

# Neural Network Training and Inversion with a Bregman Learning Framework



**Xiaoyu Wang**

Department of Applied Mathematics and Theoretical Physics  
University of Cambridge

This dissertation is submitted for the degree of  
*Doctor of Philosophy*

Fitzwilliam College

March 2023



## Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. I further state that no substantial part of my thesis has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee.

Xiaoyu Wang  
March 2023



# Abstract

Deep Neural Networks (DNNs) are powerful computing systems that have revolutionised a wide range of research domains and have achieved remarkable success in various real-world applications over the past decade. Despite their significant recent advancements, training DNNs still remains a challenging task due to the non-convex and (potentially) non-smooth nature of the objective function. Back-propagation in combination with gradient-based minimisation approaches has been the predominant strategy for training DNNs for decades. Yet the popular error backpropagation algorithm is susceptible to potential drawbacks and limitations, for example its non-parallelisability and biological implausibility, and vanishing or exploding gradients issues, etc.

Inverting DNNs to infer likely inputs of the system from given outputs, is the other side of the same coin. Early ideas of DNNs inversion trace back to the 1990s, but research interests in more generic network inversion problems have been rekindled and primarily driven due to the rapid advancements in generative modelling in recent years. While several approaches for the inversion of DNNs have been proposed, the stability of the inversion is an often neglected crucial aspect. The neural network inversion problem is ill-posed as the solution does not depend continuously on the input datum hence can be highly sensitive to perturbations.

The core theme of this thesis is at the training of DNNs. Built up on distributed optimisation approaches, this work contributes to both the learning problems and the inversion problems of DNNs. In particular, we propose a lifted Bregman learning framework that goes beyond the classical back-propagation approach, and aims to address unresolved and overlooked issues in training and the inversion of DNNs.

More specifically, we propose a family of loss (penalty) functions that are based on a tailored Bregman distance. We provide detailed mathematical analysis on the derived Bregman learning framework and propose a whole range of deterministic and stochastic optimisation strategies to enable solving the learning problem. Bringing techniques and tools from Inverse Problems and Regularisation Theory, we provide theoretical guarantees as well as computational optimisation strategies for the stable, model-based inversion of neural networks.



## Acknowledgements

The journey of my PhD has been truly fulfilling and incredible, and I owe it all to the amazing support of so many wonderful people. I want to express my heartfelt gratitude to each and every one of you who played a part in this unforgettable journey.

I am immensely grateful to my supervisors, Martin Benning and Carola-Bibiane Schönlieb. Martin, I cannot thank you enough for your firm support, patience, and encouragement. It has been an incredible stroke of luck to have crossed path with you. The wealth of knowledge I have gained from you throughout this experience has been invaluable. Carola, I want to express my deep gratitude for your guidance and unwavering support whenever I needed it. Your belief in me and affirmations have been instrumental. Both of you have not only been outstanding mentors but also true and supportive friends.

I would like to extend my heartfelt appreciation to Hamza Fawzi and Matthias Ehrhardt for reviewing the thesis and for the engaging and fruitful discussions that enriched my viva examination.

I am truly grateful for the funding of my PhD provided by the Cantab Capital Institute for the Mathematics of Information, University of Cambridge, and Fitzwilliam College, which made my research possible.

To all my friends and colleagues at the Cambridge Centre for Analysis and Cambridge Image Analysis group, especially Angelica Aviles-Rivero and Yury Korolev. Thank you for your utmost care and encouragement. I will cherish all the joyful moments of the dinner gatherings and garden parties we had.

I want to wholeheartedly express my sincere gratitude to all the wonderful friends I made at Cambridge, especially my close friends at Fitz. It is here that I came to understand the profound truth that "it's all about the people." The bonds we have formed have led to lifelong friendships and beautiful memories. I am deeply thankful for each of my close friends, for accompanying me and always being there for me. I am truly thankful for your presence in my life.

Last but definitely not least, I am infinitely thankful to my parents, Wang Ying and Liu Weixin. Thank you for your patient guidance, love, and care. I could not have

achieved this without you. Your understanding and belief in me, as well as standing by my decisions on this journey, mean the world to me. Thank you from the bottom of my heart.

Life is indeed a tapestry of serendipitous moments. Looking back, I would never have imagined meeting my ten-year-ago self and saying, "This is where you'll be in a decade." Yet, here I stand, in awe of the remarkable journey that has unfolded. I am deeply grateful to all who have made this adventure unforgettable. Every ending holds the seed of a new beginning, and I am excited to see what awaits.

# Table of contents

|                                                                        |             |
|------------------------------------------------------------------------|-------------|
| <b>List of figures</b>                                                 | <b>xiii</b> |
| <b>1 Introduction</b>                                                  | <b>1</b>    |
| <b>2 Preliminaries</b>                                                 | <b>9</b>    |
| 1 Convex Analysis . . . . .                                            | 9           |
| 2 Optimisation Methods . . . . .                                       | 17          |
| 2.1 Gradient Descent Method . . . . .                                  | 18          |
| 2.2 Subgradient Method . . . . .                                       | 24          |
| 2.3 Proximal Gradient Method . . . . .                                 | 27          |
| 2.4 Alternating Minimisation . . . . .                                 | 32          |
| 2.5 Saddle Point Method . . . . .                                      | 34          |
| 2.6 Stochastic Gradient Method . . . . .                               | 41          |
| 3 Inverse Problems and Regularisation Theory . . . . .                 | 46          |
| 3.1 Inverse Problems . . . . .                                         | 46          |
| 3.2 Variational Regularisation Model . . . . .                         | 47          |
| 3.3 Non-linear Regularisation Analysis . . . . .                       | 48          |
| 4 Machine Learning and Deep Learning . . . . .                         | 53          |
| 4.1 Machine Learning . . . . .                                         | 53          |
| 4.2 Deep Learning and Artificial Neural Networks . . . . .             | 57          |
| <b>3 Generalised Perceptron Learning</b>                               | <b>69</b>   |
| 1 Rosenblatt's Perceptron . . . . .                                    | 71          |
| 1.1 Convergence of Rosenblatt's Perceptron Learning Rule . . . . .     | 72          |
| 2 Adaptive Linear Neuron (ADALINE) . . . . .                           | 74          |
| 3 Generalised Perceptrons . . . . .                                    | 75          |
| 3.1 Delta Rule of ADALINE . . . . .                                    | 77          |
| 4 Learning Perceptrons via Minimising Bregman Loss Functions . . . . . | 78          |

|          |                                                                |            |
|----------|----------------------------------------------------------------|------------|
| 4.1      | A Motivational Example . . . . .                               | 79         |
| 5        | Application and Numerical Experiments . . . . .                | 83         |
| 6        | Discussion . . . . .                                           | 85         |
| <b>4</b> | <b>Lifted Bregman Training of Neural Networks</b>              | <b>87</b>  |
| 1        | A Little History on Training Neural Networks . . . . .         | 89         |
| 2        | Training Feedforward Neural Networks . . . . .                 | 91         |
| 2.1      | First-order Methods and Back-propagation . . . . .             | 91         |
| 2.2      | Distributed Optimisation Approaches . . . . .                  | 95         |
| 3        | Lifted Bregman Training Framework . . . . .                    | 99         |
| 3.1      | The Bregman Penalty Function . . . . .                         | 101        |
| 3.2      | Lifted Bregman Training . . . . .                              | 107        |
| 3.3      | Related Works . . . . .                                        | 107        |
| 3.4      | Training Residual Neural Network . . . . .                     | 109        |
| 4        | Numerical Realisation . . . . .                                | 111        |
| 4.1      | Deterministic Approaches . . . . .                             | 111        |
| 4.2      | Stochastic Approaches . . . . .                                | 120        |
| 5        | Example Problems . . . . .                                     | 125        |
| 5.1      | Classification . . . . .                                       | 125        |
| 5.2      | Sparse Autoencoder . . . . .                                   | 126        |
| 5.3      | Sparse Denoising Autoencoder . . . . .                         | 126        |
| 6        | Numerical Results . . . . .                                    | 127        |
| 6.1      | Classification . . . . .                                       | 127        |
| 6.2      | Sparse Autoencoder . . . . .                                   | 129        |
| 6.3      | Sparse Denoising Autoencoder . . . . .                         | 134        |
| 7        | Discussion . . . . .                                           | 144        |
| <b>5</b> | <b>Lifted Bregman Inversion of Neural Networks</b>             | <b>149</b> |
| 1        | Introduction . . . . .                                         | 151        |
| 2        | Model-based Inversion of Feedforward Neural Networks . . . . . | 152        |
| 3        | Convergence Analysis and Error Estimates . . . . .             | 154        |
| 3.1      | Well-definedness . . . . .                                     | 154        |
| 3.2      | Error Estimates . . . . .                                      | 156        |
| 4        | Computational Implementation . . . . .                         | 161        |
| 4.1      | Inverting Perceptrons . . . . .                                | 161        |
| 4.2      | Inverting Feedforward Neural Networks . . . . .                | 163        |
| 5        | Numerical Results . . . . .                                    | 164        |

---

|          |                                                       |            |
|----------|-------------------------------------------------------|------------|
| 5.1      | Perceptrons . . . . .                                 | 165        |
| 5.2      | Feedforward Neural Networks . . . . .                 | 169        |
| 6        | Discussion . . . . .                                  | 173        |
| <b>6</b> | <b>Conclusion and Outlook</b>                         | <b>175</b> |
| 1        | Summary . . . . .                                     | 175        |
| 1.1      | Generalised Perceptron Learning . . . . .             | 176        |
| 1.2      | Lifted Bregman Training of Neural Networks . . . . .  | 176        |
| 1.3      | Lifted Bregman Inversion of Neural Networks . . . . . | 177        |
| 2        | Outlook . . . . .                                     | 178        |
| 3        | Concluding Remarks . . . . .                          | 180        |
|          | <b>References</b>                                     | <b>181</b> |



# List of figures

|     |                                                                                                     |     |
|-----|-----------------------------------------------------------------------------------------------------|-----|
| 2.1 | Convex sets and convex functions . . . . .                                                          | 11  |
| 2.2 | Visualisation of the Bregman distance . . . . .                                                     | 13  |
| 2.3 | Bias-variance decomposition . . . . .                                                               | 56  |
| 2.4 | Heaviside step and hyperbolic tangent activation function . . . . .                                 | 60  |
| 2.5 | ReLU and soft-thresholding activation function . . . . .                                            | 62  |
| 2.6 | Visualisation of a standard feedforward neural network . . . . .                                    | 64  |
| 2.7 | Visualisation of LeNet-5 . . . . .                                                                  | 65  |
| 2.8 | Visualisation of an autoencoder . . . . .                                                           | 66  |
| 2.9 | Visualisation of a residual block . . . . .                                                         | 67  |
| 3.1 | Rosenblatt's perceptron . . . . .                                                                   | 71  |
| 3.2 | Schematic of ADALINE . . . . .                                                                      | 75  |
| 3.3 | Proximal activation functions . . . . .                                                             | 79  |
| 3.4 | Sample Fashion-MNIST images . . . . .                                                               | 84  |
| 3.5 | Convergence comparison of the Rosenblatt-ISTA Algorithm with baseline approaches . . . . .          | 86  |
| 4.1 | The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) . . . . .                            | 90  |
| 4.2 | Bregman (penalty) loss functions . . . . .                                                          | 106 |
| 4.3 | Convergence comparison of sparse autoencoder trained on MNIST-1K dataset . . . . .                  | 131 |
| 4.4 | Convergence comparison of sparse autoencoder trained on MNIST-1K dataset for 750 epochs . . . . .   | 133 |
| 4.5 | Visualisation of sparse autoencoder reconstructed training sample images                            | 134 |
| 4.6 | Visualisation of sparse autoencoder reconstructed validation sample images                          | 134 |
| 4.7 | Convergence comparison of sparse denoising autoencoder trained on Fashion-MNIST-1K images . . . . . | 136 |

|      |                                                                                                                       |     |
|------|-----------------------------------------------------------------------------------------------------------------------|-----|
| 4.8  | Convergence comparison of sparse denoising autoencoder trained on Fashion-MNIST-10K images . . . . .                  | 137 |
| 4.9  | Convergence comparison of sparse denoising autoencoders trained on Fashion-MNIST-10K images for 1500 epochs . . . . . | 139 |
| 4.10 | Convergence comparison of sparse denoising autoencoders with under-complete autoencoders . . . . .                    | 140 |
| 4.11 | Visualisation of sparse denoising autoencoder denoised training sample images . . . . .                               | 141 |
| 4.12 | Visualisation of sparse denoising autoencoder denoised validation sample images . . . . .                             | 141 |
| 4.13 | Visualisation of the sparse code . . . . .                                                                            | 142 |
| 4.14 | Visualisation of denoised images from the Fashion-MNIST-10K training dataset . . . . .                                | 143 |
| 4.15 | Visualisation of denoised images from the Fashion-MNIST-10K validation dataset . . . . .                              | 143 |
| 4.16 | More visualisation of the denoised images from the MNIST dataset . . . . .                                            | 145 |
| 4.17 | More visualisation of the denoised images from the Fashion-MNIST-1K dataset . . . . .                                 | 146 |
| 4.18 | More visualisation of the denoised images from the Fashion-MNIST-10K dataset . . . . .                                | 147 |
| 5.1  | Visualisation of noisy data $y^\delta$ . . . . .                                                                      | 165 |
| 5.2  | Groundtruth solution circle image . . . . .                                                                           | 166 |
| 5.3  | Reconstructed circle image via Landweber iterative regularisation . . . . .                                           | 166 |
| 5.4  | Reconstructed circle image via TV regularisation . . . . .                                                            | 166 |
| 5.5  | Reconstructed MNIST training and validation sample images from the perceptron inversion . . . . .                     | 167 |
| 5.6  | Reconstructed training sample images from the blurring autoencoder inversion . . . . .                                | 168 |
| 5.7  | Reconstructed validation sample images from the blurring autoencoder inversion . . . . .                              | 168 |
| 5.8  | Reconstructed training and validation sample images from the convolutional autoencoder inversion . . . . .            | 171 |
| 5.9  | Visual comparison between reconstructed and decoded image against noise                                               | 172 |
| 5.10 | Comparison of the PSNR value between reconstructed and decoded image against noise . . . . .                          | 172 |

# Chapter 1

## Introduction

Deep Neural Networks (DNNs) are powerful computing systems that have revolutionised a wide range of research domains and have achieved remarkable success in various real-world applications including computer vision, natural language processing, speech recognition, and scientific discovery over the past decade (Bronstein et al., 2017; Devlin et al., 2018; Dosovitskiy et al., 2020; Goodfellow et al., 2016a; He et al., 2016; Jumper et al., 2021; Krizhevsky et al., 2017; OpenAI, 2023; Silver et al., 2016). As the title suggests, the core theme of this thesis is at the training of neural networks. In what follows, we will divide our discussion with respect to training DNNs into two main aspects: the forward learning problem and the backward learning problem. While the forward learning problem is concerned with predicting or inferring outcomes with given inputs, the backward learning problem seeks to determine the likely input or possible cause to have produced an observation or consequence.

### The Forward-direction Learning Problem

On a generic level in the supervised learning setting, the goal of learning neural networks in the forward direction is to find a mapping between observed data sample pairs. For such forward learning problems, a collection of labeled training data  $\{(x, y)\}$  is provided. We are interested in computationally optimising the parameters of a neural network  $\mathcal{N}_{\Theta}$ , such that it can accurately predict the output  $y$  for new, unseen data.

$$y = \mathcal{N}_{\Theta}(x)$$

The forward learning problem is of interest to many classification and regression problems and is seen the most among machine learning tasks. Despite significant

advancements in recent years, learning deep neural networks in the forward direction remains a challenging task due to the non-convex and potentially non-smooth nature of the objective function.

Back-propagation (Rumelhart et al., 1986) in combination with gradient-based minimisation approaches (Duchi et al., 2011; Kingma and Welling, 2013; Robbins and Monro, 1951; Tieleman et al., 2012; Zeiler, 2012) has been the predominant strategy for training artificial neural networks for several decades. Thanks to the modern automatic differentiation framework, back-propagation is made easy to implement and its effectiveness has been showcased by the accomplishments in a great variety of applications. Despite its convenience and popularity, the error back-propagation algorithm has several issues that can limit its performance and applicability.

- The first issue is the inability of the gradient based error backpropagation framework to handle non-differentiable nonlinearities such as those arise from discontinuous activation functions. This means subgradient, instead of gradient based algorithms are required, leading to the disadvantage of inferior convergence rates (Bertsekas, 1999; Chambolle and Pock, 2016a; Polyak, 1987).
- The second significant issue is the vanishing and exploding gradient problem. During back-propagation, gradients are computed backwards from the last output layer to the initial input layer of the network. This mechanism can result in the computed gradients becoming vanishingly small or exponentially large by applying the chain rule recursively through multiple layers of a deep neural network or via backpropagation through time in recurrent neural networks. As a result, updating parameters in the initial layers can be difficult and ineffective, leading to slow convergence or unstable training performances (Bengio et al., 1994; Goodfellow et al., 2016a; Hochreiter and Schmidhuber, 1997; Pascanu et al., 2013; Riedmiller and Braun, 1993).
- The third issue is that the current error back-propagation framework is not straightforward to parallelise. This is in particular challenging when it comes to updating parameters across different layers of the neural network. Because of the nature of the back-propagation algorithm, computations in one layer need to be completed before moving on to the next layer, which can limit the performance on parallel computing architectures (Huo et al., 2018; Zhuang et al., 2021).
- Last but not least, the algorithm’s biological implausibility has been a topic of debate in the field of neuroscience (Bartunov et al., 2018; Guerguiev et al., 2017;

Krotov and Hopfield, 2019; Lee et al., 2015; Meulemans et al., 2020; Millidge et al., 2022; Sacramento et al., 2018; Sjöström et al., 2010). Back-propagation requires neurons to compute derivatives with respect to a global error signal and propagate through each layer. However there is little evidence to suggest that such global control or coordination of gradient propagation exists in the brain. While some suggests that neurons can perform some form of error propagation, local error feedback is more biologically realistic.

Several alternative approaches have been proposed in attempt to partially address some aforementioned computational issues. For example, the introduction of Rprop (Riedmiller and Braun, 1993) to improve the back-propagation algorithm; the use of non-saturating activation functions such as the rectified linear units (ReLU) (Nair and Hinton, 2010) or its variants to avoid vanishing gradient at certain layers; and the introduction of skip-connections to inspire new architectural designs (He et al., 2015a; Ronneberger et al., 2015; Srivastava et al., 2015) for improved training performance. Nevertheless, the challenge of computing gradients with respect to a highly nested objective function still persists as a fundamental problem.

On the other hand, multiple works have proposed distributed training schemes to enable parallel network parameters updates. This includes several works on model parallelism where the entire neural network is split across different computing devices, where each device is responsible for learning a subset of the network (Huang et al., 2019; Narayanan et al., 2019; Zhuang et al., 2021). In spite of its successful application on training gigantic neural networks, such as in natural language processing or computer vision applications, model parallelism requires careful partitioning of the network parameters and costly communications between devices.

## The Backward-direction Learning Problem

The backward learning problem is concerned with learning neural networks in the backward direction, or the inversion problem of neural networks. We are interested in obtaining a plausible solution  $x$  that could have produced the observation  $y$ . However, in finding an inverted neural network, the goal of acquiring an inverse mapping of the neural network  $\mathcal{N}_{\Theta}^{-1}$  is challenging, as the inversion problem is often ill-conditioned.

$$x = \mathcal{N}_{\Theta}^{-1}(y)$$

The concept of inverting neural networks can be traced back to the 1990s (Jensen et al., 1999; Kindermann and Linden, 1990; Linden and Kindermann, 1989; Lu et al., 1999; Ogawa et al., 1998). More recently, learning in the backward direction has received much attention for optimising with respect to network inputs. In particular, finding adversarial examples in the input space, i.e. generating neural network inputs that are difficult for the network to classify correctly. This area of research aims for providing better understandings of the trained network’s robustness and generalisation properties (Goodfellow et al., 2014; Moosavi-Dezfooli et al., 2016; Szegedy et al., 2013). Nevertheless, research interests in more generic neural network inversion problems over the last several years have been primarily driven due to the advancements in generative modelling (Antoniou et al., 2017; Dinh et al., 2015; Frid-Adar et al., 2018; Ho et al., 2020; Kingma and Welling, 2013; Rezende and Mohamed, 2015; Sohl-Dickstein et al., 2015; Wang et al., 2018; Zhu et al., 2017).

While several approaches for inverting neural networks have been proposed in particular in the context of generative modelling (Behrmann et al., 2019, 2021; Gal et al., 2022; Xia et al., 2022), an important aspect that is often neglected is the stability of the inversion. In the sense of Hadmard (Hadamard, 1902), the well-posedness of a problem requires the solution to the problem to depend continuously on the input datum. The inversion of neural networks does not necessarily guarantee stability and the process can be highly sensitive and unstable when faced with minor changes in the input datum.

Regularisation theory has been an essential tool for addressing ill-posedness in inverse problems. It emphasises on finding stable approximation of ill-conditioned inverses through regularisations (Benning and Burger, 2018; Engl et al., 1996; Scherzer et al., 2009). In the work of Mahendran and Vedaldi (2015), an inversion approach is proposed in the context of understanding image representations. The authors aim to infer the network input from the network output with total variation regularised inversion. However, this work only provides empirical evidence and lacks a rigorous proof that the proposed approach is a variational regularisation.

While there has also been substantial work that focuses on utilising neural networks as nonlinear operators in variational regularisation methods (Arridge et al., 2019; Li et al., 2020; Lunz et al., 2018; Mukherjee et al., 2021; Schwab et al., 2019), these approaches are often coupled with forward learning problems where the back-propagation algorithm is used. Therefore, the aforementioned computational issues with forward learning problems continue to exist in these formulations.

## Thesis Contributions

Lately, there has been an increasing trend of exploring distributed optimisation approaches in the context of training neural networks (Askari et al., 2018; Carreira-Perpinan and Wang, 2014; Choromanska et al., 2019; Gu et al., 2020; Høier and Zach, 2020; Lau et al., 2018; Li et al., 2019; Taylor et al., 2016; Zach and Estellers, 2019; Zeng et al., 2019; Zhang and Brand, 2017). However, many of these approaches still suffer from certain limitations such as differentiating non-differentiable non-linearities (Carreira-Perpinan and Wang, 2014), trained affine-linear networks (Askari et al., 2018), or overly restrictive assumptions (Gu et al., 2020; Li et al., 2019).

Extending this line of research, this thesis is a contribution to both the forward-direction learning and the backward-direction learning problems. In particular, we propose a Bregman learning framework that goes beyond the classical back-propagation framework and aims to address several unresolved and overlooked issues in the training and inversion of neural networks as discussed in the previous sections. This thesis consists of three main chapters, where in each chapter we focus on different aspects of the proposed Bregman learning framework.

- **Chapter 3** establishes a key observation that commonly used activation functions can in fact also be interpreted as proximal mappings. We propose a loss function that is based on tailored Bregman distances and analyse properties of the proposed Bregman loss function. We expand the discussion to the learning problem of a generalised perceptron model using the proposed Bregman loss function and confirm the commonly held belief that the original Rosenblatt’s perceptron learning algorithm is an incremental gradient method with respect to a particular Bregman loss function. This energy minimisation interpretation also generalises the classical ADALINE delta rule. Through numerical results we demonstrate that the proposed approach outperforms standard training approaches. This chapter is based on the joint work with Martin Benning and resulted in Wang and Benning (2020).
- **Chapter 4** establishes the lifted Bregman framework for the training of multi-layer neural networks. This framework generalises the Method of Auxiliary Coordinates and provides an alternative learning approach than using the back-propagation based (sub-)gradient methods over a nested objective function. We provide detailed mathematical analysis on the Bregman penalty function and propose a wide range of deterministic and stochastic optimisation strategies

that are able to computationally solve the training problem. We show that the proposed approach has an additional advantage on solving regularised learning problems with explicit variational regularisation imposed on hidden variables of the network, by presenting superior learning performances of the proposed framework on learning sparse (denoising) autoencoders with sparse codes. We demonstrate that in contrary to wide-spread belief, expressive feedforward networks can overfit to training data without generalising well when non-standard optimisation methods are being used. This chapter is based on the joint work with Martin Benning and resulted in Wang and Benning (2022).

- **Chapter 5** establishes the lifted Bregman formulation for the regularised inversion of neural networks. We propose a family of variational regularisations based on a tailored Bregman distance and provide theoretical guarantees for the stable, model-based inversion of neural networks. In particular, we prove that for the single-layer perceptron case, the proposed model is a convergent variational regularisation method and provide general error estimates as well as error estimates for a concrete perceptron case. We also propose proximal first-order optimisation strategies for solving the variational regularisation method for multi-layer neural networks and present several numerical examples that support the effectiveness of the proposed model-based regularisation approach. This chapter is based on the joint work with Martin Benning and resulted in Wang and Benning (2023).

## Thesis Overview

In **Chapter 2**, we introduce adequate mathematical background and major tools for the main topics in this thesis, these include the following three areas 1) convex analysis and optimisation algorithms, 2) inverse problems and regularisation theory, and 3) machine learning and deep neural networks. The remainder of this thesis is structured as follows.

In **Chapter 3**, we begin with a short review of the history of perceptron learning and examine the original Rosenblatt’s perceptron learning algorithm as well as the Widrow-Hoff delta rule. After formulating the generalised perceptron learning problem, we introduce the Bregman loss function that is based on the generalised Bregman distance and analyse its favorable properties. We next demonstrate an energy minimisation interpretation of the Rosenblatt’s learning algorithm and how it unifies the two historical approaches. We then consider an example application of learning sparse perceptrons and present comparison against standard approaches in the numerical results section.

In **Chapter 4**, we first describe historical approaches of training neural networks, including the development of the error back-propagation algorithm and an overview of recent developments in distributed optimisation training methods. We then introduce our proposed lifted Bregman framework and provide thorough analysis of a variety of mathematical properties of the proposed Bregman penalty function which guarantee the advantages over other distributed learning approaches. We next provide an overview of a wide variety of deterministic and stochastic optimisation approaches to computationally solve the lifted Bregman training problem as well as a discussion on suitable regularisation strategies for neural networks. After describing several example problems, we provide numerical results and an extensive comparison against more conventional approaches.

In **Chapter 5**, we begin by introducing the lifted Bregman formulation for the model-based inversion of feedforward neural networks. Then we present convergence result for the regularised inversion of perceptrons and provide general error estimates, followed by error estimates for a concrete perceptron case. Next, we discuss implementation details of the proposed variational regularisation scheme and algorithmic approaches for solving the inversion problem for both the single-layer perceptron and the multi-layer neural network setting. Through numerical results, we demonstrate empirically that the proposed model-based regularisation approach is indeed effective.

In **Chapter 6**, we conclude the thesis with a summary of key results and contributions, followed by a discussion of possible extensions and future works relating to this thesis.



# Chapter 2

## Preliminaries

A major theme of this thesis is the formulation of a novel framework for learning artificial neural networks and the analysis of its mathematical properties. To lay foundations for later chapters, in this chapter we aim to provide an introduction to the mathematical tools and theories ranging from the topics of convex analysis, optimisation, inverse problems and regularisation theory as well as important notions and concepts from machine learning and deep learning.

This chapter is structured as follows. Section 1 covers a range of notions and definitions that will later be used in this thesis. In Section 2 we expand on optimisation problems and introduce algorithmic approaches for solving them. Section 3 covers the topic of inverse problems as well as reviewing concepts from regularisation theory. Finally in Section 4 we review notions from machine learning, deep learning and artificial neural networks. We assume that readers are familiar with some basic knowledge of analysis, linear algebra and probability.

### 1 Convex Analysis

We will begin by reviewing basic notions and definitions in convex analysis, which will be helpful for discussions in later sections. This section is mainly based on materials from Beck (2017) and Boyd et al. (2004).

We will begin with defining some key assumptions for the functions we frequently use throughout the rest of the sections.

**Definition 1.1** (Proper function). *A function is proper if it is not identical to  $+\infty$  and nowhere  $-\infty$ .*

In other words, its effective domain  $\text{dom}(f)$  is not empty. The effective domain of a function  $f$  is defined as follows.

**Definition 1.2** (Effective domain). *The effective domain of a function  $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$  is defined as*

$$\text{dom}(f) := \{x \in \mathbb{R}^n \mid f(x) < \infty\}.$$

Another important assumption is lower semi-continuity, which is defined as follows.

**Definition 1.3** (Lower semi-continuity). *A function  $f$  is lower semi-continuous (l.s.c.) if, for all  $x \in \text{dom}(f)$ , if  $x_n \rightarrow x$ , then*

$$f(x) \leq \liminf_{n \rightarrow \infty} f(x_n).$$

Another useful notion we recall here is the epigraph of a function.

**Definition 1.4** (Epigraph). *The epigraph of a function  $f$  is defined as*

$$\text{epi } f := \{(x, t) \in \text{dom}(f) \times \mathbb{R} \mid t \geq f(x)\}.$$

Using the notion of epigraph we can characterise the closedness of a function  $f$ .

**Definition 1.5** (Closed function). *A function  $f$  is said to be closed if its epigraph is closed.*

**Remark 1.1.** *A necessary and sufficient condition for a function  $f$  to be closed is that lower semi-continuity holds for each points in the domain.*

We will next recall the definition of convexity. We begin by defining a convex set, which is characterised as follows.

**Definition 1.6** (Convex set). *A convex set  $\Omega \subset \mathbb{R}^n$  has the property that*

$$x, y \in \Omega \iff (1 - \alpha)x + \alpha y \in \Omega \quad \text{for all } \alpha \in [0, 1].$$

In other words, for all pairs of points  $(x, y)$  be contained in a convex set  $\Omega$ , the line segment between  $x$  and  $y$  must also be contained in  $\Omega$ . As visualised in Figure 2.1, the polygon is a convex set. A closely related notion is the concept of a convex hull.

**Definition 1.7** (Convex hull). *For any set  $S$ , the convex hull of  $S$  is the smallest convex set enclosing  $S$ .*

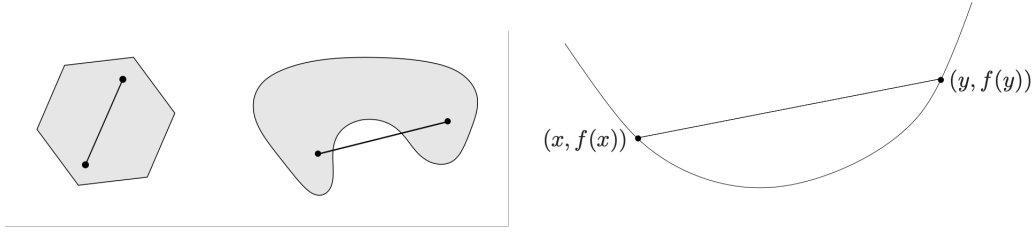


Fig. 2.1 **Left:** A convex set. **Middle:** A non-convex set. **Right:** A convex function. Figures taken from Boyd et al. (2004).

Convex functions play an important role in the study of optimisation problems. Here we define convex functions with the following characterisation.

**Definition 1.8** (Convex function). *A function  $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$  is called convex if it satisfies the following*

$$f((1 - \alpha)x + \alpha y) \leq (1 - \alpha)f(x) + \alpha f(y), \quad \text{for all } x, y \in \mathbb{R}^n \text{ such that } x \neq y, \alpha \in (0, 1).$$

*In addition, we say  $f$  is strictly convex if the above inequality holds strictly.*

Geometrically, this states that the line segment connecting any two points  $(x, f(x))$  and  $(y, f(y))$  should lie entirely above the graph of the function  $y$ , as visualised in Figure 2.1. Alternatively, convex functions can also be characterised by the convexity of their epigraph. A function is convex if and only if its epigraph, as defined in Definition 1.4, is a convex set.

**Remark 1.2.** *A proper convex function is closed if and only if it is lower semi-continuous.*

For the ease of notation, we define the following set of frequently used functions that are proper, convex and lower semi-continuous:

$$\Gamma_0 := \{f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\} \mid f \text{ is proper, convex and lower semi-continuous.}\} \quad (2.1)$$

We will next recall subdifferentiability and subgradients as they provide important characterisations of minimisers when solving non-smooth minimisation problems.

**Definition 1.9** (Subdifferential). *Let  $f \in \Gamma_0$ . We say that the function  $f$  is subdifferentiable at  $x \in \mathbb{R}^n$  if there exists  $p \in \mathbb{R}^n$ , such that*

$$f(y) \geq f(x) + \langle p, y - x \rangle$$

holds for all  $y \in \mathbb{R}^n$ . Furthermore,  $p$  is called a subgradient at the point  $x$ , whereas the collection of all subgradients at point  $x$ :

$$\partial f(x) := \{p \in \mathbb{R}^n \mid f(y) \geq f(x) + \langle p, y - x \rangle \text{ for all } y \in \mathbb{R}^n\}$$

is called the subdifferential of  $f$  at  $x$ .

**Remark 1.3** (Fermat's theorem of stationary points). *Using the subgradient notion, we can generalise Fermat's stationary conditions to non-smooth convex functions where  $x \in \mathbb{R}^n$  is a global minimiser of  $f$  if and only if  $0 \in \partial f(x)$  (cf. (Beck, 2017)).*

Another important definition we would like to recall here is the Fenchel conjugate.

**Definition 1.10** (Fenchel conjugate). *Let  $f \in \Gamma_0$ . Then the Fenchel- or convex-conjugate  $f^* : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$  is defined as*

$$f^*(y) := \sup_{x \in \mathbb{R}^n} \{\langle x, y \rangle - f(x)\}.$$

The conjugate operation can be invoked twice, which results in the biconjugate operation.

**Definition 1.11** (Biconjugate). *For a proper function  $f$ , its biconjugate is defined as follows*

$$f^{**}(x) := \sup_{y \in \mathbb{R}^n} \{\langle x, y \rangle - f^*(y)\}.$$

In particular, for  $f \in \Gamma_0$ , we have  $f = f^{**}$  (cf. (Beck, 2017, Theorem 4.8)).

Subgradients of convex conjugates can be conveniently linked with the following Theorem.

**Theorem 1.1** (Conjugate subgradient theorem). *Let  $f \in \Gamma_0$ . The following three claims are equivalent for any  $x \in \mathbb{R}^n$  and  $p \in \mathbb{R}^n$ .*

- $\langle x, p \rangle = f(x) + f^*(p)$
- $p \in \partial f(x)$
- $x \in \partial f^*(p)$

*Proof.* The proof for this theorem can be found in Beck (2017, Theorem 4.20).

The Bregman distance (Benning and Riis, 2021; Bregman, 1967; Burger, 2016; Censor and Lent, 1981; Kiwi, 1997) is a vital concept in this thesis. We will next recall the generalised Bregman distance for subdifferentiable functions.

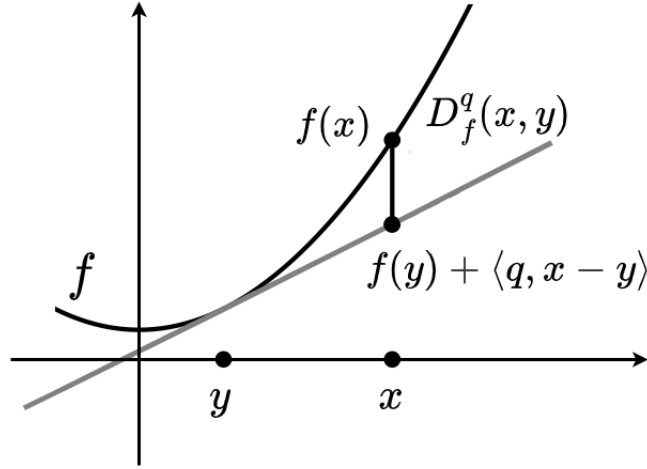


Fig. 2.2 Visualisation of the Bregman distance  $D_f^q(x, y)$ .

**Definition 1.12** (Generalised Bregman distance). *Let  $f \in \Gamma_0$ . The generalised Bregman distance of  $f$  is defined as*

$$D_f^q(x, y) := f(x) - f(y) - \langle q, x - y \rangle,$$

for all  $x \in \mathbb{R}^n$  and  $y \in \text{dom}(f)$ . Here,  $q \in \partial f(y)$  is a subgradient of  $f$  at argument  $y$ .

Figure 2.2 shows a graphical illustration of the Bregman distance between two points  $x$  and  $y$ .

**Remark 1.4** (Bregman distance conjugate characterisation). *The generalised Bregman distance can be alternatively characterised with the following definition. Follows directly from (Rockafellar, 1970, Theorem 23.5), we can apply Theorem 1.1 and substitute  $f(y) = \langle q, y \rangle - f^*(q)$  to obtain:*

$$D_f^q(x, y) = f(x) + f^*(q) - \langle x, q \rangle.$$

Notice that Bregman distances are similar to metrics, yet they do not satisfy the triangle inequality nor symmetry. However, one can obtain a symmetric Bregman distance with the following.

**Definition 1.13** (Symmetric Bregman distance). *Let  $f \in \Gamma_0$ . The symmetric Bregman distance or Jeffreys distance (Jeffreys, 1946) between two vectors  $x$  and  $\tilde{x}$  is defined as*

the sum of two Bregman distances with interchanged arguments, i.e.

$$D_f^{symm}(x, y) := D_f^p(x, y) + D_f^q(y, x) = \langle x - y, p - q \rangle,$$

for  $x, y \in \text{dom}(f)$  with  $p \in \partial f(x)$  and  $q \in \partial f(y)$ .

As a renown bounded symmetrisation of the Kullback–Leibler divergence, the Jensen–Shannon divergence generalises to the Burbea–Rao divergence, which is defined by the following.

**Definition 1.14** (Burbea–Rao divergence). *Let  $F : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$  be in  $\Gamma_0$ . The Burbea–Rao divergence (Burbea and Rao, 1982a,b; Nielsen and Boltz, 2011) is defined as*

$$J_F(x, y) := \frac{1}{2} \left( F(x) + F(y) - 2F\left(\frac{x+y}{2}\right) \right), \quad (2.2)$$

for all  $x, y \in \text{dom}(F)$ .

Another key notion we would like to review is smoothness. But before that, let us first recap the definition of Lipschitz-continuity.

**Definition 1.15** (Lipschitz continuity). *For an operator  $G : \mathbb{R}^n \rightarrow \mathbb{R}^m$ , it is said to be Lipschitz continuous if there exists a non-negative constant  $L$  such that for all  $x, y \in \mathbb{R}^n$ , the following holds*

$$\|G(x) - G(y)\| \leq L\|x - y\|.$$

For continuously differentiable functions with Lipschitz continuous gradients, we can characterise them with the following definition of smoothness.

**Definition 1.16** ( $L$ -smoothness). *We say that a continuously differentiable function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is  $L$ -smooth, or has  $L$ -Lipschitz gradients, if its gradient  $\nabla f : \mathbb{R}^n \rightarrow \mathbb{R}^n$  is Lipschitz with Lipschitz constant  $L$ , i.e.*

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|, \quad \text{for all } x, y \in \mathbb{R}^n.$$

Given an  $L$ -smooth function  $f$ , an immediate consequence we obtain is characterised by the following Lemma, where it asserts that  $f$  can be upper-bounded by a certain quadratic function.

**Lemma 1.1** (Descent lemma). *For  $L$ -smooth function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , we have for any  $x, y \in \mathbb{R}^n$  that*

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2} \|y - x\|^2.$$

*We can conveniently characterise the descent lemma using the Bregman distance:*

$$D_f^{\nabla f(x)}(y, x) = f(y) - f(x) - \langle \nabla f(x), y - x \rangle \leq \frac{L}{2} \|y - x\|^2.$$

*Proof.* The proof of this statement can be found in Beck (2017, Lemma 5.7).

On the other hand, if there exists a quadratic lower bound on the growth of a function  $f$ , we can characterise  $f$  with the notion of strong convexity.

**Definition 1.17** (Strongly convex function). *A differentiable function  $f$  is called strongly convex with parameter  $m > 0$  if the following inequality holds for all points  $x, y$  in its domain:*

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{m}{2} \|y - x\|^2.$$

*Using the notion of Bregman distance, the above condition can be equivalent written as:*

$$D_f^{\nabla f(x)}(y, x) = f(y) - f(x) - \langle \nabla f(x), y - x \rangle \geq \frac{m}{2} \|y - x\|^2.$$

**Remark 1.5.** *Strong convexity does not necessarily require the function to be differentiable. When  $f$  is non-smooth,  $\nabla f(x)$  is replaced with a subgradient of  $f$  at point  $x$ .*

The following theorem establishes a connection between smoothness and strong convexity.

**Theorem 1.2** (Conjugate correspondence theorem). *Let  $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$  be an extended real-valued function and  $f^* : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$  be the conjugate function of  $f$ , let  $\tau > 0$ , then the connection between smoothness and strong convexity can be characterised by the following*

- *If  $f$  is a  $\frac{1}{\tau}$ -smooth convex function, then  $f^*$  is  $\tau$ -strongly convex with respect to the dual norm  $\|\cdot\|_*$ .*
- *If  $f$  is a proper closed  $\tau$ -strongly convex function with respect to the norm  $\|\cdot\|$ , then  $f^*$  is  $\frac{1}{\tau}$ -smooth.*

*Proof.* The proof of this theorem can be found in Beck (2017, Theorem 5.26).

Proximal mapping, or also referred to as Moreau's proximal mapping (Moreau, 1962, 1965), is a crucial concept for many algorithms that we will discuss in the next section. We will first recall the definition of this operator before we recap some of its key properties.

**Definition 1.18** (Proximal map). *The proximal map  $\text{prox}_{\mu f} : \mathbb{R}^n \rightarrow \text{dom}(f) \subset \mathbb{R}^n$  of a function  $f \in \Gamma_0 : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$  is defined as*

$$\text{prox}_{\mu f}(x) := \arg \min_{y \in \mathbb{R}^n} \left\{ f(y) + \frac{1}{2\mu} \|x - y\|^2 \right\} = (I + \mu \partial f)^{-1}(x). \quad (2.3)$$

where  $(I + \mu \partial f)^{-1}$  is known as the resolvent of operator  $\partial f$  with parameter  $\mu > 0$ .

As an example, for a characteristic or indicator function of a closed, convex and nonempty set  $C$ , defined by

$$\delta_C(x) = \begin{cases} 0 & \text{if } x \in C, \\ +\infty & \text{otherwise.} \end{cases} \quad (2.4)$$

The proximal mapping of the characteristic function is the orthogonal projection operator onto the set  $C$ , i.e.

$$\text{prox}_f(x) := \arg \min_{y \in \mathbb{R}^n} \left\{ \delta_C(y) + \frac{1}{2} \|x - y\|^2 \right\} = \arg \min_{y \in C} \frac{1}{2} \|y - x\|^2 = P_C(x),$$

where  $P$  here denotes the orthogonal projection operator. Another notable example is the proximal mapping of the  $\ell_1$ -norm, where in this case the proximal map is given by the soft-thresholding operator. More details will be discussed in Section 2.3.

A key property of the proximal mapping is the Moreau decomposition theorem, which connects the notion of proximal maps and convex conjugates.

**Theorem 1.3** (Moreau decomposition). *Let  $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$  be in  $\Gamma_0$  and  $f^*$  denotes its Fenchel conjugate. Then*

$$x = \text{prox}_f(x) + \text{prox}_{f^*}(x)$$

holds true for all  $x \in \mathbb{R}^n$ .

*Proof.* The proof for this theorem can be found in Beck (2017, Theorem 6.45).

Next we define Moreau-Yosida regularisation (Moreau, 1965; Yosida, 1964), also known as the Moreau envelope of a function  $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$ .

**Definition 1.19** (Moreau-Yosida regularisation). *Let  $f \in \Gamma_0$ . The Moreau-Yosida regularisation of  $f$ , with  $\mu > 0$  is defined as*

$$M_f^\mu(x) = \min_{y \in \mathbb{R}^n} \left\{ f(y) + \frac{1}{2\mu} \|x - y\|_2^2 \right\}. \quad (2.5)$$

where  $\mu$  is called the smoothing parameter (Beck, 2017).

Notice that the minimisation problem (2.5) has a unique solution, given by  $\text{prox}_{\mu f}(x)$ .

The following well-known lemma characterises the differentiability of the Moreau-Yosida regularisation.

**Lemma 1.2** (Smoothness of Moreau-Yosida regularisation). *Let  $f \in \Gamma_0$  and  $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$ , and  $\mu > 0$ . Then  $M_f^\mu(x)$  is  $\frac{1}{\mu}$ -smooth and the gradient of the Moreau-Yosida regularisation of  $f$  can be computed by  $\nabla M_f^\mu(x) = \frac{1}{\mu}(x - \text{prox}_{\mu f}(x))$ .*

*Proof.* The proof for this statement can be found in Bauschke et al. (2011, Proposition 12.30) and Beck (2017, Theorem 6.60).

Finally, the following two properties will come in handy when we analyse algorithms in the next section.

**Definition 1.20** (Monotone Operator). *An operator  $G : \mathcal{X} \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$  is called a monotone operator if*

$$\langle G(x) - G(y), x - y \rangle \geq 0 \quad \text{for all } x, y \in \mathcal{X}.$$

**Definition 1.21** (Non-expansiveness). *Recall Definition 1.15, for a Lipschitz continuous operator  $G : \mathcal{X} \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$  where*

$$\|G(x) - G(y)\| \leq L\|x - y\| \quad \text{for all } x, y \in \mathcal{X},$$

*we say  $G$  is non-expansive when  $L = 1$ . On the other hand, if  $L < 1$ , we say  $G$  is a contraction with factor  $L$ .*

## 2 Optimisation Methods

In this section, we review several classical algorithmic approaches when it comes to computationally solving optimisation problems. For the scope of this thesis, we will mainly limit our discussion to first-order optimisation methods. The discussion in

---

**Algorithm 1:** Gradient Descent Method

---

Initialize  $x^0$  and  $\tau_0$ .

