill directions. The simplest gradient-based algorithm is gradient descent, which was originally suggested by Cauchy [1847]. In each iteration, every parameter is updated with a small step, η , in the direction of the negative gradient:

$$\hat{\theta}_{k+1} = \hat{\theta}_k - \eta \cdot \nabla f(\hat{\theta}_k). \quad (4.1)$$

When fitting a regression model to data, the optimization process is often referred to as training the model, since the model performance (ideally) increases with each iteration.

Table 4.1: Update directions when using steepest descent for the ℓ_2 -, ℓ_∞ - and ℓ_1 -norms, where $m := \operatorname{argmax}_p |\nabla f(\hat{\theta})_p|$ and e_m denotes the m -th standard basis vector. The normalized and non-normalized versions of $\Delta\hat{\theta}$ differ in the magnitude, but not the direction, of the vector.

Norm	$\Delta\hat{\theta}$	Name
$\ \cdot\ _2$	$\nabla f(\hat{\theta})/\ \nabla f(\hat{\theta})\ _2$	(Normalized) Gradient Descent
$\ \cdot\ _\infty$	$\operatorname{sign}(\nabla f(\hat{\theta}))$	Sign Gradient Descent
$\ \cdot\ _1$	$\left(\operatorname{sign}(\nabla f(\hat{\theta}))\right)_m \cdot e_m$	Coordinate Descent

4.1 Steepest Descent

Steepest descent [Boyd et al., 2004] is a framework that generalizes many gradient-based optimization algorithms. The objective $\min_{\theta} f(\theta)$ is minimized by applying Equation 4.2 iteratively for a given norm, $\|\cdot\|$.

$$\begin{aligned} \Delta\hat{\theta}_k &= \operatorname{argmax}_{v: \|v\|=1} v^\top \nabla f(\hat{\theta}_k) \\ \hat{\theta}_{k+1} &= \hat{\theta}_k - \eta \cdot \Delta\hat{\theta}_k. \end{aligned} \tag{4.2}$$

Depending on which norm is used, different algorithms are obtained. In Table 4.1 we present the algorithms obtained from steepest descent when using the ℓ_2 -, ℓ_∞ - and ℓ_1 -norms. In addition to gradient descent, which is obtained for the ℓ_2 -norm, we obtain sign gradient descent from the ℓ_∞ -norm, and coordinate descent from the ℓ_1 -norm. While all three algorithms move downhill in the optimization landscape, they follow slightly different trajectories. For gradient descent, in each iteration, all components in $\hat{\theta}$ are updated, and those with large gradient values are updated more than those with small values. For sign gradient descent, the magnitudes of all updates are equal. For coordinate descent, only the component with the largest gradient value is updated, while all others are kept fixed.

4.2 Accelerating Gradient Descent

There exist several methods for accelerating Equation 4.1. The ones reviewed in this section build on one, or both, of two ideas.

The first idea allows not only for current, but also for past, gradient values to

influence the update direction. This is similar to how a ball rolls down a slope; once it has gained momentum it does not respond immediately to changes in the slope. Consequently, the algorithm is referred to as gradient descent with momentum [Polyak, 1964], for which an iteration is given by

$$\hat{\theta}_{k+1} = \hat{\theta}_k + \gamma (\hat{\theta}_k - \hat{\theta}_{k-1}) - \eta \cdot \nabla f(\hat{\theta}_k),$$

where $\gamma \in [0, 1)$ denotes the strength of the momentum. Standard gradient descent is obtained as a special case for $\gamma = 0$. Nesterov accelerated gradient [Nesterov, 1983] is a modification of gradient descent with momentum. Instead of evaluating the gradient at $\hat{\theta}_k$, it is evaluated at the best guess of $\hat{\theta}_{k+1}$:

$$\hat{\theta}_{k+1} = \hat{\theta}_k + \gamma (\hat{\theta}_k - \hat{\theta}_{k-1}) - \eta \cdot \nabla f \left(\hat{\theta}_k + \gamma (\hat{\theta}_k - \hat{\theta}_{k-1}) \right).$$