**for**  $k = 0, 1, 2, \dots$  **do**

$$x^{k+1} = x^k - \tau_k \nabla f(x^k).$$

**end**


---

this section is mainly based on materials from Beck (2017); Benning and Riis (2021); Chambolle and Pock (2016a); Parikh and Boyd (2014); Wright and Recht (2022).

## 2.1 Gradient Descent Method

We begin our discussion with the classical gradient descent method. The gradient descent method has been a long studied algorithmic approach (Courant, 1943; Hadamard, 1908; Lemaréchal, 2012). Although simple in its form, the gradient descent method is an effective tools for solving minimisation problems when one has access to the gradient information (Kantorovich and Akilov, 2016). To begin with, we would like to first consider the unconstrained minimisation of a smooth function  $f(x)$ , i.e.

$$\min_{x \in \mathbb{R}^n} f(x) \tag{2.6}$$

where we assume Lipschitz continuity on its gradient (Definition 1.16).

As summarised in Algorithm 1, the gradient descent method is defined by the recursive iterates  $x^{k+1} = x^k - \tau_k \nabla f(x^k)$ , where  $\tau_k > 0$  denotes the step size. This method generally works well if we know that  $f$  is sufficiently smooth. Hence if we assume  $f$  is  $L$ -smooth with  $L \geq 0$ , by the Descent lemma 1.1, when choosing  $y = x - \tau \nabla f(x)$ , we obtain the following:

$$f(x - \tau \nabla f(x)) \leq f(x) - \tau \|\nabla f(x)\|^2 + \tau^2 \frac{L}{2} \|\nabla f(x)\|^2. \tag{2.7}$$

Notice that (2.7) quantifies the amount of decrease after one iterate, hence we can optimise the right hand side of (2.7) with respect to  $\tau$ . Then one has to ensure  $0 = -\|\nabla f(x)\|^2 + \tau L \|\nabla f(x)\|^2$ , which then implies the minimising  $\tau^* = \frac{1}{L}$ . Substituting this choice of  $\tau$  into (2.7), we obtain:

$$f(x^{k+1}) \leq f(x^k) - \frac{1}{2L} \|\nabla f(x^k)\|^2, \tag{2.8}$$

which gives us a key inequality for the convergence analysis of gradient descent method. If we assume convexity of  $f$  as defined in Definition 1.8 in addition to  $f$  being  $L$ -smooth, we can show the following convergence result for the gradient descent method with the step size choice  $\tau = \frac{1}{L}$ .

**Theorem 2.1** (Convergence of gradient descent method). *Suppose  $f$  is convex and  $L$ -smooth, and suppose there exists a minimiser  $x^*$  to Problem (2.6), where the optimal value is defined as  $f(x^*)$ . Then the gradient descent method with step size choice  $\tau = 1/L$  generates a sequence  $\{x^k\}_{k=0}^\infty$  that satisfies*

$$f(x^k) - f(x^*) \leq \frac{L}{2k} \|x^0 - x^*\|^2, \quad k = 1, 2, \dots$$

*Proof.* By the definition of convexity of  $f$ , we have  $f(x^*) \geq f(x^k) + \langle \nabla f(x^k), x^* - x^k \rangle$ . Substituted into (2.8), we have:

$$\begin{aligned} f(x^{k+1}) &\leq f(x^*) + \langle \nabla f(x^k), x^* - x^k \rangle - \frac{1}{2L} \|\nabla f(x^k)\|^2 \\ &= f(x^*) + \frac{L}{2} \left( \|x^k - x^*\|^2 - \|x^k - x^* - \frac{1}{L} \nabla f(x^k)\|^2 \right) \\ &= f(x^*) + \frac{L}{2} \left( \|x^k - x^*\|^2 - \|x^{k+1} - x^*\|^2 \right). \end{aligned}$$

If we sum over  $k$  iterations, we obtain:

$$\begin{aligned} \sum_{n=0}^k (f(x^{n+1}) - f(x^*)) &\leq \frac{L}{2} \sum_{n=0}^k \left( \|x^n - x^*\|^2 - \|x^{n+1} - x^*\|^2 \right) \\ &= \frac{L}{2} \left( \|x^0 - x^*\|^2 - \|x^{k+1} - x^*\|^2 \right) \\ &\leq \frac{L}{2} \|x^0 - x^*\|^2. \end{aligned}$$

Since  $\{f(x^k)\}$  is a non-decreasing sequence we have:

$$f(x^k) - f(x^*) \leq \frac{1}{k} \sum_{n=0}^k (f(x^n) - f(x^*)) \leq \frac{L}{2k} \|x^0 - x^*\|^2.$$

□

An improved convergence result can be shown if we further assume strong convexity of  $f$ . Recall Definition 1.17, a strongly convex function  $f$  can be lower bounded by quadratic functions. We will begin with the following lemma, which will come in handy

when proving a linear convergence rate when applying gradient descent to a strongly convex function  $f$ .

**Lemma 2.1.** *Let  $f$  be a continuously differentiable and strongly convex function with parameter  $m$  and denote the unique minimiser of Problem (2.6) by  $x^*$ , then we have*

$$\begin{aligned} f(x) - f(x^*) &\leq \frac{\|\nabla f(x)\|^2}{2m}, \\ \|x - x^*\| &\leq \frac{2}{m} \|\nabla f(x)\|. \end{aligned}$$

*Proof.* From the definition of strong convexity we have:

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{m}{2} \|y - x\|^2. \quad (2.9)$$

Now if we fix  $x$  and minimise both sides of the inequality with respect to  $y$  the inequality still holds. It is straightforward to see that the left hand side attains its minimum at  $y = x^*$ . The right hand side attains its minimum when  $y = x - \frac{\nabla f(x)}{m}$ . Plugging these choices of  $y$  back into (2.9) we get:

$$\begin{aligned} f(x^*) &\geq f(x) - \langle \nabla f(x), \frac{1}{m} \nabla f(x) \rangle + \frac{m}{2} \left\| \frac{1}{m} \nabla f(x) \right\|^2 \\ &\geq f(x) - \frac{1}{2m} \|\nabla f(x)\|^2, \end{aligned}$$

hence after rearranging the terms we get  $\|\nabla f(x)\|^2 \geq 2m(f(x) - f(x^*))$ . Similarly we can show:

$$\begin{aligned} f(x^*) &\geq f(x) - \langle \nabla f(x), x^* - x \rangle + \frac{m}{2} \|x - x^*\|^2 \\ &\geq f(x) - \|\nabla f(x)\| \|x^* - x\| + \frac{m}{2} \|x - x^*\|^2, \end{aligned}$$

where the second line uses the Cauchy-Schwarz inequality. After rearranging the terms, we obtain the desired estimate

$$\|x - x^*\| \leq \frac{2}{m} \|\nabla f(x)\|.$$

This concludes the proof. □

**Theorem 2.2** (Linear convergence of gradient descent). *Suppose  $f$  is strongly convex and  $L$ -smooth. Denote the unique minimiser of Problem (2.6) by  $x^*$  and define the optimal function value by  $f(x^*)$ . Then, for the gradient descent method with step size*

---

**Algorithm 2:** Implicit Gradient Descent Method

---

Initialize  $x^0$  and  $\tau$ .**for**  $k = 0, 1, 2, \dots$  **do**

$$x^{k+1} = x^k - \tau \nabla f(x^{k+1}). \quad (2.10)$$

**end**

---

$\tau = 1/L$ , after  $k$  steps we have that

$$f(x^k) - f(x^*) \leq \left(1 - \frac{m}{L}\right)^k (f(x^0) - f(x^*)) ,$$

*Proof* Taking the bound on function value obtained in Lemma 2.1 and substituting into (2.8) gives:

$$\begin{aligned} f(x^{k+1}) &\leq f(x^k) - \frac{1}{2L} \|\nabla f(x^k)\|^2 \\ &\leq f(x^k) - \frac{m}{L} (f(x^k) - f(x^*)) . \end{aligned}$$

Now, if we subtract  $f(x^*)$  from both sides we get:

$$f(x^{k+1}) - f(x^*) \leq \left(1 - \frac{m}{L}\right) (f(x^k) - f(x^*)) .$$

Applying this recursion for  $k$  steps we obtain the desired linear convergence result.  $\square$

### 2.1.1 Implicit Gradient Descent Method

Throughout the analysis in the last section, one key assumption we made is that  $f$  needs to be sufficiently smooth, i.e.  $\nabla f$  is Lipschitz continuous. The step size  $\frac{1}{L}$  is determined by the global Lipschitz constant of the gradient. Without such condition on the smoothness of  $f$ , and for more general descent methods, one needs to choose step size  $\tau_k$  carefully to guarantee convergence at a particular rate. One approach is to use line-search methods for choosing step sizes. In this section, we discuss an alternative option which is to use the implicit gradient descent method. A key motivation for using this method is the less restricted choices of step size. For simplicity, we illustrate the idea by assuming a constant step size  $\tau$  for the rest of this section.

As summarised in Algorithm 2, the implicit gradient descent method is defined by iterate  $x^{k+1} = x^k - \tau \nabla f(x^{k+1})$ . Notice that in the explicit gradient descent method, the gradient is evaluated at the previous iterate  $x^k$ , whereas in the implicit method,

the gradient is evaluated at the current iterate  $x^{k+1}$ , which begs the question on how to evaluate  $\nabla f(x^{k+1})$ .

Observe that for such a  $x^{k+1}$ , we can rewrite (2.10) to

$$\nabla f(x^{k+1}) + \frac{1}{\tau}(x^{k+1} - x^k) = 0 ,$$

which suggests  $x^{k+1}$  is a critical point of the function

$$f(x) + \frac{1}{2\tau}\|x - x^k\|^2 . \quad (2.11)$$

If we remember the definition of proximal maps in Definition 1.18, we see immediately that, for  $f \in \Gamma_0$

$$x \rightarrow \arg \min_x \left\{ f(x) + \frac{1}{2\tau}\|x - x^k\|^2 \right\}$$

defines  $\text{prox}_{\tau f}$  evaluated at  $x^k$  and  $x^{k+1}$  is the unique minimiser.

Note that in this interpretation, we do not require any smoothness assumption on  $f$ . Now further recall the definition of the Moreau-Yosida regularisation of  $f$  with parameter  $\tau$  (Definition 1.19):

$$M_f^\tau(x) = \inf_y \left\{ f(y) + \frac{1}{2\tau}\|x - y\|_2^2 \right\} .$$

We know from Lemma (1.2) that the gradient of the Moreau envelope  $\nabla M_f^\tau(x)$  is  $\frac{1}{\tau}$  Lipschitz and computed as

$$\nabla M_f^\tau(x) = \frac{1}{\tau}(x - \text{prox}_{\tau f}(x)) ,$$

which can further be rewritten as

$$\text{prox}_{\tau f}(x) = x - \tau \nabla M_f^\tau(x) .$$

Hence an iterate of the implicit gradient descent method applied to  $f$  can be written as

$$x^{k+1} = \text{prox}_{\tau f}(x^k) = (I + \partial f)^{-1}(x^k) = x^k - \tau \nabla M_f^\tau(x^k) ,$$

which is indeed an iteration of the explicit gradient descent method on the Moreau envelope  $M_f^\tau$ , for which we know that it has the same minimisers as  $f$ . The convergence analysis of the implicit gradient method is hence similar to what has been shown earlier

in the convergence result of the steepest descent method in Theorem (2.1) but now loosening the restrictions on step size parameter.

The iterate  $x^{k+1} = \text{prox}_{\tau f}(x^k)$  in fact takes form after a more general family of methods called the proximal point method (Martinet, 1970, 1972), which is a special case of the proximal gradient method. We will discuss more on the analysis of proximal gradient method in Section 2.3.

### A Unified Framework with Bregman Distance

Note that with the notion of Bregman distances (Definition 1.12), we can unify both the explicit and the implicit gradient descent method into one framework (Benning and Riis, 2021). Consider the Bregman proximal method (Censor and Zenios, 1992; Chen and Teboulle, 1993; Eckstein, 1993; Teboulle, 1992) described by the following iterates

$$x^{k+1} = \arg \min_x \{f(x) + D_J(x, x^k)\} , \quad (2.12)$$

where  $D_J$  denotes the Bregman distance with the following associating  $J$

$$J(x) = \begin{cases} \frac{1}{\tau} R(x) & \text{implicit} \\ \frac{1}{\tau} R(x) - f(x) & \text{explicit} . \end{cases}$$

Evaluating the Bregman distance with respect to the above choice of  $J$ , we obtain

$$x^{k+1} = \arg \min_x \begin{cases} f(x) + \frac{1}{\tau} D_R(x, x^k) & \text{implicit} \\ f(x^k) + \langle \nabla f(x^k), x - x^k \rangle + \frac{1}{\tau} D_R(x, x^k) & \text{explicit} . \end{cases} \quad (2.13)$$

Note that the classical implicit and explicit gradient descent algorithm are special cases of the formulation in (2.13) with the choice  $R(x) = \frac{1}{2}\|x\|^2$ . Furthermore, the convergence of the Bregman proximal method (2.12) is usually guaranteed when guarantee on the convexity of  $J$ . In the explicit setting, this is equivalent to choosing  $\tau$  and  $R$  so that convexity of  $\frac{1}{\tau}R - f$  is ensured. If  $R$  is a quadratic function, this is equivalent to require  $f$  being  $\frac{1}{\tau}$  smooth (Bauschke et al., 2017; Benning et al., 2021; Bolte et al., 2018).

---

**Algorithm 3:** Subgradient Method

---

Initialize  $x^0$  and  $\tau_0$ .

**for**  $k = 0, 1, 2, \dots$  **do**

$$x^{k+1} = x^k - \tau_k g^k, \quad g^k \in \partial f(x^k). \quad (2.14)$$

**end**


---

## 2.2 Subgradient Method

In this section, we begin the discussion on solving non-smooth minimisation problems, where we will consider unconstrained minimisation of a non-smooth function  $f(x)$ ,

$$\min_{x \in \mathbb{R}^n} f(x). \quad (2.15)$$

As subgradients have generalised the notion of gradients, a natural generalisation of the classical gradient descent method to the non-smooth case when  $f$  is not differentiable results in the subgradient method. The method is described by the iterates  $x^{k+1} = x^k - \tau_k g^k$  where  $g^k$  is a subgradient of  $f$  at point  $x^k$  and  $\tau_k$  is the step size.

Note that a crucial difference here is that for the gradient descent method, a small step in the negative gradient direction will always guarantee decrease in the function value, while this is not necessarily the case for the subgradient method. Hence the subgradient method is not necessarily a descent method.

To analyse convergence of subgradient methods, a key inequality is summarised in the following lemma.

**Lemma 2.2.** *Suppose  $f$  is proper, closed and convex. Denote a minimiser of Problem (2.15) by  $x^*$  and the optimal value of the problem by  $f(x^*)$ . Let  $\{x^k\}$  be the sequence generated by Algorithm 3, then*

$$\|x^{k+1} - x^*\|^2 \leq \|x^k - x^*\|^2 - 2\tau_k(f(x^k) - f(x^*)) + \tau_k^2 \|g^k\|^2. \quad (2.16)$$

*Proof* We can expand  $\|x^{k+1} - x^*\|^2$  by writing

$$\begin{aligned} \|x^{k+1} - x^*\|^2 &= \|x^k - \tau_k g^k - x^*\|^2 \\ &= \|x^k - x^*\|^2 - 2\tau_k \langle g^k, x^k - x^* \rangle + \tau_k^2 \|g^k\|^2 \\ &\leq \|x^k - x^*\|^2 - 2\tau_k(f(x^k) - f(x^*)) + \tau_k^2 \|g^k\|^2, \end{aligned}$$

where the last inequality follows the definition of subgradients (Definition 1.9).  $\square$

In order to show convergence of subgradient methods, we need the subgradients to be bounded in norm. In particular, we assume that there exists some positive constant  $C$  for which  $\|g\| \leq C$  for all  $g \in \partial f(x)$ .

Notice if we optimise the right hand side of (2.16) with respect to  $\tau_k$ , a natural choice of step size is

$$\tau_k^* = \frac{f(x^k) - f(x^*)}{\|g^k\|^2}, \quad (2.17)$$

where we can further define  $\tau_k = 1$  when  $g^k = 0$ . Choosing step size parameter at  $\tau_k^*$  is an optimal solution and is referred to as Polyak's step size rule (Polyak, 1987). In the next theorem we summarise a general convergence result when following (2.17).

**Theorem 2.3.** *Suppose  $f$  is proper, closed and convex. Denote a minimiser of Problem (2.15) by  $x^*$  and the optimal value of the problem by  $f(x^*)$ . Further assume that there exists some positive constant  $C$  for which  $\|g\| \leq C$  for all  $g \in \partial f(x)$ . Let  $\{x^k\}$  be the sequence generated by the subgradient method (Algorithm 3) using Polyak's step size rule (2.17) and denote  $f_{best}^k = \min_{\{0,1,\dots,k\}} f(x^k)$ , then*

$$\begin{aligned} (1) & \|x^{k+1} - x^*\|^2 \leq \|x^k - x^*\|^2 \\ (2) & f(x^k) \rightarrow f(x^*) \quad \text{as } k \rightarrow \infty \\ (3) & f_{best}^k - f(x^*) \leq \frac{C\|x^0 - x^*\|^2}{\sqrt{k+1}} \end{aligned}$$

*Proof.* A proof to this theorem can be found as a special instance of Beck (2017, Theorem 8.13).

Yet one drawback of the Polyak's step size rule is that it requires  $f(x^*)$  to be known ahead of time. Without a good knowledge or a good estimate of  $f(x^*)$ , this optimal step size is incomputable. However, there are two alternative options: to either use a fixed step size  $\tau$  or a diminishing step size  $\tau_k$ . We will next analyse convergence of Algorithm 3 for these two types of step size choices.

First observe that if we sum up (2.16) over  $k$  iterations, we can get the following bound after rearranging some terms:

$$\sum_{n=0}^k \tau_n (f(x^n) - f(x^*)) \leq \frac{1}{2} \|x^0 - x^*\|^2 + \frac{1}{2} \sum_{n=0}^k \tau_n^2 \|g^n\|^2.$$

Furthermore, if we use the assumption  $\|g^n\| \leq C$  and that  $f(x^n) \geq f_{\text{best}}^k$ , we obtain

$$\left(\sum_{n=0}^k \tau_n\right) (f_{\text{best}}^k - f(x^*)) \leq \frac{1}{2} \|x^0 - x^*\|^2 + \frac{C^2}{2} \sum_{n=0}^k \tau_n^2,$$

therefore

$$f_{\text{best}}^k - f(x^*) \leq \frac{\|x^0 - x^*\|^2}{2 \sum_{n=0}^k \tau_n} + \frac{C^2 \sum_{n=0}^k \tau_n^2}{2 \sum_{n=0}^k \tau_n}.$$

Now observe that if one were to simply use a fix step size  $\tau$ , the optimal function value cannot be achieved in limit as

$$\lim_{k \rightarrow \infty} f_{\text{best}}^k \leq f(x^*) + \frac{\tau C^2}{2}.$$

On the other hand, if we use diminishing step sizes, then the sequence  $\{\tau_n\}$  needs to satisfy the following condition

$$\frac{\sum_{n=0}^k \tau_n^2}{\sum_{n=0}^k \tau_n} \rightarrow 0 \text{ as } k \rightarrow \infty.$$

The convergence result depends on the choices of step size. A typical choice that satisfies the above condition is  $\tau_k = \frac{1}{k+1}$  which is square summable but not summable. Alternatively, we have the following convergence result.

**Theorem 2.4** (Convergence rate of subgradient method). *Suppose  $f$  is proper, closed and convex, and there exist some positive constant  $C$  for which  $\|g\| \leq C$  for all  $g \in \partial f(x)$ . Let  $\{x^k\}$  be the sequence generated by the subgradient method, and the sequence of diminishing step size generated by  $\tau_k = \frac{1}{\|g^k\| \sqrt{k+1}}$ , then*

$$f_{\text{best}}^k - f(x^*) \leq C \frac{1 + \|x^0 - x^*\|^2}{2\sqrt{k+1}}.$$

*Proof.* A detailed proof of the convergence result can be found in Nesterov (2003), see also Chambolle and Pock (2016a) and Beck (2017).

Note that although we obtain convergence under a diminishing step size rule, this comes at the price of slow convergence rate, which is a strong limitation of Algorithm 3. In the next section, we introduce a broad family of algorithms called proximal gradient method. With many classical algorithmic approaches falling under the umbrella of this type of method, the proximal gradient method is better suited for many non-smooth minimisation problems.

### 2.3 Proximal Gradient Method

Throughout this section, we consider the following composite objective

$$\min_{x \in \mathcal{X}} F(x) := f(x) + g(x), \quad (2.18)$$

where  $f$  is a proper, closed and smooth function, while  $g \in \Gamma_0$  is a proper, closed and convex function. Note that in many imaging and machine learning problems, this objective often consists of a smooth part  $f$  and a non-smooth part  $g$ . Recall from earlier discussion in Section 2.1.1, we show that proximal point method (Martinet, 1970, 1972) can be utilised to minimise  $g$  without assuming its smoothness. Indeed, a core idea in the proximal gradient method is to combine an explicit gradient descent step on  $f$  and an implicit step with respect to  $g$ .

To better illustrate the idea behind proximal gradient methods, we first look at the specific example problem

$$\min_{x \in \mathcal{X}} f(x) + \delta_C, \quad (2.19)$$

where  $\delta_C$  is the characteristic function (2.4) over a non-empty closed convex set  $C$ . Equivalently, this is a convex constrained minimisation problem

$$\min_{x \in C} f(x), \quad (2.20)$$

which we can solve via projected gradient descent method (cf. Beck (2017))

$$x^{k+1} = P_C(x^k - \tau_k \nabla f(x^k)),$$

where  $\tau_k$  is the step size at iterate  $k$ . As we have shown in (2.13), an explicit gradient step with step size  $\tau_k$  can be interpreted as finding a point  $x^{k+1}$  that minimises the linearisation of  $f$  around  $x^k$  while staying close to the previous iterate  $x^k$ . The projected gradient step searches for such a point that lies in the set  $C$ , i.e.

$$x^{k+1} = \arg \min_{x \in C} f(x^k) + \langle \nabla f(x^k), x - x^k \rangle + \frac{1}{2\tau_k} \|x - x^k\|^2.$$

Please note that this is equivalent to solving the following convex unconstrained minimisation problem

$$x^{k+1} = \arg \min_{x \in \mathcal{X}} \delta_C(x) + f(x^k) + \langle \nabla f(x^k), x - x^k \rangle + \frac{1}{2\tau_k} \|x - x^k\|^2.$$

---

**Algorithm 4:** Proximal Gradient Method

---

Initialize  $x^0$  and  $\tau_0$ .

**for**  $k = 0, 1, 2, \dots$  **do**

$$x^{k+1} = T_\tau(x^k) = \text{prox}_{\tau_k g}(x^k - \tau_k \nabla f(x^k)).$$

**end**


---

Now consider a more general non-smooth function  $g$  that replaces the characteristic function. After adding and removing constant terms and with some simple algebraic manipulation, we obtain the following

$$x^{k+1} = \arg \min_{x \in \mathcal{X}} \left\{ \tau_k g(x) + \frac{1}{2} \|x - (x^k - \tau_k \nabla f(x^k))\|^2 \right\}, \quad (2.21)$$

where by the definition of the proximal operator (Definition 1.18), (2.21) can be written as

$$x^{k+1} = \text{prox}_{\tau_k g}(x^k - \tau_k \nabla f(x^k)).$$

The above iterate characterises the proximal gradient method (cf. Teboulle (2018)), which is also known under the name forward-backward splitting (Lions and Mercier, 1979). For notation simplicity, we can introduce the prox-grad operator  $T_\tau(x) = \text{prox}_{\tau g}(x - \tau \nabla f(x))$ . The resulting proximal gradient method is more compactly summarised in Algorithm 4.

As shown earlier, solving a convex constrained minimisation problem with the projected gradient descent method is a special use case of the proximal gradient method with  $g$  being the indicator function. Another example of the proximal gradient method is when solving optimisation problems with additional regularisation terms, where  $g$  will be a non-smooth regularisation function. The proximal gradient method comes in handy when it is "simple" in the sense that the proximal mapping has a closed-form that is easy to evaluate. An example of this is the  $\ell_1$ -norm regularised minimisation, where  $g(x) = \lambda \|x\|_1$ . This problem can be approached with the iterative shrinkage thresholding algorithm (ISTA) (Daubechies et al., 2004), where in each iteration the soft-thresholding or shrinkage operator is applied.

We next discuss convergence results of the proximal gradient method. The following theorem summarises a key inequality on which our analysis is based.

**Theorem 2.5.** Suppose that  $f$  is a proper closed and  $L$ -smooth function,  $g$  is a proper closed and convex function. Let  $T_\tau(x) = \text{prox}_{\tau g}(x - \tau \nabla f(x))$ . For any  $x, y$  satisfying

$$f(T_\tau(y)) \leq f(y) + \langle \nabla f(y), T_\tau(y) - y \rangle + \frac{1}{2\tau} \|T_\tau(y) - y\|^2, \quad (2.22)$$

it holds that

$$F(x) - F(T_\tau(y)) \geq \frac{1}{2\tau} \|x - T_\tau(y)\|^2 - \frac{1}{2\tau} \|x - y\|^2 + D_f(x, y), \quad (2.23)$$

where  $D_f(u, v) = f(u) - f(v) - \langle \nabla f(v), u - v \rangle$  is a Bregman distance function (Definition 1.12).

*Proof.* Consider the following  $\frac{1}{\tau}$ -strongly convex function  $\varphi(u)$

$$\varphi(u) = f(y) + g(u) + \langle \nabla f(y), u - y \rangle + \frac{1}{2\tau} \|u - y\|^2.$$

By the definition of strong convexity (Definition 1.17), we have that for any  $x, y$

$$\varphi(y) \geq \varphi(x) + \langle \partial \varphi(x), y - x \rangle + \frac{1}{2\tau} \|x - y\|^2.$$

On the other hand, we know that if  $x^*$  is the unique minimiser of a closed strongly convex function  $\varphi$ , then it follows from Fermat's optimality condition (Remark 1.3) that  $0 \in \partial \varphi(x^*)$ , and we obtain

$$\varphi(x) - \varphi(x^*) \geq \frac{1}{2\tau} \|x - x^*\|^2.$$

Note that as  $T_\tau(y) = \arg \min_u \varphi(u)$ , the following holds:

$$\varphi(x) - \varphi(T_\tau(y)) \geq \frac{1}{2\tau} \|x - T_\tau(y)\|^2. \quad (2.24)$$

Now we can write  $\varphi(T_\tau(y))$  as

$$\begin{aligned} \varphi(T_\tau(y)) &= f(y) + \langle \nabla f(y), T_\tau(y) - y \rangle + \frac{1}{2\tau} \|T_\tau(y) - y\|^2 + g(T_\tau(y)) \\ &\geq f(T_\tau(y)) + g(T_\tau(y)) = F(T_\tau(y)), \end{aligned}$$

where the second inequality follows (2.22). And by (2.24), we know that for any  $x$ ,

$$\varphi(x) - F(T_\tau(y)) \geq \frac{1}{2\tau} \|x - T_\tau(y)\|^2. \quad (2.25)$$

Using the expression of  $\varphi(x)$  we can expand (2.25) to obtain

$$f(y) + \langle \nabla f(y), x - y \rangle + g(x) + \frac{1}{2\tau} \|x - y\|^2 - F(T_\tau(y)) \geq \frac{1}{2\tau} \|x - T_\tau(y)\|^2.$$

After rearranging terms we get

$$F(x) - F(T_\tau(y)) \geq \frac{1}{2\tau} \|x - T_\tau(y)\|^2 - \frac{1}{2\tau} \|x - y\|^2 + D_f(x, y).$$

□

Note that the above result holds without making the assumption on  $f$  being convex. Now using the convexity of  $f$ , we can show an  $O(1/k)$  rate of convergence for the proximal gradient method as described in Algorithm 4.

**Theorem 2.6** (Convergence rate of proximal gradient method). *Suppose  $f$  is a proper, closed, convex and  $L$ -smooth function, and  $g$  is a proper, closed, and convex function. Let  $\{x^k\}$  be the sequence generated by the proximal gradient descent method in Algorithm 4 with constant step size  $\tau = \frac{1}{L}$ . Denote the minimiser of Problem 2.18 with  $x^*$  and the optimal function value by  $F(x^*)$ , then*

$$F(x^k) - F(x^*) \leq \frac{L \|x^0 - x^*\|^2}{2k}. \quad (2.26)$$

*Proof.* Using the key proximal gradient inequality in Theorem 2.5 and take  $x = x^*$ ,  $y = x^n$ . We obtain

$$\begin{aligned} \frac{2}{L} (F(x^*) - F(x^{n+1})) &\geq \|x^* - x^{n+1}\|^2 + \|x^* - x^n\|^2 + \frac{2}{L} D_f(x, x^n) \\ &\geq \|x^* - x^{n+1}\|^2 + \|x^* - x^n\|^2, \end{aligned}$$

where in the second line of the inequality we use the convexity of  $f$ . Summing both sides of the inequality over  $k$  iterations for  $n = 0 \dots k-1$ , we get

$$\frac{2}{L} \sum_{n=0}^{k-1} (F(x^*) - F(x^{n+1})) \geq \|x^* - x^k\|^2 - \|x^* - x^0\|^2.$$

Hence

$$\sum_{n=0}^{k-1} (F(x^{n+1}) - F(x^*)) \leq \frac{L}{2} \|x^* - x^0\|^2 - \frac{L}{2} \|x^* - x^k\|^2 \leq \frac{L}{2} \|x^* - x^0\|^2.$$

Using the key proximal gradient inequality in Theorem 2.5 again, but this time for the choice  $x = y = x^n$ , we obtain

$$F(x^k) - F(x^{k+1}) \geq \frac{L}{2} \|x^k - x^{k+1}\|^2,$$

which in particular suggests  $F(x^k) \geq F(x^{k+1})$ . Hence we validate that the proximal gradient method produces a non-increasing sequence of function values.

By the monotonicity of  $\{F(x^n)\}$ , we can conclude that

$$k(F(x^k) - F(x^*)) \leq \sum_{n=0}^{k-1} (F(x^{n+1}) - F(x^*)) \leq \frac{L}{2} \|x^* - x^0\|^2.$$

Consequently we obtain

$$F(x^k) - F(x^*) \leq \frac{L \|x^* - x^0\|^2}{2k}.$$

□

## Bregman Proximal Gradient Method

Note that we can also consider the problem of minimising the composite objective in (2.18) with the Bregman framework as described in Benning and Riis (2021). Apply the implicit method in (2.13) with  $J(x) = \frac{1}{\tau} \|x\|^2 - f(x)$ , then an iterate of the Bregman proximal method (2.12) can be written down as:

$$\begin{aligned} x^{k+1} &= \arg \min_x \left\{ f(x) + g(x) + D_J(x, x^k) \right\} \\ &= \arg \min_x \left\{ \frac{1}{2} \|x - (x^k - \tau \nabla f(x^k))\|^2 + \tau g(x) \right\} \\ &= (I + \tau g)^{-1} (x^k - \tau \nabla f(x^k)), \end{aligned}$$

which recovers the classical proximal gradient method, and  $(I + \tau g)^{-1}$  is known as the proximal mapping or resolvent (Definition 1.18). More general proximal gradient methods can be derived using alternative choices of  $J$  and  $g$  (Beck, 2017; Benning and Riis, 2021).

With this we conclude the section on proximal gradient method. For more detailed discussion, we refer readers to Beck (2017); Boyd et al. (2004); Lions and Mercier (1979); Parikh and Boyd (2014). In the next section, we will introduce the alternating minimisation strategy, a popular scheme in solving joint minimisation problems.

---

**Algorithm 5:** Alternating Minimisation Method

---

```

Set  $x^{k,0} = x^k$ .
for  $i = 1, 2, \dots, p$  do
     $\tilde{y} \in \arg \min_{y \in \mathbb{R}_i^n} f(x^{k,i-1} + U_i(y - x_i^k))$ .
    Update  $x^{k,i} = x^{k,i-1} + U_i(\tilde{y} - x_i^k)$ .
end
Set  $x^{k+1} = x^{k,p}$ .

```

---

## 2.4 Alternating Minimisation

In this section we introduce an alternating minimisation scheme and discuss joint minimisation problems. Namely, instead of solving for the minimising arguments for all variables all at once, we search for the minimum in one direction at a time. Generally speaking, we will consider the following optimisation problem

$$\min_{x_1 \in \mathbb{R}_1, x_2 \in \mathbb{R}_2, \dots, x_p \in \mathbb{R}_p} f(x_1, x_2, \dots, x_p), \quad (2.27)$$

where  $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$  is a proper function and  $\mathbb{R}^n = \mathbb{R}_1^n \times \mathbb{R}_2^n \dots \times \mathbb{R}_p^n$ . We denote  $x \in \mathbb{R}^n$  by  $x = (x_1, x_2, \dots, x_p)$ , where  $x$  is also partitioned into  $p$  blocks.

In each iteration of the alternating minimisation scheme, we successively choose a block of variables  $x_i$ , and optimise the objective function with respect to the selected block of variables then update these selected ones only. Denote the  $x$  variable in the  $k$ -th iterate by  $x^k = (x_1^k, x_2^k, \dots, x_p^k)$ , the alternating minimisation scheme can be summarised in the following iterate  $x_i^{k+1} \in \arg \min_{x_i \in \mathbb{R}_i^n} f(x_1^{k+1}, x_2^{k+1}, \dots, x_{i-1}^{k+1}, x_i, x_{i+1}^k, \dots, x_p^k)$ .

We adopt the notation from Beck (2017) and introduce the linear transform  $U_i : \mathbb{R}_i^n \rightarrow \mathbb{R}^n$  defined as

$$U_i(y) = (0, \dots, 0, y, 0, \dots, 0),$$

where only the  $i$ -th block is non-zero and set to the value of  $y$ .

If we denote  $x^{k,i-1} = (x_1^{k+1}, x_2^{k+1}, \dots, x_{i-1}^{k+1}, x_i^k, x_{i+1}^k, \dots, x_p^k)$ , then the alternating minimisation scheme can equivalently be summarised in Algorithm 5. As we can see, each iteration involves  $p$  sub-iterations. In the following analysis, we show that under certain assumptions, the sequence generated by the alternating minimisation scheme (Algorithm 5) converges to a coordinate-wise minimum.

**Definition 2.1** (Coordinate-wise minimum). *A point  $x^* \in \mathbb{R}^n$  is called a coordinate-wise minimiser of  $f$ , if*

$$f(x^*) \leq f(x^* + U_i(y)) \quad \text{for all } i = 1, 2, \dots, p \text{ and } y \in \mathbb{R}_i^n.$$

Next, we analyse the convergence of the alternating minimisation scheme with the following composite objective. The problem reads

$$\min_{x_1 \in \mathbb{R}_1^n, x_2 \in \mathbb{R}_2^n, \dots, x_p \in \mathbb{R}_p^n} f(x_1, x_2, \dots, x_p) + \sum_{i=1}^p g_i(x_i). \quad (2.28)$$

Here  $g(x_1, x_2, \dots, x_p) = \sum_{i=1}^p g_i(x_i)$  has a separable structure. In what follows, we will show that points of coordinate-wise minima of Problem (2.28) are necessarily stationary points, which means in the convex setting they will also be global minima points.

**Lemma 2.3.** *Let  $g$  be proper, closed and convex for any  $i = 1, 2, \dots, p$ , and let  $f$  be a closed function and differentiable over the interior of its domain. Denote a coordinate-wise minima point of  $f(x) + g(x)$  as defined in (2.28) by  $x^*$ , then  $x^*$  is a stationary point of Problem (2.28).*

*Proof.* Since  $x^*$  is a coordinate-wise minima point, the following inclusion holds for all  $i = 1, 2, \dots, p$ :

$$x_i^* \in \arg \min_{y \in \mathbb{R}_i^n} \tilde{f}_i(y) + g_i(y),$$

where  $\tilde{f}_i(y) = f(x^* + U_i(y - x_i^*)) = f(x_1^*, x_2^*, \dots, x_{i-1}^*, y, x_{i+1}^*, \dots, x_p^*)$ . We know that if  $x^*$  is a local optimal solution of the composite problem, and  $f$  is differentiable at  $x^*$ , then by Fermat's theorem of stationary points (Remark 1.3) we obtain  $-\nabla \tilde{f}_i(x_i^*) \in \partial g_i(x_i^*)$ . Denote the gradient of  $f$  with respect to the  $i$ -th block by  $\nabla_i f$ . Since  $\nabla \tilde{f}_i(x_i^*) = \nabla_i f(x^*)$ , we have that for all  $i = 1, 2, \dots, p$  and obtain  $-\nabla_i f(x^*) \in g_i(x^*)$ . Hence  $-\nabla f(x^*) \in g(x^*)$ , namely  $x^*$  is a stationary point of the composite problem.  $\square$

Note that in the derivation of the above result we do not assume the convexity of  $f$ . Hence in order for the generated sequence from Algorithm 5 to converge to a stationary point, we need to assume that the problem  $\min_{y \in \mathbb{R}_i^n} f(x + U_i(y - x_i))$  has a unique minimiser, which is a quite strong condition. However, we can remove this condition if the convexity of  $f$  is explicitly assumed. For a more general scheme where we consider  $p$  blocks of partition, one can in fact achieve an  $O(1/k)$  convergence rate using the alternating minimisation scheme. A detailed proof of this convergence rate can be found in Beck (2017) and more discussions can be found in Wright (2015);

Wright and Recht (2022). We also refer interested readers to Fercoq and Richtárik (2015); Lu et al. (2018); Lu and Xiao (2015); Nesterov (2012) for extended discussions.

## 2.5 Saddle Point Method

In this section, we introduce an important class of optimisation methods called the saddle point method, via the use of duality. As discussed earlier in Section 2.3, projected gradient descent or proximal gradient descent method can be applied for solving simple constrained minimisation problems. Saddle point method on the other hand, also works effectively well on constrained optimisation problems, particularly on problems where the constraints assume more complicated structures.

To begin with, consider the following constrained minimisation problem with a linear equality constraint, i.e.

$$\min_x f(x) \quad \text{s.t. } Ax = b. \quad (2.29)$$

One natural approach to solve (2.29) is to relax the constraint by incorporating it into the objective. More specifically, one can introduce a penalty function which adds value to the objective function when the constraint is violated. A typical choice of the penalty function is the quadratic penalty (QP), which gives the following approximation to the original Problem (2.29):

$$\min_x f(x) + \frac{1}{2\tau} \|Ax - b\|^2, \quad (2.30)$$

where  $\tau > 0$  is the parameter that controls the penalty. Problem (2.30) can be solved with penalty methods, where one starts with very little penalty to obtain an initial minimiser of (2.30). Then in each iteration one decreases the penalty parameter, solve (2.30) and use that solution as the initial guess for the next iteration. The generated sequence of solutions will converge to the solution of Problem (2.29) asymptotically.

Note that alternatively, we can formulate the following min-max problem that is equivalent to Problem (2.30):

$$\min_x \max_{\lambda} f(x) + \langle \lambda, Ax - b \rangle - \frac{\tau}{2} \|\lambda\|^2. \quad (2.31)$$

To see the equivalence, we can optimise with respect to  $\lambda$  to obtain  $\lambda^* = -(Ax - b)/\tau$ . Inserting this back into (2.31), we recover (2.30).

In this formulation, one advantage is that we can use a fixed  $\tau$ , instead of decreasing  $\tau$  after each iteration, by explicitly solving for the optimal  $\lambda$  in the inner maximisation

problem. This formulation is also well defined for the limiting case when  $\tau = 0$ , and writes

$$\min_x \max_{\lambda} f(x) + \langle \lambda, Ax - b \rangle. \quad (2.32)$$

Here the function  $\mathcal{L}(x, \lambda) = f(x) + \langle \lambda, Ax - b \rangle$  is called the Lagrangian function associated with the constrained problem (2.29) and  $\lambda$  is known as the Lagrange multiplier.

Note that the above min-max problem (2.32) is equivalent to problem (2.29). To see this, observe that the inner maximisation problem yields  $f(x)$  only when the constraint is satisfied. Otherwise the maximisation with respect to  $\lambda$  will achieve  $\infty$ . In other words, the outer minimisation problem only considers candidate solutions  $x$  that satisfies  $Ax - b = 0$ .

Under mild conditions (Ekeland and Temam, 1999; Rockafellar, 1970; Sion, 1958; v. Neumann, 1928) we can swap the min and max to achieve

$$\max_{\lambda} \min_x f(x) + \langle \lambda, Ax - b \rangle. \quad (2.33)$$

If we denote  $Q(\lambda) = \min_x f(x) + \langle \lambda, Ax - b \rangle$ , the dual problem is known as:  $\max_{\lambda} Q(\lambda)$  and (2.29) is referred to as the primal problem. Now using the Fenchel conjugate (Definition 1.10), we write the dual problem as

$$\begin{aligned} \max_{\lambda} Q(\lambda) &= \max_{\lambda} \left( -\max_x \langle -A^* \lambda, x \rangle - f(x) + \langle b, \lambda \rangle \right) \\ &= \max_{\lambda} -f^*(-A^* \lambda) - \langle b, \lambda \rangle. \end{aligned} \quad (2.34)$$

Since the Fenchel conjugate is always convex, the dual problem is always a concave maximisation problem.

Furthermore, the maximum value of the dual problem is always a lower bound of the optimal value of the primal problem. This result is known as weak duality. If  $x^*$  is any solution of the primal problem, and  $\lambda^*$  is a solution to the dual problem, then  $(x^*, \lambda^*)$  is a saddle point that satisfies

$$\mathcal{L}(x^*, \lambda) \leq \mathcal{L}(x^*, \lambda^*) \leq \mathcal{L}(x, \lambda^*). \quad (2.35)$$

Under strong duality, the primal-dual gap  $\mathcal{L}(x, \lambda^*) - \mathcal{L}(x^*, \lambda)$  vanishes and one can construct minimisers of the primal problem from the solution of the dual problem.

---

**Algorithm 6:** Dual Subgradient Method

---

```

Initialise  $x^0, \lambda^0$  and  $\tau_0$ .
for  $k = 1, 2, \dots$  do
     $x^{k+1} = \arg \min \mathcal{L}(x, \lambda^k)$ ,
     $\lambda^{k+1} = \lambda^k + \tau_k(Ax^{k+1} - b)$ .
end

```

---

In the following, we introduce a more general dual subgradient method where the dual problem is solved using a subgradient method (Algorithm 3). When  $Q(\lambda)$  is sufficiently smooth, this method reduces to the dual ascent method.

As summarised in Algorithm 6, the dual subgradient method alternately updates  $x^k$  and  $\lambda^k$ , where  $\tau_k$  is the step size parameter. In the first step of the algorithm, we minimise over  $x$ . Then in the second step we perform an update on the dual variable  $\lambda$ . However, while the dual problem is concave, it is often non-smooth. Using the dual subgradient method usually leads to slow convergence. In addition, the step size parameter needs to be appropriately chosen to ensure convergence.

The conjugate correspondence theorem (Theorem 1.2) suggests that one can smoothen the dual function by adding a strongly convex term (Definition 1.17) to the primal function. Following this idea, we can formulate the augmented Lagrangian function by adding a quadratic term to obtain

$$\mathcal{L}_\tau(x, \lambda) = f(x) + \langle \lambda, Ax - b \rangle + \frac{1}{2\tau} \|Ax - b\|^2, \quad (2.36)$$

where  $\tau$  is known as the penalty parameter. Notice the resemblance of this formulation to the quadratic penalty method. Indeed, the augmented Lagrangian can be viewed as the Lagrangian associated with Problem (2.29) added to a quadratic penalty term.

Applying the dual ascent method to the augmented objective, we obtain the following augmented Lagrangian method, historically also referred to as the method of multipliers. In Algorithm 7, we summarise the method of multipliers. Notice that the penalty parameter  $\tau$  is now used as the step size parameter.

As a matter of fact, the augmented Lagrangian method can be interpreted as the proximal point method (Martinet, 1970, 1972) applied to the dual problem. This perspective also explains the faster and more robust convergence performance of the augmented Lagrangian method than the dual subgradient method (Algorithm 6). To

**Algorithm 7:** Method of Multipliers

---

```

Initialise  $x^0, \lambda^0$  and  $\tau$ .
for  $k = 1, 2, \dots$  do
     $x^{k+1} = \arg \min \mathcal{L}_\tau(x, \lambda^k),$ 
     $\lambda^{k+1} = \lambda^k + \tau(Ax^{k+1} - b).$ 
end

```

---

see this, observe that the optimality condition for  $x^{k+1}$  is

$$0 \in \partial f(x^{k+1}) + A^\top(\lambda^k + \frac{1}{\tau}(Ax^{k+1} - b)),$$

which can be rewritten as  $-A^\top \lambda^{k+1} \in \partial f(x^{k+1})$ . Using the conjugate subgradient theorem (Theorem 1.1), we can characterise this inclusion equivalently with  $x^{k+1} \in \partial f^*(-A^\top \lambda^{k+1})$ .

If we now multiply both sides with  $-A$  and express  $Ax^{k+1}$  using  $\lambda^{k+1} = \lambda^k + \tau(Ax^{k+1} - b)$ , we obtain  $0 \in \partial A f^*(-A^\top \lambda^{k+1}) + b + \frac{1}{\tau}(\lambda^{k+1} - \lambda^k)$ , which is exactly the optimality condition for the proximal point update on the dual problem

$$\lambda^{k+1} = \arg \max_{\lambda} \left\{ Q(\lambda) + \frac{1}{2\tau} \|\lambda - \lambda^k\| \right\}.$$

From the convergence result of Algorithm 4, we know the dual objective converges at rate of  $O(1/k)$ :

$$Q(\lambda^*) - Q(\lambda^k) \leq \frac{\|\lambda^* - \lambda^0\|^2}{2\tau k}.$$

Further, observe that in some cases our objective function  $f$  is separable, i.e. it can be written in the following form

$$f(x) = \sum_{i=1}^N f_i(x_i),$$

where  $x_1, x_2 \dots x_N$  are  $N$  partitions of the vector  $x$ . One can accordingly split  $Ax$  as  $Ax = \sum_{i=1}^N A_i x_i$ . Hence, the Lagrangian is now also separable:

$$\mathcal{L}(x, \lambda) = \sum_{i=1}^N \mathcal{L}_i(x_i, \lambda) = \sum_{i=1}^N \left( f_i(x_i) + \langle \lambda, A_i x_i \rangle - \frac{1}{N} \langle \lambda, b \rangle \right).$$

---

**Algorithm 8:** Alternating Direction Method of Multipliers

---

Initialise  $x^0, z^0, \lambda^0$  and  $\tau$ .  
**for**  $k = 1, 2, \dots$  **do**  
     $x^{k+1} = \arg \min_x \mathcal{L}_\tau(x, z^k, \lambda^k),$   
     $z^{k+1} = \arg \min_z \mathcal{L}_\tau(x^{k+1}, z, \lambda^k),$   
     $\lambda^{k+1} = \lambda^k + \tau(Ax^{k+1} + Bz^{k+1} - c).$   
**end**

---

If we solve the minimisation problem using the dual subgradient method, the separable structure of the above Lagrangian allows the minimisation step in  $x$  variables to be performed in parallel by solving  $N$  sub-problems. The dual subgradient method in this case is usually referred to as the dual decomposition method. However, the augmented Lagrangian method loses such separability because of the additional quadratic penalty term. We will therefore discuss another type of algorithm, namely the alternating direction method of multipliers and see how it circumvents the decomposition issue.

### 2.5.1 Alternating Direction Method of Multipliers

The alternating direction method of multipliers (ADMM) (Boyd et al., 2011; Gabay, 1983) aims at solving problems of the form

$$\min_{x,z} f(x) + g(z) \quad \text{s.t.} \quad Ax + Bz = c, \quad (2.37)$$

where  $x \in \mathcal{X}, z \in \mathcal{Z}$ , and  $\mathcal{X}, \mathcal{Z}$  are closed convex sets. Notice that this formulation is different than problem (2.29) in the sense that the objective function  $f$  is separable across the partitions  $x$  and  $z$ . The augmented Lagrangian is formulated as

$$\mathcal{L}_\tau(x, z, \lambda) = f(x) + g(z) + \langle \lambda, Ax + Bz - c \rangle + \frac{1}{2\tau} \|Ax + Bz - c\|^2.$$

Following the method of multipliers (Algorithm 7), we obtain the iterates as below

$$(x^{k+1}, z^{k+1}) = \arg \min_{x,z} \mathcal{L}_\tau(x, z, \lambda^k) \quad (2.38)$$

$$\lambda^{k+1} = \lambda^k + \tau(Ax^{k+1} + Bz^{k+1} - c). \quad (2.39)$$

In the first step, the augmented Lagrangian is jointly minimised with respect to  $x$  and  $z$ . ADMM instead considers an alternating minimisation technique (Algorithm 5) and updates  $x$  and  $z$  sequentially. In Algorithm 8, we summarises the ADMM algorithm.

Instead of the joint minimisation, ADMM performs block coordinate descent on the primal problem. Furthermore, separating the minimisation problem between  $x$  and  $z$  now permits decomposition. The proof of the convergence result can be found in Boyd et al. (2011), where more detailed discussions can also be found in Boyd et al. (2011); Chambolle and Pock (2016a); Parikh and Boyd (2014).

### Bregman Alternating Direction Method of Multipliers

Note that ADMM can be modified to fit the Bregman framework. Namely, we can modify the update steps in Algorithm 8 into

**Definition 2.2** (Bregman ADMM).

$$x^{k+1} = \arg \min_x \left\{ \mathcal{L}_\tau(x, z^k, \lambda^k) + D_{J_x}(x, x^k) \right\} \quad (2.40)$$

$$z^{k+1} = \arg \min_z \left\{ \mathcal{L}_\tau(x^{k+1}, z, \lambda^k) + D_{J_z}(z, z^k) \right\} \quad (2.41)$$

$$\lambda^{k+1} = \arg \max_\lambda \left\{ \lambda^k + \tau(Ax^{k+1} + Bz^{k+1} - c) + D_{J_\lambda}(\lambda, \lambda^k) \right\}. \quad (2.42)$$

where  $D_J(u, v)$  is the generalised Bregman distance as defined earlier in Definition 1.12. The adaption of ADMM to more general Bregman functions was first proposed in Wang and Banerjee (2014). Notice that with the choices of  $J_x = J_z = 0$  and  $J_\lambda = \frac{1}{2\tau}\|\lambda\|^2$ , update steps in (8) recover the classical ADMM setting. With various choices of  $J_x(x)$ ,  $J_z(z)$  and  $J_\lambda(\lambda)$ , one can produce other useful variations of ADMM. For example, with the following choices of  $J$

$$\begin{aligned} J_x(x) &= \frac{1}{2\tau\tau_x}\|x\|^2 - \frac{1}{2\tau}\|Ax\|^2 \\ J_z(z) &= \frac{1}{2\tau\tau_z}\|z\|^2 - \frac{1}{2\tau}\|Bz\|^2 \\ J_\lambda(\lambda) &= \frac{1}{2\tau_\lambda}\|\lambda\|^2, \end{aligned}$$

we obtain a variant of ADMM with the following iterates

$$\begin{aligned} x^{k+1} &= (I + \tau\tau_x\partial f)^{-1} \left( x^k - \tau_x A^\top (Ax^k + Bz^k - c + \tau\lambda^k) \right) \\ z^{k+1} &= (I + \tau\tau_z\partial g)^{-1} \left( z^k - \tau_z B^\top (Ax^{k+1} + Bz^k - c + \tau\lambda^k) \right) \\ \lambda^{k+1} &= \lambda^k + \tau_\lambda (Ax^{k+1} + Bz^{k+1} - c). \end{aligned}$$

Note that in this variant, the update steps are now explicit with respect to the operators  $A$  and  $B$ .

---

**Algorithm 9:** Primal-Dual Hybrid Gradient Method

---

Initialise  $x^0, y^0$ , step size  $\tau$  and  $\lambda$ .  
**for**  $k = 1, 2, \dots$  **do**  
     $x^{k+1} = \text{prox}_{\lambda g}(x^k - \lambda K^* y^k)$ ,  
     $y^{k+1} = \text{prox}_{\tau f^*}(y^k + \tau K(2x^{k+1} - x^k))$ .  
**end**

---

With this we conclude the section on ADMM approaches. We refer interested readers to Franca et al. (2018); Goldstein et al. (2014); Poon and Liang (2019); Zhang and Shen (2019) for further discussions.

### 2.5.2 Primal-Dual Hybrid Gradient Method

Last but not least, we want to briefly review a special saddle point problem in this section and discuss a method that plays a crucial role in many imaging and machine learning problems, namely the primal-dual hybrid gradient (PDHG) method. In particular, let us consider the following composite objective

$$\min_{x \in \mathcal{X}} f(Kx) + g(x), \quad (2.43)$$

where  $f, g \in \Gamma_0$  are proper, convex, l.s.c. functions and  $K$  is a bounded linear operator. The structure of problem is standard in many imaging problems such as image compression, image denoising and image deblurring, etc. Since  $f$  is convex, its biconjugate (Definition 1.11) is just itself, i.e.  $f = f^{**}$ . We can then rewrite

$$\min_x f(Kx) + g(x) = \min_x \max_y \langle y, Kx \rangle - f^*(y) + g(x).$$

Again, we can swap the min and max to obtain the following:

$$\begin{aligned} \min_x f(Kx) + g(x) &= \min_x \max_y \langle y, Kx \rangle - f^*(y) + g(x) \\ &= \max_y \min_x \langle K^* y, x \rangle + g(x) - f^*(y). \end{aligned}$$

One natural idea to proceed is to perform a proximal descent step with respect to variable  $x$  and a proximal ascent step with respect to variable  $y$ , resulting in the

---

**Algorithm 10:** Stochastic Gradient Method

---

Initialise  $x^0$  and  $\tau_k$ .**for**  $k = 1, 2, \dots$  **do**

$$x^{k+1} = x^k - \tau_k g(x^k, \xi^k). \quad (2.46)$$

**end**

---

following update steps

$$x^{k+1} = \text{prox}_{\lambda g}(x^k - \lambda K^* y^k) \quad (2.44)$$

$$y^{k+1} = \text{prox}_{\tau f^*}(y^k + \tau K x^{k+1}). \quad (2.45)$$

However, convergence of the above alternating scheme is not guaranteed. To ensure convergence, there have been several strategies to modify the iterates. One of the earliest ideas is to incorporate a type of extra-gradient (Korpelevich, 1976), which proposed to replace  $y^k$  in the update step of  $x^{k+1}$  in (2.44) by a proximal point  $\text{prox}_{\tau f^*}(y^k + \tau K^* x^k)$ .

The PDHG method (Esser et al., 2010; Pock et al., 2009) is another variation with similar properties that computes solutions of the min-max problem, but without additional proximal point computations. In Algorithm 9, we summarise the update iterates of the PDHG method.

Note that in the update step of  $y^{k+1}$ , we replace  $x^{k+1}$  with  $x^{k+1} + (x^{k+1} - x^k)$ . This can be viewed as an approximation to the extra-gradient discussed in Korpelevich (1976). With this modification, one can indeed show (ergodic) convergence rate of  $O(1/k)$ . The proof of this convergence result can be found in Chambolle and Pock (2016b). Here we describe one specific form of PDHG, for more general forms and detailed discussion, we refer readers to Benning and Riis (2021); Chambolle and Pock (2011, 2016a,b); Esser et al. (2010); Pock et al. (2009); Zhu and Chan (2008).

## 2.6 Stochastic Gradient Method

Previously, we study deterministic approaches on minimising  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ , whereas in this section, we begin reviewing strategies in the stochastic setting. In particular, we consider the scenario where instead of having available information about the exact gradient or subgradient of  $f$ , now we only have access to a vector  $g(x, \xi) \in \mathbb{R}^n$  that is a function of  $x$  and a random variable  $\xi$ .

---

**Algorithm 11:** Incremental Gradient Method

---

Initialise  $x^0$  and  $\tau_k$ .

**for**  $k = 1, 2, \dots$  **do**

    select  $i_k$  in  $\{1, \dots, N\}$ .

$$x^{k+1} = x^k - \tau_k \nabla f_{i_k}(x^k). \quad (2.47)$$

**end**

---

Throughout the rest of the section, we assume that  $g(x, \xi)$  is an unbiased estimator of  $\nabla f$ . Denote the underlying probability distribution of the random variable  $\xi$  by  $P$  and denote by  $\mathbb{E}_\xi$  the expectation of  $\xi$  according to  $P$ . This unbiased estimation assumption is written as

$$\mathbb{E}_\xi[g(x, \xi)] = \nabla f(x).$$

In Algorithm 10, we summarise the stochastic gradient method (Kiefer and Wolfowitz, 1952; Robbins and Monro, 1951), where  $\tau_k$  is the step size and  $\xi^k$  is chosen according to  $P$  at each iterate and independent of choices made at previous iterates. The stochastic gradient method is a particularly popular optimisation method in modern deep learning practices due to its efficiency in learning with large-scale datasets, where only random estimates of  $\nabla f$  are accessible.

A typical illustrative example is the noisy gradient problem, where the queried gradient information contains noise, i.e.

$$g(x, \xi) = \nabla f(x) + \xi,$$

where  $\xi$  is an additive noise described by some noise process. We can see that  $g(x, \xi)$  is an unbiased estimator of  $\nabla f(x)$  when  $\mathbb{E}(\xi) = 0$ . Another notable example is the incremental gradient method. In this problem, we assume the objective function  $f$  is a finite sum:

$$f(x) = \frac{1}{N} \sum_{i=1}^N f_i(x).$$

In Algorithm 11, we summarise the incremental gradient method. A typical application for Algorithm 11 is when the computation of the gradient estimate  $g(x^k, \xi^k)$  is cheaper than evaluating the full gradient  $\nabla f(x^k)$ . We can see that  $\mathbb{E}_\xi(g(x, \xi)) =$

$\nabla f(x) = \frac{1}{N} \sum_{i=1}^N f_i(x)$  when randomly choosing  $i_k$  under a uniform distribution, where the underlying  $P$  is such that  $P(i) = \frac{1}{N}$  for  $i = 1, \dots, N$ .

We will next discuss the convergence analysis of stochastic gradient method (Algorithm 10). In the scope of this chapter, we will only consider the following scenario, where the norm of the gradient estimate is bounded. In particular, we consider the case when the stochastic gradient  $g(x, \xi)$  is bounded from above almost surely for all  $x$  with some non-negative constant  $B$ , that is

$$\mathbb{E}_\xi[\|g(x, \xi)\|^2] \leq B^2.$$

**Theorem 2.7.** *Suppose  $f$  is convex and  $L$ -smooth and suppose there is a minimiser  $x^*$  to Problem 2.6, where the optimal value is defined by  $f(x^*)$ . With the step size  $\tau = 1/L$ , we generate a sequence  $\{\bar{x}^k\}$  where*

$$\bar{x}^k = \frac{\sum_{n=0}^k \tau_n x^n}{\sum_{n=0}^k \tau_n},$$

*that is the average of all iterates so far, weighted by the step size  $\tau_n$ . Then, the sequence  $\{\bar{x}^k\}$  satisfies*

$$\mathbb{E}[f(\bar{x}^k) - f(x^*)] \leq \frac{\|x - x^*\|^2}{2 \sum_{n=0}^k \tau_n} + \frac{B^2 \sum_{n=0}^k \tau_n^2}{2 \sum_{n=0}^k \tau_n}.$$

*Proof.* After  $k$  iterations, we have the following estimate

$$\mathbb{E}[f(\bar{x}^k) - f(x^*)] \leq \mathbb{E} \left[ \frac{1}{\sum_{n=0}^k \tau_n} \sum_{n=0}^k \tau_n (f(x^n) - f(x^*)) \right] \quad (2.48)$$

$$\leq \frac{1}{\sum_{n=0}^k \tau_n} \sum_{n=0}^k \tau_n \mathbb{E} [\langle \nabla f(x^n), x^n - x^* \rangle], \quad (2.49)$$

where this follows the definition of  $\bar{x}^k$  and the convexity of  $f$ . Next we want to derive a bound on the expected error to a minimiser  $x^*$ . We start by expanding the terms

$$\begin{aligned} \|x^{k+1} - x^*\|^2 &= \|x^k - \tau_k g(x^k, \xi^k) - x^*\|^2 \\ &= \|x^k - x^*\|^2 - 2\tau_k \langle g(x^k, \xi^k), x^k - x^* \rangle + \tau_k^2 \|g(x^k, \xi^k)\|^2. \end{aligned}$$

For the second and the third term in the above equation, notice that when taking expectation with respect to all random variables up to iteration  $k$ , by the law of iterated

expectation and the fact that  $x^k$  depends on  $\xi^0, \xi^1, \dots, \xi^{k-1}$  but not on  $\xi^k$ , we have

$$\begin{aligned}\mathbb{E}[\langle g(x^k, \xi^k), x^k - x^* \rangle] &= \mathbb{E} \left[ \mathbb{E}_{\xi^k} \left[ \langle g(x^k, \xi^k), x^k - x^* \rangle \mid \xi^0, \xi^1, \dots, \xi^{k-1} \right] \right] \\ &= \mathbb{E} \left[ \langle \mathbb{E}_{\xi^k} [g(x^k, \xi^k) \mid \xi^0, \xi^1, \dots, \xi^{k-1}], x^k - x^* \rangle \right].\end{aligned}$$

Furthermore, we know that  $g(x^k, \xi^k)$  depends on  $\xi^k$ . Hence we can take expectation of  $g(x^k, \xi^k)$  explicitly with respect to  $\xi^k$  to obtain  $\mathbb{E}_{\xi^k} [g(x^k, \xi^k) \mid \xi^0, \xi^1, \dots, \xi^{k-1}] = \nabla f(x^k)$ . Hence we get

$$\mathbb{E}[\langle g(x^k, \xi^k), x^k - x^* \rangle] = \mathbb{E}[\langle \nabla f(x^k), x^k - x^* \rangle].$$

Similarly, we obtain the bound

$$\mathbb{E}[\|g(x^k, \xi^k)\|^2] = \mathbb{E}[\mathbb{E}_{\xi^k} [\|g(x^k, \xi^k) \mid \xi^1, \dots, \xi^{k-1}\|^2]] \leq B^2,$$

where the last step follows from the bounded norm assumption on the gradient estimate. Putting everything together, we have

$$\mathbb{E}[\|x^{k+1} - x^*\|^2] \leq \mathbb{E}[\|x^k - x^*\|^2] - 2\tau_k \mathbb{E}[\langle \nabla f(x^k), x^k - x^* \rangle] + \tau_k^2 B^2.$$

After rearranging the terms, we have

$$\tau_k \mathbb{E}[\langle \nabla f(x^k), x^k - x^* \rangle] \leq \frac{1}{2} [\mathbb{E}[\|x^k - x^*\|^2] - \mathbb{E}[\|x^{k+1} - x^*\|^2] + \tau_k^2 B^2]. \quad (2.50)$$

Plugging (2.50) back to (2.48) we obtain

$$\begin{aligned}\mathbb{E}[f(\bar{x}^k) - f(x^*)] &\leq \frac{1}{\sum_{n=0}^k \tau_n} \sum_{n=0}^k \left[ \frac{1}{2} (\mathbb{E}[\|x^n - x^*\|^2] - \mathbb{E}[\|x^{n+1} - x^*\|^2]) + \frac{1}{2} \tau_n^2 B^2 \right] \\ &= \frac{1}{2 \sum_{n=0}^k \tau_n} \left[ \mathbb{E}[\|x^n - x^*\|^2] - \mathbb{E}[\|x^{n+1} - x^*\|^2] + B^2 \sum_{n=0}^k \tau_n^2 \right] \\ &\leq \frac{\|x^0 - x^*\|^2 + B^2 \sum_{n=0}^k \tau_n^2}{2 \sum_{n=0}^k \tau_n}.\end{aligned}$$

□

Note that in the previous convergence result we assume  $f$  being  $L$ -smooth hence the choice of step size  $\tau = 1/L$ . In the next section, we will briefly discuss an extension from explicit stochastic gradient method to its implicit counterpart as seen in Section 2.1.

---

**Algorithm 12:** Implicit Stochastic Gradient Method

---

```

Initialise  $x^0$  and  $\tau_k$ .
for  $k = 1, 2, \dots$  do
  |  $x^{k+1} = x^k - \tau_k g(x^{k+1}, \xi^{k+1})$ .
end

```

---

### 2.6.1 Implicit Stochastic Gradient Method

The duo of explicit-implicit updates in gradient descent method as discussed in Section 2.1.1 can also be extended to the stochastic setting. Similarly, we can derive the following implicit stochastic gradient method (Toulis and Airoldi, 2017), summarised in Algorithm 12.

As we have discussed in Algorithm 2, the current iterate  $x^{k+1}$  now appears on both sides of the equation, but in a stochastic setting where we only have an unbiased estimator of the gradient. Most of the analysis from Section 2.1.1 can be translated into the stochastic setting. We will illustrate with the following concrete example

$$\min_x \frac{1}{N} \sum_{i=1}^N f_i(x),$$

where we solve the above problem with the incremental gradient method as described in Algorithm 11.

Assuming that the stochastic estimate of the gradient  $g(x^{k+1}, \xi^{k+1})$  satisfies that  $g(x^{k+1}, \xi^{k+1}) \in \partial f_i(x^{k+1})$ , we then have

$$x^{k+1} = \text{prox}_{\tau f_i}(x^k) = (I + \partial f_i)^{-1}(x^k) = x^k - \tau \nabla M_{f_i}^\tau(x^k),$$

which is an iteration of the explicit incremental gradient method on the Moreau envelope  $M_{f_i}^\tau$ . The convergence analysis of the implicit incremental gradient method hence follows similarly to the convergence result of its explicit version described in the previous section.

With this we bring our discussion on stochastic gradient method to a close. We refer interested readers to Bertsekas (2011); Bottou (2010); Lacoste-Julien et al. (2012) for more further discussions.

### 3 Inverse Problems and Regularisation Theory

Inverse problems concern the problems of using observations to infer the values of the parameters of a modelled system. In this era of big-data, we see more and more interactions evolving between the mathematical field of inverse problems and deep learning models. In this section, we review some key concepts and notions in inverse problems and regularisation theory, where we will in particular focus on variational regularisation. The discussion in this section is mainly based on Arridge et al. (2019); Benning and Burger (2011, 2018); Benning and Ehrhardt (2016); Engl et al. (1996).

#### 3.1 Inverse Problems

An inverse problem is typically referred to as the problem of the inversion of a forward model. For  $u \in \mathcal{U}$  and  $y \in \mathcal{V}$ , given the following data generation model:

$$y = Ku, \tag{2.51}$$

where  $K : \mathcal{U} \rightarrow \mathcal{V}$  is typically a linear operator. Observations  $y \in \mathcal{V}$  are generated by an underlying forward operation  $Ku$ . In the case when  $y$  is corrupted with observational or measurement noise  $\delta$ , we denote the noisy observation by  $y^\delta$ . The inverse problem usually consists of solving for the unknown solution  $u \in \mathcal{U}$  from indirect observations  $y$  or  $y^\delta$  and challenges in finding a solution lie in the fact that many inverse problems are ill-posed in the sense of negating well-posedness (Hadamard, 1902). We say a problem is well-posed if it satisfies three defining properties:

1. There exists a solution to the problem.
2. The solution is unique.
3. The solution depends continuously on the input datum.

For example, the uniqueness condition is violated when there are multiple possible  $u$  that are all consistent with the observation  $y$ . Or the stability condition can be easily violated if small errors in observation  $y$  lead to amplified errors in the solution  $u$ .

Regularisation theory is a key tool in solving ill-posed inverse problems. The central idea of a regularisation method is to construct a well-posed approximation to the ill-posed problem that is convergent. In the next section, we will limit our discussion to variational regularisation methods and review their properties.

### 3.2 Variational Regularisation Model

In this section, we will discuss the basic form of the variational approach of regularisation. For the rest of the section, we will make the assumption that  $\mathcal{U}$  and  $\mathcal{V}$  are Banach spaces and  $K : \mathcal{U} \rightarrow \mathcal{V}$  is a bounded linear operator.

A variational regularisation method in general can be formulated as the following problem

$$\hat{u} \in \arg \min_{u \in \mathcal{U}} \{F(Ku, y) + \alpha J(u)\} . \quad (2.52)$$

The objective function on the right hand side consists of two functionals: a data fidelity or data discrepancy term  $F$  measuring the distance between two arguments and a regularisation functional  $J$  favoring minimisers with desirable structures and penalising potential solutions with undesired features. The two terms are balanced by a positive regularisation parameter  $\alpha > 0$ . A typical choice of the data fidelity term is usually a squared norm. As for the choice of  $J$ , it usually depends on some a-priori information that we want to incorporate when solving problem (2.52). We will next discuss some typical choices of regularisation functionals.

**$\ell_2$ -norm** One of the most common choice of regularisation functional is the  $\ell_2$ -norm  $\|u\|_2$ , which is well studied in the classical Tikhonov or Tikhonov-Phillips regularisation. The  $\ell_2$ -norm regularisation ensures the squared norm of the solution is not too large. Solving an Ordinary Least Squares regression problem with  $\ell_2$  regularisation is known as the ridge regression in statistics literature. The weight decay technique in deep learning practices is also an example of the  $\ell_2$ -norm regularisation.

**$\ell_1$ -norm** Another popular choice of regularisation functional is the  $\ell_1$ -norm  $\|u\|_1$ . The  $\ell_1$ -norm regularisation is often used as a regulariser to enforce sparsity in the solution. The  $\ell_1$ -norm can be used as an approximation to the  $\|\cdot\|_0$  norm which counts the number of non-zero elements in the solution. The  $\ell_1$ -norm is typically seen in many imaging problems and also in compressed sensing and basis pursuit problems. The regression problem with  $\ell_1$ -norm regularisation is known as the Least Absolute Shrinkage and Selection Operator (LASSO) in statistics literature (Tibshirani, 1996).

**Total variation** Total variation (TV) regularisation has been one of the major driving force for the development in variational regularisation methods. Total variation regularisation was first introduced by Rudin, Osher and Fatemi in Rudin et al. (1992)

for its edge preserving properties and that it favors images with sparse gradients (Chambolle et al., 2010). In many cases of imaging problems, one often favors non-smooth solutions and particularly, to allow discontinuities in solutions, such as the sudden change of intensity at edges in images. The total variation can be defined as

$$\text{TV}(u) = |u|_{\text{BV}} = \sup \left\{ \int_{\Omega} \langle u, \text{div}(\psi) \rangle \mid \psi \in C_0^\infty(\Omega; \mathbb{R}^n), |\psi| \leq 1 \right\}. \quad (2.53)$$

where  $\Omega$  is an open subset of  $\mathbb{R}^n$  denoting the image domain. The space of functions of bounded variations (BV) is defined as

$$\text{BV}(\Omega) = \left\{ u \in L^1(\Omega) \mid \text{TV}(u) \leq \infty \right\}. \quad (2.54)$$

In practice, we often consider the discrete total variation  $\|\nabla u\|_{p,1}$ , (Chambolle and Lions, 1997; Rudin et al., 1992). Consider a two-dimensional scalar-valued image  $u \in \mathbb{R}^{H \times W}$ , we can define a finite forward difference discretisation of the gradient operator  $\nabla : \mathbb{R}^{H \times W} \rightarrow \mathbb{R}^{H \times W \times 2}$  as

$$(\nabla u)_{i,j,1} = \begin{cases} u_{i+1,j} - u_{i,j} & \text{if } 1 \leq i < H, \\ 0 & \text{else,} \end{cases}, \quad (\nabla u)_{i,j,2} = \begin{cases} u_{i,j+1} - u_{i,j} & \text{if } 1 \leq j < W, \\ 0 & \text{else.} \end{cases}.$$

The discrete total variation is defined as the  $\ell_1$  norm of the  $p$ -norm of the pixel-wise image gradients, i.e.

$$\|\nabla u\|_{p,1} = \sum_{i=1}^H \sum_{j=1}^W |(\nabla u)_{i,j}|_p = \sum_{i=1}^H \sum_{j=1}^W \left( (\nabla u)_{i,j,1}^p + (\nabla u)_{i,j,2}^p \right)^{1/p}.$$

This concludes the review on variational regularisation models. In the next section, we will recall basic definitions from non-linear regularisation analysis and how we can derive general error estimates.

### 3.3 Non-linear Regularisation Analysis

After having introduced several example variational regularisation models, in this section we will recap some fundamental concepts in the analysis of nonlinear regularisation methods.

As mentioned earlier, the core idea of a regularisation method is constructing convergent approximations to the desired solution. To begin with, we first introduce the notion of a best approximate solution.

**Definition 3.1** (Best approximate solution). *Given an error measure  $F : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}_+ \cup \{\infty\}$ , where  $\mathbb{R}_+$  denotes the non-negative real numbers. We call  $\hat{u} \in \mathcal{U}$  a best approximate solution of (2.51) with respect to  $F$  if*

$$F(K\hat{u}, y) \leq F(Ku, y) \quad \text{for all } u \in \mathcal{U} . \quad (2.55)$$

The next definition defines the selection of via prior knowledge by a (multivalued) selection operator.

**Definition 3.2** (Selection operator). *Denote the range and kernel of  $K$  with  $\mathcal{R}(K)$  and  $\mathcal{N}(K)$  respectively. A multivalued operator  $\mathcal{S} : \mathcal{R}(K) \rightrightarrows \mathcal{U}$  is called a selection operator if  $\mathcal{S}(Ku) \subset u + \mathcal{N}(K)$  for all  $u \in \mathcal{U}$ . A best approximate solution  $\hat{u}$  is called prior selected solution of (2.51) if and only if  $\hat{u} \in \mathcal{S}(K\hat{u})$ .*

We first define the effective range of the forward operator as

$$\mathcal{R}_F(K) = \left\{ y \in \mathcal{V} \mid \arg \min_{u \in \mathcal{U}, J(u) < +\infty} F(Ku, y) \neq \emptyset \right\} .$$

Then in the scalar case where  $\alpha \geq 0$ , we can construct one selection operator by minimising the regularisation functional on the set of best approximate solutions. Let  $y \in \mathcal{R}_F(K)$ , then we define

$$\mathcal{S}(y) = \arg \min_{u \in \mathcal{U}} \left\{ J(u) \mid u \in \arg \min_{\hat{u} \in \mathcal{U}} F(K\hat{u}, y) \right\} .$$

A generic variational regularisation operation can then be defined by the following.

**Definition 3.3** (Variational regularisation). *Let  $F : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}_+$  be continuous with  $F(y, y) = 0$  for all  $y \in \mathcal{R}_F(K)$  and  $J : \mathcal{U} \times \mathbb{R}_+ \rightarrow \mathbb{R} \cup \{\infty\}$  be proper, lower semi-continuous and convex functionals. Then the (set-valued) operator  $R : \mathcal{V} \times \mathbb{R}_+ \rightrightarrows \mathcal{U}$  defined as*

$$R(y^\delta, \alpha) = \arg \min_{u \in \mathcal{U}} \left\{ F(Ku, y^\delta) + J(u, \alpha) \right\} ,$$

*is said to be a variational regularisation, for fixed regularisation parameters  $\alpha \in \mathbb{R}_+$ .*

In the next step, we will define what do we mean by a convergent regularisation method.

**Definition 3.4.** Let  $y = Ku^\dagger$  for some exact solution  $u^\dagger$ . Given some error measure  $D : \mathcal{U} \times \mathcal{U} \rightarrow \mathbb{R}_+ \cup \{+\infty\}$ , where  $D$  being a Bregman distance, a regularisation method is called  $D$ -convergent if

$$\limsup_{\delta \rightarrow 0} \left\{ D(u^\alpha, u^\dagger) \mid u^\alpha \in R(y^\delta, \alpha), y^\delta \in \mathcal{V}, F(y, y^\delta) \leq \delta \right\} = 0.$$

A regularisation method is called convergent at order  $v$  if there exists a constant  $C_v$  such that for all data satisfying the error bound, we obtain the estimate

$$D(R(y^{\delta, \alpha(\delta, y^\delta)}), u^\dagger) \leq C_v \delta^v. \quad (2.56)$$

We will next summarise basic analysis of the variational regularisation methods. We will start by making the following assumptions:

**Assumption 3.1.** Let  $\mathcal{U} = Z^*$  for some normed space  $Z$  and let the weak star topology on  $\mathcal{U}$  be metrisable on bounded sets. Furthermore, we assume

- $K = L^*$  for a bounded linear operator  $L : \mathcal{V} \rightarrow Z$ .
- $J$  is nonnegative with  $J = H^*$  for some proper functional  $H : Z \rightarrow \mathbb{R} \cup \{+\infty\}$ .
- $F$  is proper, nonnegative, convex in its first argument and continuous in its second argument. And for every  $g \in \mathcal{V}$ , there exists  $u$  with  $F(Ku, g) + \alpha J(u) < \infty$ .
- For each  $g \in \mathcal{V}$  and  $\alpha \in \mathbb{R}_+$ , there exists a constant  $c = (a, b, \|g\|)$  depending monotonically non-decreasing on all arguments such that

$$\|u\|_{\mathcal{U}} \leq c \quad \text{if } F(Ku, g) \leq a, \quad \alpha J(u) \leq b.$$

First we will verify the well-posedness of the selection operator  $\mathcal{S}$  with the following theorem.

**Theorem 3.1.** Let Assumption 3.1 hold, then for every  $y \in \mathcal{R}_F(K)$ , the selection operator  $\mathcal{S}$  of the  $J$ -minimising solutions is well-defined for  $\alpha \in \mathbb{R}_+$ .

*Proof.* The proof of this theorem follows directly from Benning and Burger (2018, Lemma 5.5). For  $y \in \mathcal{R}_F(K)$ , we know there exists a minimiser  $u^*$  such that  $J(u^*) < \infty$ . In the definition of  $\mathcal{S}$ , we consider the minimisation only with respect to the set of  $u$  such that  $F(Ku, y) = F(Ku^*, y)$ , which provides an upper bound on the data fidelity term. On the other hand, in such set of  $u$ , we can also obtain an upper bound on

the regularisation functional by restricting to  $u$  with  $J(u) \leq J(u^*)$ . Hence by the last item in Assumptions 3.1, for the set of such  $u$ , the norm in  $\mathcal{U}$  is bounded and for each minimising sequence there exists a weak-star convergence subsequence  $u_n$ . Moreover, with the assumptions above, we see that  $\alpha J(\cdot)$  is sequentially weak-star lower semi-continuous and  $F(\cdot, y)$  is weakly lower semi-continuous. From the first item in Assumptions 3.1, we see that  $K$  is continuous from the weak-star topology of  $\mathcal{U}$  to the weak topology of  $\mathcal{V}$ . As a consequence  $F(\cdot, y) + \alpha J(\cdot)$  is weak-star lower semi-continuous. Hence the weak-star limit of  $u_n$  is a minimiser, i.e.  $\mathcal{S}(y) \neq \emptyset$ .  $\square$

We can then verify the well-definedness of the regularisation operator  $R(y^\delta, \alpha)$ .

**Theorem 3.2.** *Let Assumption 3.1 hold. Then for every  $y \in \mathcal{V}$  the variational regularisation model has a minimiser in  $\mathcal{U}$  for every  $\alpha \in \mathbb{R}_+$ , i.e. the regularisation operator  $R$  is well-defined. Moreover,  $R(y^\delta, \alpha)$  is a convex set.*

*Proof.* The proof of this theorem follows directly from Benning and Burger (2018, Theorem 5.6). Suppose there exists an  $\tilde{u}$  with  $F(K\tilde{u}, y) + \alpha J(\tilde{u}) < \infty$ . Denote the functional value attained by  $\tilde{u}$  with value  $a$ . Then we can restrict the minimisation to the set of  $u$  such that

$$F(Ku, y) + \alpha J(u) \leq F(K\tilde{u}, y) + \alpha J(\tilde{u}) .$$

Since both terms are non-negative in the full functional  $F(Ku, y) + \alpha J(u)$ , by Assumption 3.1, we obtain the boundedness of the norm on this subset of  $u$ . The remaining weak-star compactness and lower semi-continuous argument are similar to Lemma 3.1. We can hence verify the existence of a minimiser. As we assume the functional  $F$  to be convex, the convexity of  $R(y^\delta, \alpha)$  follows from the fact that the set of minimisers of  $F$  is also convex.  $\square$

The next theorem verifies the generalised stability of the variational regularisation.

**Theorem 3.3.** *Let Assumption 3.1 hold and let  $F$  be continuous in its second variable. Then for  $\alpha \in \mathbb{R}_+$  and every sequence  $y_n \rightarrow y \in \mathcal{V}$ , there exists a subsequence  $u_{n_k} \in R(y_{n_k}, \alpha)$  converging to an element  $u^* \in R(y, \alpha)$*

*Proof.* The proof of this theorem follows directly from Benning and Burger (2018, Theorem 5.7). By the definition of the regularisation operator, we find for  $u_n \in R(y_n, \alpha)$  such that for any  $u \in \mathcal{U}$

$$F(Ku_n, y_n) + \alpha J(u_n) \leq F(Ku, y_n) + \alpha J(u) .$$

Due to the convergence of  $y_n$  and the continuity of  $F$  with respect to its second argument, we can uniformly bound  $F(Ku, y_n) + \alpha J(u)$  by some value  $a$ . Hence we also obtain a uniform bound for both terms on the left hand side. Invoke the last item in Assumption 3.1, we have that  $\|u_n\| \leq c(a, a, \|y_n\|)$ . The boundedness of  $\|y_n\|$  and monotone dependence on  $c$  yields a uniform bound on  $\|u_n\|$  hence a weakly converging subsequence. Using a similar lower semi-continuity argument and the continuity of  $F$  in its second argument, we can see that for the limit  $u^*$  the following holds

$$\begin{aligned} F(Ku^*, y) + \alpha J(u^*) &\leq \liminf F(Ku_{n_k}, y_{n_k}) + \alpha J(u_{n_k}) \leq \lim F(Ku, y_{n_k}) + \alpha J(u) \\ &= F(Ku, y) + \alpha J(u). \end{aligned}$$

Hence  $u^* \in R(y, \alpha)$ . □

Given some unknown ground truth solution  $u^\dagger \in \mathcal{S}(y, \alpha)$ , in order to derive error estimate, we want to ensure that there exists data  $y_\alpha^\dagger$  such that  $u^\dagger \in R(y_\alpha^\dagger, \alpha)$  is a solution of the corresponding variational regularisation problem. More rigorously, we have the following range condition.

**Definition 3.5** (Range condition). *We say an element  $u^\dagger \in \mathcal{S}(y, \alpha)$  for  $y \in \mathcal{R}_F(K)$  satisfies the range condition if  $u^\dagger \in \mathcal{R}(R(\cdot, \alpha))$ . If  $K \in \mathcal{L}(\mathcal{U}, \mathcal{V})$ ,  $F$  is convex and Frechét-differentiable with respect to its first argument, and  $J(\cdot, \alpha)$  is proper, convex and lower semi-continuous, then this is equivalent to the existence of  $p^\dagger \in \partial J(u^\dagger, \alpha)$  and  $y_\alpha^\dagger \in \mathcal{V}$  such that*

$$p^\dagger = -K^* \partial F(Ku^\dagger, y_\alpha^\dagger).$$

Recall that the source condition (cf. (Benning and Burger, 2018; Engl et al., 1996)) on the other hand is defined as follows.

**Definition 3.6** (Source condition). *An element  $u^\dagger \in \mathcal{S}(y, \alpha)$  for  $y \in \mathcal{R}_K(F)$  satisfies the source condition if*

$$\mathcal{R}(K^*) \cap \partial J(u^\dagger, \alpha) \neq \emptyset.$$

*This is equivalent to the existence of  $p^\dagger \in \partial J(u^\dagger, \alpha)$  and  $v \in \mathcal{V}^* \setminus \{0\}$  such that*

$$p^\dagger = K^* v.$$

The following theorem (cf. (Benning and Burger, 2011)) builds the basis of error estimate by estimating Bregman distances between  $u^\alpha \in R(y^\delta, \alpha)$  and  $u^\dagger$  for general variational regularisation methods.

**Theorem 3.4.** *Let the source condition (Definition 3.6) be satisfied. Then the error estimate*

$$F(Ku^\alpha, y^\delta) + D_{J(\cdot, \alpha)}^{\text{symm}}(u^\dagger, u^\alpha) \leq F(Ku^\dagger, y^\delta) + \alpha \langle v, Ku^\alpha - Ku^\dagger \rangle$$

*holds for every  $u^\alpha \in R(y^\delta, \alpha)$ .*

*Proof.* For the proof of this theorem we refer readers to Benning and Burger (2011, Theorem 3.3).

This brings us to the end of our discussion on inverse problems and variational regularisation method. For more information about regularisation method, we refer interested readers to Benning and Burger (2018); Engl et al. (1996) for an overview of deterministic regularisation, to Stuart (2010) for a Bayesian perspective on regularisation and De Vito et al. (2005, 2021) for regularisation in the context of machine learning.

## 4 Machine Learning and Deep Learning

In this section, we discuss topics in modern deep learning practices. We will begin by introducing machine learning problems on a generic level as well as recapping the important concept of the bias-variance trade-off. Then we dive into topics in deep learning and artificial neural networks, where we introduce some fundamental concepts and example architectures of DNNs. The discussion in this section is mainly based on materials from Bishop and Nasrabadi (2006); Goodfellow et al. (2016a); Murphy (2012); Zhang et al. (2021a).

### 4.1 Machine Learning

In machine learning problems, learning tasks such as classification, regression are often evaluated via quantifying expected errors over the underlying distribution  $P$  where the data is sampled or generated from. As early as discussed in Tsytkin and Nikolic (1971), the goal of a learning system and in particular in a supervised learning setting, is to approximate an unknown target function such that it minimises the expected risk

(also known as the population risk or expected error), defined as

$$\begin{aligned} E[f] &= \mathbb{E}_{(x,y) \sim P} [\ell(f(x), y)] \\ &= \int_{(x,y) \sim P} \ell(f(x), y) p(x, y) dx dy. \end{aligned} \quad (2.57)$$

That is, the expectation taken over the data space of  $(x, y)$  according to distribution  $P$ , where  $p$  is the unknown joint probability density function. The error or loss function is denoted by  $\ell$ , which measures the difference between  $f(x)$  and  $y$ . If we assume there exists an underlying fixed family of functions  $\mathcal{F}$ , we can define  $\hat{f}$  as the function that minimises the population risk, i.e.

$$\hat{f} = \arg \min_{f \in \mathcal{F}} \mathbb{E}_{(x,y) \sim P} [\ell(f(x), y)] .$$

However, as the ground truth distribution is unknown, the expected risk function (2.57) is not directly accessible. Nevertheless, in practice one can approximate the expected risk using a finite collection of observed independent identically distributed (i.i.d.) data samples  $\{(x^i, y^i)\}_{i=1}^s$ . This approximation is defined as the empirical risk

$$\frac{1}{s} \sum_{i=1}^s \ell(f(x^i), y^i) . \quad (2.58)$$

Subsequently, the difference between the population risk (2.57) and the empirical risk (2.58) is known as the generalisation error

$$\mathbb{E}_{(x,y) \sim P} [\ell(f(x), y)] - \frac{1}{s} \sum_{i=1}^s \ell(f(x^i), y^i) . \quad (2.59)$$

Note that when the number of samples  $s$  is sufficiently large, minimising the empirical risk will give good estimates of the expected risk minimisers (Vapnik, 1982).

Now assume that we want to best approximate the expected risk minima with some parametric function  $f_\theta$  from a family of parametrised functions  $\mathcal{F}_\theta$ . Computationally,  $\theta$  is obtained by minimising the empirical risk over the collection of data samples  $\{(x^i, y^i)\}_{i=1}^s$ , which we now refer to as the training dataset. The associated minimisation problem (2.58) is usually known as the learning or training problem. Accordingly, the empirical risk attained with  $f_\theta$  is known as the training error

$$\frac{1}{s} \sum_{i=1}^s \ell(f_\theta(x^i), y^i) . \quad (2.60)$$

Suppose there is another i.i.d. collection of data  $\{(x^j, y^j)\}_{j=1}^v$  also sampled from  $P$ , which we refer to as the validation dataset. The validation error is defined as the evaluated empirical risk of  $f_\theta$  on the validation dataset, i.e.

$$\frac{1}{v} \sum_{j=1}^v \ell(f_\theta(x^j), y^j). \quad (2.61)$$

Recall from earlier that the generalisation error tells us how well the empirical error minima  $f_\theta$  approximates the expected error minima  $\hat{f}$ . Yet since  $E$  is not accessible, the generalisation cannot be computed either. Nevertheless, the generalisation gap can be approximated using the difference between the validation error and the training error, i.e.

$$\frac{1}{v} \sum_{j=1}^v \ell(f_\theta(x^j), y^j) - \frac{1}{s} \sum_{i=1}^s \ell(f_\theta(x^i), y^i). \quad (2.62)$$

We say the parametric function  $f_\theta$  generalises well if this gap converges to zero as the number of validation samples goes to  $\infty$ .

#### 4.1.1 Bias–variance Decomposition

Essentially, in a supervised learning setting, we try to find a  $f_\theta$  that computationally minimises the empirical risk function but also yields small generalisation gap (2.62). In other words, we want the trained model capable of generalising well to data samples outside of the training dataset. In this section we introduce the bias-variance decomposition, which provides a way of analysing the expected generalisation error of the  $f_\theta$  that is found through a particular learning algorithm. Suppose we have the following data model

$$y_\epsilon = \hat{f}(x) + \epsilon,$$

where  $\hat{f}$  is the underlying target mapping we aim to learn.  $S_t = \{x^i, y_\epsilon^i\}_{i=1}^s$  is the collection of training dataset where the data labels are contaminated with noise  $\epsilon$ . We assume the noise  $\epsilon$  is a random variable with zero mean and variance  $\sigma^2$ .