The second idea tries to mitigate the issue of some gradient components being small relative to others, which causes the corresponding parameters to update more slowly. One solution to this is to replace the gradient in Equation 4.1 with something more similar to the sign of the gradient, where all components are updated equally. This is usually realized by replacing $\nabla f(\hat{\theta}_k)$ with

$$\frac{\nabla f(\hat{\theta}_k)}{\sqrt{\mathbf{g}_k^2 + \varepsilon}},$$

where

$$\mathbf{g}_k^2 = \alpha \cdot (\nabla f(\hat{\theta}))^2 + (1 - \alpha) \cdot \mathbf{g}_{k-1}^2, \quad \alpha \in [0, 1],$$

is an exponential smoother of the squared gradients (with the square taken element-wise). Here, $\varepsilon > 0$ is a small constant to avoid division with zero. For $\alpha = 0$, we obtain standard gradient descent (scaled with the factor \mathbf{g}_0^2 , which can be absorbed into η by rescaling the optimization step-size), and for $\alpha = 1$ we obtained sign gradient descent (assuming ε is negligible).

Two algorithms leveraging on this idea are RMSprop [Hinton et al., 2012] and Adam [Kingma and Ba, 2014], where Adam is basically RMSprop with momentum.

4.3 Proximal Gradient Descent

Gradient descent can only be used when the objective function is differentiable. All penalties in Chapter 3, except for the ridge and Jacobian penalties, have

discontinuities in their gradients and are thus not compatible with standard gradient descent. However, whenever the objective function f can be decomposed into $f(\hat{\boldsymbol{\theta}}) = g(\hat{\boldsymbol{\theta}}) + h(\hat{\boldsymbol{\theta}})$, where g is differentiable and h is not, which is the case in this situation, proximal gradient descent can be used. An iteration using proximal gradient descent is defined as

$$\hat{\boldsymbol{\theta}}_{k+1} = \text{prox}_{\eta h}(\hat{\boldsymbol{\theta}}_t - \eta \cdot \nabla g(\hat{\boldsymbol{\theta}}_t)),$$

where prox denotes the proximal operator for h , defined as

$$\text{prox}_{\eta h}(\hat{\boldsymbol{\theta}}) := \underset{\mathbf{x}}{\text{argmin}} \frac{1}{2\eta} \|\mathbf{x} - \hat{\boldsymbol{\theta}}\|_2^2 + h(\mathbf{x}). \quad (4.3)$$

When Equation 4.3 has a cheaply available solution, proximal gradient descent is a convenient option for optimizing non-differentiable functions.

Below we state the proximal operators for the regularizers in Chapter 3. For greater detail, see Parikh et al. [2014].

Ridge:

$$\left(\text{prox}_{\eta\lambda\|\cdot\|_2^2}(\hat{\boldsymbol{\theta}}) \right)_i = \frac{\hat{\theta}_i}{1 + \eta\lambda}.$$

Lasso:

$$\left(\text{prox}_{\eta\lambda\|\cdot\|_1}(\hat{\boldsymbol{\theta}}) \right)_i = \text{sign}(\hat{\theta}_i) \cdot \max(0, |\hat{\theta}_i| - \eta\lambda) = \hat{\theta}_i \cdot \max\left(0, 1 - \frac{\eta\lambda}{|\hat{\theta}_i|}\right).$$

The Elastic Net:

$$\left(\text{prox}_{\eta\lambda(\alpha\|\cdot\|_1 + (1-\alpha)\|\cdot\|_2^2)}(\hat{\boldsymbol{\theta}}) \right)_i = \frac{\text{sign}(\hat{\theta}_i) \cdot \max(0, |\hat{\theta}_i| - \eta\alpha\lambda)}{1 + \eta(1-\alpha)\lambda}.$$

Group Lasso:

$$\left(\text{prox}_{\eta\lambda\sum_k \|\cdot\|_2}(\hat{\boldsymbol{\theta}}) \right)_i = \hat{\theta}_i \cdot \max\left(0, 1 - \frac{\eta\lambda}{\sqrt{\sum_{\hat{\theta}_j \in g_k} \hat{\theta}_j^2}}\right).$$

Exclusive Lasso:

$$\begin{aligned} \left(\text{prox}_{\eta\lambda \sum_k \|\cdot\|_1^2}(\hat{\boldsymbol{\theta}})\right)_i &= \text{sign}(\hat{\theta}_i) \cdot \max\left(0, |\hat{\theta}_i| - \eta\lambda \sum_{\hat{\theta}_j \in g_k} |\hat{\theta}_j|\right) \\ &= \hat{\theta}_i \cdot \max\left(0, 1 - \frac{\eta\lambda \sum_{\hat{\theta}_j \in g_k} |\hat{\theta}_j|}{|\hat{\theta}_i|}\right). \end{aligned}$$