During training, we want to find  $f_\theta$  to approximate  $\hat{f}$  by computationally solving the empirical risk minimisation problem (ERM) over  $S_t$ . To measure this approximation, in the following we use the expected squared error  $\ell(\cdot) = \|\cdot\|^2$  as an example. For an unseen pair of data  $(x, y)$ , the expected error can be decomposed into the following

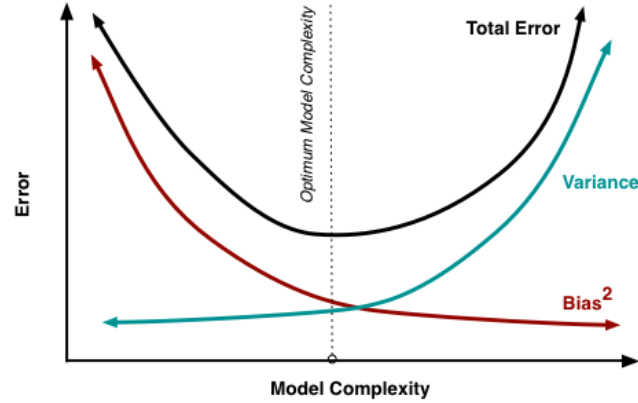


Fig. 2.3 Visualisation of how bias and variance contribute to the total error. Figure taken from Fortman-Roe (2012).

three components

$$\begin{aligned}
 \mathbb{E}_{S_t, \epsilon} [\|y_\epsilon - f_\theta(x)\|^2] &= \mathbb{E}_{S_t, \epsilon} [\|\hat{f}(x) + \epsilon - f_\theta(x)\|^2] \\
 &= \mathbb{E}_{S_t, \epsilon} [\|\epsilon\|^2] + 0 + \mathbb{E}_{S_t, \epsilon} [\|\hat{f}(x) - f_\theta(x)\|^2] \\
 &= \text{Var}_\epsilon[\epsilon] + \mathbb{E}_{S_t, \epsilon} [\|\hat{f}(x) - \mathbb{E}_{S_t}[f_\theta] + \mathbb{E}_{S_t}[f_\theta] - f_\theta(x)\|^2] \\
 &= \sigma^2 + \mathbb{E}_{S_t, \epsilon} [\|\hat{f}(x) - \mathbb{E}_{S_t}[f_\theta]\|^2] + \mathbb{E}_{S_t, \epsilon} [\|\mathbb{E}_{S_t}[f_\theta] - f_\theta(x)\|^2] \\
 &= \sigma^2 + \|\hat{f}(x) - \mathbb{E}_{S_t}[f_\theta]\|^2 + \mathbb{E}_{S_t, \epsilon} [\|f_\theta(x) - \mathbb{E}_{S_t}[f_\theta]\|^2] \\
 &= \sigma^2 + \text{Bias}[f_\theta]^2 + \text{Var}[f_\theta(x)] .
 \end{aligned}$$

The first component is the noise variance from the data generation model. This is an irreducible error from the measurements. The second term is the squared bias term of  $f_\theta$ . It is a systematic model error that is dependent on the choice of the family of parametrised functions  $\mathcal{F}_\theta$ . For instance, if  $f$  is non-linear, yet we approximate it with a family of simple linear functions. This hence produces a systematic bias in model fitting. The third term is the variance of  $f_\theta$ , which measures how much  $f_\theta$  moves around its mean. This error is caused by capturing too much random noise  $\epsilon$  in  $S_t$ .

In Figure 2.3, we present a visualisation of the described bias-variance trade-off. If we make too simplistic model assumptions, we are unlikely to capture the true relations between  $x$  and  $y$ . This phenomenon is known as underfitting. In contrast, if we make too complex model assumptions that even tiny fluctuations in  $S_t$  can be captured, then the model is unlikely to generalise well to data outside of our training samples. This phenomenon is known as overfitting.

Regularisation is an effective tool to combat overfitting. In Section 3, we have seen examples where variational regularisation methods have been utilised in this context. In deep learning practices, explicit regularisation such as weight decay, or trimming connectivity of layer functions such as adding dropout layers have been proposed and used as standard practices to this day. However understanding the intrinsic generalisation properties of trained neural networks still remains an ongoing research topic (Bartlett et al., 2020; Belkin, 2021; Belkin et al., 2019; Kawaguchi et al., 2017; Loog et al., 2020; Zhang et al., 2021b).

This brings us to the end of this section. For further discussions on classical machine learning, we refer interested readers to Bishop and Nasrabadi (2006); Hastie et al. (2009); Murphy (2012); Shalev-Shwartz and Ben-David (2014).

## 4.2 Deep Learning and Artificial Neural Networks

After having defined the learning problem in the classical machine learning setting, we review the deep learning approaches and neural networks in this section. In particular, we first recap fundamental concepts of Artificial Neural Networks (ANNs) before carry on explaining architectural details of some popular neural network models.

A neural network is typically a composition of parametrised functions that defines a mapping between a set of input data to a collection of output values. Historically, ANNs are inspired by the structure and the function of biological neural networks (Hubel and Wiesel, 1959; McCulloch and Pitts, 1943). When it comes to learning or training a neural network, there are two distinct phases, namely the forward phase and the backward phase. The forward phase refers to the process where the network takes input data and processes it through a series of interconnected layers to produce an output or prediction. Hence also known the forward pass or inference phase. The backward phase is commonly known as the back-propagation and refers to the process of computing the gradients of the loss function with respect to the neural network parameters. This is achieved via using the chain rule to propagate the gradients backwards through the entire network, a procedure often known as error back-propagation (Rumelhart et al., 1986). Computed gradients are then used to update network parameters following some optimisation strategies such as the stochastic gradient method (Algorithm 10).

### 4.2.1 Artificial Neuron

Basic units of ANNs are artificial neurons. In 1943, Warren McCulloch and Walter Pitts proposed the first model for artificial neurons, where they named it the Threshold

Logic Unit (TLU), or Linear Threshold Unit (McCulloch and Pitts, 1943). Each neuron computes a weighted sum of its inputs and passes through a fixed value threshold function to produce a binary output. Assume the fixed threshold value is  $\alpha$ , the McCulloch and Pitts neuron is described by

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^n w_i x_i > \alpha, \\ 0 & \text{otherwise} \end{cases} \quad (2.63)$$

where  $x \in \mathbb{R}^n$  and  $y \in \{0, 1\}$  denote the input vector and output value respectively. We say an artificial neuron is activated if the input value is greater than the threshold and is deactivated otherwise.

Although the proposed model piqued great interests, it lacks the ability to learn as it is essentially a hard-wired logic device. Truly learnable networks began with the proposal of the Rosenblatt's Perceptron model (Rosenblatt., 1957).

#### 4.2.2 Layer Functions

A multi-layer neural network is typically composed of several layers of interconnected nodes, where the output of each neuron is fed as input to the neurons in its subsequent layers. Depending on how nodes across layers are interconnected, one can design different architectures of ANNs. Modern deep neural networks are characterised by their notable depth, where transformations from one hidden layer to another is attained via using various choice of layer functions.

As an example, one of the most typical layer functions is the fully connected layer, which corresponds to an interconnection pattern where each neuron is connected to every neuron in the adjacent layer. The fully connected layer function is typically represented by matrix multiplication with a dense matrix  $W \in \mathbb{R}^{m \times n}$ . For the ease of notation, in what follows, we consider a generic layer function defined by  $f(x, \Theta)$  to represent a broader range of interconnection patterns, where  $\Theta$  encodes parameters that parameterise a layer function  $f$ . Fully connected layer for instance is represented by  $f(x, \Theta) = Wx + b$ , with  $\Theta = (W, b)$ .

**Convolutional layer** Another popular choice of the layer function is the convolutional layer, where its name is borrowed from the convolution operation. As an example, the convolution operation on 1- $D$  over two functionals  $x$  and  $k$  is defined as  $s(t) = \int x(a) k(t - a) da$ , denoted by  $s(t) = (x * k)(t)$ . In practice, we usually define the discrete convolution operation and implement it as a finite element summation.

For an image signal  $x$  embedded in a 2- $D$  grid that convolves with a 2- $D$  kernel  $K$ , the convolutional layer in many neural network libraries implements the following cross-correlation operation, where the kernel  $K$  is not flipped:

$$S(i, j) = (K * x)(i, j) = \sum_m \sum_n x(i + m, j + n) K(m, n).$$

The output of a convolutional layer is commonly referred to as the feature map.

An important feature of the convolutional layer is that it exploits the strong spatially local correlation that presents in natural images. The design of convolutional kernels encourage information to be extracted from spatially close inputs. In addition, convolution operation is translational and rotational invariant, which provides convolutional layer the ability to capture local patterns or features regardless of the exact location.

To some extent, convolutional layer can be considered as a regularised version of fully connected layer where the aforementioned properties are hard-coded a priori to encourage sparsity in its connectivity pattern.

**Pooling layer** Pooling layer functions perform down-sampling operations on their inputs. Depends on the sub-sampling size, the input is split into disjoint patches. Each patch is then mapped to a single value output depends on the choice of the pooling operator. Typical options of pooling operators are max pooling operators or average pooling operators, through which the maximum value or mean value is provided respectively. The output of a pooling layer is reduced in spatial dimension but its local information within each patch is maintained. Pooling layers help reduce sensitivity of variations in its input and further improves local translational invariance of a neural network.

### 4.2.3 Activation Functions

Activation functions are usually denoted by  $\sigma(\cdot)$  and they play crucial roles in building artificial neural networks. Non-linear activation functions enable ANNs to have more expressive power to model complex and nonlinear mappings between input and output data pairs (Berner et al., 2021; Cybenko, 1989; Funahashi, 1989; Lu et al., 2017). In this section, we discuss some popular choices of activation functions and their properties.

**Heaviside step functions** Historically, one of the earliest activation functions that is used is the Heaviside step function. The Heaviside step function is a piecewise constant threshold function, as visualised in Figure 2.4. The input is compared to a

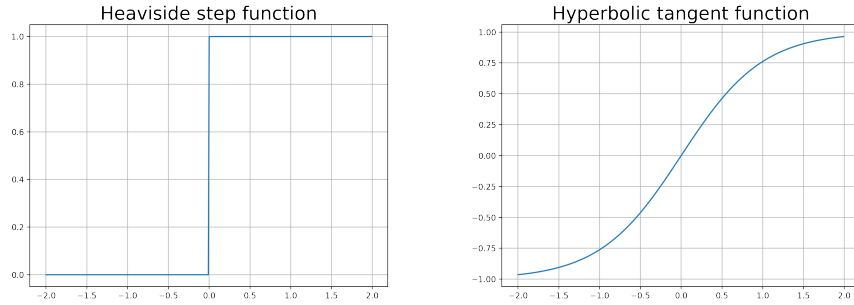


Fig. 2.4 **Left:** the Heaviside function. **Right:** the hyperbolic tangent function.

threshold value to produce an output.

$$H(x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x \geq 0. \end{cases} \quad (2.64)$$

However, there are two major concerns for using the Heaviside step function as an activation function. First of all it is non-smooth as the gradient at zero is not well defined. Secondly, for any  $x \neq 0$  the gradient always evaluates to zero. Both limitations hinder the process of learning, i.e. adaptively modifying neuron parameters' values based on error feedback.

The first limitation can be alleviated by approximating the step function with smooth analytical approximations. Examples are the logistic function and the smooth-step function, which both belong to the family of sigmoid functions.

**Sigmoid functions** A sigmoid function is often characterised by its *S*-shaped curve and continuously differentiability. Due to the easiness to calculate its derivative, sigmoid functions used to be desirable choices of activation functions for it enables the usage of gradient-based optimisation methods in training neural networks.

The hyperbolic tangent function  $\tanh(h)$  is one specific example of sigmoid functions that once enjoyed much popularity. It is defined as

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \quad (2.65)$$

Note that unlike the logistic function, the output of  $\tanh(h)$  is centered around zero, which prevents its outputs to be constantly positive. In addition, note that the hyperbolic tangent function has two horizontal asymptote  $-1$  and  $1$ . As visualised in

Figure 2.4, the further away the input is from the center, the closer its gradient value approaches zero.

This last issue above is in general known as the saturating gradient problem, where we say an activation function  $\sigma$  is saturating if  $\lim_{\|x\| \rightarrow \infty} \|\nabla \sigma(x)\| = 0$ . This caveat is more noticeable for training multi-layer neural networks that use traditional sigmoid activation functions such as  $\tanh(h)$  and logistic sigmoid functions. When the gradients are computed via back-propagation, as a result of the saturation of gradients in sigmoid neurons, the computed gradients will be vanishingly small, eventually preventing network parameters to be effectively updated. This phenomenon is often referred to as the vanishing gradient problem.

**Ramp functions** Perhaps the most popular choice of activation function to date is the rectifier or ramp function, defined as

$$\sigma(x) = \max(0, x) = x \mathbf{1}_{x \geq 0}. \quad (2.66)$$

As visualised in Figure 2.5, note that the Rectified linear units (ReLU) is non-smooth since the derivative does not exist at zero. ReLU suffers less from the vanishing gradient problem, as it only saturates in one direction (Glorot et al., 2011; Krizhevsky et al., 2017). In addition, as long as the input falls in the negative range, the output value will always be zero. As a consequence, fewer neurons will be activated, where eventually one obtains a sparse network in which only few interconnections of the nodes exist, and therefore leads to higher computational efficiency (Glorot et al., 2011).

Despite its popularity, the aforementioned sparsity inducing characteristic of ReLU also gives rise to a concerning issue: the dying ReLU phenomenon. Since the output of ReLU is zero for all negative inputs and the gradient with respect to negative inputs is also zero, a rectified linear neuron might get stuck at zero during training and not contributing to the learning process anymore, thus described as dead. Alternative popular rectifier variants are proposed as attempts to address this issue, where they typically introduce a small positive gradient for negative inputs.

**Soft-thresholding function** Another activation function that also introduces sparsity to the neural network's interconnection is the soft-thresholding or soft-shrinkage activation function, which takes form of the soft-thresholding operator with a non-

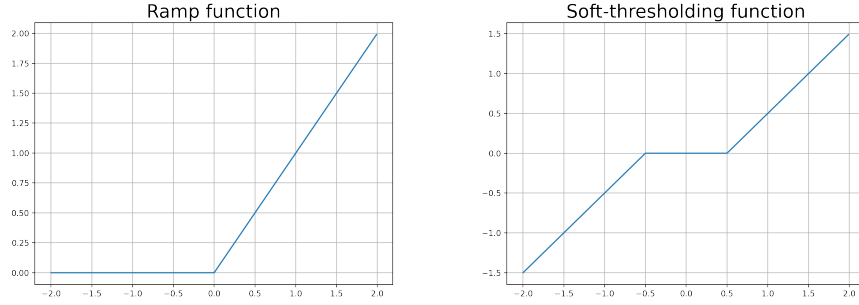


Fig. 2.5 **Left:** the ramp function. **Right:** the soft-thresholding function.

negative parameter  $\alpha \geq 0$ .

$$\sigma(x) = \begin{cases} x - \alpha & \text{if } x \geq \alpha \\ 0 & \text{if } -\alpha \leq x \leq \alpha \\ x + \alpha & \text{if } x \leq -\alpha \end{cases} \quad (2.67)$$

We visualise the soft-thresholding activation function in Figure 2.5.

**Softmax function** Last but not least is the softmax activation function. The softmax function is a multi-dimension generalisation of the sigmoid function. For  $x \in \mathbb{R}^n$ , the softmax activation function is defined as

$$\sigma(z)_j = \frac{\exp(z_j)}{\sum_{k=1}^n \exp(z_k)}, \quad \forall j \in \{1, \dots, n\} \quad (2.68)$$

The softmax function is often used in a supervised learning setting for classification tasks, where it is applied to normalise the network output to a probability distribution.

#### 4.2.4 Activation function as proximal maps

In fact, besides the interpretation as derivatives of convex potentials, most of the activation functions can also be realised as proximal maps (cf. Definition 1.18, Combettes and Pesquet (2020a,b)). This type of activation functions are also introduced in Hasannasab et al. (2020); Hertrich et al. (2021) by the term stable activation functions. In Chapter 3 we will characterise this observation in more details. Here we provide two illustrations.

**Ramp function** The rectifier or ramp function can be interpreted as the proximal map of the characteristic function (Definition 2.4) over the non-negative orthant

$$\delta(x) := \begin{cases} 0 & x \in [0, \infty)^n \\ \infty & \text{otherwise} \end{cases}$$

**Soft-shrinkage function** The well-known soft-shrinkage or soft-thresholding activation function is a proximal map for a positive multiple  $\alpha$  of the  $\ell_1$ -norm  $\|\cdot\|_1$ .

After having reviewed the building blocks, we now have all the ingredients we need to assemble an ANN.

#### 4.2.5 Example Neural Network Architecture

In this section we will introduce some example neural network architectures as well as reviewing their properties. We will begin by defining a general feedforward neural network.

**Feedforward neural network** A standard feedforward neural network  $\mathcal{N} : \mathbb{R}^n \times \mathcal{P} \rightarrow \mathbb{R}^m$  with  $L$  layers can be generically defined as the composition of parametrised functions of the form

$$\mathcal{N}(x, \Theta) = \sigma_L(f(\sigma_{L-1}(f(\dots \sigma_1(f(x, \Theta_1)) \dots)), \Theta_L)), \quad (2.69)$$

for given input data  $x \in \mathbb{R}^n$  and parameters  $\Theta \in \mathcal{P}$ . Here  $\{\sigma_l\}_{l=1}^L$  denotes the collection of nonlinear activation functions and  $f$  denotes a generic layer function parametrised by parameters  $\{\Theta_l\}_{l=1}^L$ . To ease the notation, we use  $\Theta$  to denote  $\{\Theta_l\}_{l=1}^L$ . As an example, a rectified linear unit (ReLU) (Nair and Hinton, 2010) can be constructed via the composition of  $\sigma_l(\cdot) := \max(0, \cdot)$  and  $f(x, \Theta_l) = W_l x + b_l$  for  $\Theta_l = (W_l, b_l)$ , where the matrix  $W_l$  is usually referred to as the weights and the vector  $b_l$  as the bias term of layer  $l$ . In Figure 2.6 we visualise a fully connected feedforward neural network. The width of a layer is characterised by the number of neurons in that layer. The width of a neural network is determined by the width of the widest layer of a neural network (Lu et al., 2017). The depth of a neural network is characterised by the number of layers  $L$ .

As a special instance, the perceptron or a single-layer perceptron is a feedforward neural network with one layer, i.e.  $\mathcal{N}(x, \Theta_1) = \sigma(f(x, \Theta_1))$ .

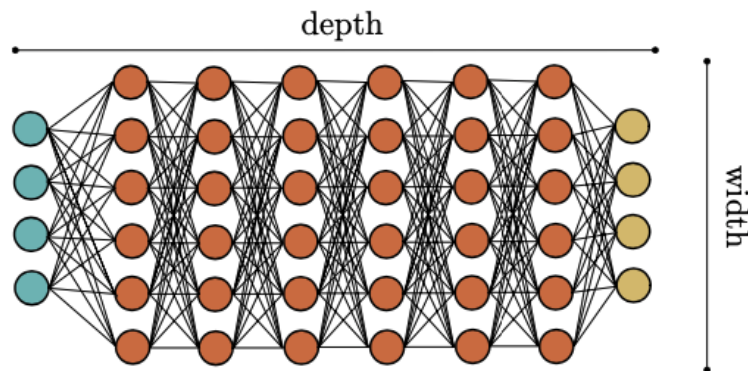


Fig. 2.6 Visualisation of a standard fully connected feedforward neural network. Figure taken from Berner et al. (2021).

**Convolutional neural networks** Convolutional neural networks (CNNs) are biologically inspired variants of feedforward neural networks (Fukushima, 1980; Hubel and Wiesel, 1959) that have shown great success in various computer vision tasks and imaging problems in the past decade (He et al., 2015b; Krizhevsky et al., 2017; Long et al., 2015; Ren et al., 2015; Simonyan and Zisserman, 2014). Historically, it has been observed that the convolution process shares similarities with how a neuron in the visual cortex generates response to a specific stimulus (Hubel and Wiesel, 1959). Inspired by the work of Hubel and Wiesel, the neocognitron (Fukushima, 1980) was introduced by Kunihiro Fukushima in 1980. The neocognitron model is one of the earliest attempts to employ a convolutional layer in its architecture.

Broadly speaking, a CNN refers to a neural network that incorporates convolution operation as a layer function in at least one of its layers. A convolutional block is the basic unit in an CNN, usually consists of a convolutional layer, an activation function, and a subsequent pooling layer, that applied on the feature map produced from the previous convolutional layer. Modern CNNs are often built by stacking multiple convolutional blocks before passing forward into a fully connected neural network to produce outputs.

LeNet-5 (LeCun et al., 1998) is one of the first successful convolutional neural networks trained for the purpose of recognising handwritten digits in images. Introduced by Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner in 1998, LeNet soon gained significant attention for its performance. LeNet-5 consists of two parts: 1) two successive stacks of convolutional blocks to reduce the size of the spatial dimension while increasing the number of learned channels and 2) a three-layer fully connected neural network to produce the predicted outputs. Each convolutional block is composed

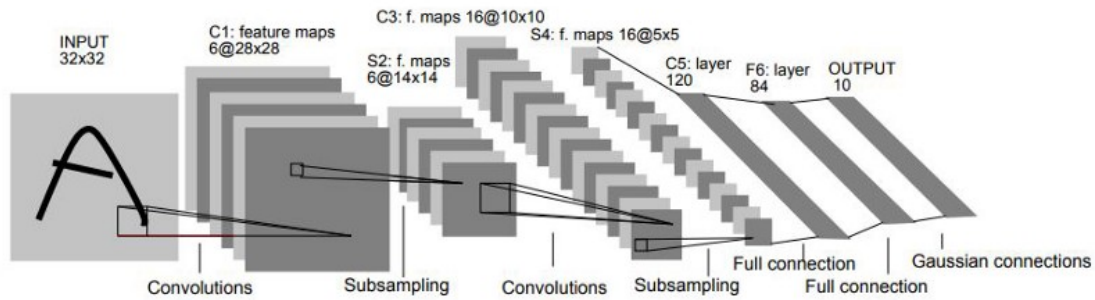


Fig. 2.7 Visualisation of the architecture of LeNet-5. Figure taken from LeCun et al. (1998).

of a convolutional layer, a sigmoid activation function, and an average pooling layer. A visualisation of the architectural design of LeNet-5 is provided in Figure 2.7.

**Autoencoders** Autoencoders are special types of feedforward neural networks that are trained to reconstruct the network inputs. In terms of architecture, an autoencoder is a composition of two mathematical operators: an encoder  $\mathcal{E}$  and a decoder  $\mathcal{D}$ , as visualised in Figure 2.8. The encoder maps input data onto latent variables in the latent space. The latent variables are usually referred to as the code. The decoder aims to recover the input data from the code.

The most standard autoencoder is the undercomplete autoencoder where one constrains the code to have a smaller dimension than the input dimension. In the scope of this thesis, a topic of interest are sparse autoencoders. Unlike undercomplete autoencoders, the latent space of a sparse autoencoder can have the same or even larger dimension than the input space. The compression of the input data is achieved by ensuring that only relatively few coefficients of the code are non-zero. Their advantage over conventional undercomplete autoencoders is that the position of the non-zero coefficients can vary for individual input data.

Multiple approaches have been proposed to achieve code sparsity. For instance, one approach is to penalise the Kullback–Leibler divergence between neurons activation distribution in the code and an ideal sparse distribution (Ng et al., 2011). Alternatively, one can also directly impose  $l_1$ -norm regularisation on the hidden code to enforce sparsity.

Denoising autoencoder (Vincent et al., 2010) is another type of regularised autoencoder. Instead of explicitly adding regularisation terms in the training objective, denoising autoencoders consider a different learning objective that is to reconstruct clean input data from the noise corrupted version. Denoising autoencoders hence can

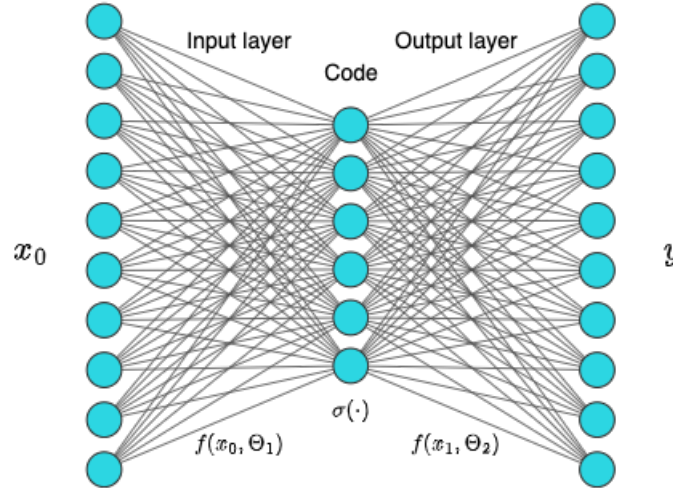


Fig. 2.8 Visualisation of a simple two-layer fully connected undercomplete autoencoder.

be viewed as autoencoders with implicit regularisation that are forced to capture the structure of the training data (Alain and Bengio, 2014; Bengio et al., 2013b).

**Residual neural networks** Another type of neural network we want to review are Residual Neural networks (ResNet). First proposed by He et al. (2016) and also discussed in the context of highway networks in Srivastava et al. (2015), residual neural networks have had a significant influence on deep learning practices. To this date, we see its concept in various neural network architectures and applications (Kipf and Welling, 2016; Redmon and Farhadi, 2018; Ren et al., 2015; Vaswani et al., 2017).

The major difference between residual neural networks and standard feedforward neural networks is the way they handle the flow of information through the network. In standard feedforward neural networks, outputs in each layer flow entirely to their immediate next layer as inputs. Whereas residual neural networks introduces skip connections that connect a layer to a later layer that is not directly adjacent to it in the neural network.

The residual block stands out as the most notable characteristic of a residual network architecture. The work of He et al. (2016) was inspired by the observation of only marginal gain when constructing deeper neural networks, leading them to the question of how to ensure that adding a new layer to the original model would be at

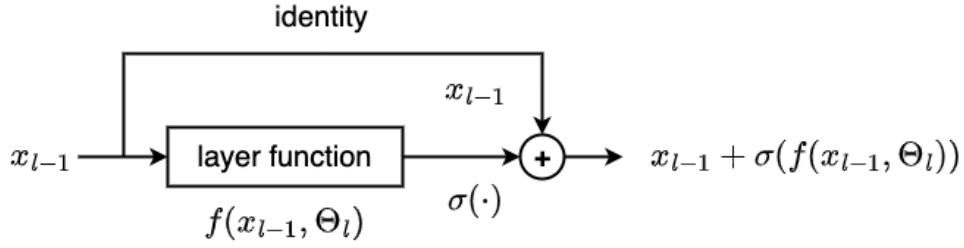


Fig. 2.9 Visualisation of the scheme of a residual block.

least as effective as using the original model alone. To solve this problem, the authors proposed a simple yet highly effective solution: the new layer should be capable of effortlessly learning an identity function, which was the inspiration behind the design of the residual block.

In Figure 2.9, we provide the visualisation of a residual block, where we denote the activated output at the  $l$ -th layer by  $x_l = \sigma_l(f(x_{l-1}, \Theta_l))$ . By introducing residual blocks, the flow of information in a neural network is improved by creating a shortcut that allows information to bypass certain layers effectively.

The U-net (Ronneberger et al., 2015) architecture proposes a similar concept that utilises skip connections for image segmentation tasks. In this architecture, the expanding path is connected to the contracting path through skip connections, enabling the network to leverage both low-level and high-level features for more effective segmentation. Generically speaking, any two activated outputs in an  $L$ -layer neural network  $x_{l_1}, x_{l_2} \in \{x_0, \dots, x_L\}$  can be connected via skip connections, resulting in neural networks that are described by the following relation

$$x_{l_2} = x_{l_1} + \sigma_l(f(x_{l_1}, \Theta_{l_1})) \quad \text{for} \quad l_1, l_2 \in \{0, 1, \dots, L\}.$$

This concludes our discussion on deep learning and artificial neural networks in this section. We refer interested readers to Berner et al. (2021); Bottou (1998); Goodfellow et al. (2016a); Higham and Higham (2019); Vidal et al. (2017); Zhang et al. (2021a) for further and more extensive discussions.



# Chapter 3

## Generalised Perceptron Learning

In this chapter, we begin with the problem of learning one of the simplest forms of a neural network, namely the perceptron, also known as a single-layer ANN. The development of perceptrons is an important milestone in the history of ANNs. As one of the precursors in building intelligent learning systems and constructing learnable machines, this model has played a crucial role in shaping early forms of neural networks.

In particular, we consider the problem of training a broader family of generalised perceptrons with non-smooth activation functions. Since the learning problem involves minimising non-smooth objectives, standard training procedures often adopt subgradient-based optimisation strategies. These approaches often have undesired properties such as slow convergence rates (Bertsekas, 1999; Chambolle and Pock, 2016a; Polyak, 1987). Through the example of learning perceptrons, we want to motivate the use of a tailored loss function as the training objective function.

We will show how this generalisation is capable of mitigating potential issues brought forward in the standard gradient-based training approaches that use the back-propagation algorithm. Our main contributions are three-fold:

- We establish a key observation that commonly used activation functions can be interpreted as proximal mappings and propose the use of a tailored Bregman distance as the training loss function.
- We analyse properties of the proposed Bregman loss function and show the key advantage that its partial derivative does not require the differentiation with respect to non-smooth activation functions.
- We confirm that the historical Rosenblatt’s perceptron learning algorithm (Rosenblatt., 1957) indeed can be interpreted as an incremental gradient descent method

with respect to a special choice of energy. This energy minimisation perspective also generalises the classical ADALINE delta rule.

The energy minimisation interpretation allows us to devise training approaches that avoid the computation of sub-differentials with respect to non-smooth activation functions and therefore provide better handling of the non-smoothness in the overall minimisation objective.

### **Organisation of the Chapter**

This chapter is organised as follows. We will first review the history of perceptrons and two well-known perceptron learning rules in Section 1 and Section 2. Next, we consider the family of generalised perceptrons and formulate the perceptron learning problem in Section 3. We introduce an energy function based on the generalised Bregman distance and demonstrate how Rosenblatt's learning algorithm can be interpreted as an incremental gradient method in Section 4. In Section 5 we consider example applications of learning sparse perceptrons. We present numerical results and compare the results to those obtained from classical subgradient-based methods. We conclude this chapter with a brief discussion in Section 6.

# 1 Rosenblatt's Perceptron

The history of developing artificial intelligence begins with attempts on constructing learnable machines (Vapnik, 1999). The 1960s saw the proposal of one of the first model of such a learnable machine, suggested by Frank Rosenblatt and was named the Perceptron (Rosenblatt., 1957).

Rosenblatt's perceptron is based on the well-established McCulloch & Pitts neuron model (McCulloch and Pitts, 1943), where the perceptron takes an  $n$ -dimensional vector input  $x \in \mathbb{R}^n$  and produces a scalar-valued binary output  $y \in \{0, 1\}$ . The output is described by the following mapping:

$$f(x) = \begin{cases} 1, & \text{if } w^\top x > 0 \\ 0, & \text{otherwise} \end{cases}, \quad (3.1)$$

where the Heaviside step function (cf. Chapter 2 Definition 2.64) is used as the activation function and  $w \in \mathbb{R}^n$  is a learnable adaptive parameter.

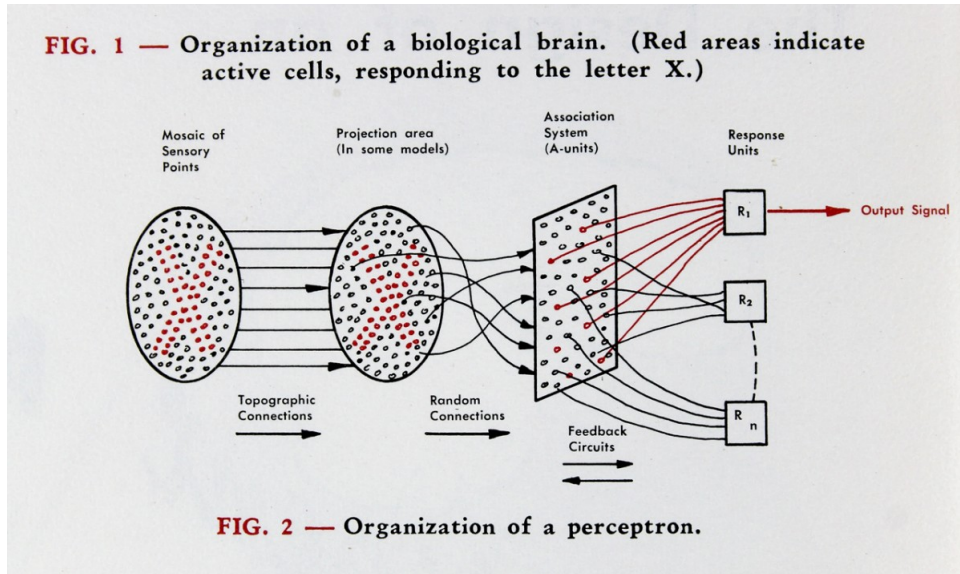


Fig. 3.1 An image of Rosenblatt's perceptron model (Cornell Chronicle, 2022).

Rosenblatt initially considered a composite model that consists of several levels of neurons. However, at the time, it was unclear and impractical to determine suitable parameters for all neurons in a simultaneous manner. Instead, Rosenblatt proposed a two-part architecture, which is shown in Figure 3.1. The architecture consists of 1) a pre-processing part in which all the neurons' parameters are fixed, and 2) a single level

adaptive processing unit, where parameters of this last layer of neurons are learned during the training process.

Along with the perceptron model, Rosenblatt also introduced a simple iterative procedure for learning the parameters in a supervised learning setting. In what follows, we describe this iterative procedure.

1. Initialise  $w^0 = \mathbf{0}$  and let  $\{(x^1, y^1), (x^2, y^2), \dots, (x^s, y^s)\}$  be the collection of  $s$  input samples to the learnable perceptron.
2. At each step, one training sample  $(x^k, y^k)$  is fed into the perceptron.
  - If the training sample  $(x^k, y^k)$  is classified correctly, i.e.  $(2y^k - 1)(w^{k\top} x^k) > 0$ , the parameters remain unchanged:

$$w^{k+1} = w^k.$$

- If the training sample is classified incorrectly, i.e.  $(2y^k - 1)(w^{k\top} x^k) < 0$ , the parameters are modified according to the rule:

$$w^{k+1} = w^k + (2y^k - 1)(x^k)^\top.$$

Note that in the update step where there is a misclassified data sample, we have written the two possible situations: 1)  $y^k = 1, f(x^k) = 0$  and 2)  $y^k = 0, f(x^k) = 1$ , in one compact expression as  $2y^k - 1$ .

## 1.1 Convergence of Rosenblatt's Perceptron Learning Rule

Rosenblatt's original perceptron learning rule can be proved convergent only when the training data is linearly separable, or equivalently when a hyperplane can be found that perfectly separates the two classes (Rosenblatt, 1961). More rigorously, we recall the following definition for linear separability.

**Definition 1.1** (Linear separability). *Let  $S_1$  and  $S_2$  be two sets of points in an  $n$ -dimensional Euclidean space. We say  $S_1$  and  $S_2$  are linearly separable if there exist  $n+1$  real numbers  $w_1, w_2, \dots, w_n, b$  such that every point  $x \in S_1$  satisfies  $\sum_{i=1}^n w_i x_i > b$  and every point  $x \in S_2$  satisfies  $\sum_{i=1}^n w_i x_i < b$ , where  $x_i$  is the  $i$ -th component of  $x$ .*

Equivalently, linear separability can be characterised by convex sets. Two sets are linearly separable if their respective convex hulls (cf. Chapter 2 Definition 1.7) are disjoint. We make the following assumption on linear separability of the training data.

**Assumption 1.1** (Linear separability). *Let us assume there exists an optimal  $w^* \in \mathbb{R}^n$  where without loss of generality let  $\|w^*\| = 1$ , and some  $\gamma > 0$ , such that for all  $k \in \{0, 1, \dots, s\}$ , the following holds:*

$$(2y^k - 1) \cdot w^{*\top} x^k > \gamma.$$

In addition, we also make the following assumption on the bounded norm for all data points  $x^k \in \mathbb{R}^n$ .

**Assumption 1.2** (Bounded norm). *For all  $k \in \{0, 1, \dots, s\}$ , there exists a positive constant  $B > 0$ , such that  $\|x^k\| \leq B$ .*

Under the two assumptions, we have the following convergence theorem (Collins, 2012; Kalyanakrishnan, 2017), where the Rosenblatt's perceptron learning algorithm as described above will guarantee finding optimal parameters  $w^*$  that classifies every training sample correctly.

**Theorem 1.1.** *Let the linear separability (Assumption 1.1) and bounded norm (Assumption 1.2) both be satisfied. Then, the Rosenblatt's perceptron learning algorithm terminates after at most  $\frac{B^2}{\gamma^2}$  updates according to misclassified cases, and returns a separating hyperplane.*

*Proof.* Note that when we make an update on  $w^{k+1}$  resulted from a misclassified point, we have

$$\begin{aligned} \langle w^*, w^{k+1} \rangle &= \langle w^*, w^k + (2y^k - 1)(x^k)^\top \rangle \\ &= \langle w^*, w^k \rangle + \langle w^*, (2y^k - 1)(x^k)^\top \rangle \\ &> \langle w^*, w^k \rangle + \gamma. \end{aligned}$$

It follows from induction that  $\langle w^*, w^{k+1} \rangle > k\gamma$ .

On the other hand we know  $\langle w^*, w^{k+1} \rangle \leq \|w^*\| \|w^{k+1}\| = \|w^{k+1}\|$ . Hence we obtain

$$\|w^{k+1}\| > k\gamma.$$

To obtain an upper bound, we have that

$$\begin{aligned}
\|w^{k+1}\|^2 &= \|w^k + (2y^k - 1)(x^k)^\top\|^2 \\
&= \|w^k\|^2 + \|(2y^k - 1)(x^k)\|^2 + 2\langle w^k, (2y^k - 1)(x^k)^\top \rangle \\
&\leq \|w^k\|^2 + \|x^k\|^2 \\
&\leq \|w^k\|^2 + B^2.
\end{aligned}$$

By induction we have  $\|w^{k+1}\|^2 \leq kB^2$ . Therefore we obtain both a lower bound and an upper bound for  $\|w^{k+1}\|^2$  as

$$k^2\gamma^2 < \|w^{k+1}\|^2 \leq kB^2.$$

Hence  $k < \frac{B^2}{\gamma^2}$ . □

However, as pointed out by Minsky and Papert in 1969 (Minsky and Papert, 1969), a major limitation of Rosenblatt's original algorithm is its lack of robustness in the learning process. Indeed, although the iteration stops once no more mistakes are made on classifying the training set, the decision boundary returned is sub-optimal as the generalisation error could be potentially large albeit having zero error on the training data.

## 2 Adaptive Linear Neuron (ADALINE)

Adaptive Linear Neuron (ADALINE) (Widrow, 1960) is another early learning system that was developed at about the same time as Rosenblatt's Perceptron learning rule, by Widrow and Hoff in 1960. The author developed the idea of ADALINE and is widely regarded as a crucial foundational element in the advancement of artificial neural networks (Goodfellow et al., 2016a).

The ADALINE learning system offers an alternative method for training perceptrons. The output of the system was also based on the McCulloch & Pitts neuron model but originally used the sign function as the threshold function, described as following:

$$f(x) = \begin{cases} 1, & \text{if } w^\top x > 0 \\ -1, & \text{otherwise} \end{cases}. \quad (3.2)$$

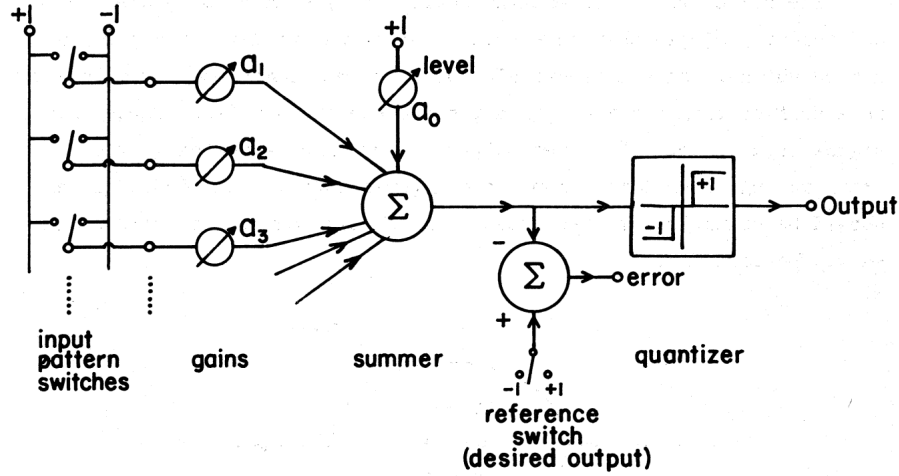


Fig. 3.2 Schematic of ADALINE (Widrow, 1960).

With a given collection of  $s$  input data samples  $\{(x^1, y^1), \dots, (x^s, y^s)\}$ , the learnable parameters  $w$  is updated according to the delta rule during training:

$$w^{k+1} = w^k + \tau(w^{k\top} x^k - y^k)(x^k)$$

where  $\tau$  is the step size parameter.

The ADALINE delta rule is an improvement over Rosenblatt's perceptron learning rule. Even in the case when training data is not linearly separable, it is still able to converge to a minimum squared 2-norm error solution using sufficiently small learning step size. In fact the ADALINE delta rule owes its success to its disregard for the discontinuity that comes with the non-linear threshold functions. But more importantly, the delta rule can be seen as an iterative procedure of minimising a specific choice of energy.

In the next sections, we first introduce a more general perceptron model. Then, from the perspective of empirical risk minimisation, we establish a connection between the two earliest perceptron learning rules and gradient-based minimisation.

### 3 Generalised Perceptrons

Recall from (2.69), a generalised perceptron can be considered as an artificial neuron of the form

$$y = \sigma(w^\top x + b). \quad (3.3)$$

Here  $\sigma$  denotes a (point-wise) non-linear activation function,  $w \in \mathbb{R}^{m \times n}$  is the weight-matrix and  $b \in \mathbb{R}^n$  is the bias-vector. The vector  $x \in \mathbb{R}^m$  and the vector  $y \in \mathbb{R}^n$  denote the input, respectively the output, of the perceptron. As described in Section 1, Rosenblatt proposed the first perceptron learning algorithm. Here we summarise it in Algorithm 13 with respect to using generalised perceptrons, where  $s$  denotes the number of training samples.

---

**Algorithm 13:** Generalised Rosenblatt's Perceptron Learning Algorithm

---

```

Initialize  $w^0, b^0$ .
for  $k = 1, 2, \dots$  do
    for  $i = 1, \dots, s$  do
         $e^i = y^i - \sigma((w^k)^\top x^i + b^k)$ ,
         $w^{k+1} = w^k + e^i x^{i\top}$ ,
         $b^{k+1} = b^k + e^i$ .
    end
end

```

---

By the late 1960s the field of statistical learning theory had been developed. The learning problem is framed as a function estimation problem on the basis of empirical data (Vapnik, 1999). As early as discussed in Tsyppkin and Nikolic (1971), the goal of a learning system comprises finding the minimum of the empirical risk function, which is an approximation of the expected risk function using a finite collection of observed data samples (cf. Chapter 2 Definition 2.58).

Therefore the problem of learning the weights and biases of a generalised perceptron (3.3) can be formulated as the empirical risk minimisation problem of the following form:

$$\min_{w,b} J(w,b) := \frac{1}{s} \sum_{i=1}^s \ell(y^i, \sigma(w^\top x^i + b)) + \alpha R(w,b), \quad (3.4)$$

where  $J$  is the overall cost- or objective-function and  $\ell$  is the loss function that is chosen a-priori (usually based on prior assumptions on the statistical distribution of the data). The function  $R$  is a regularisation function that allows to encode prior information on weight and bias which can be useful to combat ill-conditioning of the data matrix (Benning and Burger, 2018) and help to control the validation error (Goodfellow et al., 2016a; Shalev-Shwartz and Ben-David, 2014). The two terms are balanced with a positive regularisation parameter  $\alpha$ .

The overall objective function in Equation (3.4) is usually minimised by a gradient- or subgradient-based algorithm such as (sub-)gradient descent. If we for instance

**Algorithm 14:** Mini-batch Subgradient Descent

---

```

Initialize  $w^0$  and  $b^0$ .
for  $k = 1, 2, \dots$  do
    Choose  $B_k \subset \{1, 2, \dots, s\}$  either at random or deterministically.
     $g_w^k = \frac{1}{|B_k|} \sum_{i \in B_k} [\sigma((w^k)^\top x^i + b^k) - y^i] \sigma'((w^k)^\top x^i + b^k) x^i$ ,
     $w^{k+1} = w^k - \tau_w^k g_w^k$ ,
     $g_b^k = \frac{1}{|B_k|} \sum_{i \in B_k} [\sigma((w^k)^\top x^i + b^k) - y^i] \sigma'((w^k)^\top x^i + b^k)$ ,
     $b^{k+1} = b^k - \tau_b^k g_b^k$ .
end

```

---

choose  $R \equiv 0$  and  $\ell(y, \sigma(w^\top x + b)) = \frac{1}{2} \|y - \sigma(w^\top x + b)\|^2$  and minimise (3.4) via mini-batch subgradient descent, we obtain the algorithm described in Algorithm 14, where  $B_k \subset \{1, 2, \dots, s\}$  is a (mini-)batch of indices chosen at iteration  $k$ . When  $|B_k| = 1$ , this corresponds to the incremental or stochastic subgradient descent method (Algorithm 11 and Algorithm 10). Here,  $\sigma'$  denotes the derivative of  $\sigma$ , respectively a subgradient if  $\sigma$  is not differentiable. Hyper-parameters  $\tau_w^k > 0$  and  $\tau_b^k > 0$  denote the learning rates at iteration  $k$ .

### 3.1 Delta Rule of ADALINE

Notice that as mentioned earlier in Section 2, the delta rule of ADALINE updates the neuron's parameters in similar fashion as the incremental or stochastic subgradient descent method. In fact, the ADALINE model introduces the idea of loss functions that evaluate the error using the linear activated output. In other words, instead of dealing with the induced loss from the output of the threshold unit  $\sigma(w^\top x + b)$ , the algorithm considers  $w^\top x$  as a linear approximation to the original induced loss. The delta rule of ADALINE, also broadly known as the Widrow-Hoff update rule, is therefore indeed an incremental gradient descent method but applied to the following loss function:

$$\frac{1}{s} \sum_{i=1}^s \frac{1}{2} \|y^i - (w^\top x^i + b)\|^2,$$

aims at solving a linear least squares problem.

We will reveal in the following section that Rosenblatt's learning algorithm can in fact also be interpreted as a special case of incremental gradient descent method but with respect to a particular choice of energy.

## 4 Learning Perceptrons via Minimising Bregman Loss Functions

In this section, we propose the use of a tailored Bregman loss function as the data term  $\ell$  in Equation (3.4) instead of using the squared two-norm. This replacement circumvents the computation of sub-differentials with respect to non-smooth activation functions when handling the non-smoothness in the overall minimisation objective, and essentially avoids resorting to subgradient based training approaches.