$\|\cdot\|_\infty$ -norm Regularization:

$$\text{prox}_{\eta\lambda \|\cdot\|_\infty}(\hat{\boldsymbol{\theta}}) = \hat{\boldsymbol{\theta}} - \eta\lambda \text{Proj}_{\|\cdot\|_1 \leq 1}\left(\frac{\hat{\boldsymbol{\theta}}}{\eta\lambda}\right),$$

where $\text{Proj}_{\|\cdot\|_1 \leq 1}(\cdot)$ denotes the projection operator onto the ℓ_1 -ball.

Remark 1: Since the squared ℓ_2 -norm is differentiable, in this case, there is no need for proximal gradient descent, but the proximal operator is included for completeness.

Remark 2: For the ℓ_1 -norm, if $|\theta_i| < \eta\lambda$, applying the proximal operator will result in an exact zero, while for the ℓ_2 -norm, if $\theta_i \neq 0$, applying the proximal operator can only result in a very small value, but never an exact zero.

Remark 3: The proximal operator for the combined ℓ_1 - and ℓ_2 -norms is the product of the corresponding proximal operators.

Remark 4: The proximal operator for adaptive lasso is the same as for standard lasso with $\lambda_j = \lambda/|\hat{\beta}_j^R|^\gamma$.

Remark 5: If each θ_i constitutes a group of its own, the proximal operators for lasso and group lasso coincide.

4.4 Gradient Flow

The update rule of gradient descent in Equation 4.1 coincides with the formulation of Euler's method for solving differential equations [Euler, 1792], where the differential equation

$$\mathbf{y}'(t) = \mathbf{g}(t, \mathbf{y}(t))$$

is solved iteratively according to

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h \cdot \mathbf{g}(t_k, \mathbf{y}_k).$$

Identifying \mathbf{y} with $\hat{\boldsymbol{\theta}}$, h with η and $\mathbf{g}(t_k, \mathbf{y}_k)$ with $-\nabla f(\hat{\boldsymbol{\theta}}_{k+1})$, we see that the differential equation corresponding to gradient descent is

$$\hat{\boldsymbol{\theta}}'(t) = -\nabla f(\hat{\boldsymbol{\theta}}(t)). \quad (4.4)$$

Alternatively Equation 4.4 can be derived by rewriting Equation 4.1 as

$$\hat{\boldsymbol{\theta}}(t + \Delta t) = \hat{\boldsymbol{\theta}}(t) - \Delta t \cdot \nabla f(\hat{\boldsymbol{\theta}}(t)),$$

rearranging and letting $\Delta t \rightarrow 0$:

$$\hat{\boldsymbol{\theta}}'(t) = \lim_{\Delta t \rightarrow 0} \frac{\hat{\boldsymbol{\theta}}(t + \Delta t) - \hat{\boldsymbol{\theta}}(t)}{\Delta t} = -\nabla f(\hat{\boldsymbol{\theta}}(t)).$$

Gradient descent with infinitesimal step size is often referred to as gradient flow.

In general, Equation 4.4 has no closed-form solution. However, for linear regression, a closed-form gradient flow solution exists. The differential equation for solving Equation 2.1 with gradient descent with infinitesimal step size is

$$\hat{\boldsymbol{\beta}}'(t) = \mathbf{X}^\top \mathbf{y} - \mathbf{X}^\top \mathbf{X} \hat{\boldsymbol{\beta}}(t).$$

Assuming $\hat{\boldsymbol{\beta}}(0) = \mathbf{0}$, the solution is given by

$$\hat{\boldsymbol{\beta}}(t) = (\mathbf{I} - \exp(-t \cdot \mathbf{X}^\top \mathbf{X})) (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \quad (4.5)$$

where \exp denotes the matrix exponential.