In this set-up, we will show that the Rosenblatt's learning Algorithm 13 can be interpreted as an incremental gradient descent method applied to a particular energy function for a special class of activation functions that can be expressed as proximal maps (Moreau, 1962) (cf. Chapter 2 Definition 1.18).

**Example 4.1** (Activation function as proximal maps). *There are numerous cases where commonly used activation functions in deep learning are also proximal maps (cf. Combettes and Pesquet (2020a); Hasannasab et al. (2020); Li et al. (2019); Zhang and Brand (2017)).*

*As we have seen in Chapter 2, the rectified Linear Unit (ReLU) (cf. Nair and Hinton (2010)) is the proximal map for the characteristic function over the non-negative orthant and the soft-thresholding activation function is a proximal map for  $\alpha \|\cdot\|_1$  for some  $\alpha > 0$ . Here we provide two more specific examples.*

*Common smooth activation functions like the hyperbolic tangent (cf. Chapter 2 Definition 2.65) can also be framed as proximal maps. The hyperbolic tangent can be recovered by choosing the characteristic function*

$$\Psi(u) := \begin{cases} u \tanh^{-1}(u) + \frac{1}{2} (\log(1 - u^2) - u^2) & |u| < 1 \\ \infty & \text{otherwise} \end{cases} \implies \sigma(z) = \tanh(z).$$

*All previous examples were separable activation functions. Our final example is the non-separable softmax activation function (cf. Chapter 2 Definition 2.68), which we can obtain as a proximal map for the characteristic function*

$$\begin{aligned} \Psi(u) &:= \begin{cases} \sum_{j=1}^n \left[ u_j \log(u_j) - \frac{1}{2} u_j^2 \right] & u_j \geq 0, \sum_{j=1}^n u_j = 1 \\ \infty & \text{otherwise} \end{cases} \\ \implies \sigma(z)_j &= \frac{\exp(z_j)}{\sum_{k=1}^n \exp(z_k)}, \quad \forall j \in \{1, \dots, n\}. \end{aligned}$$

In Figure 3.3, we show visualisations of the above mentioned proximal activation functions.

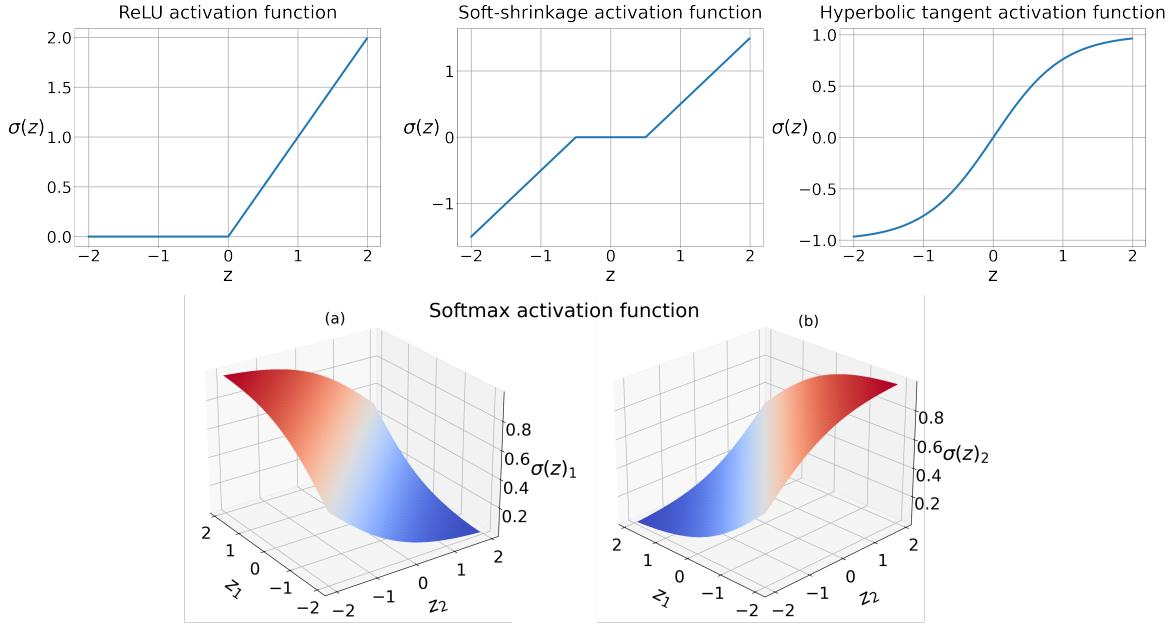


Fig. 3.3 This figure shows example activation functions that are also proximal maps. From left to right on the top row are the ReLU activation function, the soft-thresholding activation function (threshold value  $\alpha = 0.5$ ) and the hyperbolic tangent activation function respectively. The bottom two figures visualise components of the softmax activation function for the case of  $n = 2$ .

## 4.1 A Motivational Example

The loss function that we proposed here is based on the (generalised) Bregman distance (cf. Chapter 2 Definition 1.12). To demonstrate the rationale behind employing this loss function, we start with a motivating example. Consider learning a single-layer perceptron with proximal activation function. For notation simplicity, here we only consider one data point, which formulates the following minimisation problem

$$\arg \min_{w,b} \ell(y, \sigma(w^\top x + b)), \quad (3.5)$$

where  $\ell(\cdot)$  is the loss function. Observe that for proximal activation functions, the activated output  $\sigma(w^\top x + b)$  is the proximal point with respect to the transformed

input  $w^\top x + b$ . More specifically, we can define the following energy  $E_z$  where

$$E_z(u) = \frac{1}{2} \|u - z\|^2 + \Psi(u), \quad (3.6)$$

such that

$$u^* = \arg \min_u E_z(u) = \sigma(z),$$

Note that the perceptron learning problem (3.5) can hence be rewritten as a bilevel optimisation problem

$$\begin{aligned} & \arg \min_{(w,b)} \ell(y, u^*) \\ \text{s.t. } & u^* = \arg \min_u E_{(w^\top x + b)}(u). \end{aligned} \quad (3.7)$$

Let us for instance consider the squared two-norm as the loss function. Inspired from the majorisation-minimisation argument in Geiping and Moeller (2019), we can instead minimise an upper bound of (3.5) with a Bregman surrogate function since the energy term  $E_z$  is 1-strongly convex. The learning problem can be instead formulated by the following:

$$\arg \min_{(w,b)} D_E(y, \sigma(w^\top x + b)), \quad (3.8)$$

More formally, based on the definition of the generalised Bregman distance and the assumption  $y \in \text{dom}(\Psi)$ , we propose the following Bregman distance function as the data term

$$\begin{aligned} D_{E_z}(y, \sigma(z)) &= E_z(y) - E_z(\sigma(z)) - \langle \nabla E_z(\sigma(z)), y - \sigma(z) \rangle \\ &= E_z(y) - E_z(\sigma(z)), \end{aligned} \quad (3.9)$$

where for convenience, we will denote  $D_{E_z}(y, \sigma(z))$  by a shorthand notation  $D_E(y, z)$ . One motivation in using Equation (3.9) as a loss function (instead of only using the squared two norm) lies in the simplicity of form of its gradient.

**Theorem 4.1.** *For fixed  $y \in \text{dom}(\Psi)$ , and suppose  $\Psi : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$  is a proper, convex and lower semi-continuous function. The gradient of  $D_E$  as defined in Equation*

(3.9) with respect to the argument  $z$  reads

$$\nabla_z D_E(y, z) = \sigma(z) - y.$$

*Proof.* Let  $E_z(\sigma(z))$  denote the Moreau-Yosida Regularisation (Moreau, 1965; Yosida, 1964) of  $\Psi$ , where  $E_z$  is defined as the above activation energy (3.6). It is well known (cf. (Parikh and Boyd, 2014)) that the gradient of the Moreau-Yosida regularisation satisfies

$$\nabla E_z(\sigma(z)) = z - \sigma(z).$$

Therefore, we derive the gradient of  $E$  with respect to the argument  $z$  as

$$\begin{aligned} \nabla_z D_E(y, z) &= \nabla_z E_z(y) - \nabla_z E_z(\sigma(z)) \\ &= z - y - z + \sigma(z) \\ &= \sigma(z) - y. \end{aligned}$$

□

**Remark 4.1.** Note that  $\nabla E_z(\sigma(z))$  is the gradient of  $z \mapsto E_z(\sigma(z))$ , and not the gradient of  $E_z$  evaluated at  $\sigma(z)$  as the latter is zero since  $\sigma(z)$  minimises  $E_z$ .

Furthermore, we can show that the proposed Bregman loss function  $D_E$  is indeed a Bregman surrogate function of the squared two-norm loss function which provides an upper bound captured by the following

**Corollary 4.1.** For fixed  $y \in \text{dom}(\Psi)$ , the Bregman loss function  $D_E$  is an upper bound of the squared 2-norm loss function

$$D_E(y, z) = \frac{1}{2} \|y - \sigma(z)\|^2 + D_{\Psi}^{z - \sigma(z)}(y, \sigma(z)) \quad (3.10)$$

for the (valid) subgradient  $z - \sigma(z) \in \partial\Psi(\sigma(z))$  and  $D_{\Psi}$  denotes the Bregman distance with respect to the underlying functional  $\Psi$ .

*Proof.* To see this, observe that we can expand the Bregman loss function as the following

$$\begin{aligned}
E_z(y) - E_z(\sigma(z)) &= \frac{1}{2}\|y - z\|^2 + \Psi(y) - \frac{1}{2}\|\sigma(z) - z\|^2 - \Psi(\sigma(z)) \\
&= \frac{1}{2}\|y - z\|^2 - \frac{1}{2}\langle \sigma(z) - y + y - z, \sigma(z) - z \rangle + \Psi(y) - \Psi(\sigma(z)) \\
&= \frac{1}{2}\langle y - z, y - z - \sigma(z) + z \rangle - \frac{1}{2}\langle \sigma(z) - z, \sigma(z) - y \rangle + \Psi(y) - \Psi(\sigma(z)) \\
&= \frac{1}{2}\langle y - z, y - \sigma(z) \rangle + \frac{1}{2}\langle \sigma(z) - z, y - \sigma(z) \rangle + \Psi(y) - \Psi(\sigma(z)) \\
&= \frac{1}{2}\langle y - \sigma(z), y - \sigma(z) \rangle - \langle z - \sigma(z), y - \sigma(z) \rangle + \Psi(y) - \Psi(\sigma(z)) \\
&= \frac{1}{2}\|y - \sigma(z)\|^2 + \Psi(y) - \Psi(\sigma(z)) - \langle z - \sigma(z), y - \sigma(z) \rangle \\
&= \frac{1}{2}\|y - \sigma(z)\|^2 + D_{\Psi}^{z-\sigma(z)}(y, \sigma(z)).
\end{aligned}$$

□

If we use the Bregman loss data term as defined in (3.9) in the Minimisation Problem (3.4) and apply an incremental gradient descent strategy with constant step-size or learning rate one, we immediately obtain Rosenblatt's learning algorithm as defined in Algorithm 13 as a consequence of the chain rule.

In other words, an incremental or stochastic gradient descent method applied to the Bregman loss function (3.9) generalises Rosenblatt's original learning algorithm to a broader class of (proximal) activation functions, for which Algorithm 13 can be interpreted as an energy minimisation method, just with the different Energy (3.9). We also note that as a special case when the function  $\Psi$  is simply zero, the activation function  $\sigma$  is the identity function. In this case (3.9) is equivalent to the squared loss  $\frac{1}{2}\|y - w^\top x\|^2$ . As expected, applying the incremental gradient descent to this energy recovers the ADALINE delta rule.

The beauty of interpreting Rosenblatt's learning algorithm as an energy minimisation problem is that we can apply many other algorithmic strategies to the same energy. In the following section, we demonstrate how we can easily set up an iterative soft-thresholding generalisation of the Rosenblatt algorithm to train a perceptron with sparse weights without having to differentiate the activation function.

**Algorithm 15:** Rosenblatt-ISTA Algorithm

---

```

Initialize  $w^0$  and  $b^0$ .
for  $k = 1, 2, \dots$  do
    Choose  $B_k \subset \{1, 2, \dots, s\}$  either at random or deterministically.
     $g_w^k = \frac{1}{|B_k|} \sum_{i \in B_k} [\sigma((w^k)^\top x^i + b^k) - y^i] x^{i^\top}$ ,
     $\tilde{w} = w^k - \tau_w^k g_w^k$ ,
     $w^{k+1} = \arg \min_w \{ \frac{1}{2} \|\tilde{w} - w\|^2 + \alpha \|w\|_1 \}$ ,
     $g_b^k = \frac{1}{|B_k|} \sum_{i \in B_k} [\sigma((w^k)^\top x^i + b^k) - y^i]$ ,
     $b^{k+1} = b^k - \tau_b^k g_b^k$ .
end

```

---

## 5 Application and Numerical Experiments

Sparsity promoting regularisation on artificial neural networks has long received significant attention over years (Aghasi et al., 2017; Azarian et al., 2020; Collins and Kohli, 2014; Hoeffler et al., 2021; Huang and Wang, 2018; Liu et al., 2015; Scardapane et al., 2017; Wen et al., 2016). In this section, we consider an application problem where we use the energy minimisation interpretation to develop a way of learning sparse perceptrons (cf. (Jackson and Craven, 1996)), with the benefit of not having to differentiate the activation function.

We consider a LASSO-type cost function (Tibshirani, 1996), i.e. we choose the regularisation term  $R$  in Equation (3.4) to be  $R(w, b) = \|w\|_1$  in order to promote sparsity of the weight matrix. Here,  $\|\cdot\|_1$  denotes the one-norm. We consider using a rectifier activation function as described in Example 4.1.

To avoid having to differentiate the non-smooth activation function when solving minimisation problem (3.4), we propose to use the Bregman loss term (3.9) as choice for the data term  $\ell$ , and in combination with the Iterative Soft-Thresholding Algorithm (ISTA) (Daubechies et al., 2004) to handle the additional non-smooth one-norm. We name the resulting algorithm Rosenblatt-ISTA, and summarise it in Algorithm 15. In numerical experiments, we choose  $|B_k| = s$ , which corresponds to using the full (sub-)gradient. We compute the step sizes by  $\tau_w = 1.99/\|x\|_2^2$ .

We test the algorithm performances with a supervised classification example based on images from the Fashion-MNIST dataset (Xiao et al., 2017), where a few sample images are visualised in Figure 3.4.

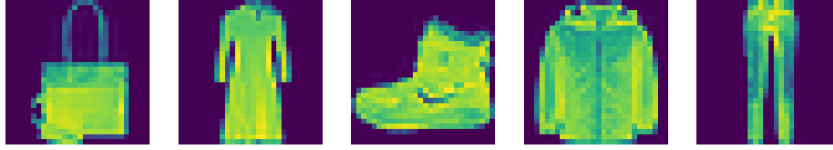


Fig. 3.4 Sample images from the Fashion-MNIST dataset (Xiao et al., 2017).

Training such a generalised classification perceptron can be described as the following minimisation problem:

$$\min_{w,b} \frac{1}{s} \sum_{i=1}^s D_E(y^i, w^\top x^i + b) + \alpha \|w\|_1, \quad (3.11)$$

where we use the Bregman loss function to measure the discrepancy between the predicted output and the one-hot encoded target values as our classification loss function. The perceptron is equipped with ReLU nonlinearities. More specifically, we use weight matrix  $w \in \mathbb{R}^{784 \times 10}$  and  $b \in \mathbb{R}^{10 \times 1}$ .

The proposed Rosenblatt-ISTA training algorithm is compared with three baseline approaches. All three baseline approaches solve the perceptron learning problem (3.4) by minimising the squared Euclidean loss as the loss function. More specifically:

- The first approach uses a standard subgradient method (cf. Chapter 2 Algorithm 3) with constant step size, denoted by Sub-GD (const). In particular, we define the sub-derivative  $\sigma'$  of the rectifier activation function at 0 to take on value 1.
- The second approach uses a subgradient method with diminishing step size following the rule  $\tau_w^k = \frac{\tau_w^0}{k}$ . This approach is denoted by Sub-GD (dimin). This approach provides guaranteed convergence since the Bregman loss function is bi-convex (i.e. convex with respect to its second argument for fixed first argument), but gives slower convergence rate.
- The third method also follows ISTA (cf. Chapter 2 Algorithm 4). But instead of taking a gradient descent step, we perform subgradient descent steps in the direction of the now non-smooth data term  $\ell$  during each iteration. We denote this approach by Sub-GD ISTA.

All methods and results are implemented in Python with the Numpy library. We train the perceptrons on 3,000 images, selected from the training dataset and validate them on 10,000 images from the test dataset. The choices of  $\alpha$  are determined via cross validation. In our proposed objective function, the regularisation parameter  $\alpha$  is

set at 0.9. We choose  $\alpha$  at 0.81, 0.81, 0.85 for the three baseline methods respectively. All four approaches are initialised with the same weight, bias and step size, following He et al. (2015a).

In each iteration, we record the overall objective value, the training and validation accuracy, and visualised in Figure 3.5. We can see that the proposed Rosenblatt-ISTA training scheme is able to converge much faster than the three baseline approaches and also achieves higher training and validation accuracy. As expected, when using a diminishing step size, Sub-GD shows the slowest convergence, and exhibits a plateaued behavior in classification accuracy. The performance of the Sub-GD ISTA algorithm on the other hand is almost identical with the Sub-GD (const) method.

## 6 Discussion

In this chapter, we have discussed the learning of a generalised perceptron model with proximal activation functions. We demonstrated that, from an energy minimisation point of view, the original Rosenblatt’s perceptron learning algorithm can also be interpreted as an incremental gradient method with respect to a particular choice of data term, based on a tailored Bregman distance. In addition, this interpretation also generalises the classical ADALINE delta rule.

We have modified this interpretation to learn sparse perceptrons with the proposed Rosenblatt-ISTA algorithm. We presented numerical results for which our approach outperforms three other subgradient-based baseline approaches in terms of achieving both faster convergence and higher accuracy results.

In the subsequent chapters we will examine the proposed Bregman loss function with a closer look. We will show extensions of the proposed generalised perceptron learning scheme to more complex architectures such as multi-layer feedforward neural networks, and with particular applications on convolutional neural networks and residual neural networks.

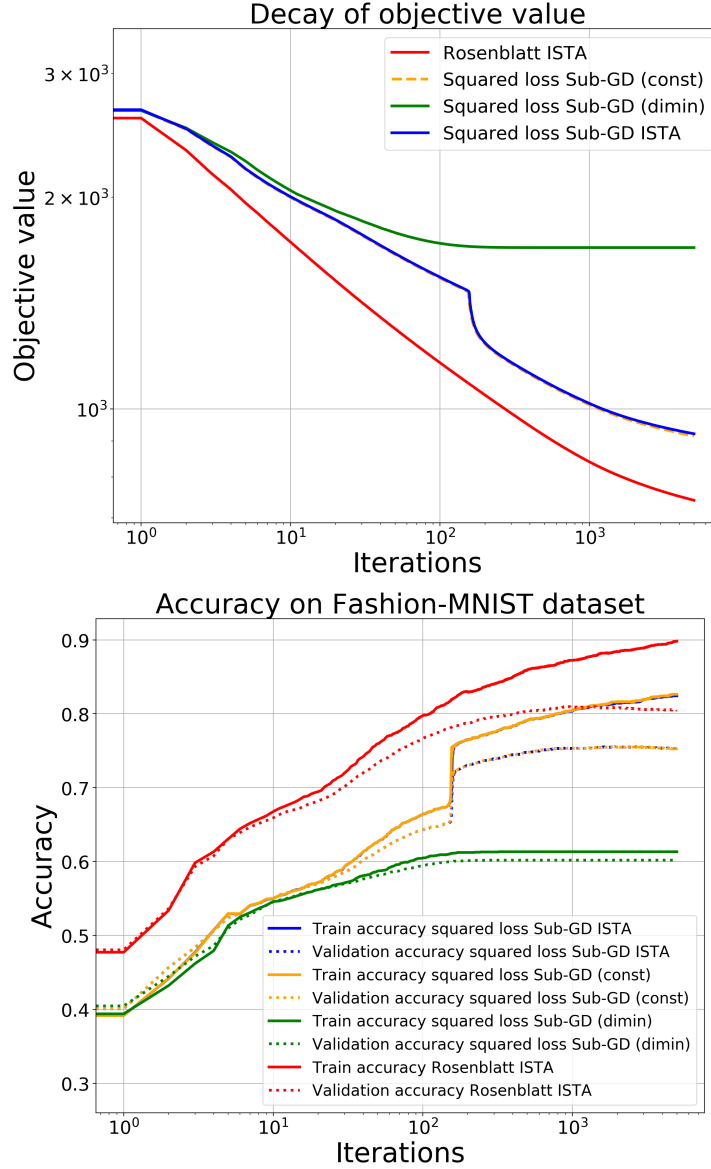


Fig. 3.5 We compare our method of training sparse perceptrons with three baseline approaches as described in Section 5. **Top:** The plot shows the decay of objective value, i.e. Bregman loss plus  $\ell^1$  regularisation for our proposed scheme and the squared Euclidean loss plus  $\ell^1$  regularisation for the baseline approaches. **Bottom:** The plot shows the change of training and validation accuracy over the number of iterations.

## Chapter 4

# Lifted Bregman Training of Feedforward Neural Networks

In this chapter, we establish a distributed optimisation framework for training multi-layer neural networks that does not require the differentiation of activation functions. Built upon Chapter 3, the discussion in this chapter can be viewed as a natural extension from learning single-layer perceptrons to their multi-layer counterparts. However, it's important to emphasise that extending the learning of perceptrons to multi-layer neural network architectures is not a trivial task. In fact, this transition has enriched the role of the Bregman loss function. More specifically, to accomplish this extension, we relax the original learning problem and formulate a lifted training approach where the Bregman loss function is used as a penalty function. The new Bregman training formulation offers an alternative solution to training neural networks, in addition to the standard back-propagation-based approaches. The main contributions of this chapter are four-fold:

- We establish the Bregman learning framework for the forward-direction training of neural networks, which generalises the Method of Auxiliary Coordinates and provides an alternative learning approach than using the back-propagation algorithm and (sub-)gradient methods over a nested objective function.
- We provide detailed analysis of the proposed Bregman loss function and propose a variety of different deterministic and stochastic iterative minimisation algorithms for computationally solving the empirical risk minimisation problem.
- We demonstrate that the proposed approach has an added advantage for solving regularised learning problems with explicit variational regularisation imposed on

network hidden variables. Specifically, we show that the proposed framework achieves superior learning performance when applied to learn regularised and denoising autoencoders.

- We compare the proposed iterative minimisation algorithms to more conventional approaches, such as stochastic gradient descent method using the back-propagation algorithm. We demonstrate that contrary to popular belief, expressive feedforward networks can suffer from overfitting to training data and fail to generalise well when non-standard optimisation methods are used.

## Organisation of the Chapter

This chapter is structured as follows. In Section 1, we begin with an introduction on historical approaches of training neural networks, as well as the development of the back-propagation algorithm and its potential drawbacks. In Section 2, we describe the standard approach of training neural networks with gradient-based algorithms and the back-propagation algorithm. We then provide an overview over recent developments in distributed optimisation methods for the training of feedforward neural networks. In Section 3, we introduce our proposed lifted Bregman framework. We first define the Bregman penalty function and its use within the method of auxiliary coordinates. We then examine a variety of mathematical properties of this function that guarantee its advantages over other approaches. In Section 4, we provide a range of deterministic and stochastic optimisation approaches that can be used to computationally solve the lifted Bregman training problem. We also discuss suitable regularisation strategies for the regularisation of network parameters and network hidden variables. In Section 5, we explore specific example problems, including classification, data compression, and denoising. For the latter two, we introduce a regularised training approach that will produce autoencoders with sparse codes. In Section 6, we present numerical results for the example problems described in Section 5 and provide a comprehensive comparison of the proposed iterative minimisation algorithms in Section 4 to more conventional approaches. We conclude this chapter with a brief discussion in Section 7.

# 1 A Little History on Training Neural Networks

Artificial Neural Networks (ANNs) have been extremely popular choices of model functions for a great variety of machine learning problems (cf. (Goodfellow et al., 2016b; Schmidhuber, 2015; Wang and Raj, 2017)). The concept of compositing multiple layers of artificial neurons to construct a multi-layer neural network traced back to the 1960s when early ANNs mainly considered shallow architectures. As discussed in Chapter 3, although Rosenblatt designed a multi-layer perceptron (Rosenblatt., 1957), it was a combination of unmodifiable initial layers and one single learnable output layer. In the late 1950s, most work on adaptive learning of network parameters were via supervised learning approaches as explored in Widrow (1960).

To this day, the back-propagation algorithm (Rumelhart et al., 1986) has established its central role in learning deep neural networks and is considered an important pillar of modern deep learning practices, yet the concept of "back-propagating errors" dates back to the early 1960s. Early researchers discussed techniques of minimising errors through gradient descent in the parameter space of complex systems (Bryson, 1961; Dreyfus, 1962; Kelley, 1960) using principles from Dynamic Programming (Bellman, 1957), before similar techniques with more explicit usage on neural network systems arose.

As early as in the 1970s, the idea of reverse mode auto differentiation came into shape in the work of Seppo Linnainmaa (Griewank, 2012; Linnainmaa, 1970, 1976). It was later described by Paul Werbos for its more efficient use on training ANNs parameters (Werbos, 2005). The term back-propagation itself was popularised by Rumelhart, Hinton and Williams in their seminal work on learning internal representations in hidden layers of an ANN (Rumelhart et al., 1986). In the late 1980s, several techniques were proposed to improve the performance of gradient descent through the use of the back-propagation algorithm (Riedmiller and Braun, 1993; Rumelhart et al., 1986). Around the same time, the back-propagation algorithm was first applied to train convolutional neural networks in 1989 (LeCun et al., 1989).

Early form of back-propagation algorithms provided an essential tool for training, in principle, ANNs of arbitrary depth. However, little empirical benefit was observed despite increasing the depth of an artificial neural network. It was not until 1990s when the vanishing gradient problem, which hindered efficient use of back-propagation on training deep neural networks was identified. In 1991, Horeiter described the long time lag problem in his work (Hochreiter, 1991) and Bengio in 1994 studied the vanishing gradient problem (Bengio et al., 1994). While the initial discussion of both works focused on learning recurrent neural networks, similar concerns are also relevant to



Fig. 4.1 The ImageNet Large Scale Visual Recognition Challenge (Russakovsky et al., 2015).

traditional deep feedforward neural networks (Bengio et al., 1994; Schmidhuber, 2015). The prediction of an "AI Winter" came true in 1987 when the AI bubble burst, leading to a decrease in investment and funding for AI research.

Made possible by advancements in computing power and GPU processing (Abi-Chahla, 2008), 2012 saw the ground breaking success of AlexNet (Krizhevsky et al., 2017) on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Russakovsky et al., 2015) where it outperformed the previous state-of-the-art methods in image classification tasks by a significant margin, reducing the top-5 error rate from 26.2% to 15.3%. The success of AlexNet paved the way for the development of more sophisticated deep learning models, particularly in computer vision applications. The interest in building deeper and more complex neural network architectures was once again reignited. However, despite the great success of deep learning in recent years, the standard approach of using gradient-based minimisation strategies in combination with the back-propagation algorithm has potential limitations (Taylor et al., 2016).

First of all, as we have briefly mentioned earlier, back-propagation suffers from vanishing gradient problems (Erhan et al., 2009), where gradients of loss functions with respect to the parameters of early network layers become vanishingly small, which renders the use of gradient-based minimisation methods ineffective. While the use of non-saturating activation functions and the introduction of skip-connections to network architectures have been proposed to mitigate vanishing gradient problems, achieving state-of-the-art performance still requires meticulous hyper-parameter tuning and good initialisation strategies. The second drawback is that many common activation functions in neural network architectures are not differentiable, which means that

subgradient- instead of gradient-based algorithms are required. And even though there are mathematical justifications for such an approach (cf. Bolte and Pauwels (2020)), subgradient-based methods have the disadvantage of inferior convergence rates over other non-smooth optimisation methods like proximal gradient descent (cf. Teboulle (2018)). Another issue of the back-propagation algorithm is that it is sequential in nature, which makes it difficult to distribute the computation of the network parameters across multiple workers efficiently. As a consequence of the aforementioned issues, numerous distributed optimisation approaches have been proposed as alternatives to gradient-based training in combination with back-propagation in recent years (Askari et al., 2018; Carreira-Perpinan and Wang, 2014; Gu et al., 2020; Høier and Zach, 2020; Li et al., 2019; Taylor et al., 2016; Zach and Estellers, 2019; Zhang and Brand, 2017). However, many of these approaches still suffer from limitations such as differentiating non-differentiable activation functions (Carreira-Perpinan and Wang, 2014), recovering only affine-linear networks (Askari et al., 2018), or overly restrictive assumptions (Gu et al., 2020; Li et al., 2019).

## 2 Training Feedforward Neural Networks

In the following sections, we revisit the notion of feedforward neural networks (cf. Chapter 2 Definition 2.69) before we summarise the classical approach of gradient-based training and back-propagation. We then discuss distributed optimisation and lifted training strategies.

A feedforward neural network  $\mathcal{N} : \mathbb{R}^n \times \mathcal{P} \rightarrow \mathbb{R}^m$  with  $L$  layers can be defined as the composition of parametrised functions of the following form

$$\mathcal{N}(x, \Theta) = \sigma_L(f(\sigma_{L-1}(f(\dots \sigma_1(f(x, \Theta_1)) \dots)), \Theta_L)), \quad (4.1)$$

for given input data  $x \in \mathbb{R}^n$  and parameters  $\Theta \in \mathcal{P}$ . Here  $\{\sigma_l\}_{l=1}^L$  denotes the collection of nonlinear activation functions and  $f$  denotes a generic layer function parametrised by parameters  $\{\Theta_l\}_{l=1}^L$ . We use  $\Theta$  to denote  $\{\Theta_l\}_{l=1}^L$ . As an example, a fully connected layer is represented by  $f(x, \Theta_l) = W_l^\top x + b_l$  for  $\Theta_l = (W_l, b_l)$ , where the matrix  $W_l$  is usually referred to as the weights and the vector  $b_l$  as the bias term of layer  $l$ .

### 2.1 First-order Methods and Back-propagation

Estimating parameters  $\Theta$  of a feedforward network in the supervised learning setting is usually framed as the minimisation of an empirical risk (cf. Chapter 2 Definition

**Algorithm 16:** Back-Propagation Algorithm

---

```

Initialise  $\Theta$ .
for  $i = 1, \dots, s$  do
    for  $l = 1, \dots, L$  do
        Perform forward pass to compute  $z_l$  and  $x_l$ :
         $z_l^i = f(x_{l-1}^i, \Theta_l)$ ,
         $x_l^i = \sigma_l(z_l^i)$ ,
    end
end
for  $i = 1, \dots, s$  do
    for  $l = L, \dots, 1$  do
        Perform backward pass to compute  $\delta_l^i$ :
         $\delta_l^i = \begin{cases} \sigma'_l(z_L^i) \odot \nabla_x \ell(y^i, x_L^i), & \text{for } l = L \\ \sigma'_l(z_l^i) \odot \frac{\partial f}{\partial x_l} \delta_{l+1}^i, & \text{for } l \in \{1, \dots, L-1\} \end{cases}$ 
    end
end
Partial derivatives:  $\frac{\partial \ell}{\partial \Theta_l^{jk}} = \delta_l^j \frac{\partial f}{\partial \Theta_l^k}$  for  $j \in \{1, \dots, n_l\}$  and  $k \in \{1, \dots, n_{l-1}\}$ .

```

---

2.58) of the form

$$\min_{\Theta} \frac{1}{s} \sum_{i=1}^s \ell(y^i, \mathcal{N}(x^i, \Theta)), \quad (4.2)$$

where  $\ell$  is a chosen data error term pertaining to the learning task (usually referred to as the loss function), while  $\{(x^i, y^i)\}_{i=1}^s$  denotes the  $s$  pairs of input and output samples that are provided a-priori.

A standard computational approach to solve (4.2) are (sub-)gradient-based algorithms such as (sub-)gradient descent (cf. Chapter 2 Algorithm 3) and stochastic variants of it (cf. Chapter 2 Algorithm 2.46). Gradient descent for solving (4.2) reads

$$\Theta^{k+1} = \Theta^k - \tau_{\Theta} \nabla_{\Theta} \left( \frac{1}{s} \sum_{i=1}^s \ell(y^i, \mathcal{N}(x^i, \Theta^k)) \right), \quad (4.3)$$

for  $k \in \mathbb{N}$ , a step-size parameter  $\tau_{\Theta} > 0$  and initial parameters  $\Theta^0$ . The evaluation of the gradient  $\nabla_{\Theta} \left( \frac{1}{s} \sum_{i=1}^s \ell(y^i, \mathcal{N}(x^i, \Theta^k)) \right)$  is computed with the back-propagation (cf. Rumelhart et al. (1986)), which is summarised in Algorithm 16.

As pointed out in LeCun et al. (1988), the back-propagation algorithm can also be deduced from a Lagrangian formulation of (4.2), where solving the corresponding optimality system leads to the individual steps in the back-propagation algorithm. For

each pair of data samples  $(x^i, y^i)$  we can define

$$x_l^i = \sigma_l(f(x_{l-1}^i, \Theta_l)) \quad \text{for} \quad l = 1, \dots, L, \quad (4.4)$$

as the so-called activation variables with  $x_0^i = x^i$ . Further, we use  $z_l^i$  to denote transformed input variables by

$$z_l^i = f(x_{l-1}^i, \Theta_l) \quad \text{for} \quad l = 1, \dots, L.$$

The introduction of auxiliary variables  $x_l$  and  $z_l$  break down the nested dependency of the function mapping  $\mathcal{N}$  into layer-wise equality constraints. To ease notation, we introduce  $\mathbf{X} = \{x_l^i\}_{l=1, \dots, L}^{i=1, \dots, s}$  and  $\mathbf{Z} = \{z_l^i\}_{l=1, \dots, L}^{i=1, \dots, s}$  to denote the two groups of auxiliary variables. Having introduced  $\mathbf{X}$  and  $\mathbf{Z}$ , we can write the minimisation problem (4.2) in the following equivalent constrained form:

$$\begin{aligned} \min_{\Theta, \mathbf{X}, \mathbf{Z}} \quad & \frac{1}{s} \sum_{i=1}^s \ell(y^i, x_L^i) \\ \text{s.t.} \quad & z_l^i = f(x_{l-1}^i, \Theta_l) \quad \text{for } l = 1, \dots, L, \\ \text{s.t.} \quad & x_l^i = \sigma_l(f(x_{l-1}^i, \Theta_l)) \quad \text{for } l = 1, \dots, L. \end{aligned} \quad (4.5)$$

We can write down the corresponding Lagrangian function for (4.5) as

$$\mathcal{L}(\Theta, \mathbf{X}, \mathbf{Z}, \mu, \delta) = \sum_{i=1}^s \left[ \ell(y, x_L^i) + \sum_{l=1}^L \langle \mu_l^i, x_l^i - \sigma(z_l^i) \rangle + \sum_{l=1}^L \langle \delta_l^i, z_l^i - f(x_{l-1}^i, \Theta_l) \rangle \right], \quad (4.6)$$

where  $\mu = \{\mu_l^i\}_{l=1, \dots, L}^{i=1, \dots, s}$  and  $\delta = \{\delta_l^i\}_{l=1, \dots, L}^{i=1, \dots, s}$  are Lagrangian multipliers.

In the following deduction of the back-propagation algorithm, we fix for one training data sample and drop the dependency on  $i$  for the ease of notation. The optimality

conditions of (4.6) can be split into the following sub-conditions:

$$\frac{\partial \mathcal{L}}{\partial x_L} = 0 \implies \nabla_{x_L} \ell(y, x_L) + \mu_L = 0 \quad (4.7)$$

$$\frac{\partial \mathcal{L}}{\partial \Theta_l} = 0 \implies \delta_l = \frac{\partial f}{\partial \Theta_l} \mu_l^\top \text{ for } l = 1, \dots, L-1 \quad (4.8)$$

$$\frac{\partial \mathcal{L}}{\partial x_l} = 0 \implies \mu_l = \frac{\partial f}{\partial x_l} \delta_{l+1} \text{ for } l = 1, \dots, L-1 \quad (4.9)$$

$$\frac{\partial \mathcal{L}}{\partial z_l} = 0 \implies \delta_l = \sigma'_l(z_l) \mu_l \text{ for } l = 1 \dots L \quad (4.10)$$

$$\frac{\partial \mathcal{L}}{\partial \mu_l} = 0 \implies x_l = \sigma_l(f(x_{l-1}, \Theta_l)) \text{ for } l = 1, \dots, L \quad (4.11)$$

$$\frac{\partial \mathcal{L}}{\partial \delta_l} = 0 \implies z_l = f(x_{l-1}, \Theta_l) \text{ for } l = 1, \dots, L \quad (4.12)$$

Notice that sub-conditions (4.11) and (4.12) carry out exactly the forward-pass of the Algorithm 16. Combining sub-conditions (4.9) and (4.10) gives

$$\delta_l = \sigma'_l(z_l) \frac{\partial f}{\partial x_l} \delta_{l+1} \text{ for } l = 1 \dots L-1 ,$$

which is the backward-pass in Algorithm 16 for computing the partial derivatives. Sub-conditions (4.7) and (4.8) on the other hand are computational steps for taking the (sub-)gradient update step (cf. Higham and Higham (2019)).

Despite its popularity for training neural networks, the approach of using (sub-)gradient based methods such as (4.3) in combination with the back-propagation algorithm 16 nevertheless suffers from several drawbacks.

- The first two of the major problems are the vanishing and exploding gradient issues, where the gradients either decrease to vanishingly small values or increase to very large values, which causes major problems for the computation of (4.3) (cf. Bengio et al. (1994); Glorot and Bengio (2010)).
- A third issue is that the back-propagation algorithm is sequential in nature and computation of individual partial derivatives cannot easily be distributed.
- The fourth drawback is that while it is easy to generalise gradient-based algorithms to proximal gradient methods that can handle non-smooth functions acting on the network parameters, it is less straight-forward to generalise these techniques so that they can handle non-smooth functions acting on the network's hidden activation variables or the transformed input variables without using subgradients.

These drawbacks and limitations of the combination of (sub-)gradient based methods and back-propagation have motivated research to seek for alternative training methods for deep neural networks. In the next section, we will summarise some of these alternatives.

## 2.2 Distributed Optimisation Approaches

Learning network parameters via distributed optimisation has been given much attention in recent years (Askari et al., 2018; Carreira-Perpinan and Wang, 2014; Gu et al., 2020; Li et al., 2019; Taylor et al., 2016; Zach and Estellers, 2019; Zhang and Brand, 2017). In these works, the overall Learning Problem (4.2) is reformulated to Problem (4.5).

This constrained optimisation Problem (4.5) can further be relaxed by replacing the constraints for the activation variables with penalty terms in the objective function. Solving each layer-wise sub-problem can then be performed independently, which allows for distributed optimisation methods to be utilised. This attempt has mainly been approached from two directions: by using quadratic penalties (Carreira-Perpinan and Wang, 2014; Taylor et al., 2016) or by framing activation functions as orthogonal projections on convex sets of constraints (Askari et al., 2018; Gu et al., 2020; Li et al., 2019; Zach and Estellers, 2019; Zhang and Brand, 2017).

### 2.2.1 Method of Auxiliary Coordinates with Quadratic Penalty (MAC-QP)

The work of Carreira-Perpinan and Wang (2014) is among the earliest to look into the quadratic penalty approach, where the authors propose the Method of Auxiliary Coordinates (MAC). The MAC approach only introduces the activation variables  $\mathbf{X}$  and relaxes the constrained problem (4.5) by replacing the constraints  $x_l^i = \sigma_l(f(x_{l-1}^i, \Theta_l))$  with a Quadratic Penalty (QP) objective where the corresponding minimisation problem reads

$$\min_{\mathbf{X}, \Theta} \frac{1}{s} \sum_{i=1}^s \left[ \ell(y^i, \sigma_L(f(x_{L-1}^i, \Theta_L))) + \sum_{l=1}^{L-1} \|x_l^i - \sigma_l(f(x_{l-1}^i, \Theta_l))\|_2^2 \right]. \quad (4.13)$$

If we consider the special case of affine-linear functions  $f(x_{l-1}, \Theta_l) = W_l^\top x_{l-1} + b_l$ , then for  $s = 1$  and a single training pair of samples  $x$  and  $y$ . The optimality condition of the MAC-QP formulation with respect to the bias parameter  $b_l$  reads

$$0 = (x_l - \sigma_l(W_l^\top x_{l-1} + b_l)) \sigma'_l(W_l^\top x_{l-1} + b_l).$$

Whenever  $\sigma'_l(W_l^\top x_{l-1} + b_l) \neq 0$ , this automatically implies

$$x_l = \sigma_l(W_l^\top x_{l-1} + b_l),$$

ensuring that a critical point satisfies (4.4).

Following the MAC-QP approach, minimisation of network parameters can easily be distributed with the right choice of optimisation method. However, minimising the MAC-QP objective with gradient-based first-order optimisation methods still requires the differentiation of the activation functions. This is different for lifted training approaches that we briefly discuss in the later section.

### 2.2.2 Connecting Quadratic Penalty Approach to Back-propagation

In the work of Frerix et al. (2017), a connection between the classical back-propagation algorithm and quadratic penalty approach is established. After introducing the auxiliary variables  $\mathbf{Z}$ , the authors consider the following quadratic penalty function:

$$E(\Theta, \mathbf{X}, \mathbf{Z}) = \frac{1}{s} \sum_{i=1}^s \left[ \ell(y^i, f(x_{L-1}^i, \Theta_L)) + \sum_{l=1}^{L-1} \left( \frac{\gamma}{2} \|x_l^i - \sigma(z_l^i)\|_2^2 + \frac{\rho}{2} \|z_l^i - \sigma_l(f(x_{l-1}^i, \Theta_l))\|_2^2 \right) \right]. \quad (4.14)$$

Instead of directly optimising (4.14). Their work shows that by carrying out 1) a forward pass to update relevant variables, and 2) sequential gradient steps on the quadratic penalty function (4.14), the updated network parameters  $\Theta$  are equivalent to what received from a classical gradient descent step update. The proposed algorithm is summarised in Algorithm 17. In other words, with specific choices of step size parameters  $\gamma = \rho = \frac{1}{\tau}$ , Algorithm 17 computes one gradient descent iteration on the original nested objective (4.2).

For simplicity, we consider only one pair of data points, i.e.  $s = 1$ . The gradient step on  $E(\Theta, \mathbf{X}, \mathbf{Z})$  with respect to  $\Theta_L$  gives

$$\Theta_{L-1}^{k+1} = \Theta_{L-1}^k - \tau \nabla \ell(y, f(x_{L-1}, \Theta_{L-1})), \quad (4.15)$$

whereas a gradient descent step with respect to  $x_{L-1}$  is

$$x_{L-1}^{k+1/2} = x_{L-1}^k - \tau \nabla_x f(x_{L-1}, \Theta_L) \cdot \nabla_f \ell(y, f(x_{L-1}, \Theta_L)). \quad (4.16)$$

We use the half update notation  $x_{L-1}^{k+1/2}$  to denote the updated variable  $x_{L-1}^k$  before the forward pass of the next iteration. Note that for all layers  $l \in \{1, \dots, L-1\}$ , the

---

**Algorithm 17:** Quadratic Penalty Formulation of the Back-propagation Algorithm

---

Initialise  $\Theta$ .  
 Forward pass:  
**for**  $l \in \{1, \dots, L-1\}$  **do**  
      $z_l^k = f(x_{l-1}^k, \Theta_l^k)$ ,  
      $x_l^k = \sigma(z_l^k)$ .  
**end**  
 Perform minimisation on (4.14).  
 (a) Gradient step on  $\ell(y^i, f(x_{L-1}^i, \Theta_L))$ , with respect to  $(\Theta_L, x_{L-1})$ .  
**for**  $l \in \{L-1, \dots, 1\}$  **do**  
     (b) Gradient step on  $E$  with respect to  $z_l$  and  $x_{l-1}$ .  
     (c) Gradient step on  $E$  with respect to  $\Theta_l$ .  
**end**  
 Output updated  $\Theta^{k+1}$ .

---

following holds due to the forward pass:

$$\sigma(z_l^k) = x_l^k, \quad (4.17)$$

$$f(x_{L-1}, \Theta_{L-1}) = z_l^k. \quad (4.18)$$

Therefore, in step (b) we get the following updates:

$$z_l^{k+1/2} = z_l^k - \tau \gamma \nabla \sigma(z_l^k) \cdot (\sigma(z_l^k - x_l^{k+1/2})) = z_l^k - \nabla \sigma(z_l^k) \cdot (x_l^k - x_l^{k+1/2}), \quad (4.19)$$

where with respect to  $x_{l-1}$  we have

$$\begin{aligned}
 x_{l-1}^{k+1} &= x_{l-1}^k - \tau \rho \nabla_x f(x_{l-1}, \Theta_l) \cdot (f(x_{l-1}^k, \Theta_l^k) - z_l^{k+1/2}) \\
 &= x_{l-1}^k - \nabla_x f(x_{l-1}^k, \Theta_l^k) \cdot (z_l^k - z_l^{k+1/2}).
 \end{aligned} \quad (4.20)$$

Combining (4.19) and (4.20), we obtain

$$z_l^k - z_l^{k+1/2} = \nabla \sigma(z_l^k) \nabla_x f(x_l^k, \Theta_{l+1}^k) \cdot (z_{l+1}^k - z_{l+1}^{k+1/2}), \quad (4.21)$$

Using (4.16) and (4.20) the difference between  $z_l^k$  and the half update  $z_l^{k+1/2}$  can be tracked from the loss function  $\ell$  across  $L-1$  layers, where we obtain

$$z_l^k - z_l^{k+1/2} = \tau \left( \prod_{q=l}^{L-1} \nabla \sigma(z_q^k) \nabla_x f(x_q^k, \Theta_{q+1}^k) \right) \cdot \nabla_f \ell(y, f(x_{L-1}^k, \Theta_L^k)) \quad (4.22)$$

On the other hand, follows the gradient step in (c) we get

$$\Theta_l^{k+1} = \Theta_l^k - \nabla_{\Theta} f(x_{l-1}^k, \Theta_l^k) \cdot (z_l^k - z_l^{k+1/2}). \quad (4.23)$$

Now insert (4.22) into the gradient step with respect to  $\Theta_l$  in (4.23), we obtain the gradient descent update over the nested objective function via the chain rule as shown in (4.3).