Remark: Equation 4.5 is a special case of the more general closed-form solution for Equation 2.2 with $\lambda > 0$ and $\hat{\boldsymbol{\beta}}(0) = \hat{\boldsymbol{\beta}}_0$,

$$\begin{aligned} \hat{\boldsymbol{\beta}}(t) &= (\mathbf{I} - \exp(-t \cdot (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}))) (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y} \\ &\quad + \exp(-t \cdot (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})) \hat{\boldsymbol{\beta}}_0. \end{aligned}$$

4.5 Early Stopping as a Regularizer

It is well known that early stopping, i.e. to stop training before convergence, has a regularizing effect. Intuitively this makes sense: If parameters have small values at the beginning of the training and the model complexity increases with parameter size, stopping before convergence results in a model with smaller parameter values, and thus in a less complex model. More rigorously, let us

compare the ridge estimate of β of Equation 2.3,

$$\begin{aligned}\hat{\beta}(\lambda) &= (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y} \\ &= (\mathbf{I} - (\mathbf{I} + 1/\lambda \cdot \mathbf{X}^\top \mathbf{X})^{-1}) (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y},\end{aligned}$$

to the gradient flow solution of Equation 4.5,

$$\begin{aligned}\hat{\beta}(t) &= (\mathbf{I} - \exp(-t \cdot \mathbf{X}^\top \mathbf{X})) (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \\ &= (\mathbf{I} - \exp(t \cdot \mathbf{X}^\top \mathbf{X})^{-1}) (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}.\end{aligned}$$

We note that the ridge estimate can be obtained from the gradient flow estimate by replacing $\exp(t \cdot \mathbf{X}^\top \mathbf{X})$ by its first order Taylor expansion, $(\mathbf{I} + t \cdot \mathbf{X}^\top \mathbf{X})$, and by defining $\lambda = 1/t$. Thus, thinking about training time as an inverse penalty, and solving linear regression with gradient descent, we obtain a solution that is similar, but not identical, to the solution of ridge regression. This was first proposed by Ali et al. [2019].

Similarly, the close resemblance between coordinate descent with early stopping and lasso has been well studied [Efron et al., 2004, Hastie et al., 2007], and there are also striking similarities between sign gradient descent and ℓ_∞ -norm regularization, which are addressed in Paper V. Interestingly, the connections between an explicit norm regularization and a gradient-based algorithm with early stopping (i.e. gradient descent and ℓ_2 -regularization, coordinate descent and ℓ_1 -regularization, sign gradient descent, and ℓ_∞ -regularization) are the same as the norms of steepest descent as presented in Table 4.1.

4.6 Hyperparameter Optimization

Hyperparameters are parameters that need to be selected before starting to train the model and can thus be thought of as model selection parameters: First, the model class is specified through the hyperparameters and then the model parameters are optimized to obtain the optimal model within the selected class.

One option is to treat the hyperparameters as any other model parameters and include them in the model parameter optimization, something that is known as marginal likelihood optimization. One drawback of this approach is, however, that problems that are convex given the hyperparameters, tend to be non-convex when treating the hyperparameters as model parameters. Thus, marginal likelihood optimization often gets stuck in local, non-optimal, minima.

Another option is to manually select a set of hyperparameter values to consider and split the data into training and validation data. Then the training data is used to train multiple models, one for each potential hyperparameter combination, and the model that performs best on validation data is selected. Thus, local optima for the hyperparameters are avoided, but at the expense of a brute force approach, which might both be computationally expensive and overlook potentially favorable values. To reduce the impact of exactly how the data is split into training and validation data, the split is often performed multiple times, and the final decision is based on the average of the different splits, something that is known as cross-validation. In k -fold cross-validation, the data is split into k subsamples, folds. Then each potential model is evaluated k times, where each time one (different) fold is used as validation data and the remaining $k - 1$ folds are used as training data. This way, each fold is used for validation exactly once (and as part of the training data $k - 1$ times). The case when $k = n$, and each fold contains one observation, is known as leave-one-out cross-validation. Generalized cross-validation [Golub et al., 1979] is an approximation of leave-one-out cross-validation, that generally is more computationally efficient.

5 Summaries of the Included Papers

The five papers included in this thesis are categorized in Table 5.1. The backbone of each paper is either sparse regression or kernel regression, in combination with one or more of the techniques of neural networks, Jacobian regularization, and early stopping.

The first three papers cover sparse regression, the process of increasing interpretability by excluding parameters. Papers I and II combine explicit, sparse regularization with neural network regression to obtain interpretable, yet flexible, models. In Paper I, adaptive lasso is used with a neural network to obtain an additive model. In Paper II, group lasso is used together with an autoencoder to obtain sparse, non-linear dimensionality reduction. In Paper III, an iterative method mimicking elastic net regularization is proposed, replacing explicit regularization with early stopping, and thus considerably increasing the computational speed.

Table 5.1: Categorization of the five papers included in the thesis. Note that Papers II and V occur twice. The papers either focus on sparse or kernel regression, in combination with neural networks, Jacobian regularization, or early stopping.

	Sparse Regression	Kernel Regression
Neural Network Regression	Paper I Paper II	Paper V
Jacobian Regularization	Paper II	Paper IV
Early Stopping as Regularization	Paper III	Paper V

The last two papers cover kernel regression. In Paper IV, we use Jacobian regularization to propose a closed-form, computationally efficient bandwidth selection heuristic for kernel regression with the Gaussian kernel. In Paper V, we investigate using gradient-based optimization methods for kernel regression. The effect of this is two-fold: First, it allows for decreasing training time by replacing explicit regularization with early stopping. Second, it allows for updating the kernel during training, obtaining a more flexible method, resembling the behavior of neural networks. Since Paper V is quite rich in content, it will be split into two papers, one covering regularization through early stopping, and one covering non-constant kernels.

5.1 Paper I

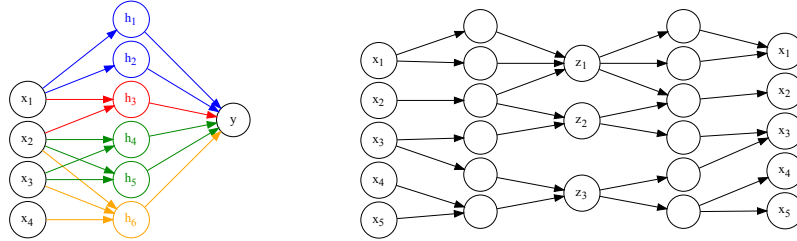
We investigate how a sparsely regularized neural network can be used as a non-parametric additive model by eliminating less important links. When using one hidden layer and one output node, a sparse neural network can be interpreted as an additive model, as sketched in Figure 5.1a, where the network corresponds to

$$\mathbb{E}[y] = f_1(x_1) + f_2(x_1, x_2) + f_3(x_2, x_3) + f_4(x_2, x_3, x_4).$$

In this representation, f_1 is realized by the hidden nodes h_1 and h_2 , f_2 is realized by h_3 , f_3 by h_4 and h_5 , and f_4 by h_6 .

In order to introduce as little bias as possible, adaptive, rather than standard, lasso, is used, something we show is crucial for obtaining satisfactory results. The sparse network is obtained through proximal gradient descent, a method that is not compatible with accelerated gradient descent methods such as RMSProp and Adam. However, the training time is heavily reduced by a multi-stage training algorithm, which uses Adam in the initial stages of training, and does not apply proximal gradient descent until in the last stage.

We apply the algorithm to a data set with air pollution in the U.K., presented in Wood et al. [2017], and compare it to the model suggested by the authors, where the functions were manually selected. We conclude that our algorithm selects similar functions with the same predictive performance. Replacing the adaptive lasso with standard lasso results in functions with more input variables, thus being less interpretable.



(a) Sketch of the network in Paper I

(b) Sketch of the network in Paper II

Figure 5.1: Sketches of network architectures produced Papers I and II. Figure 5.1a shows a neural network with one hidden layer, regularized with the adaptive lasso. The four subfunctions are color-coded to increase readability. In Figure 5.1b, a path lasso regularized autoencoder is sketched. The z -nodes are connected only to a subset of the x -nodes and vice versa.

Main Contributions

- Demonstration of how a sparsely regularized neural network can be used for non-parametric additive modeling.
- Introduction of a computationally efficient, yet accurate, method based on adaptive lasso and proximal gradient descent.
- Demonstration of the superiority of adaptive lasso compared to standard lasso.

5.2 Paper II

We introduce path lasso, which builds on the group lasso penalty and penalizes the number of connected nodes in two non-adjacent layers of a feed-forward neural network. As an example, in the sketch in Figure 5.1b, the only node in the third layer connected to node x_1 is z_1 .