After having established a connection between the quadratic penalty approach and the back-propagation algorithm, we will next discuss the lifted training approaches for neural networks.

### 2.2.3 Lifted Training Approach

The second line of work to approach Problem (4.5) is motivated by the observation that the ReLU activation function  $\sigma(z, 0) = \max(z, 0)$  itself can be viewed as the solution to a constrained minimisation problem. More specifically, in the work of Zhang and Brand (2017), the authors characterise the activated output of the ReLU activation function  $\sigma_l(W_l^\top x_{l-1} + b_l)$  as the solution to the following non-smooth convex optimisation problem (or respectively a constrained convex optimisation problem with smooth objective):

$$\begin{aligned} x_l &= \sigma_l(W_l^\top x_{l-1} + b_l) \\ &= \arg \min_{x \geq 0} \|x - (W_l^\top x_{l-1} + b_l)\|_2^2. \end{aligned}$$

This implies that  $x_l$  is the vector closest to  $W_l^\top x_{l-1} + b_l$  that lies in the non-negative orthant.

Hence, instead of writing out the dependency relation of activation variables  $\mathbf{X}$  explicitly as shown in Equation (4.5), nonlinear constraints can be re-written into conditions similarly to the above as argmin maps. In the classical lifted approach (Askari et al., 2018), Problem (4.5) is consequently approximated with the following constrained formulation where restraints on activation variables are now directly imposed:

$$\min_{\Theta, \{x_l^i \in \mathbb{R}_+^n\}_{l=1}^{L-1}} \frac{1}{s} \sum_{i=1}^s \left[ \ell(y^i, W_L^\top x_{L-1}^i + b_L) + \sum_{l=1}^{L-1} \lambda_l \|x_l^i - W_l^\top x_{l-1}^i - b_l\|_2^2 \right], \quad (4.24)$$

where  $\{x_l^i \in \mathbb{R}_+^n\}_{l=1}^{L-1}$  denotes that the activation variables across layers  $l = 1, \dots, L-1$  for each sample  $i$  lie in the non-negative orthant, i.e.  $\mathbb{R}_+^n := \{v \in \mathbb{R}^n \mid v_j \geq 0, \forall j \in \{1 \dots n\}\}$ . The parameters  $\lambda_l$  for  $l = 1, \dots, L-1$  are positive hyperparameters.

This approach is often referred to as the "lifted" approach because the search space for network parameters is lifted to become a higher dimensional space that now also includes auxiliary variables. However, this idea of "lifting" is also present in the MAC-QP approach.

Even though the overall learning problem (4.24) is not convex, each sub-problem is convex in each individual variable when keeping all other variables fixed. Individual updates are also relatively easy to compute using orthogonal projections onto the non-negative orthant. In terms of distributing the computation of parameters, this approach also enjoys the same benefits that the MAC-QP scheme enjoys.

However, one major limitation that this approach suffers from is that it does not solve the original Problem (4.2), but essentially trains an affine-linear network (cf. Zach and Estellers (2019)). This restriction can be shown by examining the optimality system. Consider the case with  $s = 1$  and one single training sample pair  $(x, y)$ , then the optimality condition of (4.24) with respect to parameters  $b_l$  reads

$$\lambda_l(W_l^\top x_{l-1} + b_l - x_l) = 0 \quad \implies \quad x_l = W_l^\top x_{l-1} + b_l. \quad (4.25)$$

Despite this major limitation this approach has been considered for computing initialisations of network parameters for other optimisation algorithms (Li et al., 2019; Zach and Estellers, 2019).

### 3 Lifted Bregman Training Framework

Building upon the previous approaches, we introduce a lifted Bregman framework for the training of feedforward networks. This framework will fix the aforementioned issues and 1) be capable of recovering a network with nonlinear activation functions, while 2) taking partial derivatives with respect to network parameters will not require computing derivatives of the activation functions.

Our approach differs from Problem (4.13) and Problem (4.24) in the sense that instead of penalising the quadratic Euclidean norm of the nonlinear constraint  $x_l^i = \sigma_l(f(x_{l-1}^i, \Theta_l))$  we propose an alternative penalisation. From now on we assume that the activation functions  $\sigma_l$  that we consider are proximal activation functions, namely they come from the large class of proximal maps (cf. Chapter 2 Definition 1.18).

In particular, the following remark gives an alternative characterisation of the proximal map for convex functions  $\Psi$ .

**Remark 3.1.** *Note that an equivalent characterisation of the proximal map  $\sigma$  is*

$$z - \sigma(z) \in \partial\Psi(\sigma(z)),$$

*as a consequence of Fermat's theorem. Here  $\partial\Psi$  denotes the subdifferential of  $\Psi$ .*

With the help of Remark 3.1, we discover that  $x_l = \sigma(f(x_{l-1}, \Theta_l)) = \text{prox}_\Psi(f(x_{l-1}, \Theta_l))$  can also be written as

$$f(x_{l-1}, \Theta_l) - x_l \in \partial\Psi(x_l), \quad \text{respectively,} \quad f(x_{l-1}, \Theta_l) \in \partial\left(\frac{1}{2}\|\cdot\|^2 + \Psi\right)(x_l), \quad (4.26)$$

where the second inclusion follows from (Ekeland and Temam, 1999, Chapter 1, Section 5, Proposition 5.6).

The following neat equivalence for inclusion (4.26) forms the basis for our new penalisation term.

**Lemma 3.1** (Subdifferential characterisation). *Suppose the function  $\Psi : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$  is a proper, lower semi-continuous and convex function. Then the inclusion  $z \in \partial\left(\frac{1}{2}\|\cdot\|^2 + \Psi\right)(x)$  is equivalent to*

$$\frac{1}{2}\|x\|^2 + \Psi(x) + \left(\frac{1}{2}\|\cdot\|^2 + \Psi\right)^*(z) = \langle x, z \rangle. \quad (4.27)$$

Here,  $\left(\frac{1}{2}\|\cdot\|^2 + \Psi\right)^*$  denotes the Fenchel conjugate of  $\frac{1}{2}\|\cdot\|^2 + \Psi$ .

A proof for this lemma can be found in (Rockafellar, 1970, Theorem 23.5), where a more general result is proven. Now, instead of enforcing (4.27) as a hard constraint, we define the penalisation function

$$B_\Psi(x, z) := \frac{1}{2}\|x\|^2 + \Psi(x) + \left(\frac{1}{2}\|\cdot\|^2 + \Psi\right)^*(z) - \langle x, z \rangle, \quad (4.28)$$

and replace the squared Euclidean norm in (4.24) with it, which results in the unconstrained optimisation problem

$$\min_{\Theta, \mathbf{X}} \sum_{i=1}^s \left[ \ell(y^i, x_L^i) + \sum_{l=1}^{L-1} \lambda_l B_\Psi(x_l^i, f(x_{l-1}^i, \Theta_l)) \right]. \quad (4.29)$$

for positive scalars  $\{\lambda_l\}_{l=1}^{L-1}$ .

We name this penalisation function  $B_\Psi(x, z)$  Bregman penalty, as we are going to show that it belongs to the family of generalised Bregman distances (cf. Chapter 2 Definition 1.12). In contrast to (4.13) and (4.24), these penalisations incorporate prior information encoded in the choice of the function  $\Psi$ . In the following sections, we provide a more detailed analysis of the Bregman penalty term and show how it can improve the classical lifted training approach and overcome its limitations.

### 3.1 The Bregman Penalty Function

We first want to verify that the proposed Bregman penalty function (4.28) is a non-negative generalised Bregman distance (Bregman, 1967; Kiwiel, 1997) and satisfies some advantageous properties compared to the mean-squared error loss used in (4.24). Please note that the Bregman distance can also be expressed by means of the convex conjugate (cf. Chapter 2 Definition 1.4) of  $\Phi$ . Hence, if we choose  $\Phi = \frac{1}{2}\|\cdot\|^2 + \Psi$  and  $z \in \partial\Phi(\sigma(z))$ , we immediately observe

$$B_\Psi(x, z) = D_\Phi^z(x, \sigma(z)).$$

Here  $z \in \partial\Phi(\sigma(z))$  is a valid subgradient, since  $z \in \partial(\frac{1}{2}\|\cdot\|^2 + \Psi)(\sigma(z))$  is equivalent to  $z - \sigma(z) \in \partial\Psi(\sigma(z))$ , respectively  $\sigma(z) = \text{prox}_\Psi(z)$  as pointed out in Remark 3.1. Since this is true by definition of  $\sigma$ , we have established that  $B_\Psi(x, z)$  is a Bregman distance. Since Bregman distances are non-negative, this automatically implies non-negativity of  $B_\Psi(x, z)$  for all arguments  $x, z$ , but we easily get a better lower bound if we split the Bregman distance, i.e.

$$B_\Psi(x, z) = D_\Phi^z(x, \sigma(z)) = D_{\frac{1}{2}\|\cdot\|^2}^{\sigma(z)}(x, \sigma(z)) + D_\Psi^{z-\sigma(z)}(x, \sigma(z)), \quad (4.30)$$

$$= \frac{1}{2}\|x - \sigma(z)\|^2 + D_\Psi^{z-\sigma(z)}(x, \sigma(z)) \geq \frac{1}{2}\|x - \sigma(z)\|^2, \quad (4.31)$$

which we capture with the following corollary.

**Corollary 3.1.** *The loss function  $B_\Psi$  as defined in (4.28) is a Bregman distance and bounded from below by  $\frac{1}{2}\|x - \sigma(z)\|^2$ .*

*Proof.* This follows from the fact that  $D_\Psi^{z-\sigma(z)}(x, \sigma(z))$  is a valid generalised Bregman distance, because of  $z - \sigma(z) \in \partial\Psi(\sigma(z))$  by definition of  $\sigma$ . Hence,  $D_\Psi^{z-\sigma(z)}(x, \sigma(z)) \geq 0$ , which concludes the proof.  $\square$

As a consequence, we can characterise the exact discrepancy between  $B_\Psi(x, z)$  and  $\frac{1}{2}\|x - \sigma(z)\|^2$  in terms of  $D_\Psi^{z-\sigma(z)}(x, \sigma(z))$ , which means we establish

$$\begin{aligned} B_\Psi(x, z) &= \frac{1}{2}\|x - \sigma(z)\|^2 + \Psi(x) - \Psi(\sigma(z)) - \langle z - \sigma(z), x - \sigma(z) \rangle, \\ &= \frac{1}{2}\|x - \sigma(z)\|^2 + \Psi(x) + \Psi^*(z - \sigma(z)) - \langle x, z - \sigma(z) \rangle, \end{aligned} \quad (4.32)$$

We can further simplify this term with the help of the Moreau decomposition. Recall Chapter 2 Theorem 1.3, we have  $\sigma(x) := \text{prox}(x)$  and  $\sigma^*(x) := \text{prox}_{\Psi^*}(x)$ . Then we can rewrite (4.32) to

$$B_\Psi(x, z) = \frac{1}{2}\|x - \sigma(z)\|^2 + \Psi(x) + \Psi^*(\sigma^*(z)) - \langle x, \sigma^*(z) \rangle.$$

If we define  $E_z(x) := \frac{1}{2}\|x - z\|^2 + \Psi(x)$  as the function for which we have  $\sigma(z) = \text{prox}(z) = \arg \min_x E_z(x)$  and consequently  $0 \in \partial E_z(\sigma(z))$ , we can further conclude

$$\begin{aligned} B_\Psi(x, z) &= \frac{1}{2}\|x - \sigma(z)\|^2 + \Psi(x) - \Psi(\sigma(z)) - \langle z - \sigma(z), x - \sigma(z) \rangle, \\ &= \frac{1}{2}\|x\|^2 + \Psi(x) - \langle x, z \rangle + \frac{1}{2}\|\sigma(z)\|^2 - \Psi(\sigma(z)) + \langle z - \sigma(z), \sigma(z) \rangle, \\ &= \frac{1}{2}\|x - z\|^2 + \Psi(x) - \frac{1}{2}\|z\|^2 - \frac{1}{2}\|\sigma(z)\|^2 - \Psi(\sigma(z)) + \langle z, \sigma(z) \rangle, \\ &= \frac{1}{2}\|x - z\|^2 + \Psi(x) - \frac{1}{2}\|\sigma(z) - z\|^2 - \Psi(\sigma(z)), \\ &= E_z(x) - E_z(\sigma(z)) = D_{E_z}^0(x, \sigma(z)), \end{aligned}$$

which is an alternative characterisation that is a valid Bregman distance because  $0 \in \partial E_z(\sigma(z))$  is a valid subgradient of  $E_z$  at  $\sigma(z)$ . This characterisation nicely links to recent work in parametric majorisation for data-driven energy minimisation methods (Geiping and Moeller, 2019) as we have shown in Chapter 3.

In Chapter 3, we have motivated the use of the Bregman loss function and showed its gradient evaluation through its form in (4.30). Here we take a closer look at its role as penalisation functions and provide an alternative characterisation to its gradient evaluation with respect to the second argument through its form in (4.28).

**Lemma 3.2** (Gradient of (4.28) with respect to second argument). *Suppose  $\Psi : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$  is a proper, convex and lower semi-continuous function with proximal map  $\sigma := \text{prox}_\Psi$ . Then  $B_\Psi$  as defined in (4.28) is differentiable with respect to its second argument for all  $x \in \text{dom}(\Psi)$  and the gradient with respect to its second argument, i.e.*

$\nabla_2 B_\Psi(x, z)$ , is

$$\nabla_2 B_\Psi(x, z) = \sigma(z) - x,$$

for all  $x \in \text{dom}(\Psi)$ .

*Proof.* Note that the term  $\Phi(x) = \frac{1}{2}\|x\|^2 + \Psi(x)$  in (4.28) is constant with respect to the argument  $z$  and bounded because of  $x \in \text{dom}(\Psi)$ , which implies

$$\nabla_2 B_\Psi(x, z) = \nabla_z [\Phi^*(z) - \langle x, z \rangle]$$

for fixed  $x$ . The gradient of  $\langle x, z \rangle$  with respect to the second argument  $z$  is simply  $x$ ; hence, what remains to be shown is that

$$\nabla \Phi^*(z) = \sigma(z). \quad (4.33)$$

Note that differentiability of  $\Phi^*$  follows directly from (Bauschke et al., 2011, Corollary 18.12) or (Beck, 2017, Theorem 5.26), and that the result (4.33) is well-known. However, for the sake of completeness we include the derivation of the gradient. Since  $\Psi$  is proper, convex and lower semi-continuous, the Fenchel conjugate  $\Phi^*(z) = (\frac{1}{2}\|\cdot\|^2 + \Psi)^*(z) = \sup_x \langle x, z \rangle - \frac{1}{2}\|x\|^2 - \Psi(x)$  of  $\Phi$  is well-defined and the supremum over  $\langle z, x \rangle - \frac{1}{2}\|x\|^2 - \Psi(x)$  is attained. More importantly, we have

$$\begin{aligned} \Phi^*(z) &= \sup_x \langle x, z \rangle - \frac{1}{2}\|x\|^2 - \Psi(x), \\ &= \frac{1}{2}\|z\|^2 + \sup_x -\frac{1}{2}\|x - z\|^2 - \Psi(x), \\ &= \frac{1}{2}\|z\|^2 - \inf_x \left\{ \frac{1}{2}\|x - z\|^2 + \Psi(x) \right\}. \end{aligned}$$

We know that  $\sigma(z) = \text{prox}(z) = \arg \min_{\tilde{x}} \left\{ \frac{1}{2}\|\tilde{x} - z\|^2 + \Psi(\tilde{x}) \right\}$ , which implies  $x = \sigma(z)$  and

$$\Phi^*(z) = \frac{1}{2}\|z\|^2 - \frac{1}{2}\|\sigma(z) - z\|^2 - \Psi(\sigma(z)).$$

The quantity  $M(z) := \frac{1}{2}\|\sigma(z) - z\|^2 + \Psi(\sigma(z))$  is the Moreau-Yosida regularisation of  $\Psi$  (cf. Chapter 2 Definition 1.19), for which we know that its gradient is  $\nabla M(z) = z - \text{prox}(z) = z - \sigma(z)$  (cf. Chapter 2 Lemma 1.2). The gradient of  $\frac{1}{2}\|z\|^2$  with respect to  $z$  is simply  $z$ , which is why we have verified (4.33).  $\square$

**Remark 3.2.** Note that  $B_\Psi(x, z)$  may be unbounded from below for some choices of  $x$ , which implies that  $\nabla_2 B_\Psi(x, z)$  may never be equal to zero for certain choices of  $x$ . More precisely,  $\nabla_2 B_\Psi(x, z) = 0$  is equivalent to  $z - x \in \partial\Psi(x)$ , which requires  $\partial\Psi(x) \neq \emptyset$ . Hence, if  $x$  is chosen such that  $\partial\Psi(x) = \emptyset$ , we can never guarantee  $\nabla_2 B_\Psi(x, z) = 0$ . A simple example is  $\Psi(x) = \delta_{\geq 0}(x)$  with negative argument  $x$ . In that case, there obviously cannot exist an element  $z$  such that  $\sigma(z) = \max(z, 0) = x$  holds true because  $\max(z, 0) \geq 0$  while  $x < 0$ .

We summarise all previous findings in the following theorem.

**Theorem 3.1** (Bregman penalty function). *The Bregman penalty function as defined in (4.28) satisfies the following properties.*

1.  $B_\Psi(x, z) = E_z(x) - E_z(\sigma(z))$ .
2.  $B_\Psi(x, z) = D_\Phi^z(x, \sigma(z)) = D_{\frac{1}{2}\|\cdot\|^2 + \Psi}(x, \sigma(z))$ .
3.  $B_\Psi(x, z) = \frac{1}{2}\|x - \sigma(z)\|^2 + D_\Psi^{z - \sigma(z)}(x, \sigma(z)) \geq \frac{1}{2}\|x - \sigma(z)\|^2$ .
4.  $B_\Psi(x, z) = \frac{1}{2}\|x - \sigma(z)\|^2 + \Psi(x) - \Psi(\sigma(z)) - \langle z - \sigma(z), x - \sigma(z) \rangle$ .
5. For fixed first argument  $x$ , the function  $B_\Psi$  is continuously differentiable w.r.t the second argument if the first argument satisfies  $x \in \text{dom}(\Psi)$ .
6.  $\nabla_2 B_\Psi(x, z) = \sigma(z) - x$ .
7. The global minimum of  $B_\Psi$  is zero, which is attained for all arguments  $x$  and  $z$  that satisfy  $x = \sigma(z)$  and  $\partial\Psi(x) \neq \emptyset$ .
8.  $B_\Psi$  is a bi-convex function, i.e. it is convex w.r.t.  $x$  for fixed  $z$  and convex w.r.t.  $z$  for fixed  $x$ .
9. The operator  $G_x(z) := \nabla_2 B_\Psi(x, z) = \sigma(z) - x$  is monotone.
10. The operator  $G_x$  is Lipschitz-continuous with Lipschitz constant one, i.e.

$$\|G_x(z_1) - G_x(z_2)\| \leq \|z_1 - z_2\|.$$

11. The operator  $G_y$  is firmly non-expansive, i.e.

$$\langle G_x(z_1) - G_x(z_2), z_1 - z_2 \rangle \geq \|G_x(z_1) - G_x(z_2)\|^2.$$

*Proof.* The only properties left to prove are Items 8–11. Item 8 follows directly from the definition of  $B_\Psi(x, z)$  in (4.28), since both  $\frac{1}{2}\|\cdot\|^2 + \Psi - \langle \cdot, z \rangle$  and  $\left(\frac{1}{2}\|\cdot\|^2 + \Psi\right)^* - \langle x, \cdot \rangle$  are convex functions for fixed  $x$  and  $z$ , respectively. Item 10 follows directly from the 1-Lipschitz continuity of the proximal map  $\sigma$  (cf. (Beck, 2017, Theorem 6.42 (b))) via  $\|G_x(z_1) - G_x(z_2)\| = \|\sigma(z_1) - x - (\sigma(z_2) - x)\| = \|\sigma(z_1) - \sigma(z_2)\| \leq \|z_1 - z_2\|$ . In similar fashion, Item 11 follows directly from the firm non-expansiveness of the proximal map  $\sigma$  (cf. (Beck, 2017, Theorem 6.42 (a))) via  $\langle G_x(z_1) - G_x(z_2), z_1 - z_2 \rangle = \langle \sigma(z_1) - x - (\sigma(z_2) - x), z_1 - z_2 \rangle = \langle \sigma(z_1) - \sigma(z_2), z_1 - z_2 \rangle \geq \|\sigma(z_1) - \sigma(z_2)\|^2 = \|\sigma(z_1) - x - (\sigma(z_2) - x)\|^2 = \|G_x(z_1) - G_x(z_2)\|^2$ . The firm non-expansiveness then automatically implies Item 9, since  $\|G_x(z_1) - G_x(z_2)\|^2 \geq 0$  for all  $z_1, z_2$ .  $\square$

**Remark 3.3.** *Alternatively, one can introduce parameters  $\gamma$  in Item 3 to formulate:*

$$B_\Psi(x, z) = \frac{1}{2}\|x - \sigma(z)\|^2 + \gamma D_\Psi^{z - \sigma(z)}(x, \sigma(z)).$$

*Notice that this modification provides a true generalisation of the quadratic penalty term in (4.13) as  $B_\Psi(x, z)$  reduces to  $\frac{1}{2}\|x - \sigma(z)\|^2$  when  $\gamma = 0$ . In the scope of this thesis, we consider  $\gamma = 1$ , but it would be of interest to consider other possible values of  $\gamma$ . In particular, using varying  $\gamma$  allows one to control and balance different  $B_\Psi(x, z)$  terms across different layers, which could potentially lead to better convergence performance.*

**Example 3.1.** *In the following, we show the obtained Bregman penalty associated with popular choices of proximal activation functions. For the ReLU activation function we have:*

$$B_\Psi(x, z) = \begin{cases} \frac{1}{2}\|x - \max(z, 0)\|^2 + \langle x, \max(-z, 0) \rangle & x \in [0, \infty)^n \\ \infty & \text{otherwise} \end{cases}.$$

*Similarly, the Bregman penalty function associated with the soft-thresholding activation function for the scalar case  $x, z \in \mathbb{R}$  can be derived as:*

$$B_\Psi(x, z) = \begin{cases} \frac{1}{2}(x - z + \alpha)^2 + \alpha(|x| - x) & z > \alpha \\ \frac{1}{2}x^2 + \alpha|x| - zx & |z| \leq \alpha \\ \frac{1}{2}(x - z - \alpha)^2 + \alpha(|x| + x) & z < -\alpha \end{cases}.$$

*for  $\alpha > 0$ .*

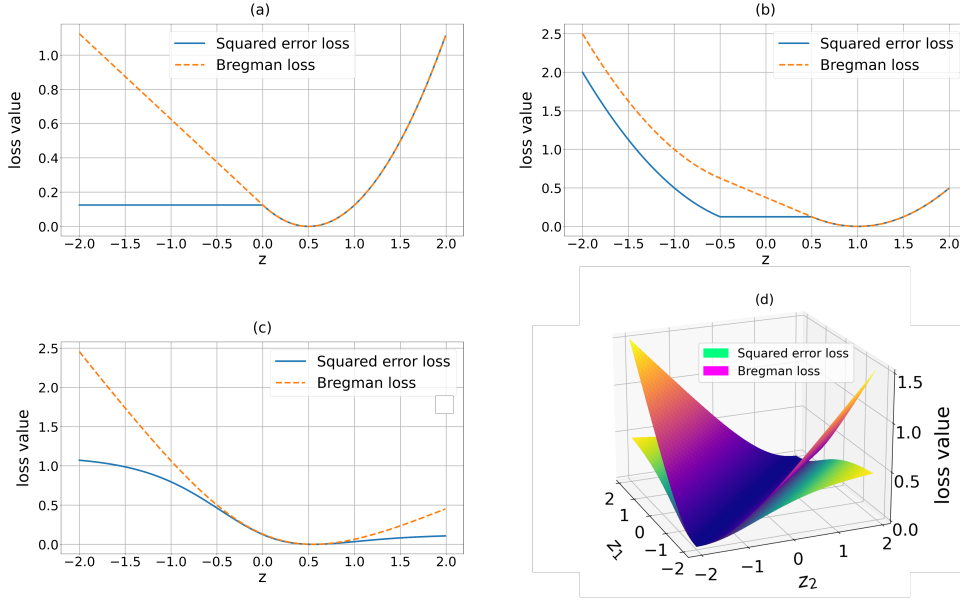


Fig. 4.2 This figure plots the comparison between the Bregman penalty function  $B_\Psi(x, z)$  and the squared Euclidean loss  $\frac{1}{2}\|x - \sigma(z)\|^2$  for  $-2 \leq z \leq 2$ . We show four cases where each  $\sigma$  is: **a)** the ReLU activation function, **b)** the soft-thresholding activation function, **c)** the hyperbolic tangent activation function and **d)** the softmax activation function for  $n = 2$  respectively. The target value  $x$  is set to 0.5 when  $n = 1$  and  $x = (0.5, 0.5)$  when  $n = 2$ .

For the hyperbolic tangent activation function, its induced Bregman penalty function is computed as

$$B_\Psi(x, z) = \begin{cases} \frac{1}{2} \log\left(\frac{1-x^2}{1-\tanh^2 z}\right) + x(\tanh^{-1}(x) - z) & |x| < 1 \\ \infty & \text{otherwise} \end{cases}.$$

Finally, when considering the non-separable softmax activation function, its associated Bregman penalty function is

$$B_\Psi(x, z) = \begin{cases} \sum_{j=1}^n x_j(\log(x_j) - z_j) + \log(\sum_{j=1}^n \exp(z_j)) & x_j \geq 0, \sum_{j=1}^n x_j = 1 \\ \infty & \text{otherwise} \end{cases},$$

which is basically a shifted version of the multinomial logistic regression loss function. In Figure 4.2, we visualise how each example Bregman penalty function compares to the squared Euclidean distance.

### 3.2 Lifted Bregman Training

By replacing the squared Euclidean penalisations in (4.13) with the Bregman penalty function as defined in (4.28), we have modified (4.13) to (4.29).

To demonstrate the advantage of (4.29), let us consider the case with  $s = 1$  and one single training sample pair  $(x, y)$  again. Using Lemma 3.2, the gradient of (4.29) with respect to  $\Theta_l$  reads

$$\nabla_{\Theta_l} B_{\Psi}(x_l, f(x_{l-1}, \Theta_l)) = (\sigma(f(x_{l-1}, \Theta_l)) - x_l) J_f^{\Theta}(x_{l-1}, \Theta_l),$$

where  $J_f^{\Theta}$  denotes the Jacobian of  $f$  with respect to argument  $\Theta$ , and for the specific choice  $f(x_{l-1}, \Theta_l) = W_l^{\top} x_{l-1} + b_l$  we observe

$$\sigma(W_l^{\top} x_{l-1}^i + b_l) - x_l^i = 0,$$

as the optimality condition of (4.29) w.r.t. the bias term  $b_l$ .

This shows that we guarantee network consistency, i.e.  $x_l^i = \sigma(W_l^{\top} x_{l-1}^i + b_l)$ , for all layers  $l$ . In other words, the network is truly nonlinear in contrast to the lifted networks described in Section 2.2.3, and computing the optimality condition with respect to network parameters does not require differentiation of the activation function as in the MAC-QP approach.

### 3.3 Related Works

In this section we review some other works related to the lifted Bregman training framework and briefly address similarities and differences.

To tackle the major limitations of the classical lifted training approach, several recent works also formulate training of lifted networks differently. Fenchel lifted neural network (Gu et al., 2020) and Lifted Proximal Operator Machine (LPOM) (Li et al., 2019) both extend the observation in Zhang and Brand (2017) to a broader class of activation functions and demonstrate improved learning performance. In this section, we briefly present the approaches proposed in these works and point out their connections and differences to the proposed lifted Bregman framework.

Fenchel lifted neural networks (Gu et al., 2020) extend the classical lifted network approach (Askari et al., 2018) as discussed in Section 2.2.3 by reformulating the learning problem (4.2) of a neural network with affine-linear function  $f(x_{l-1}, \Theta_l) = W_l^{\top} x_{l-1} + b_l$

as

$$\begin{aligned} \min_{\mathbf{X}, \{W_l, b_l\}_{l=1}^L} \quad & \sum_{i=1}^s \ell(y^i, W_L^\top x_{L-1}^i + b_L) \\ \text{s.t.} \quad & B_l(x_l, W_l^\top x_{l-1}^i + b_l) \leq 0, \text{ for } l = 0 \dots L-1. \end{aligned}$$

where the equality constraints in Problem (4.5) are converted into constraints on a collection of pre-defined biconvex functions  $B_l$ . For one fixed data pair  $(x, y)$ , the constraint  $B_l(x_l, W_l^\top x_{l-1}) \leq 0$  provides the characterisation of an activation function via

$$x_l^i = \sigma(W_l^\top x_{l-1}^i + b_l) \quad \Leftrightarrow \quad B_l(x_l^i, W_l^\top x_{l-1}^i + b_l) \leq 0.$$

The Lagrange formulation of the constrained minimisation problem yields

$$\min_{\mathbf{X}, \{W_l, b_l\}_{l=1}^L} \quad \sum_{i=1}^s \left[ \ell(y^i, W_L^\top x_{L-1}^i) + \sum_{l=1}^L \lambda_l B_l(x_{l+1}, W_{l+1}^\top x_l) \right],$$

where  $\lambda_l$  are the Lagrange multipliers. The bi-convex functions  $B_l$  essentially take on the same role as the Bregman penalty function (4.28), but without making the connection to Bregman distances and without many of the theoretical results provided in this paper. In the case of the ReLU activation function  $\sigma(z) = \max(z, 0)$ , the functions  $B_l$  are defined as

$$B_l(v, u) = \begin{cases} \frac{1}{2}v^2 + \frac{1}{2}u^2 - uv & \text{if } v \geq 0 \\ \infty & \text{otherwise} \end{cases}.$$

Using Item 4 in Theorem 3.1, it is not difficult to see that this is the scalar-valued equivalent of the Bregman penalty function

$$B_\Psi(x, z) = \begin{cases} \frac{1}{2}\|x\|^2 + \frac{1}{2}\|\max(z, 0)\|^2 - \langle z, x \rangle & \text{if } x \geq 0 \\ \infty & \text{otherwise} \end{cases}.$$

However, without using generalised Bregman distances and proximal activation functions, the derivation of  $B_l$  in Gu et al. (2020) requires deducing the energy from the optimality system. Further, derivations for activation functions other than ReLU seem possible but not as straightforward. Moreover, the analysis in Gu et al. (2020) requires the activation function  $\sigma$  to be strictly monotone, while such strong restrictions are not required in this work.

LPOM on the other hand takes a similar approach to Gu et al. (2020) through the use of proximal operators. The authors define

$$f(x) := \int_0^x (\sigma^{-1}(y) - y) dy \quad \text{and} \quad g(x) := \int_0^x (\sigma(y) - y) dy$$

such that the proximal map of  $f$  as defined in Definition 2.3 is  $\sigma$ , and the proximal map of  $g$  is  $\sigma^{-1}$ . The authors of Li et al. (2019) propose to approximate (4.5) with

$$\min_{\mathbf{x}, \{W_l, b_l\}_{l=1}^L} \sum_{i=1}^s \ell(y^i, W_L^\top x_{L-1}^i + b_L) + \sum_{l=1}^{L-1} \mu_l (f(x_l^i) + g(W_l^\top x_{l-1}^i) + \frac{1}{2} \|x_l^i - (W_l^\top x_{l-1}^i + b_l)\|^2),$$

such that the optimality condition of the objective with respect to  $x_l^i$  reads

$$0 = \mu_l (\sigma^{-1}(x_l^i) - (W_l^\top x_{l-1}^i + b_l)) + \mu_{l+1} (W_{l+1}) (\sigma(W_{l+1}^\top x_l^i + b_l) - x_{l+1}^i) .$$

Similar to the proposed approach, this approach ensures that the network constraints  $x_l^i = \sigma(W_l^\top x_{l-1}^i + b_l)$  are satisfied. However, like the Fenchel lifted neural network approach, the LPOM work requires more restrictive assumptions such as invertibility of the activation function  $\sigma$ , and provides fewer mathematical insights into the newly defined penalisation functions.

### 3.4 Training Residual Neural Network

In this section, we demonstrate that the lifted Bregman framework for training standard feedforward neural networks can be easily extended to the scheme of training residual neural networks (He et al., 2015a). Notice that the residual block at the  $l$ -th layer of a residual neural network (cf. Chapter 2 Definition 2.9) can be conveniently described as

$$x_l^i = x_{l-1}^i + \sigma(f(x_{l-1}^i, \Theta_l)), \quad (4.34)$$

where  $x_l$  are the auxiliary activation variables as described in Section 2. The training problem can be conveniently formulated with the lifted Bregman framework by:

$$\min_{\Theta, \mathbf{x}} \sum_{i=1}^s \left[ \ell(y^i, x_L^i) + \sum_{l=1}^{L-1} \lambda_l B_\Psi(x_l^i - x_{l-1}^i, f(x_{l-1}^i, \Theta_l)) \right],$$

with  $\lambda_l$  being positive scalars. Furthermore, observe that the layer-wise relation (4.34) can be viewed as a discretisation of the dynamical system:

$$\frac{dx}{dt} = \sigma(f(x_{l-1}, \Theta_l)).$$

If we consider constant step size in this discretisation scheme by setting the step size to be 1 at the  $l$ -th time step, we obtain

$$r_l^i = x_l^i - x_{l-1}^i = \sigma(f(x_{l-1}^i, \Theta_l)),$$

where  $r_l$  is introduced as an auxiliary variable called the residual variable. With the introduction of  $r_l$ , we can express each  $x_l$  variable for  $l = \{1, 2, \dots, L-1\}$  by the following scheme

$$x_l^i = x_0^i + \sum_{j=1}^l r_j^i \quad \text{for } l = \{1, 2, \dots, L-1\}, \quad (4.35)$$

where  $x_0$  denotes the input data to the neural network. Hence the learning problem of residual neural networks can be alternatively formulated as

$$\begin{aligned} \min_{\Theta, \mathbf{X}, \mathbf{r}} \sum_{i=1}^s \left[ \ell(y^i, x_L^i) + \sum_{l=1}^{L-1} \lambda_l B_{\Psi}(r_l^i, f(x_{l-1}^i, \Theta_l)) \right] \\ \text{s.t.} \quad r_l^i = x_l^i - x_{l-1}^i, \end{aligned}$$

where  $\mathbf{r} = \{r_l^i\}_{l=1, \dots, L-1}^{i=1, \dots, s}$ . Or equivalently as the following form, where all the  $x_l$  variables except for  $x_L$  in the last layer, are expressed via the recursion in (4.35), i.e.

$$\min_{\Theta, \mathbf{r}} \sum_{i=1}^s \left[ \ell(y^i, x_L^i) + \sum_{l=1}^{L-1} \lambda_l B_{\Psi} \left( r_l^i, f \left( x_0^i + \sum_{j=1}^{l-1} r_j^i, \Theta_l \right) \right) \right]. \quad (4.36)$$

In the next section, we will discuss a range of deterministic and stochastic strategies that can be applied to computationally solve the training problem for both type of neural networks.

## 4 Numerical Realisation

In this section we discuss different computational strategies to solve Problem (4.29) and Problem 4.36 numerically. We divide the discussion into deterministic and stochastic strategies.

### 4.1 Deterministic Approaches

For simplicity of notation, we consider only one pair of samples  $(x, y)$  in Problem (4.29) and we assume that the loss function  $\ell$  can also be written as a Bregman loss function, and set  $\lambda_l = 1$  for all  $l$ . In order to efficiently solve (4.29) we split the objective in (4.29) into differentiable and non-differentiable parts as follows:

$$\begin{aligned}
 E(\Theta, \mathbf{X}) &= \sum_{l=1}^L B_{\Psi}(x_l, f(x_{l-1}, \Theta_l)) \\
 &= \underbrace{\sum_{l=1}^L \frac{1}{2} \|x_l\|^2 + \Psi(x_l)}_{=:G(\mathbf{X})} + \underbrace{\sum_{l=1}^L \left[ \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right)^* (f(x_{l-1}, \theta_l)) - \langle x_l, f(x_{l-1}, \theta_l) \rangle \right]}_{=:H(\Theta, \mathbf{X})} \\
 &= G(\mathbf{X}) + H(\Theta, \mathbf{X}).
 \end{aligned}$$

Here  $H$  is a smooth-function in both arguments, while  $G$  is potentially non-smooth but with closed-form proximal map.

**Remark 4.1.** We want to emphasise that (4.29) can also be split such that the squared Euclidean norm  $\frac{1}{2} \|x_l\|^2$  of each variable  $x_l$  is part of  $H$ , and not of  $G$ . Both splittings are perfectly reasonable, and we arbitrarily chose the one that includes  $\frac{1}{2} \|x_l\|^2$  in  $G$  because computing the proximal map of  $G$  with added squared Euclidean norm requires only a trivial modification of the proximal map (cf. (Beck, 2017, Theorem 6.13)).

#### 4.1.1 Proximal Gradient Descent

A straight-forward approach for the computational minimisation of  $E$  is proximal gradient descent (cf. Chapter 2 Algorithm 4), also known as forward-backward splitting (Lions and Mercier, 1979). This method is a special instance of the following Bregman proximal algorithm (cf. Chapter Algorithm 2.12) that aims at finding minimisers of  $E$  via the iteration

$$\begin{pmatrix} \Theta^{k+1} \\ \mathbf{X}^{k+1} \end{pmatrix} = \arg \min_{\Theta, \mathbf{X}} \left\{ E(\Theta, \mathbf{X}) + D_J \left( \begin{pmatrix} \Theta \\ \mathbf{X} \end{pmatrix}, \begin{pmatrix} \Theta^k \\ \mathbf{X}^k \end{pmatrix} \right) \right\}. \quad (4.37)$$

Here,  $D_J$  denotes the Bregman distance with respect to a function  $J$ , which in case of forward-backward splitting for  $E(\Theta, \mathbf{X}) = G(\mathbf{X}) + H(\Theta, \mathbf{X})$  reads

$$J(\Theta, \mathbf{X}) = \frac{1}{2\tau_\Theta} \sum_{l=1}^L \|\Theta_l\|^2 + \frac{1}{2\tau_{\mathbf{X}}} \sum_{l=1}^{L-1} \|x_l\|^2 - H(\Theta, \mathbf{X}).$$

Here,  $\tau_\Theta$  and  $\tau_{\mathbf{X}}$  are positive step-size parameters that are usually chosen to guarantee local or global convexity of  $J$  in order to guarantee that (4.37) converges locally or globally. Please note that local or global Lipschitz-continuity of the gradient of  $H$  automatically implies existence of  $\tau_\Theta$  and  $\tau_{\mathbf{X}}$  that guarantee local or global convexity of  $J$  (cf. (Bauschke et al., 2017; Benning et al., 2021; Benning and Riis, 2021)). Please note that properties 8, 9, 10 and 11 of Theorem 3.1 are particularly useful in this regard. With these choices of  $E$  and  $J$  the optimisation problem (4.37) simplifies to

$$\begin{aligned} \Theta^{k+1} &= \Theta^k - \tau_\Theta \nabla_\Theta H(\Theta^k, \mathbf{X}^k), \\ \mathbf{X}^{k+1} &= \arg \min_{\mathbf{X}} \left\{ \frac{1}{2} \left\| \mathbf{X} - \left( \mathbf{X}^k - \tau_{\mathbf{X}} \nabla_{\mathbf{X}} H(\Theta^k, \mathbf{X}^k) \right) \right\|^2 + \tau_{\mathbf{X}} G(\mathbf{X}) \right\}, \\ &= \text{prox}_{\tau_{\mathbf{X}} G} \left( \mathbf{X}^k - \tau_{\mathbf{X}} \nabla_{\mathbf{X}} H(\Theta^k, \mathbf{X}^k) \right) \end{aligned}$$

for initial values  $\Theta^0$  and  $\mathbf{X}^0$ . More precisely, the updates with the previous definition of  $G$  and  $H$  read

$$\Theta_l^{k+1} = \Theta_l^k - \tau_\Theta \left( \text{prox}_\Psi \left( f(x_{l-1}^k, \Theta_l^k) \right) - x_l^k \right) J_f^\Theta(x_{l-1}^k, \Theta_l^k), \quad (4.38a)$$

$$\begin{aligned} x_j^{k+1} &= \text{prox}_{\frac{\tau_{\mathbf{X}}}{1+\tau_{\mathbf{X}}} \Psi} \left( \frac{1}{1+\tau_{\mathbf{X}}} \left( x_j^k - \tau_{\mathbf{X}} \left( \left( \text{prox}_\Psi \left( f(x_j^k, \Theta_{j+1}^k) \right) - x_{j+1}^k \right) J_f^x(x_j^k, \Theta_{j+1}^k) \right. \right. \right. \\ &\quad \left. \left. \left. - f(x_{j-1}^k, \Theta_j^k) \right) \right) \right), \end{aligned} \quad (4.38b)$$

for  $l \in \{1, \dots, L\}$  and  $j \in \{1, \dots, L-1\}$ , where  $J_f^\Theta$  and  $J_f^x$  denote the Jacobians of  $f$  with respect to  $\Theta$  and  $x$ , respectively.

**Example 4.1.** Suppose we design a feedforward network architecture with  $\Psi = \chi_{\geq 0}$ , which implies  $\text{prox}_\Psi(z) = \max(z, 0)$ , and  $f(x, \Theta_l) = W_l^\top x + b_l$ , for  $\Theta_l = (W_l, b_l)$ . Then

(4.38) reads

$$\begin{aligned} W_l^{k+1} &= W_l^k - \tau_W x_{l-1}^k \left( \max \left( (x_{l-1}^k)^\top W_l^k + (b_l^k)^\top, 0 \right) - (x_l^k)^\top \right), \\ b_l^{k+1} &= b_l^k - \tau_b \left( \max \left( (W_l^k)^\top x_{l-1}^k + b_l^k \right) - x_l^k, 0 \right), \\ x_j^{k+1} &= \max \left( \frac{x_j^k - \tau_x \left( W_{j+1}^k \max \left( (W_{j+1}^k)^\top x_j^k + b_{j+1}^k, 0 \right) - x_{j+1}^k \right) - (W_j^k)^\top x_{j-1}^k - b_j^k}{1 + \tau_x}, 0 \right), \end{aligned}$$

for  $l \in \{1, \dots, L\}$ ,  $j \in \{1, \dots, L-1\}$  (with input  $x_0 = x$  and output  $x_L = y$ ),  $k \in \mathbb{N}$  and the step-size parameters  $\tau_{\Theta} = (\tau_W, \tau_b)$  and  $\tau_{\mathbf{X}} = \tau_x$ .

Note that many modifications of proximal gradient descent can be applied, such as proximal gradient descent with Nesterov or Heavy-ball acceleration (Huang et al., 2013; Mukkamala et al., 2020; Nesterov, 1983; Teboulle, 2018).

#### 4.1.2 Coordinate Descent

The function  $E$  in Section 4 is convex in each individual variable if all other variables are kept fixed. It therefore makes sense to consider alternating minimisation approaches, also known as coordinate descent (cf. Chapter 2 Algorithm 5) for the minimisation of  $E$ , i.e.

$$\Theta_l^{k+1} = \arg \min_{\Theta_l} E \left( \Theta_1^{k+1}, \dots, \Theta_{l-1}^{k+1}, \Theta_l, \Theta_{l+1}^k, \dots, \Theta_L, x_1^k, \dots, x_{L-1}^k \right), \quad (4.39a)$$

respectively

$$x_j^{k+1} = \arg \min_{x_j} E \left( \Theta_1^{k+1}, \dots, \Theta_L^{k+1}, x_1^{k+1}, \dots, x_{j-1}^{k+1}, x_j, x_{j+1}^k, \dots, x_{L-1}^k \right), \quad (4.39b)$$

for  $l \in \{1, \dots, L\}$  and  $j \in \{1, \dots, L-1\}$ . Obviously, the order of updating the individual parameters is completely arbitrary and can be replaced with permutations. However, we want to point out that it does make sense to alternate with respect to the  $\Theta$  and  $\mathbf{X}$  block, and to further alternate between the block of even and odd indices within the  $\mathbf{X}$  block. Suppose  $L$  is even, then this alternating optimisation between

blocks can be written as

$$\Theta^{k+1} = \arg \min_{\Theta_1, \dots, \Theta_L} E(\Theta_1, \dots, \Theta_L, x_1^k, \dots, x_{L-1}^k), \quad (4.40a)$$

$$\{x_{2l-1}^{k+1}\}_{l=1}^{\frac{L}{2}} = \arg \min_{x_1, x_3, \dots, x_{L-1}} E(\Theta_1^{k+1}, \dots, \Theta_L^{k+1}, x_1, x_2^k, x_3, \dots, x_{L-2}^k, x_{L-1}) , \quad (4.40b)$$

$$\{x_{2l}^{k+1}\}_{l=1}^{\frac{L}{2}} = \arg \min_{x_2, x_4, \dots, x_{L-2}} E(\Theta_1^{k+1}, \dots, \Theta_L^{k+1}, x_1^{k+1}, x_2, x_3^{k+1}, \dots, x_{L-2}, x_{L-1}^{k+1}) . \quad (4.40c)$$

This alternating scheme has the advantage that the individual optimisation problems are still convex, but whilst the three blocks have to be updated sequentially, every variable in each of the three blocks can be updated in parallel. This becomes evident if we write down the individual optimisation problems, i.e.

$$\Theta_l^{k+1} = \arg \min_{\Theta_l} \left\{ \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right)^* (f(x_{l-1}^k, \Theta_l)) - \langle x_l^k, f(x_{l-1}^k, \Theta_l) \rangle \right\}, \quad (4.41a)$$

$$\begin{aligned} x_i^{k+1} = \arg \min_{x_i} & \left\{ \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right) (x_i) - \langle x_i, f(x_{i-1}^k, \Theta_i^{k+1}) \rangle \right. \\ & \left. + \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right)^* (f(x_i, \Theta_{i+1}^{k+1})) - \langle f(x_i, \Theta_{i+1}^{k+1}), x_{i+1}^k \rangle \right\}, \end{aligned} \quad (4.41b)$$

$$\begin{aligned} x_j^{k+1} = \arg \min_{x_j} & \left\{ \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right) (x_j) - \langle x_j, f(x_{j-1}^{k+1}, \Theta_j^{k+1}) \rangle \right. \\ & \left. + \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right)^* (f(x_j, \Theta_{j+1}^{k+1})) - \langle f(x_j, \Theta_{j+1}^{k+1}), x_{j+1}^{k+1} \rangle \right\}, \end{aligned} \quad (4.41c)$$

for  $l \in \{1, \dots, L\}$ ,  $i \in \{1, 3, \dots, L-1\}$  and  $j \in \{2, 4, \dots, L-2\}$ .

Note that the individual updates in (4.39) and in (4.40), respectively (4.41), are still implicit. If we, for instance, want to make the updates with respect to  $\Theta$  explicit, we can modify (4.39a) to

$$\begin{aligned} \Theta_l^{k+1} = \arg \min_{\Theta_l} & \left\{ E(\Theta_1^{k+1}, \dots, \Theta_{l-1}^{k+1}, \Theta_l, \Theta_{l+1}^k, \dots, \Theta_L^k, x_1^k, \dots, x_{L-1}^k) \right. \\ & \left. + D_J \left( (\Theta_1^{k+1}, \dots, \Theta_{l-1}^{k+1}, \Theta_l, \Theta_{l+1}^k, \dots, \Theta_L^k), (\Theta_1^{k+1}, \dots, \Theta_{l-1}^{k+1}, \Theta_l^k, \Theta_{l+1}^k, \dots, \Theta_L^k) \right) \right\}, \end{aligned} \quad (4.42)$$

where  $D_J$  denotes the Bregman distance w.r.t. the function  $J(\Theta) = \sum_{l=1}^L \frac{1}{2\tau_{\Theta_l}} \|\Theta_l\|^2 - E(\Theta, \mathbf{X}^k)$ . Similar modifications can be deployed for the block-coordinate descent scheme (4.40), respectively (4.41), and the auxiliary parameters  $\mathbf{X}$ . In the next Section 4.1.3, we briefly address how to computationally solve the individual sub-problems (4.41a), (4.41b) and (4.41c), with a special case of alternating minimisation approach: the alternating direction method of multipliers (ADMM), see Gabay (1983).

### 4.1.3 Alternating Direction Method of Multipliers

In the previous section we have explored how to break the optimisation problem into different blocks via coordinate descent. Each of these blocks could be solved with proximal gradient descent as described in Section 4.1.1. However, we can alternatively solve these blocks with ADMM (cf. Chapter 2 Algorithm 8) instead. Beginning with sub-problem (4.41a), we can introduce auxiliary variables  $z_l = f(x_{l-1}^k, \Theta_l)$  to replace (4.41a) with the constrained formulation

$$(\Theta_l^{k+1}, z_l) = \arg \min_{\Theta, z} \left\{ \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right)^* (z) - \langle x_l^k, z \rangle \right\} \quad \text{subject to} \quad z = f(x_{l-1}^k, \Theta),$$

for all  $l \in \{1, \dots, L\}$ . By introducing Lagrange multipliers  $\{\mu_l\}_{l=1}^L$ , we can transform these problems into saddle-point problems of the form

$$(\Theta_l^{k+1}, z_l, \mu_l) = \arg \min_{\Theta, z} \max_{\mu} \left\{ \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right)^* (z) - \langle x_l^k, z \rangle + \langle \mu, z - f(x_{l-1}^k, \Theta) \rangle \right\}.$$

ADMM computationally solves this saddle-point problem by alternatingly minimising and maximising the associated augmented Lagrangian

$$\mathcal{L}_l^k(\Theta, z; \mu) = \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right)^* (z) - \langle x_l^k, z \rangle + \langle \mu, z - f(x_{l-1}^k, \Theta) \rangle + \frac{\delta_l}{2} \|z - f(x_{l-1}^k, \Theta)\|^2,$$

for a positive parameter  $\delta_l$ . In the context of (4.41a), these updates read

$$\begin{aligned} \Theta_l^{j+1} &= \arg \min_{\Theta} \left\{ \frac{\delta_l}{2} \|f(x_{l-1}^k, \Theta) - z_l^j\|^2 - \langle \mu_l^j, f(x_{l-1}^k, \Theta) \rangle \right\}, \\ z_l^{j+1} &= \text{prox}_{\frac{1}{\delta_l}(\frac{1}{2}\|\cdot\|^2 + \Psi)^*} \left( \frac{1}{\delta_l} (x_l^k - \mu_l^j + \delta_l f(x_{l-1}^k, \Theta_l^{j+1})) \right), \\ \mu_l^{j+1} &= \mu_l^j + \delta_l (z_l^{j+1} - f(x_{l-1}^k, \Theta_l^{j+1})). \end{aligned}$$

Please note that the proximal map  $\text{prox}_{\frac{1}{\delta_l}(\frac{1}{2}\|\cdot\|^2 + \Psi)^*}$  can easily be expressed via the extended Moreau decomposition (cf. Chapter 2 Definition 1.3), i.e.  $\delta_l \text{prox}_{\frac{1}{\delta_l}(\frac{1}{2}\|\cdot\|^2 + \Psi)^*}(x/\delta_l) = x - \text{prox}_{\delta_l(\frac{1}{2}\|\cdot\|^2 + \Psi)}(x)$ . We further know  $\text{prox}_{\delta_l(\frac{1}{2}\|\cdot\|^2 + \Psi)}(x) = \text{prox}_{\frac{\delta_l}{1+\delta_l}\Psi}(x/(1+\delta_l))$ ; hence, we observe

$$\begin{aligned} z_l^{j+1} &= \text{prox}_{\frac{1}{\delta_l}(\frac{1}{2}\|\cdot\|^2 + \Psi)^*} \left( \frac{1}{\delta_l} (x_l^k - \mu_l^j + \delta_l f(x_{l-1}^k, \Theta_l^{j+1})) \right), \\ &= \frac{1}{\delta_l} \left( x_l^k - \mu_l^j + \delta_l f(x_{l-1}^k, \Theta_l^{j+1}) - \text{prox}_{\frac{\delta_l}{1+\delta_l}\Psi} \left( \frac{1}{1+\delta_l} (x_l^k - \mu_l^j + \delta_l f(x_{l-1}^k, \Theta_l^{j+1})) \right) \right), \end{aligned}$$

which is why we can compute the update solely based on a scaled version of the proximal map of  $\Psi$ , which is the activation function of the neural network.

**Example 4.2.** *In analogy to Example 4.1 we design a feedforward network architecture with  $\Psi = \chi_{\geq 0}$ , implying  $\text{prox}_{\Psi}(z) = \max(z, 0)$ , and  $f(x, \Theta_l) = W_l^\top x + b_l$ , for  $\Theta_l = (W_l, b_l)$ . Computing a solution to (4.41a) via ADMM yields the algorithm*

$$\begin{aligned} W_l^{j+1} &= \left( x_{l-1}^k (x_{l-1}^k)^\top \right)^{-1} x_{l-1}^k \left( z_l^j + \frac{1}{\delta_l} \mu_l^j - b_l^j \right)^\top, \\ b_l^{j+1} &= z_l^j - \left( \frac{1}{\delta_l} \mu_l^j - (W_l^{j+1})^\top x_{l-1}^k \right), \\ r_l^{j+1} &= x_l^k - \mu_l^j + \delta_l \left( (W_l^{j+1})^\top x_{l-1}^k + b_l^{j+1} \right), \\ z_l^{j+1} &= \frac{1}{\delta_l} \left( r_l^{j+1} - \text{prox}_{\frac{\delta_l}{1+\delta_l} \Psi} \left( \frac{r_l^{j+1}}{1+\delta_l} \right) \right), \\ \mu_l^{j+1} &= \mu_l^j + \delta_l \left( z_l^{j+1} - \left( (W_l^{j+1})^\top x_{l-1}^k + b_l^{j+1} \right) \right), \end{aligned} \quad (4.43)$$

for  $l \in \{1, \dots, L\}$ ,  $j \in \mathbb{N}$  and suitable initial values  $b_l^0$ ,  $z_l^0$  and  $\mu_l^0$ . It is important to point out that (4.43) is not well defined for only one sample because  $x_{l-1}^k (x_{l-1}^k)^\top$  is not invertible. However, we can overcome this issue by extending  $E$  to include more samples, so that instead of requiring invertibility of  $x_{l-1}^k (x_{l-1}^k)^\top$ , we require invertibility of  $\sum_{i=1}^s x_{i,l-1}^k (x_{i,l-1}^k)^\top$ , which can be achieved if one has sufficiently distinctive samples. Or we can add the term  $\frac{\tau}{2} \|W_l - W_l^j\|_{\text{Fro}}^2$  to the augmented Lagrangian, with a positive multiple  $\tau$ , which doesn't affect the minimiser but guarantees that (4.43) is well-defined by adding a positive multiple of the identity matrix to  $x_{l-1}^k (x_{l-1}^k)^\top$ , respectively  $\sum_{i=1}^s x_{i,l-1}^k (x_{i,l-1}^k)^\top$ .

In similar fashion to solving (4.41a), we can also solve (4.41b) and (4.41c) via ADMM. Because (4.41b) and (4.41c) have identical structure but only different variables, we focus on (4.41b) without loss of generality. If we follow the same line-of-reasoning as before, the associated augmented Lagrangian reads

$$\begin{aligned} \mathcal{L}_l^k(x, z; \mu) &= \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right) (x) - \langle x, f(x_{l-1}^k, \Theta_l^{k+1}) \rangle + \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right)^* (z) - \langle z, x_{l+1}^k \rangle \\ &\quad + \langle \mu, z - f(x, \Theta_{l+1}^{k+1}) \rangle + \frac{\delta_l}{2} \|z - f(x, \Theta_{l+1}^{k+1})\|^2. \end{aligned}$$

However, the challenge lies in the optimality condition for  $x$ , which even for fixed variables  $z$  and  $\mu$  doesn't lead to a closed-form solution for  $x$  (unless  $\Psi$  is constant).

Alternatively, we can introduce another variable  $v$  with constraint  $v = x$  and formulate

$$\begin{aligned} \mathcal{L}_l^k(x, v, z; \mu) = & \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right) (x) - \langle x, f(x_{l-1}^k, \Theta_l^{k+1}) \rangle + \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right)^* (z) - \langle z, x_{l+1}^k \rangle \\ & + \left\langle \mu, \begin{pmatrix} z \\ v \end{pmatrix} - \begin{pmatrix} f(v, \Theta_{l+1}^{k+1}) \\ x \end{pmatrix} \right\rangle + \frac{\delta_l}{2} \left\| \begin{pmatrix} z \\ v \end{pmatrix} - \begin{pmatrix} f(v, \Theta_{l+1}^{k+1}) \\ x \end{pmatrix} \right\|^2. \end{aligned}$$

Alternatingly minimising and maximising this augmented Lagrangian yields the updates

$$\begin{aligned} x_l^{j+1} &= \text{prox}_{\frac{1}{1+\delta_l}\Psi} \left( \frac{\delta_l v_l^j + (\mu_l^j)_2 + f(x_{l-1}^k, \Theta_l^{k+1})}{1 + \delta_l} \right), \\ v_l^{j+1} &= \arg \min_v \left\{ \frac{\delta_l}{2} \|f(v, \Theta_{l+1}^{k+1}) - z_l^j\|^2 - \langle (\mu_l^j)_1, f(v, \Theta_{l+1}^{k+1}) \rangle \right\}, \\ z_l^{j+1} &= \text{prox}_{\frac{1}{\delta_l}(\frac{1}{2}\|\cdot\|^2 + \Psi)^*} \left( f(v_l^{j+1}, \Theta_{l+1}^{k+1}) + \frac{1}{\delta_l} (x_{l+1}^k - (\mu_l^j)_1) \right), \\ \mu_l^{j+1} &= \begin{pmatrix} z_l^{j+1} \\ v_l^{j+1} \end{pmatrix} - \begin{pmatrix} f(v_l^{j+1}, \Theta_{l+1}^{k+1}) \\ x_l^{j+1} \end{pmatrix}, \end{aligned}$$

for all  $l \in \{1, 3, \dots, L-1\}$ .

#### 4.1.4 Constrained Optimisation

Note that it is not necessarily known a-priori how to choose the scalar parameters  $\{\lambda_l\}_{l=1}^L$  in (4.29). Alternatively, we can also re-formulate the constrained problem formulation (4.5) to

$$\begin{aligned} & \min_{\Theta, \mathbf{X}} \sum_{i=1}^s \ell(y^i, x_L^i) \\ \text{subject to} \quad & B_\Psi(x_l^i, f(x_{l-1}^i, \Theta_l)) \leq 0 \quad \text{for all } l \in \{1, \dots, L-1\}, \end{aligned} \tag{4.44}$$

similar to Gu et al. (2020). Note that we can write (4.44) as

$$\min_{\Theta, \mathbf{X}} \sum_{i=1}^s \ell(y^i, x_L^i) + \sum_{l=1}^{L-1} \chi_{\leq 0} \left( B_\Psi(x_l^i, f(x_{l-1}^i, \Theta_l)) \right),$$

for the characteristic function  $\chi_{\leq 0}$  over the non-positive orthant, which is equivalent to the Lagrange formulation

$$\min_{\Theta, \mathbf{X}} \sup_{\{\lambda_l\}_{l=1}^{L-1}} \sum_{i=1}^s \ell(y^i, x_L^i) + \sum_{l=1}^{L-1} \left[ \lambda_l B_\Psi(x_l^i, f(x_{l-1}^i, \Theta_l)) - \chi_{\geq 0}(\lambda_l) \right], \tag{4.45}$$

if we express  $\chi_{\leq 0}$  in terms of its conjugate  $\chi_{\geq 0}$ . Albeit being concave in  $\{\lambda_l\}_{l=1}^{L-1}$ , the objective function is not convex in  $\Theta, \mathbf{X}$  in general, which is why swapping min and sup will not necessarily lead to the same optimisation problem. Nevertheless, we could employ coordinate descent or alternating direction method of minimisation techniques. We could for example solve (4.45) with an alternating minimisation and proximal maximisation algorithm of the form

$$\begin{aligned} (\Theta^{k+1}, \mathbf{X}^{k+1}) &\in \arg \min_{\Theta, \mathbf{X}} \left\{ \sum_{i=1}^s \ell(y^i, x_L^i) + \sum_{l=1}^{L-1} \lambda_l^k B_{\Psi}(x_l^i, f(x_{l-1}^i, \Theta_l)) \right\}, \\ \lambda_l^{k+1} &= \max \left( 0, \lambda_l^k + \tau B_{\Psi}((x_l^i)^{k+1}, f((x_{l-1}^i)^{k+1}, \Theta_l^k)) \right), \end{aligned}$$

where the minimisation with respect to  $\Theta, \mathbf{X}$  could be solved with any of the algorithms described in Section 4 or in this section.

#### 4.1.5 Regularisation of Network Parameters

In empirical risk minimisation, it is common to design regularisation methods that substitute the empirical risk minimisation process, in order to combat ill-conditioning of the empirical risk minimisation and to improve the validation error. In the context of risk minimisation, regularisations are approximate inverses of the model function with respect to the model parameters. In this section we discuss the two common approaches of variational and iterative regularisation and how to incorporate them into the lifted Bregman framework.

**Variational Regularisation.** In variational regularisation (cf. Chapter 2 Definition 3.3), a positive multiple of a regularisation function is added to the empirical risk formulation. Denoting this regularisation function by  $R$ , we can simply modify  $E$  to

$$E_R(\Theta, \mathbf{X}) = \sum_{l=1}^L B_{\Psi}(x_l, f(x_{l-1}, \Theta_l)) + R(\Theta).$$

However, there is a catch with this modification that is probably not obvious at first glance. Suppose  $R$  is differentiable for now. If we compute the optimality condition w.r.t. a particular  $\Theta_l$ , we observe

$$\begin{aligned} 0 &= \nabla_{\Theta_l} \left( B_{\Psi}(x_l^*, f(x_{l-1}^*, \Theta_l^*)) + R(\Theta_1^*, \dots, \Theta_l^*, \dots, \Theta_L^*) \right), \\ &= \left( \sigma(f(x_{l-1}^*, \Theta_l^*)) - x_l^* \right) J_f^{\Theta}(x_{l-1}^*, \Theta_l^*) + \nabla_{\Theta_l} R(\Theta^*). \end{aligned}$$

Without  $\nabla_{\Theta_l} R(\Theta^*)$ , the condition  $0 = (\sigma(f(x_{l-1}^*, \Theta_l^*)) - x_l^*) J_f^\Theta(x_{l-1}^*, \Theta_l^*)$  guarantees network consistency  $\sigma(f(x_{l-1}^*, \Theta_l^*)) = x_l^*$  up to an element in the nullspace of  $J_f^\Theta(x_{l-1}^*, \Theta_l^*)$ . Unless  $\nabla_{\Theta_l} R(\Theta^*) = 0$ , this network consistency will be violated when adding a regularisation term  $R$  that acts on the network parameters. Because of this shortcoming, we consider iterative regularisation strategies as an alternative in the next subsection.

**Iterative Regularisation.** When we iteratively update the parameters  $\Theta$  and  $\mathbf{X}$  via approaches like (4.37) or (4.42), we can convert them into iterative regularisations known as (linearised) Bregman iterations (Benning and Burger, 2018; Cai et al., 2009; Osher et al., 2005). We achieve this simply by including a regularisation function in the Bregman function. For example, in (4.37) we can choose

$$J(\Theta, \mathbf{X}) = \frac{1}{\tau_\Theta} \left( R(\Theta) + \frac{1}{2} \sum_{l=1}^L \|\Theta_l\|^2 \right) + \frac{1}{2\tau_X} \sum_{l=1}^{L-1} \|x_l\|^2 - H(\Theta, \mathbf{X})$$

instead of

$$J(\Theta, \mathbf{X}) = \frac{1}{2\tau_\Theta} \sum_{l=1}^L \|\Theta_l\|^2 + \frac{1}{2\tau_X} \sum_{l=1}^{L-1} \|x_l\|^2 - H(\Theta, \mathbf{X}).$$

Note that in contrast to including  $R$  in a modified objective  $E_R$ , incorporating  $R$  in  $J$  does not alter  $E$ , but it allows to control the regularity of the model parameters  $\Theta$ . If the objective  $E$  has multiple or even infinitely many minimisers, a different choice of  $R$  enables convergence towards network parameters with desired properties such as sparsity of the parameters.

In the context of neural networks, such a strategy has first been applied in Benning et al. (2021) to train neural networks and also control the rank of the network parameters. We can achieve the same by choosing  $R$  to be a positive multiple of the nuclear norm. Recently, in Bungert et al. (2021), the idea of (linearised) Bregman iterations has been extended to stochastic first-order optimisation in order to effectively train neural networks with sparse parameters. Networks with sparse network parameters can be obtained by setting  $R$  to a positive multiple of the one-norm.

#### 4.1.6 Regularisation of Auxiliary Variables

One of the great advantages of a lifted network approach is that it is straight-forward to also impose regularisation on the auxiliary variables  $\{x_l\}_{l=1}^{L-1}$ . This can be extremely useful in different contexts. Suppose, for example, that we want to train an autoencoder neural network with sparse encoding. This would require us to impose regularity on

the output of the encoder network, which is equal to one of the auxiliary variables in the lifted network formulation. We will discuss such a sparse autoencoder approach in greater detail in Section 5.2. In the following, we want to discuss how to adapt the two regularisation strategies discussed in the previous section.

**Variational Regularisation.** In contrast to variational regularisation of network parameters, adding a regularisation function  $R$  that acts on  $\mathbf{X}$  to the objective  $E$  does not impact the optimality system of  $E$  w.r.t. the network parameters  $\Theta$ . Hence, network consistency will not be violated and regularity can be imposed this way.

**Iterative Regularisation.** In identical fashion to the previous section, we can incorporate regularity by modifying Bregman functions, for example in (4.37) via

$$J(\Theta, \mathbf{X}) = \frac{1}{2\tau_\Theta} \sum_{l=1}^L \|\Theta_l\|^2 + \frac{1}{\tau_{\mathbf{X}}} \left( R(\mathbf{X}) + \frac{1}{2} \sum_{l=1}^{L-1} \|x_l\|^2 \right) - H(\Theta, \mathbf{X})$$

instead of

$$J(\Theta, \mathbf{X}) = \frac{1}{2\tau_\Theta} \sum_{l=1}^L \|\Theta_l\|^2 + \frac{1}{2\tau_{\mathbf{X}}} \sum_{l=1}^{L-1} \|x_l\|^2 - H(\Theta, \mathbf{X}).$$

This concludes the discussion on deterministic optimisation strategies.

## 4.2 Stochastic Approaches

Having discussed suitable deterministic approaches for the minimisation of  $E$ , we want to describe how to adopt such approaches in stochastic settings. In particular, we consider the objective  $E$  from the previous section, but for  $s$  pairs of samples  $\{(x_i, y_i)\}_{i=1}^s$ :

$$E(\Theta, \mathbf{X}) = \frac{1}{s} \sum_{i=1}^s \sum_{l=1}^L B_\Psi(x_l^i, f(x_{l-1}^i, \Theta_l)) . \quad (4.46)$$

Here,  $\Theta = \{\Theta_l\}_{l=1}^L$  is again the short-hand notation for all parameters (for all layers), but  $\mathbf{X} = \{x_l^i\}_{l=1, \dots, L}^{i=1, \dots, s}$  is the collection of auxiliary variables that now also depend on the input and output samples  $\{(x_i, y_i)\}_{i=1}^s$ . We want to investigate which stochastic minimisation methods we can formulate that only use a (random) subset of the indices at every iteration.

The most straight-forward approach is to use out-of-the-box, state-of-the-art first order methods like stochastic gradient descent (cf. Chapter 2 Algorithm 10) or variants

of it. Since the function  $E$  is additively composed of a smooth ( $H$ ) and a non-smooth ( $G$ ) function, the overall function  $E$  is non-smooth. Hence, gradient-based first-order methods applied directly to  $E$  become subgradient-based first-order methods, with potentially slower convergence and artificial critical points, even though these usually do not pose serious issues with high probability (cf. Bolte and Pauwels (2020)). However, fully explicit optimisation methods whose convergence speed depend on Lipschitz constants of gradients can easily suffer from an explosion of these constants when applied directly to non-smooth functions.

Another disadvantage of applying methods such as stochastic gradient descent and evaluating the gradient or subgradient via back-propagation is that it is not straight-forward to easily distribute the computation of parameters as it is the case for distributed optimisation approaches such as the ones described in Section 2.2, respectively (Askari et al., 2018; Carreira-Perpinan and Wang, 2014; Gu et al., 2020).

And not to forget, we lose the advantage that the differential of the Bregman penalty function  $B_\Psi$  does not require the differentiation of the activation function if we try to differentiate the non-smooth part  $G$ . We therefore discuss three alternative optimisation strategies; we begin with a discussion of stochastic proximal gradient descent, then we continue to focus on data-parallel (implicit) optimisation, before we conclude with implicit stochastic gradient descent.

#### 4.2.1 Stochastic Proximal Gradient Descent

Given the structure of  $E$ , it seems natural to consider stochastic proximal gradient descent (Duchi and Singer, 2009; Rosasco et al., 2019). However, most approaches, such as the ones discussed in Rosasco et al. (2019), assume a structure of the form

$$E(\Theta) = G(\Theta) + \frac{1}{s} \sum_{i=1}^s H_i(\Theta),$$

where every  $H_i$  is differentiable and  $G$  is proximable. In comparison, our setting is of the form

$$E(\Theta, \mathbf{X}) = \frac{1}{s} \sum_{i=1}^s (G(\mathbf{X}_i) + H(\Theta, \mathbf{X}_i)), \quad (4.47)$$

where  $\mathbf{X}_i$  is the collection  $\{x_1^i, \dots, x_{L-1}^i\}$  for each index  $i \in \{1, \dots, s\}$ . Here, the function  $G$  only depends on the samples  $\mathbf{X}$  and optimising with respect to  $\mathbf{X}$  remains deterministic, while we can optimise  $\Theta$  by only using (random) subsets of  $\{1, \dots, s\}$

at each iteration. A possible approach is to compute

$$\begin{aligned}\Theta^{k+1} &= \Theta^k - \tau_{\Theta}^k \nabla_{\Theta} H(\Theta^k, \mathbf{X}_i^k), \\ \mathbf{X}_i^{k+1} &= \text{prox}_{\tau_{\mathbf{X}_i} G} \left( \mathbf{X}_i^k - \tau_{\mathbf{X}_i} \nabla_{\mathbf{X}_i} H(\Theta^k, \mathbf{X}_i^k) \right),\end{aligned}$$

or alternating variants. It should be straight-forward to show convergence for such an algorithm with standard arguments in the simpler but unrealistic setting in which both  $H$  and  $G$  are jointly-convex in both  $\mathbf{X}$  and  $\Theta$ . However, proving such a result is beyond the scope of this work.

#### 4.2.2 Data-parallel Optimisation

Instead of minimising (4.47) for all samples at once, one can split the indices into  $m$  randomised batches  $B_p$  with  $\bigcup_{p=1}^m B_p = \{1, \dots, s\}$  and  $\bigcap_{p=1}^m B_p = \emptyset$ . We can then solve the optimisation problems for each batch individually and subsequently average all results, which is also widely known as model averaging (McDonald et al., 2010; Zinkevich et al., 2010). The optimisation problem for each batch can be solved in parallel with any of the methods described in Section 4.1. In the next section, we focus on an alternative implicit optimisation technique that can be performed sequentially for all batches or in parallel, which in its sequential form is known as implicit stochastic gradient descent.

#### 4.2.3 Implicit Stochastic Gradient Descent

Implicit stochastic gradient descent (cf. Chapter 2 Algorithm 12) is a straight-forward modification of stochastic gradient descent where each update is implicit. Note that mini-batch stochastic gradient descent for an objective  $E(\Theta) = \frac{1}{s} \sum_{i=1}^s f_i(\Theta)$ , which in its usual form reads

$$\Theta^{k+1} = \Theta^k - \frac{\tau^k}{|B_p|} \sum_{i \in B_p} \nabla f_i(\Theta^k), \quad (4.48)$$

can be formulated as

$$\Theta^{k+1} = \arg \min_{\Theta} \left\{ \frac{1}{|B_p|} \sum_{i \in B_p} f_i(\Theta) + D_{J_k}(\Theta, \Theta^k) \right\}, \quad (4.49)$$

see for instance Benning and Riis (2021), which shows that it is also a special-case of stochastic mirror descent (Nemirovski et al., 2009), and where

$$J_k = \frac{1}{2\tau^k} \|\Theta\|^2 - \frac{1}{|B_p|} \sum_{i \in B_p} f_i(\Theta),$$

with  $D_{J_k}$  being the corresponding Bregman distance.

Here, the choice of function  $J_k$  ensures the explicitness of mini-batch stochastic gradient descent (cf. (Benning and Riis, 2021, Equation (6))). We can obviously replace  $J_k$  simply with  $\frac{1}{2\tau^k} \|\Theta\|^2$ , so that (4.49) changes to

$$\Theta^{k+1} = \arg \min_{\Theta} \left\{ \frac{1}{|B_p|} \sum_{i \in B_p} f_i(\Theta) + \frac{1}{2\tau^k} \|\Theta - \Theta^k\|^2 \right\}. \quad (4.50)$$

This method, proposed and studied in Toulis and Airolidi (2017) is similar to classical stochastic gradient descent, but the update can no longer be computed explicitly and requires the implicit solution of a deterministic optimisation problem at each iteration. If we want to connect this to Section 4.2.2, we can modify (4.50) to not use the previous argument  $\Theta$  as the second argument in the Bregman distance, but to use the average of the previous epoch, i.e.

$$\Theta_p^{k+1} = \arg \min_{\Theta} \left\{ \frac{1}{|B_p|} \sum_{i \in B_p} f_i(\Theta) + \frac{1}{2\tau_p^k} \left\| \Theta - \frac{1}{m} \sum_{q=1}^m \Theta_q^k \right\|^2 \right\}, \quad (4.51)$$

for  $p \in \{1, \dots, m\}$ . The advantage of (4.51) over (4.50) is that each batch can be processed in parallel, at the cost of potentially inferior convergence speed. Applying (4.50) to objective (4.46) yields the iteration

$$(\Theta^{k+1}, \mathbf{X}^{k+1}) = \arg \min_{\Theta, \mathbf{X}} \left\{ \frac{1}{|B_p|} \sum_{i \in B_p} \sum_{l=1}^L B_{\Psi}(x_l^i, f(x_{l-1}^i, \Theta_l)) + \frac{1}{2\tau^k} \|\Theta - \Theta^k\|^2 \right\}. \quad (4.52)$$

It is important to point out that in (4.52) not all  $x$ -variables are updated at once, but only the variables for the current batch  $B_p$ . Hence, an entire epoch is required to update all  $x$ -variables once. For simplicity, we introduce a short-hand notation  $X_l^p = \{x_l^i\}_{i \in B_p}$  to denote the collection of  $x_l$ -variables associated with the current batch  $B_p$ .

**Algorithm 18:** Implicit Stochastic Lifted Bregman Learning Algorithm

---

```

Initialise  $\Theta_l^0$  for each  $l \in \{1, 2, \dots, L\}$ .
for  $k \in \{1, 2, \dots, K\}$  do
    Choose  $B_p \subset \{1, 2, \dots, s\}$  either at random or deterministically.
    Initialise  $x_j^i = \sigma_j(f(x_{j-1}^i, \Theta_j^k))$  for each  $i \in B_p$  and  $j \in \{1, 2, \dots, L-1\}$ .
    for  $l \in \{L, \dots, 1\}$  and  $j \in \{L-1, \dots, 1\}$  do
        for  $n \in \{0, 1, \dots, N-1\}$  do
            Compute step size parameters  $\tau_\Theta$  and  $\tau_x$ .
            
$$\Theta_l^{n+1} \leftarrow \Theta_l^n - \tau_\Theta \left( \frac{1}{|B_p|} \sum_{i \in B_p} (\text{prox}_\Psi(f(x_{j-1}^{i,n}, \Theta_l^n)) - x_j^{i,n} J_f^\Theta(x_{j-1}^{i,n}, \Theta_l^n)) \right. \\ \left. + \tau^k(\Theta_l^n - \Theta_l^{k-1}) \right),$$

            for  $i \in B_p$  do
                
$$x_j^{i,n+1} \leftarrow \text{prox}_{\frac{\tau_x}{|B_p| + \tau_x}(\Psi + R)} \left( \frac{|B_p|}{|B_p| + \tau_x} \left( x_j^{i,n} + \frac{\tau_x}{|B_p|} (f(x_{j-1}^{i,n}, \Theta_j^k) \right. \right. \\ \left. \left. - (\text{prox}_\Psi(f(x_j^{i,n}, \Theta_{j+1}^n)) - x_{j+1}^{i,n}) J_f^x(x_j^{i,n}, \Theta_{j+1}^n)) \right) \right),$$

            end
        end
         $\Theta_l^k \leftarrow \Theta_l^N.$ 
    end
end

```

---

As an example, consider the problem of training a feedforward network by minimising (4.29) with additional variational regularisation acting on the auxiliary variable as described in Section 4.1.6, for  $f(x_{l-1}, \Theta_l) = W_l^\top x_{l-1} + b_l$ . Using (4.52) and (4.38) for each individual minimisation problem per mini-batch, the overall algorithm is summarised in Algorithm 18. Here,  $K$  refers to the number of epochs, and  $N$  refers to the number of iterations of the inner algorithm.

Note that similar to the examples described in Section 4.1, one can replace the squared-two-norm term in (4.52) with a Bregman distance term to design explicit-implicit variants of (4.52). For example, if we replace  $\frac{1}{2\tau^k} \|\Theta - \Theta\|^2$  with the Bregman distance  $D_{J_k}(\Theta, \Theta^k)$  for the choice

$$J_k(\Theta) = \frac{1}{\tau^k} \|\Theta\|^2 - \frac{1}{|B_p|} \sum_{i \in B_p} \sum_{l=1}^L B_\Psi(x_l^i, f(x_{l-1}^i, \Theta_l)),$$

we can make (4.52) explicit with respect to the parameters  $\Theta$ , with the potential drawback of more complicated implicit optimisation problems for  $\mathbf{X}$ .

**Remark 4.2.** *We want to emphasise that all approaches for computationally solving the lifted Bregman approach require more memory because the lifted Bregman framework introduces auxiliary variables that need to be stored at all times. However, even for the standard back-propagation algorithm, the intermediate variables also have to be stored in memory for the purpose of computing gradients in the backward pass. Nevertheless, this memory requirement is a potential limitation of lifting approaches.*

## 5 Example Problems

In this section, we present some example problems and demonstrate how the lifted Bregman approach can be utilised to train neural networks. We discuss both supervised and unsupervised learning examples. We consider the supervised learning task of image classification, and the unsupervised learning task of training a sparse autoencoder and a sparse denosing autoencoder.

### 5.1 Classification

The first example problem that we consider is image classification. In the fully supervised learning setting, pairwise data samples  $\{x_i, y_i\}_{i=1}^s$  are provided, with each  $x_i$  being an input image while  $y_i$  represents the corresponding class label. The task is to train a classifier that correctly categorises images by assigning the correct labels. Training an  $L$ -layer network via the lifted Bregman approach can be formulated as the following minimisation problem:

$$\min_{\Theta, \mathbf{X}} \frac{1}{s} \sum_{i=1}^s \left[ B_{\Psi_L}(y^i, x_L^i) + \sum_{l=1}^{L-1} B_{\Psi_l}(x_l^i, f(x_{l-1}^i, \Theta_l)) \right]. \quad (4.53)$$

Note that we allow for different functions  $\Psi_l$  in order to allow for different activation functions for different layers. The loss function  $\ell$  in the Learning Problem (4.2) can also be chosen to be a Bregman (penalty) loss function  $B_{\Psi_L}(y^i, x_L^i)$ , which here measures the discrepancy between the predicted output and the target labels.

## 5.2 Sparse Autoencoder

For the next two application examples we consider sparse autoencoders. Sparse autoencoders aim at transforming signals into sparse latent representations, effectively compressing the original signals. In this section, we formulate an unsupervised, regularised empirical risk minimisation for sparse autoencoders in the spirit of Section 4.1.6.

As discussed in Chapter 2, the latent space of a sparse autoencoder can have the same or larger dimension than the input space, where only relatively few coefficients of the code are non-zero. This variant of conventional autoencoders allows for individual signals to have non-zero coefficients in different positions.

Sparsity on latent nodes can be achieved by explicitly regularising the latent representations (Ng et al., 2011; Xie et al., 2012). One approach is to penalise the Kullback Leibler divergence between the hidden node sparsity rate and the target sparsity level during training (Ng et al., 2011; Xie et al., 2012). In this work, we explore an alternative approach using  $\ell_1$  regularisation to promote sparsity of the codes in the spirit of compressed sensing (cf. Candès et al. (2006); Donoho (2006)), and proceed as discussed in Section 4.1.6. We formulate the sparse autoencoder training problem as the minimisation of the energy

$$\min_{\Theta, \mathbf{X}} \frac{1}{s} \sum_{i=1}^s \left[ B_{\Psi_L} \left( x_0^i, x_L^i \right) + \sum_{l=1}^{L-1} \lambda_l B_{\Psi_l} \left( x_l^i, f(x_{l-1}^i, \Theta_l) \right) + \alpha \|x_{L/2}^i\|_1 \right], \quad (4.54)$$

with  $\{x_0^i\}_{i=1}^s$  denoting the provided, unlabelled data. Here  $\{x_{L/2}^i\}_{i=1}^s$  are the activation variables that correspond to the code, i.e. the output of the encoder, which we arbitrarily chose at the middle layer  $\frac{L}{2}$ . The regularisation parameter  $\alpha \geq 0$  is the hyper-parameter controlling the sparsity of the codes.

As discussed in Section 4.1.6, one of the advantages of the lifted Bregman framework is that additional regularisation terms acting on the activation variables can easily be incorporated into various algorithmic frameworks.

## 5.3 Sparse Denoising Autoencoder

We present the third example that aims at learning sparse autoencoders for denoising. During training, a denoising autoencoder receives corrupted versions  $\tilde{x}$  of the input  $x$  and aims to reconstruct the clean signal  $x$ . By changing the reconstruction objective, the denoising autoencoder is compelled to learn a robust mapping against small random perturbations and must go beyond finding an approximation of the identity function

(Alain and Bengio, 2014; Bengio et al., 2013a). In analogy to (5.2), the proposed learning problem reads

$$\min_{\Theta, \mathbf{X}} \frac{1}{s} \sum_{i=1}^s \left[ B_{\Psi_L} (x_0^i, \tilde{x}_L^i) + \sum_{l=1}^{L-1} \lambda_l B_{\Psi_l} (\tilde{x}_l^i, f(\tilde{x}_{l-1}^i, \Theta_l)) + \alpha \|\tilde{x}_{L/2}^i\|_1 \right], \quad (4.55)$$

where  $\tilde{x}_0^i$  and  $x_0^i$  are the corrupted and clear input data respectively. Similar to training sparse autoencoders, we regularise the  $\ell_1$ -norm of the hidden code, i.e. the middle layer activation variable  $x_{L/2}$  to enforce activation sparsity.

## 6 Numerical Results

In this section, we present numerical results for the example applications described in Section 5. We implement the lifted Bregman framework with Algorithm 18 and compare this method to three first-order methods: 1) the classical (sub-)gradient descent method described in (4.3), 2) the stochastic (sub-)gradient descent method which follows (4.48), and 3) the implicit stochastic (sub-)gradient descent that follows (4.50).

All results have been computed using PyTorch 3.7 on an Intel Xeon CPU E5-2630 v4. Related programming code is available through the University of Cambridge data repository at <https://doi.org/10.17863/CAM.86729>.

We use the MNIST dataset (LeCun et al., 1998) and the Fashion-MNIST dataset (Xiao et al., 2017) for all numerical experiments. For both datasets, we pre-process the data by centring the images and by converting the labels to one-hot vector encodings.

For all subsequent experiments, the choice of  $\tau_k$  is made via grid search for values in  $\{0.1, 0.25, 0.33, 0.5, 0.55, 0.8, 1\}$ , where  $\tau_k$  is selected such that the training error is smallest. The number of inner iterations  $N$  is also chosen via grid search for values in  $\{2, 5, 10, 15, 30, 50, 100, 200, 300\}$ . For  $N$  larger than 30 we do not observe a notable improvement in reduction of training error.

### 6.1 Classification

For the classification task, we follow the work of Zach and Estellers (2019) and consider a fully connected network with  $L = 4$  layers with ReLU activation functions. More specifically, we use  $f(x_{l-1}, \Theta_l) = W_l^\top x_{l-1} + b_l$  with  $W_l \in \mathbb{R}^{m_{l-1} \times m_l}$  and  $b_l \in \mathbb{R}^{m_l \times 1}$  where  $m_1 = 784$ ,  $m_2 = m_3 = 64$  and  $m_4 = 10$ . In solving the classification problem described in Section 5.1, we do not include any additional regularisation term and set

$R \equiv 0$  in the learning objective (5.1). We use the Mean Squared Error (MSE) between the prediction and the one-hot encoded target values as our classification loss, which also is a Bregman (penalty) loss function (4.28) for  $\Psi_L \equiv 0$ .

We apply Algorithm 18 to train the classification network with the lifted Bregman framework. In solving each mini-batch sub-problem (4.52), we run a maximum of  $N = 15$  iterations with  $\tau^k = 0.25$  for all  $k$ . The other step-size parameters are set to  $\tau_{W_l} = 1.99/(\|X_l^p\|_2^2 + \tau_k/2)$  and  $\tau_x = 1.99/\|W_l\|_2^2$  to ensure convexity of  $J$  in (4.37). For comparison, the same network architecture is trained via the stochastic (sub-)gradient method (4.48) in combination with the back-propagation Algorithm 16, which we will refer to as the SGD-BP approach. Out of the learning rates  $\{1 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-2}, 5 \times 10^{-2}, 1 \times 10^{-1}, 5 \times 10^{-1}\}$ , we found that  $1 \times 10^{-3}$  works best empirically in terms of receiving lowest training objective values.

For both the MNIST and Fashion-MNIST datasets, we use 60,000 images for training and 10,000 images for validation. Network parameters in all experiments are identically initialised following Glorot and Bengio (2010). We choose batch size  $|B_k| = 100$  and train the network for 100 epochs.

Table 4.1 summarises the achieved training and validation classification accuracy after training. For the classical lifted training approach, we quote results from Zach and Estellers (2019). The lifted Bregman scheme achieves comparable classification accuracy to the SGD-BP approach and shows substantial improvements over the classical lifted approach that produces affine-linear networks.

| Model                   | MNIST |       | Fashion-MNIST |       |
|-------------------------|-------|-------|---------------|-------|
|                         | Train | Test  | Train         | Test  |
| Standard Lifted Network | 85.6% | 86.3% | 81.4%         | 80.0% |
| Lifted Bregman Network  | 99.3% | 96.8% | 93.5%         | 85.7% |
| SGD-BP                  | 99.6% | 96.9% | 95.8%         | 87.9% |

Table 4.1 This table summarises training and test classification accuracies for the MNIST and Fashion-MNIST datasets, where the network is trained with the lifted Bregman approach, the classical lifted training approach and the stochastic (sub-)gradient method with back-propagation.

In Table 4.2 we present the evaluated percentage of linear activations of the network under different training strategies. This rate computes the percentage of number of nodes in each hidden layer that perform linearly, i.e. ( $z^i = \max(z^i, 0)$ ). As described in Section 2.2.3, classical lifted network training produces affine-linear networks. We empirically verify that the lifted Bregman framework overcomes this limitation.

| Model                   | MNIST   |         |         | Fashion-MNIST |         |         |
|-------------------------|---------|---------|---------|---------------|---------|---------|
|                         | Layer 1 | Layer 2 | Layer 3 | Layer 1       | Layer 2 | Layer 3 |
| Standard Lifted Network | 99.9%   | 99.9%   | 99.9%   | 99.9%         | 99.9%   | 99.9%   |
| Lifted Bregman Network  | 47.4%   | 82.2%   | 70.1%   | 59.2%         | 85.7%   | 64.7%   |
| SGD-BP                  | 40.1%   | 29.7%   | 73.6%   | 32.2%         | 28.1%   | 78.4%   |

Table 4.2 In this table, we show the percentage of nodes in the network’s hidden layers that act linearly. The network is trained with the lifted Bregman approach, the classical lifted approach and the the stochastic (sub-)gradient method with back-propagation respectively, on both the MNIST and Fashion-MNIST datasets.

|             | MNIST         |           |               |
|-------------|---------------|-----------|---------------|
|             | Per iteration | Per epoch | Total runtime |
| LBN         | 0.003         | 29.45     | 3011.72       |
| <i>LBN*</i> | 0.001         | 15.02     | 1583.89       |
| SGD-BP      | 0.001         | 5.73      | 600.62        |
|             | Fashion-MNIST |           |               |
|             | Per iteration | Per epoch | Total runtime |
| LBN         | 0.003         | 29.65     | 3088.94       |
| <i>LBN*</i> | 0.001         | 16.02     | 1612.48       |
| SGD-BP      | 0.001         | 6.12      | 603.51        |

Table 4.3 This table records runtimes for the LBN approach and SGD-BP approaches on training classifiers on MNIST and Fashion-MNIST datasets respectively. Entries in *LBN\** show the runtimes of the LBN approach where the time spent on computing the step size parameters  $\tau_{w_l}$  and  $\tau_{x_l}$  is deducted.

In terms of computational cost, we record the runtimes of the LBN and SGD-BP approaches and report them in Table 4.3. It should be emphasised that LBN schemes are computationally more expensive compared to SGD-BP approaches, due to the additional iterations for solving the inner problem. Entries in *LBN\** show the runtimes of the LBN approach where the time spent on computing the step size parameters  $\tau_{w_l}$  and  $\tau_{x_l}$  is deducted.

## 6.2 Sparse Autoencoder

As discussed in Section 5.2, we train a fully connected autoencoder with  $L = 4$  layers with  $f(x_{l-1}, \Theta_l) = W_l^\top x_{l-1} + b_l$ . The hidden dimension is set to  $m_l = 784$  for  $l = 1, \dots, 4$ , i.e. there is no explicit reduction in dimension. We use the MSE reconstruction loss and the regularisation parameter  $\alpha$  is chosen at  $\alpha = 0.09$  and does

not ensure optimal validation errors, but a sparsity rate of the code of approximately 90% to guarantee an implicit reduction in dimension instead.