We define the path lasso matrix between layers k and l according to

$$\mathbf{W}_{\text{PL}} := \sqrt{(\mathbf{W}_l)^2 \cdot (\mathbf{W}_{l-1})^2 \cdot \dots \cdot (\mathbf{W}_{k+1})^2},$$

where the square and the square root are taken element-wise. Then $\mathbf{W}_{\text{PL}} \in \mathbb{R}^{p_l \times p_k}$, where p_k and p_l are the number of nodes in layers k and l . Thus, each element in \mathbf{W}_{PL} corresponds to one node in layer k and one node in layer l . We show that $(\mathbf{W}_{\text{PL}})_{i_l i_k}$ is the group lasso penalty of all paths connecting node i_k in layer k to node i_l in layer l , and that $(\mathbf{W}_{\text{PL}})_{i_l i_k} = 0$ implies that the corresponding element of the Jacobian is zero, i.e. that the two nodes are independent. We also show how the penalty on a path, consisting of multiple links between two nodes, can be translated to individual penalties on the links using non-negative matrix factorization.

We introduce a multi-stage training algorithm where, in the initial stages of training, Adam is used, and where proximal gradient descent and matrix factorization are not applied until the last stage, thus reducing the computational cost.

We apply path lasso to an autoencoder, as sketched in Figure 5.1b, to obtain non-linear, sparse dimensionality reduction. Compared to sparse principal component analysis, our algorithm is more flexible, which allows it to use the latent space more efficiently. Compared to a standard autoencoder, it is more interpretable since each latent dimension is a function only of a subset of the original dimensions.

We evaluate the path lasso penalized autoencoder on synthetic, text, and image data.

Main Contributions

- Introduction of path lasso, a penalty that penalizes connections between nodes in non-adjacent layers in feed-forward neural networks.
- Theoretical justification of path lasso.
- Introduction of a computationally efficient, yet accurate, method based on group lasso, proximal gradient descent, and non-negative matrix factorization.

5.3 Paper III

Similar to how the elastic net generalizes both lasso and ridge regression, we combine coordinate and gradient descent into an iterative optimization algorithm that we call elastic gradient descent. The update direction of elastic gradient descent is given by

$$\mathbf{I}_{\text{egd}}(\alpha) \left(\alpha \cdot \text{sign}(\nabla f(\hat{\boldsymbol{\theta}})) + (1 - \alpha) \cdot \nabla f(\hat{\boldsymbol{\theta}}) \right) \quad \alpha \in [0, 1],$$

where \mathbf{I}_{egd} is a diagonal matrix with zeros and ones on the diagonal, such that

$$\mathbf{I}_{\text{egd}}(\alpha)_{pp} = \begin{cases} 1 & \text{if } |\nabla f(\boldsymbol{\theta})_p| \geq \alpha \cdot \max_p |\nabla f(\boldsymbol{\theta})_p| \\ 0 & \text{otherwise.} \end{cases}$$

With this definition, $\mathbf{I}_{\text{egd}}(\alpha = 0)$ is the identity matrix, and $\mathbf{I}_{\text{egd}}(\alpha = 1)$ is a matrix with $\mathbf{I}_{\text{egd}}(\alpha = 1)_{mm} = 1$ and remaining elements 0, for $m := \arg\max_p |\nabla f(\boldsymbol{\theta})_p|$. Thus, for $\alpha = 0$ elastic gradient descent is exactly gradient descent, and for $\alpha = 1$ elastic gradient descent is exactly coordinate descent.

We construct a continuous time limit of elastic gradient, which we refer to as elastic gradient flow in analogy with gradient flow. We use elastic gradient flow for theoretical comparisons to the elastic net and confirm the similarities of the two algorithms on real and synthetic data. We also show on real and synthetic data that compared to the elastic net, elastic gradient descent is several orders of magnitude faster. Compared to coordinate descent, it selects a model with considerably lower prediction and estimation errors, although still sparse.

Main Contributions

- Construction of elastic gradient descent.
- Construction of a continuous time limit, from which a deeper theoretical understanding is obtained.
- Demonstration of the superior computational speed of elastic gradient descent compared to the elastic net.

5.4 Paper IV

We derive an approximate expression for the Jacobian of a function inferred by kernel ridge regression with the Gaussian kernel. We show that this expression has a local (or, in the absence of regularization, a global) minimum in the kernel bandwidth, and derive a closed-form expression for the bandwidth for which this minimum is obtained. Since smaller derivatives result in a more stable function, we use this expression as a feather-light bandwidth selection criterion for kernel ridge regression with the Gaussian kernel.

We compare our method to cross-validation, maximum marginal likelihood estimation, and Silverman's rule of thumb [Silverman, 2018] on real and synthetic data. Compared to cross-validation and maximum marginal likelihood estimation, our method is several orders of magnitude faster and much more stable in terms of bandwidth selection, with approximately the same performance. Compared to Silverman's method, our method performs significantly better.

Main Contributions

- Derivation of an approximate expression for the Jacobian of functions inferred by kernel ridge regression with the Gaussian kernel.
- Derivation of a closed-form, computationally feather-light, bandwidth selection method for kernel ridge regression with the Gaussian kernel, based on controlling the approximate Jacobian.
- Demonstration of the superior stability and computational speed of Jacobian-based bandwidth selection compared to cross-validation and marginal likelihood maximization.

5.5 Paper V

Kernel ridge regression is generally expressed as

$$\hat{\alpha} = \operatorname{argmin}_{\alpha \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{y} - \mathbf{K}\alpha\|_2^2 + \frac{\lambda}{2} \|\alpha\|_{\mathbf{K}}^2 = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y}.$$

We show that it can equivalently be expressed as

$$\hat{\alpha} = \operatorname{argmin}_{\alpha \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{y} - \mathbf{K}\alpha\|_{\mathbf{K}^{-1}}^2 + \frac{\lambda}{2} \|\alpha\|_2^2 = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{y},$$

where the weighted norm is moved from the regularization term to the reconstruction term.

We then define kernel gradient descent as solving

$$\operatorname{argmin}_{\alpha \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{y} - \mathbf{K}\alpha\|_{\mathbf{K}^{-1}}^2$$

with gradient descent, starting at $\hat{\alpha} = \mathbf{0}$, thus obtaining regularization through early stopping rather than by explicitly penalizing the norm of α . We compare the continuous time limit of kernel gradient descent, which we call kernel gradient flow, to kernel ridge regression, bounding the difference between the two algorithms.

We use the equivalent formulation of kernel ridge regression to obtain ℓ_1 - and ℓ_∞ -regularized kernel regression, which produces signal-driven, and robust, kernel regression, respectively. We further utilize the fact that analogously to the similarities between kernel ridge regression and kernel gradient flow, the solutions obtained when using these penalties are very similar to those obtained from forward stagewise regression (also known as coordinate descent), and sign gradient descent, in combination with early stopping. Thus, the need for computationally heavy proximal gradient descent algorithms can thus be alleviated. We demonstrate this on synthetic data.

We also introduce kernel gradient descent with time-dependent translational-invariant kernels, where the bandwidth is decreased to zero during training, and theoretically address the effects this has on generalization. A decreasing bandwidth circumvents the need for hyperparameter selection and enables both zero training error and a double descent behavior. These phenomena do not occur for kernel ridge regression with constant bandwidth but are known to appear for neural networks. We compare kernel gradient descent with constant

and decreasing bandwidths on real and synthetic data, demonstrating how the latter is able to capture different lengths-scales in the data, thus increasing performance.

Since the paper is quite rich in content, it will be split into two papers, one covering regularization through early stopping, and one covering non-constant kernels.

Main Contributions

- Formulations of equivalent objective functions for kernel ridge regression.
- Establishing connections between explicitly regularized kernel regression and gradient-based optimization with early stopping.
- Demonstration of the superior computational speed of kernel regression with early stopping, in comparison to explicit regularization.
- Introduction and theoretical justification of kernel gradient descent for non-constant kernels.
- Demonstration of the superior performance of kernel regression with decreasing, compared to constant, bandwidth.
- Demonstration of a double descent behavior for kernel regression with decreasing bandwidth.

6 Conclusions and Future Perspectives

This thesis investigates how to construct regression models that combine flexibility with interpretability and computational efficiency. The key elements are carefully selected regularization and effective hyperparameter selection.

With properly selected regularization, flexible, non-linear models can be made more interpretable. We demonstrate this by using sparse regularization in the context of neural networks. By applying the regularization in stages, we are able to construct computationally efficient algorithms that converge to expressive, yet informative, local optima.

Regularization can also be used as a guide for hyperparameter selection. We demonstrate this for kernel ridge regression, where a computationally feather-light bandwidth selection method is formulated with inspiration from Jacobian regularization.

Selecting an optimal hyperparameter for the regularization strength tends to be computationally expensive, since the model usually needs to be trained once for every candidate value. However, using iterative, gradient-based optimization methods with early stopping instead of explicit regularization, the computational cost can be reduced substantially. We demonstrate this for the elastic net and for kernel regression.

Hyperparameter selection can also be made more efficient by, instead of using a fixed value, integrating over multiple hyperparameter values. We show how changing the bandwidth during training for kernel regression eliminates the need for evaluating different candidate bandwidths. Furthermore, this results in a more flexible model, with a behavior similar to that of neural networks, than when using a constant bandwidth.

Approaching hyperparameter selection through optimization opens up for both more efficient algorithms and a deeper theoretical understanding of the training of complex, non-linear models. An interesting line of future research would be to further explore the connections between explicit regularization and gradient-based optimization with early stopping, trying to connect each penalty to a corresponding optimization algorithm. The connection between kernel gradient descent with a non-constant kernel and neural networks would also be interesting to investigate deeper since it could probably be used to obtain a better understanding of neural networks.

Bibliography

- Alnur Ali, J Zico Kolter, and Ryan J Tibshirani. A continuous-time view of early stopping for least squares regression. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1370–1378. PMLR, 2019.
- Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- Augustin-Louis Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *C. R. Acad. Sci.*, 25:536–538, 1847.
- George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.
- Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *Annals of statistics*, 32(2):407–499, 2004.
- Leonhard Euler. *Institutiones calculi integralis*, volume 1. impensis Academiae imperialis scientiarum, 1792.
- Carl Friedrich Gauss. *Theoria Motus Corporum Coelestium In Sectionibus Conicis Solem Ambientium*. Perthes et Besser, 1809.
- Carl-Friedrich Gauss. *Theoria Combinationis Observationum Erroribus Minimis Obnoxiae*. Henricus Dieterich, 1823.
- Gene H Golub, Michael Heath, and Grace Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979.
- Trevor Hastie, Jonathan Taylor, Robert Tibshirani, Guenther Walther, et al. Forward stagewise regression and the monotone lasso. *Electronic Journal of Statistics*, 1:1–29, 2007.

- Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. 2012. URL <http://www.cs.toronto.edu/~hinton/coursera/lecture6/lec6.pdf>.
- Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.
- Daniel Jakubovitz and Raja Giryes. Improving dnn robustness to adversarial attacks using jacobian regularization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 514–529, 2018.
- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.
- Adrien Marie Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes; par AM Legendre...* chez Firmin Didot, libraire pour lew mathématiques, la marine, l . . . , 1806.
- Andrei Andreevich Markov. Zakon bolshikh chisel i sposob naimenshikh kvadratov.(izvlechenie iz pisem aa markova k av vasilevu). *Izvestiya fiz.-mat. obschestva Kazan univ.*(2), 8:110–128, 1899.
- Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- James Mercer. Xvi. functions of positive and negative type, and their connection the theory of integral equations. *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, 209 (441-458):415–446, 1909.
- Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence $o(1/k^2)$. In *Doklady an ussr*, volume 269, pages 543–547, 1983.
- Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and trends® in Optimization*, 1(3):127–239, 2014.
- Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.
- R Tyrrell Rockafellar. Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898, 1976.

- Craig Saunders, Alexander Gammerman, and Volodya Vovk. Ridge regression learning algorithm in dual variables. *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.
- Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.
- Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.
- Vladimir Vovk. Kernel ridge regression. In *Empirical inference*, pages 105–116. Springer, 2013.
- Paul Werbos. Beyond regression:" new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.
- Simon N Wood, Zheyuan Li, Gavin Shaddick, and Nicole H Augustin. Generalized additive models for gigadata: modeling the uk black smoke network daily data. *Journal of the American Statistical Association*, 112(519):1199–1210, 2017.
- Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.
- Yang Zhou, Rong Jin, and Steven Chu-Hong Hoi. Exclusive lasso for multi-task feature selection. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 988–995, 2010.
- Hui Zou. The adaptive lasso and its oracle properties. *Journal of the American statistical association*, 101(476):1418–1429, 2006.
- Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2): 301–320, 2005.
- Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. *Journal of computational and graphical statistics*, 15(2):265–286, 2006.