Note that the additional regularity on the code can easily be implemented in the lifted Bregman framework provided we choose a suitable activation function for the activation variable that corresponds to the code. Specifically, when the activation variable  $x_2$  is activated by the soft-thresholding function, the additional non-smooth  $\ell_1$ -norm regularisation term in (5.2) can easily be incorporated by modifying the  $\Psi_2$  function in the update step of the  $x_2$  variable in Algorithm 18. In this example, we apply the soft-thresholding activation function after the second affine-linear transformation and adopt ReLU activation functions for all other layers.

**MNIST-1K** For this application example, we consider a slightly more challenging learning scenario (referred to as MNIST-1K). We limit the amount of training data, and compare how well the trained network generalises on a larger validation set. We choose  $s = 1,000$  images from the MNIST dataset at random and use it as our training dataset and use all 10,000 images from the validation dataset for validation.

We experiment with both deterministic and stochastic implementations of the lifted Bregman framework. For the stochastic implementation (referred to as LBN-S), network parameters are trained via Algorithm 18. We choose batch size  $|B_k| = 20$  and step-size  $\tau_k = 1$ . In each batch sub-problem, the minimisation problems are iterated for  $N = 15$  iterations, with step-sizes computed via  $\tau_{w_l} = 1.99s/(\|X_l^p\|_2^2 + \tau_k/2)$  and  $\tau_b = 1.99$ . The deterministic implementation (LBN-D) follows the same update steps described in Algorithm 18 with  $\tau^k = 0$  and  $|B_k| = s$ , i.e. we use all data at every epoch. The step size  $\tau_{w_l}$  is computed as  $1.99s/\|X_l\|_2^2$ .

For comparison, we train the autoencoder also with two first-order methods: (sub-)gradient descent (GD-BP), which follows (4.3), and the stochastic (sub-)gradient descent approach (SGD-BP) described in (4.48), both in combination with the back-propagation Algorithm 16 for the computation of (sub-)gradients.

As learning objective we choose the MSE loss plus  $\alpha$  times the  $\ell_1$ -norm regularisation. Various learning rates in the range  $[10^{-4}, 10^{-1}]$  have been tested. In terms of minimising the training objective values, we find that  $4 \times 10^{-2}$  and  $3 \times 10^{-2}$  work optimal for GD-BP and SGD-BP approaches, respectively.

We train the autoencoder network for 100 epochs and plot the decay of the objective values over all epochs in Figure 4.3 (Top). While the training objective value tracks the sum of reconstruction loss and  $\alpha$  times  $\ell_1$ -norm regularisation, the validation objective only records the MSE loss. The sparsity rate per epoch is computed as the percentage of zero-valued entries of the code  $X_2$ , which we visualise in Figure 4.3 (Bottom).

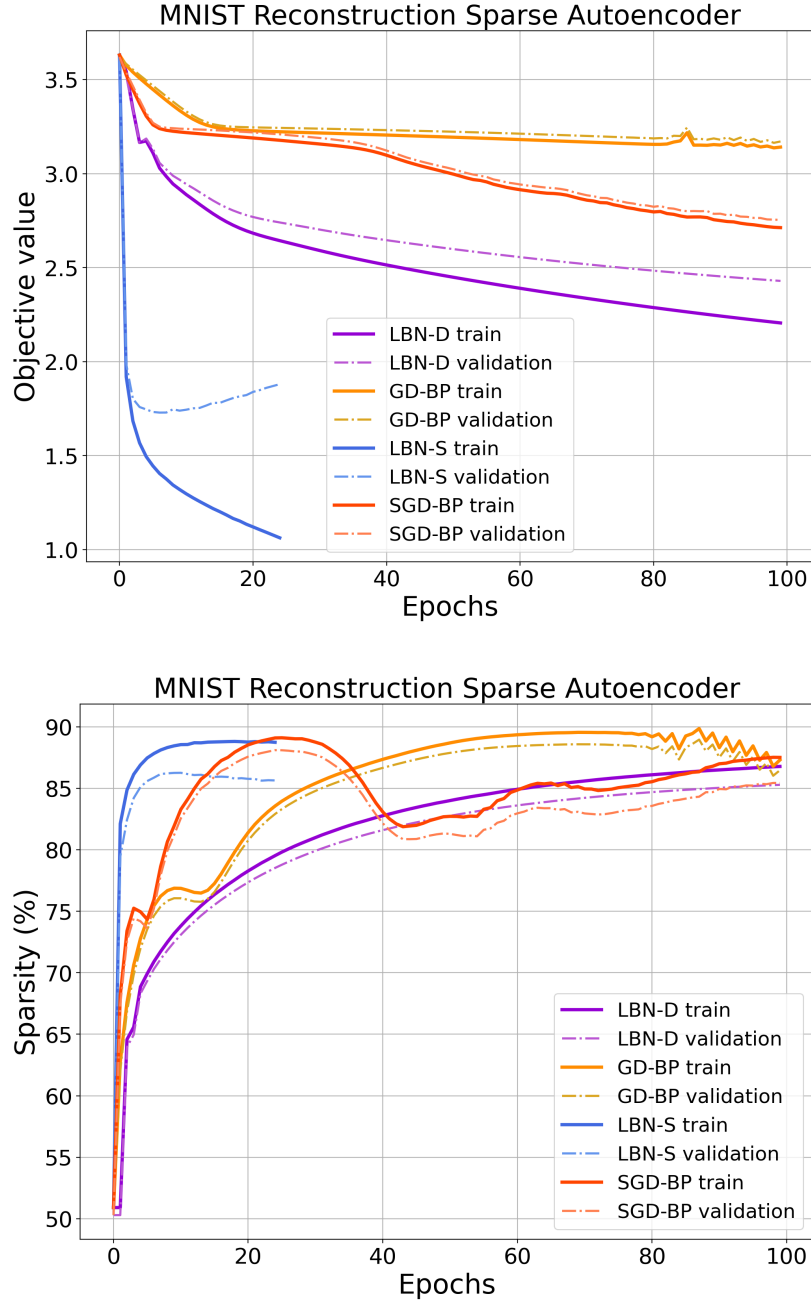


Fig. 4.3 Sparse autoencoder trained on MNIST-1K dataset. **Top:** Objective values per epoch for each learning scheme as explained in Section 6.2. The training objectives (solid lines) record MSE loss plus  $\ell_1$ -norm regularisation and the validation objectives (dashed lines) report the MSE loss values. **Bottom:** Sparsity level of the code per training epoch for each learning scheme.

Overall, the lifted Bregman implementations outperform the (sub-)gradient-based first-order methods. The reason for this discrepancy in performance is that using

first-order subgradient methods to minimise highly non-smooth functions is less efficient compared to using proximal first-order methods. Since we enforce high-levels of sparsity in this example, it is expected that the proposed lifted Bregman approach performs better. If one were to decrease the level of sparsity of the code, the discrepancy in performance will become smaller. We also observe that the stochastic variants of both learning schemes can speed up training and provide faster convergence compared to their deterministic counterparts. The effect on LBN-D is more noticeable than on GD-BP as we observe significant drops in objective value in early training epochs.

The validation loss curves recovered with GD-BP and SGD-BP follow their training curves quite closely. LBN-S on the other hand shows a drop of the validation loss before it increases again after around 7 epochs. A similar increase could be observed for LBN-D if we were to train the parameters for more epochs. This demonstrates that we can overfit this autoencoder with 3136 parameters to the 1000 training data samples of dimension 784 with the LBN algorithms, but we do not observe the same phenomenon with the GD-BP and SGD-BP algorithms. A comparison of the LBN-D, GD-BP and SGD-BP approaches when trained for more epochs can be found in Figure 4.4. In light of Zhang et al. (2021b), this raises the question whether good generalisation properties of neural networks are more down to the choice of optimisation technique than architectural design choices.

When looking at a sparsity rate comparison per epoch, we notice the oscillatory behaviours of both (sub-)gradient based methods. As their sparsity levels climb higher, their corresponding objective value curves remain stagnant. The decreases in objective values for both GD-BP and SGD-BP roughly coincide with when their sparsity rate drops. In contrast, we can see that the lifted Bregman implementations handle the same level of regularisation more smoothly. They achieve similar sparsity levels in the end but in a more stable and controlled manner.

In Figure 4.5, we visualise ground truth images and reconstructions of two randomly selected images from the training dataset for all four training approaches. In Figure 4.6 we visualise the same quantities but based on two randomly selected images from the validation dataset. All four approaches are capable of providing good quality reconstructions in light of the high sparsity levels, for both the training and testing datasets. However, the proposed LBN-D and LBN-S approaches provide sharper edges and finer details than the GD-BP and SGD-BP approaches. LBN-S slightly outperforms LBN-D and is capable of defining even clearer and sharper edges. More reconstructed images can be found at the end of this chapter.

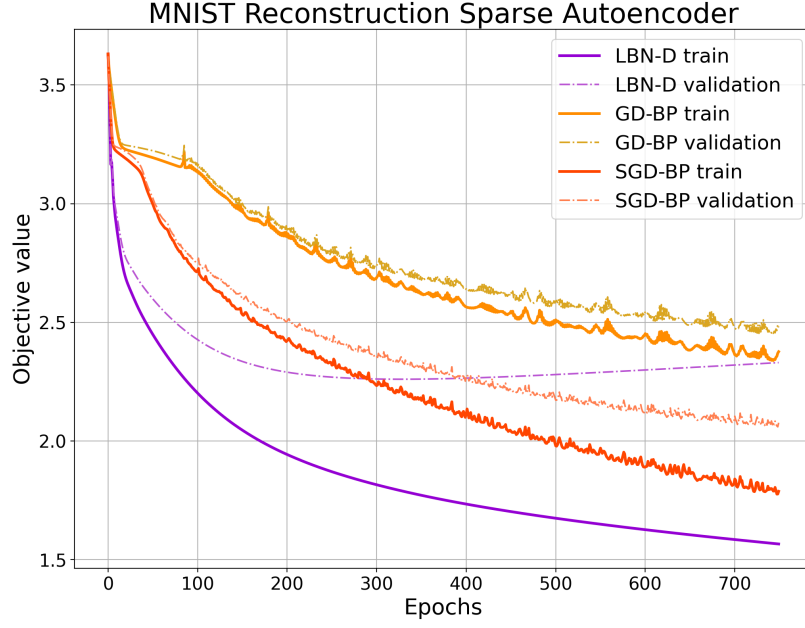


Fig. 4.4 Objective values for the sparse autoencoder trained on MNIST-1K dataset for 750 epochs for the LBN-D, GD-BP and SGD-BP scheme as explained in Section 6.2. The training objectives (solid lines) record MSE loss plus  $\ell_1$ -norm regularisation and the validation objectives (dashed lines) report the MSE loss values.

|           | Per iteration | Per epoch | Total runtime |
|-----------|---------------|-----------|---------------|
| LBN-D     |               | 0.33      | 90.35         |
| $LBN-D^*$ |               | 0.24      | 61.98         |
| GD        |               | 0.17      | 53.71         |
| LBN-S     | 0.14          | 98.09     | 11204.28      |
| $LBN-S^*$ | 0.014         | 2.045     | 257.98        |
| SGD-BP    | 0.011         | 0.237     | 60.81         |

Table 4.4 This table records runtimes for each approach of training sparse autoencoders on the MNIST-1K dataset. Entries in  $LBN - D^*$  and  $LBN - S^*$  record the runtimes where the time spent on computing step size parameters  $\tau_{w_l}$  and  $\tau_{x_l}$  is deducted.

For all experiments, we record runtimes for each approach for comparison and visualise them in Table 4.4. For deterministic approaches, we record only per epoch runtime as it is the same as the per iteration runtime. Entries in  $LBN-D^*$  and  $LBN-S^*$  show runtimes where the time spent on computing the step size parameters  $\tau_{w_l}$  and  $\tau_{x_l}$  has been deducted. We note that the difference in runtime between the LBN and SGD-BP approach per iteration is mainly a result of the computation of the step size

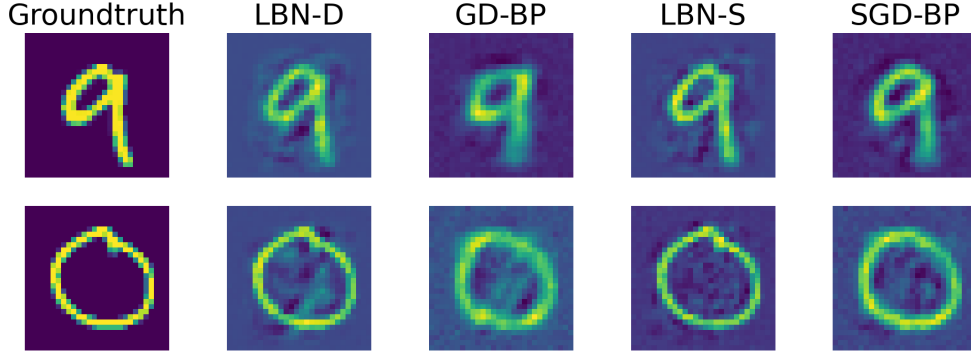


Fig. 4.5 Reconstructed images from the MNIST training dataset for the LBN-D, LBN-S, GD-BP and SGD-BP training strategies respectively, along with ground truth images.

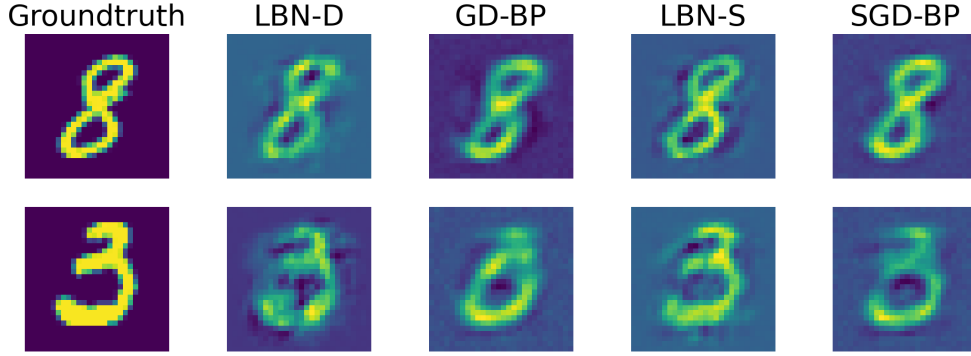


Fig. 4.6 Reconstructed images from the MNIST validation dataset for the LBN-D, LBN-S, GD-BP and SGD-BP training strategies respectively, along with ground truth images.

parameters  $\tau_{w_l}$  and  $\tau_{x_l}$ . We will present a fairer comparison with in the upcoming sparse denoising autoencoder section.

### 6.3 Sparse Denoising Autoencoder

The sparse denoising autoencoder adopts the same network architecture as the sparse autoencoder described in Section 6.2, which sets the hidden dimension to 784 across all layers. To produce noisy images, instances of Gaussian random variables with mean zero and standard deviation  $10^{-3}$  are added to each pixel. We use the MSE as the last layer loss function to measure the difference of reconstruction and noisy images. We consider two training scenarios to evaluate model performances: 1) In the first scenario, we train the network with limited data from the Fashion-MNIST dataset, similar to Section 6.2, where we take 1,000 training images (referred to as

Fashion-MNIST-1K) and validate on 10,000 images. 2) For the second scenario, the training dataset consists of 10,000 images (referred to as Fashion-MNIST-10K) and the validation dataset consists of 10,000 images.

**Fashion-MNIST-1K** In the first group of experiments, we train a sparse autoencoder with imposed sparsity regularisation during training on the Fashion-MNIST-1K dataset. For the lifted Bregman training approach (LBN) we apply Algorithm 18 to minimise the learning objective (5.3) with  $\alpha = 0.09$ . We set the batch size to  $|B_k| = 20$ , the step-size parameter  $\tau_k$  to  $\tau_k = 0.5$ , for all  $k$ , and the inner iterations  $N$  to  $N = 15$  for solving each mini-batch sub-problem (4.52) via (4.38). For comparison, we consider a vanilla stochastic (sub-)gradient method (SGD-BP) as described in (4.48) and the stochastic (sub-)gradient descent approach with implicit parameter update (ISGD-BP) that follows (4.50) to train the network parameters. Both approaches use the learning rate  $1 \times 10^{-3}$  and apply the back-propagation Algorithm 16 for computing the (sub-)gradients. The ISGD-BP does so by solving the inner problem with GD-BP for  $N$  iterations.

All approaches train for 50 epochs. A log-scale plot of the objective value decay for all three training approaches is visualised in Figure 4.7 (Top) and the tracked changes of the hidden node sparsity level are plotted in Figure 4.7 (Bottom).

When trained on the Fashion-MNIST-1K dataset, we observe that LBN achieves faster convergence and outperforms ISGD-BP and SGD-BP especially at early training epochs. Towards later epochs, ISGD-BP starts to align and eventually achieves comparable training and validation errors.

The ISGD-BP approach and LBN approach are capable of achieving and maintaining higher sparsity levels from earlier epochs onwards, while the SGD-BP approach on the other hand seems to increase the sparsity level much more slowly compared to the other two approaches. It should, however, be emphasised that the SGD-BP algorithm is computationally less expensive per epoch.

**Fashion-MNIST-10K** We conduct the second set of experiments on the Fashion-MNIST-10K dataset. We choose a bigger batch size of  $|B_k| = 200$  and penalise the  $\ell_1$ -norm regularisation with  $\alpha = 0.055$  during training. For both the LBN and the ISGD-BP approach, we set  $\tau_k = 1$  and perform  $N = 30$  iterations in each mini-batch sub-problem. We choose a learning rate of  $4 \times 10^{-4}$  for the ISGD-BP approach as well as for the SGD-BP approach.

In Figure 4.8 we visualise the training and validation loss curves (Top) and the sparsity rate (Bottom) over the 50 training epochs when training the proposed algorithms on the Fashion-MNIST-10K dataset. As expected, the gap between training

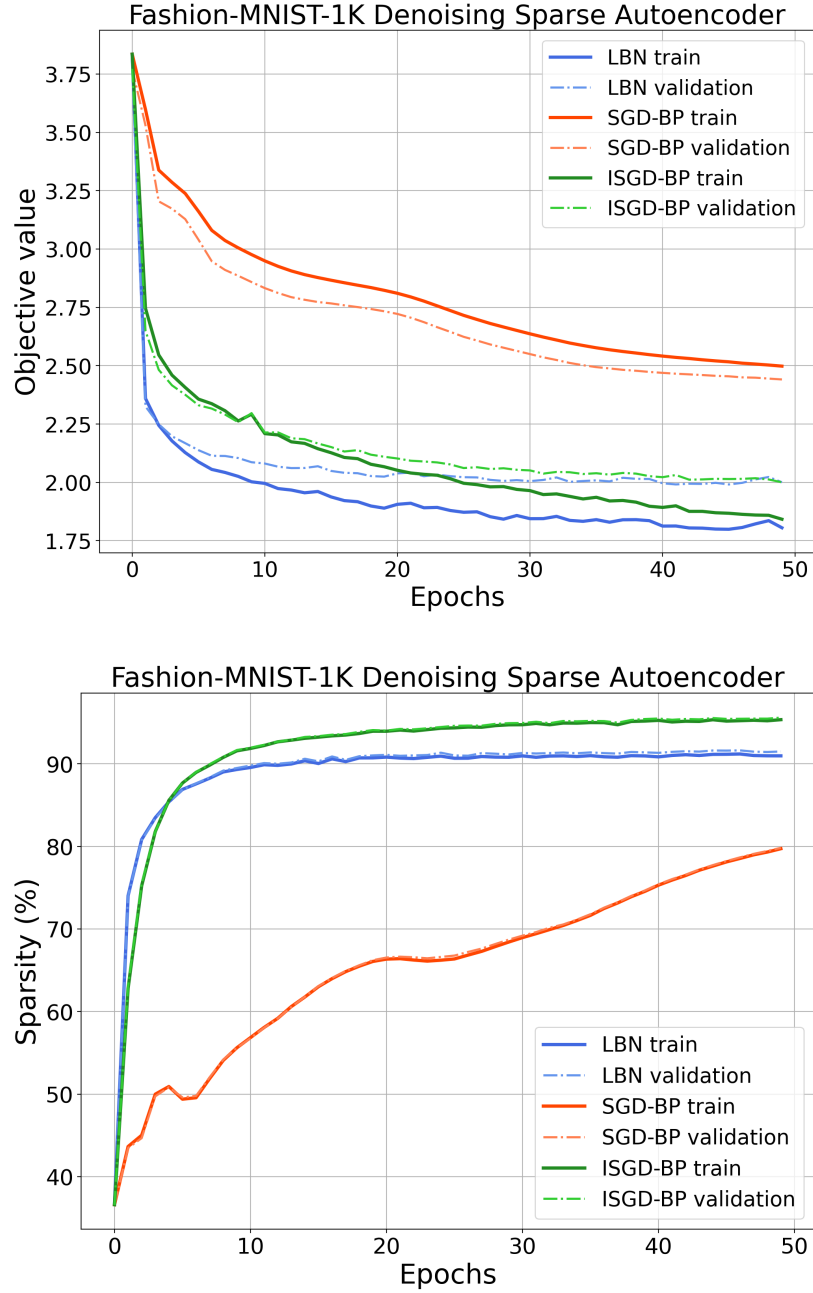


Fig. 4.7 Sparse denoising autoencoder trained on Fashion-MNIST-1K images. **Top:** Objective values per epoch for each learning approach. The training loss record MSE reconstruction error plus  $\alpha$  times  $\ell_1$ -norm regularisation while the validation loss record the MSE reconstruction error. **Bottom:** Sparsity level of the code per training epoch for each learning scheme.

and validation loss curves in this experiment are closer due to larger amount of training samples while network complexity is unchanged. Note that the proposed LBN approach

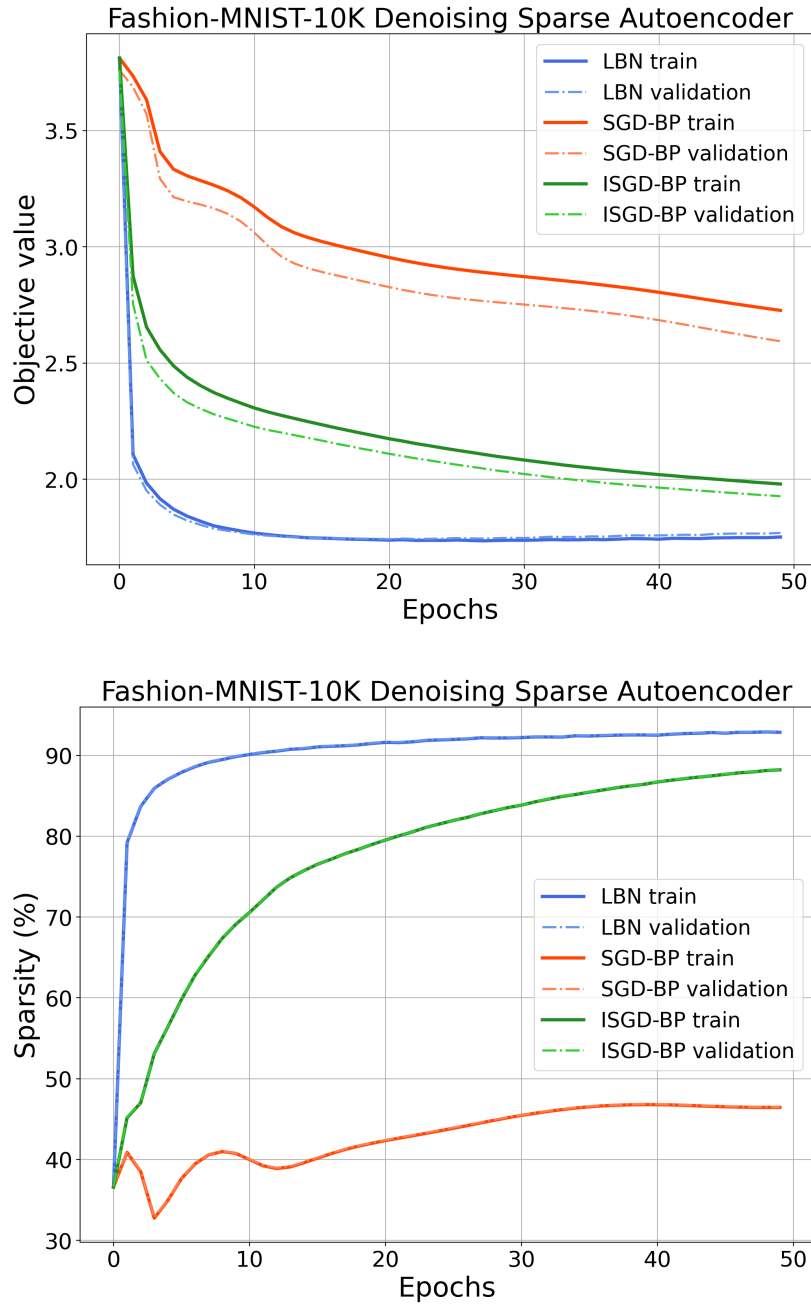


Fig. 4.8 Sparse denoising autoencoder trained on Fashion-MNIST-10K images. **Top:** Objective values per epoch for each learning approach. The training loss record MSE reconstruction error plus  $\alpha$  times  $\ell_1$ -norm regularisation while validation loss curves record the MSE reconstruction error. **Bottom:** Sparsity level of the code over the 50 training epochs for each learning scheme.

exhibits stronger performance and outperforms the other two (sub-)gradient based methods when a larger amount of training data is available. Not only is the LBN

approach capable of achieving lower reconstruction loss values after 50 epochs but also achieves and maintains higher sparsity rates more quickly.

In terms of computational cost, we record the runtimes for each approach and visualise them in Table 4.5. If we deduce the time spent on computing the step size parameters, as shown in the entries  $LBN^*$  in Table 4.5, the total runtime largely decreases. Hence, as both the LBN and ISGD-BP approaches require 30 inner iterations per batch sub-problem, we also compare both approaches to the SGD-BP approach for 1500 epochs to ensure the comparison between the total number of iterations for all three approaches is fairer. As visualised in Figure 4.9, we verify that the SGD-BP approach matches training and validation errors of ISGD-BP, but falls short of achieving training and validation values as low as those obtained with LBN.

|            | Per iteration | Per epoch | Total runtime |
|------------|---------------|-----------|---------------|
| LBN        | 0.153         | 236.76    | 12768         |
| $LBN^*$    | 0.021         | 33.11     | 1720.96       |
| ISGD-BP    | 0.027         | 50.79     | 2439.76       |
| SGD-BP     | 0.018         | 2.17      | 144.32        |
| $SGD-BP^*$ | 0.018         | 2.09      | 4205.41       |

Table 4.5 This table records runtimes for each approach of training sparse denoising autoencoders on the Fashion-MNIST-10K dataset. Entries in  $LBN^*$  record the runtimes where the time spent on computing step size parameters  $\tau_{w_l}$  and  $\tau_{x_l}$  is deducted. Entries in  $SGD-BP^*$  record the runtimes when the SGD-BP approach trains for 1500 epochs.

Although LBN approaches require more runtime per iteration than standard SGD-BP approaches due to the additional computation of the auxiliary variables  $X_l$ , we want to stress that parallelisation can benefit LBN approaches better than the standard SGD-BP based approaches. The standard back-propagation algorithm is sequential in nature, whereas the LBN approaches allow parallel computation of the weight and auxiliary variables as discussed in section 4.1.2.

**Comparison with Undercomplete Autoencoders.** So far, we haven’t properly motivated the use of sparse autoencoders in comparison to traditional autoencoders that explicitly reduce the dimension of the code. A sparse code with  $m$  non-zero entries requires basically the same amount of memory than a  $m$ -dimensional code but offers greater flexibility because the location of the non-zero entries can vary for different network inputs. This advantage should lead to better validation errors of sparse autoencoders compared to traditional, undercomplete autoencoders. In order to verify the advantages of training sparse autoencoders over training undercomplete autoencoders (UAE), we conduct two additional experiments where we train under-

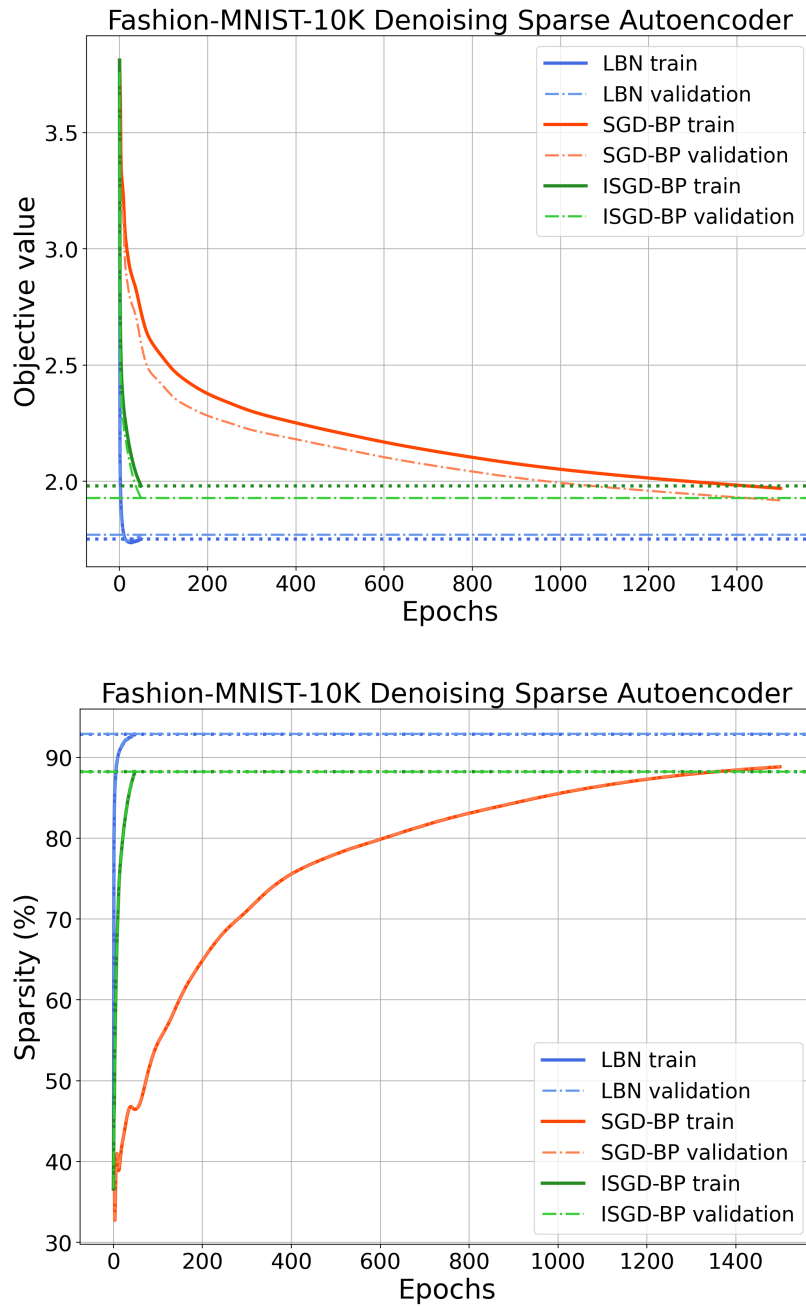


Fig. 4.9 Comparison of sparse denoising autoencoders trained on the Fashion-MNIST-10K dataset with LBN, ISGD-BP and SGD-BP for 1500 epochs, and of sparse denoising autoencoders with undercomplete autoencoders. **Top:** Objective values per epoch for all learning approaches. Horizontal lines mark end-of-training and end-of-validation objective values of LBN and ISGD-BP. **Bottom:** Sparsity level of the code over the 1500 training epochs. Horizontal lines show end-of-training sparsity rates of LBN and ISGD-BP.

complete autoencoders on the Fashion-MNIST-10K dataset without imposing any additional regularisation.

We observe that the sparse denoising autoencoder trained with  $\alpha = 0.055$  eventually reaches 91% sparsity rate after 50 epochs, which suggests that on average 70 nodes in the code of the final model are activated. Hence, we train a 4-layer fully connected undercomplete autoencoder with ReLU activation function for every layer but set the number of nodes in the middle layer to 70, which implies  $W_2 \in \mathbb{R}^{784 \times 70}$  and  $W_3 \in \mathbb{R}^{70 \times 784}$ . The undercomplete autoencoder is trained with both LBN and SGD-BP and we compare if a reconstruction quality comparable to the sparse autoencoder can be achieved.

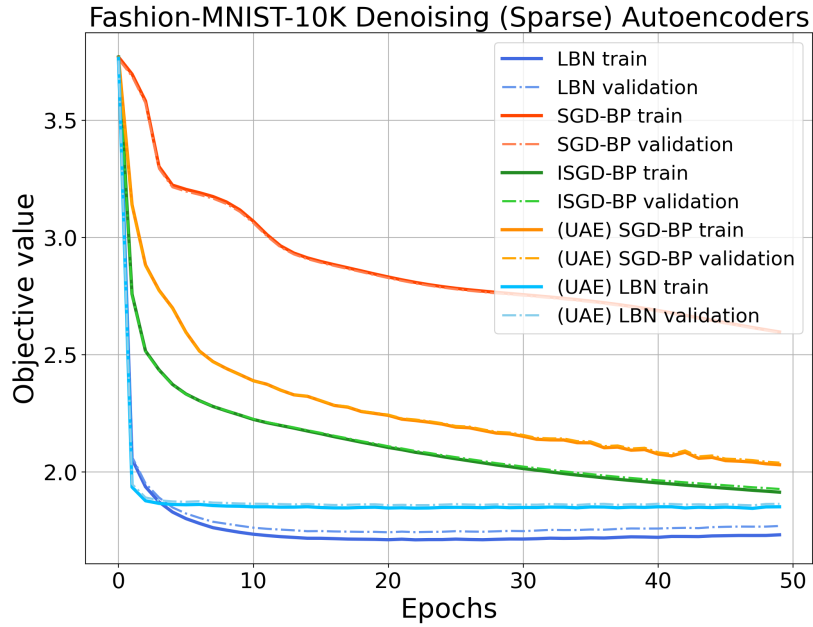


Fig. 4.10 Comparison of sparse denoising autoencoders trained on the Fashion-MNIST-10K dataset with LBN, ISGD-BP and SGD-BP, and of sparse denoising autoencoders with undercomplete autoencoders. Plot shows objective values per epoch for each learning approach. The training and validation loss curves report MSE reconstruction errors over 50 epochs.

In Figure 4.10, we validate that the lifted Bregman approach helps to improve both undercomplete and sparse autoencoder to achieve faster convergence and lower objective values compared to the (sub-)gradient based methods. We also confirm that after 50 epochs the sparse denoising autoencoder achieves lower training and validation errors compared to the UAE approach.

More specifically, the UAE model trained with the lifted Bregman approach sees bigger reconstruction loss decreases in the early epochs, but is outperformed by the

sparse autoencoder at later epochs. This experiment seems to suggest that by leveraging the power of sparsity, sparse autoencoders trained via lifted Bregman approaches are capable of finding more flexible data representations compared to autoencoders with explicit dimension reduction.

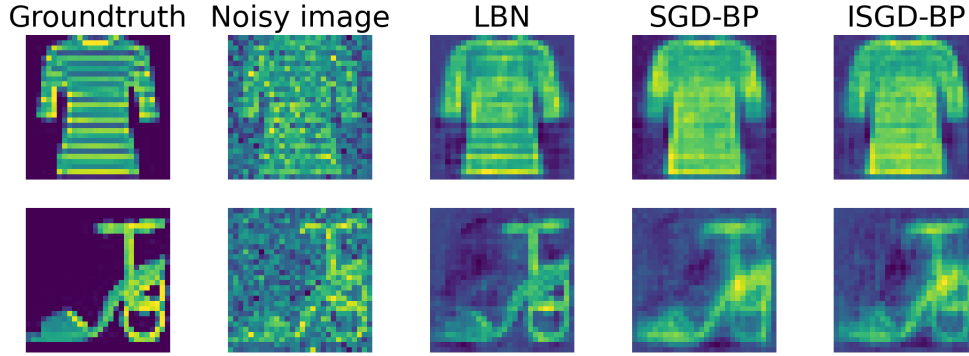


Fig. 4.11 Denoised images from the Fashion-MNIST-1K training dataset, computed with the LBN, the SGD-BP and the ISGD-BP training strategies for 50 epochs, along with the noise-free and noisy images.

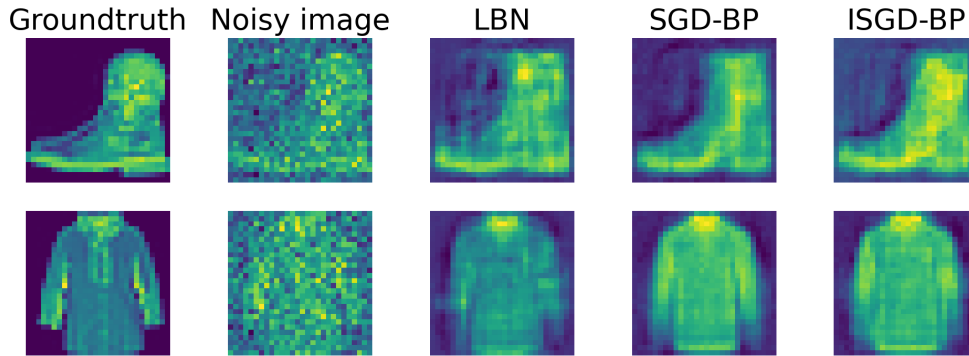


Fig. 4.12 Denoised images from the Fashion-MNIST-1K validation dataset computed with the LBN, the SGD-BP and the ISGD-BP training strategies for 50 epochs, along with the noise-free and noisy images.

The compression of the input data is achieved by ensuring that only relatively few coefficients of the code are non-zero. Hence, the advantage over conventional undercomplete autoencoders is that the position of the non-zero coefficients can vary for every input. We provide visualisations of the sparse code from a sample input image computed with the SGD-BP, ISGD-BP and LBN approach in Figure 4.13.

We also visualise a selection of denoised sample images from the Fashion-MNIST-1K training dataset in Figure 4.11 and from the Fashion-MNIST-10K training dataset in Figure 4.14. Although the loss values of LBN and ISGD-BP were comparable after 50

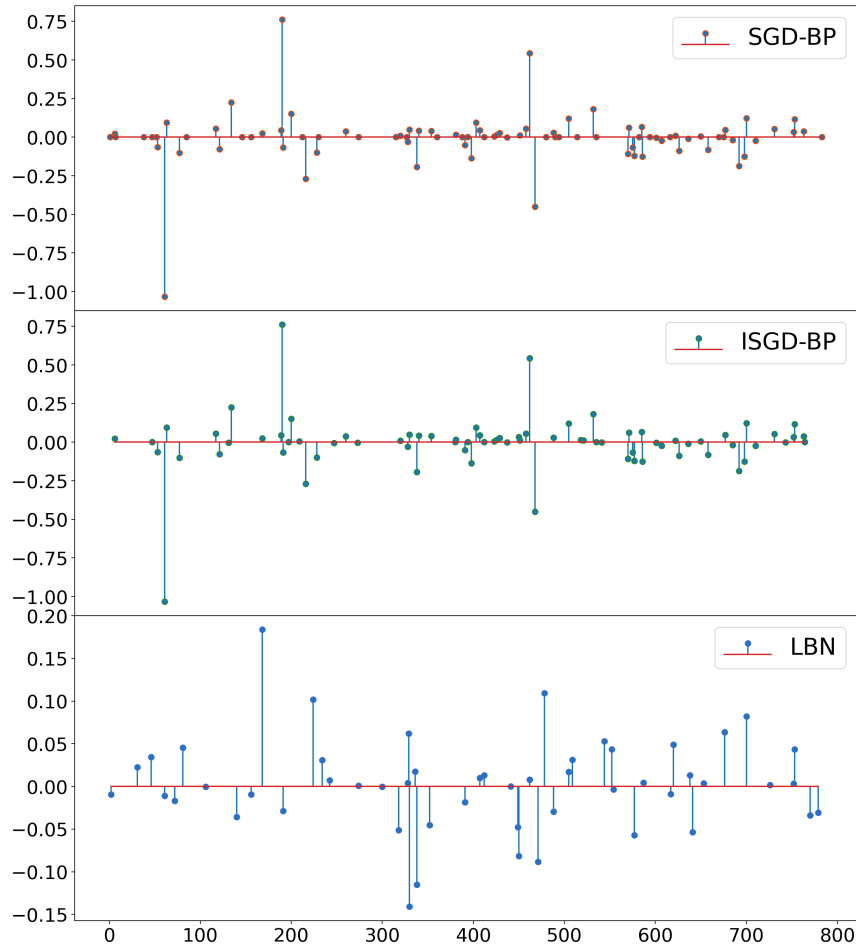


Fig. 4.13 Visualisation of the sparse coding of the sandal image object from Figure 4.14 computed with the SGD-BP, ISGD-BP, and LBN, respectively. Note that the scaling of the sparse coefficients is slightly different for LBN in comparison to SGD-BP and ISGD-BP.

epochs, the visual comparison between the network outputs suggests that the network trained with LBN is able to restore finer details (for instance the stripes on the t-shirt).

The same network also exhibits better denoising performance for the validation images as seen in Figure 4.12.

Similar observations can be made when we examine denoised sample images from the models trained on the Fashion-MNIST-10K dataset (Figure 4.14). The LBN trained network is capable of recovering fine structures such as the dotted backpack object. Both SGD-BP and ISGD-BP struggle with fine details and are prone to producing coarse approximations. The observations still hold true for the validation dataset as can be seen from Figure 4.15. For example, the LBN approach is able to recover the

curvature of the trousers, but SGD-BP and ISGD-BP only recover straight-legged trousers.

Additional visualisations of the sparse denoising autoencoder outputs can be found at the end of this chapter.

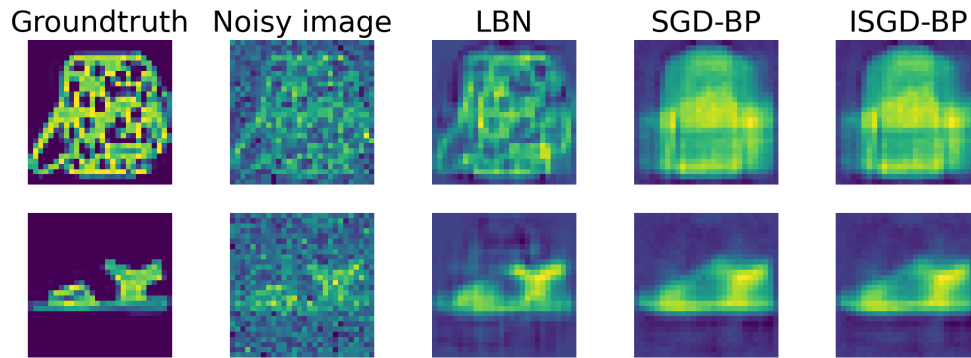


Fig. 4.14 Denoised images from the Fashion-MNIST-10K training dataset computed with the LBN, the SGD-BP (trained for 1500 epochs) and the ISGD-BP training strategies respectively, along with the noise-free and noisy images.

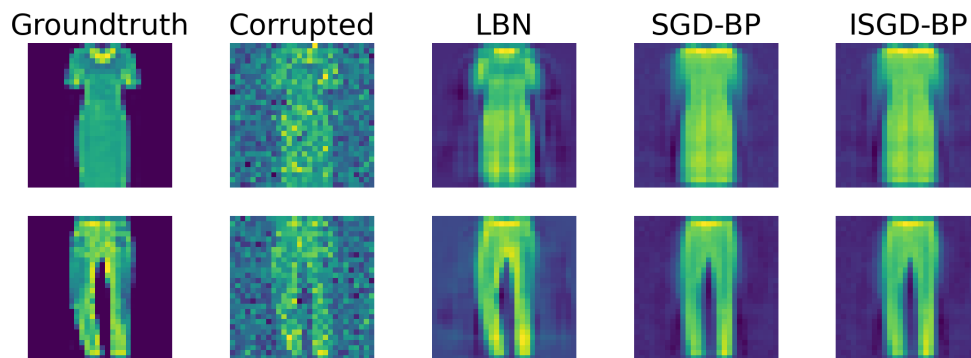


Fig. 4.15 Denoised images from the Fashion-MNIST-10K validation dataset computed with the LBN, the SGD-BP (trained for 1500 epochs) and the ISGD-BP training strategies respectively, along with the noise-free and noisy images.

## 7 Discussion

In this chapter, we discussed the Bregman framework for learning parameters of feedforward neural networks and residual neural networks. To achieve that, we have introduced a penalty function based on Bregman distances and we provided detailed mathematical analysis of this new penalty function. By replacing the quadratic penalties in the MAC-QP approach, we modified MAC-QP and lifted network frameworks as introduced in (Askari et al., 2018; Carreira-Perpinan and Wang, 2014; Gu et al., 2020; Li et al., 2019) and established rigorous mathematical properties. The advantage of the proposed framework is that it formulates the training of neural networks as a minimisation problem in which computing partial derivatives of the network parameters does not require differentiation of the activation functions. We have demonstrated that this framework can be realised computationally with a variety of different deterministic and stochastic minimisation algorithms.

With numerical experiments in classification and compression, we compared the implementation of the proposed framework to explicit and implicit stochastic gradient methods implemented with back-propagation.

- For classification, we have shown that the trained network is non-linear in contrast to the lifted network approach proposed in Askari et al. (2018), while also achieving similar classification accuracy as networks trained with first-order methods with back-propagation.
- For compression, we showed with our sparse autoencoder example that training parameters with proximal gradient descent or implicit stochastic gradient descent can be more efficient than first-order methods with back-propagation. We have also shown that it is straightforward to integrate regularisation acting on the activation variables. This enabled us to design an autoencoder with sparse codes that is shown to produce more adaptive compressions compared to conventional autoencoders.

This chapter established the lifted Bregman framework for solving forward-direction learning problems of neural networks. In the upcoming chapter, we will illustrate how this framework inspires a solution to the backward-direction learning problem, via the regularised inversion of neural networks.

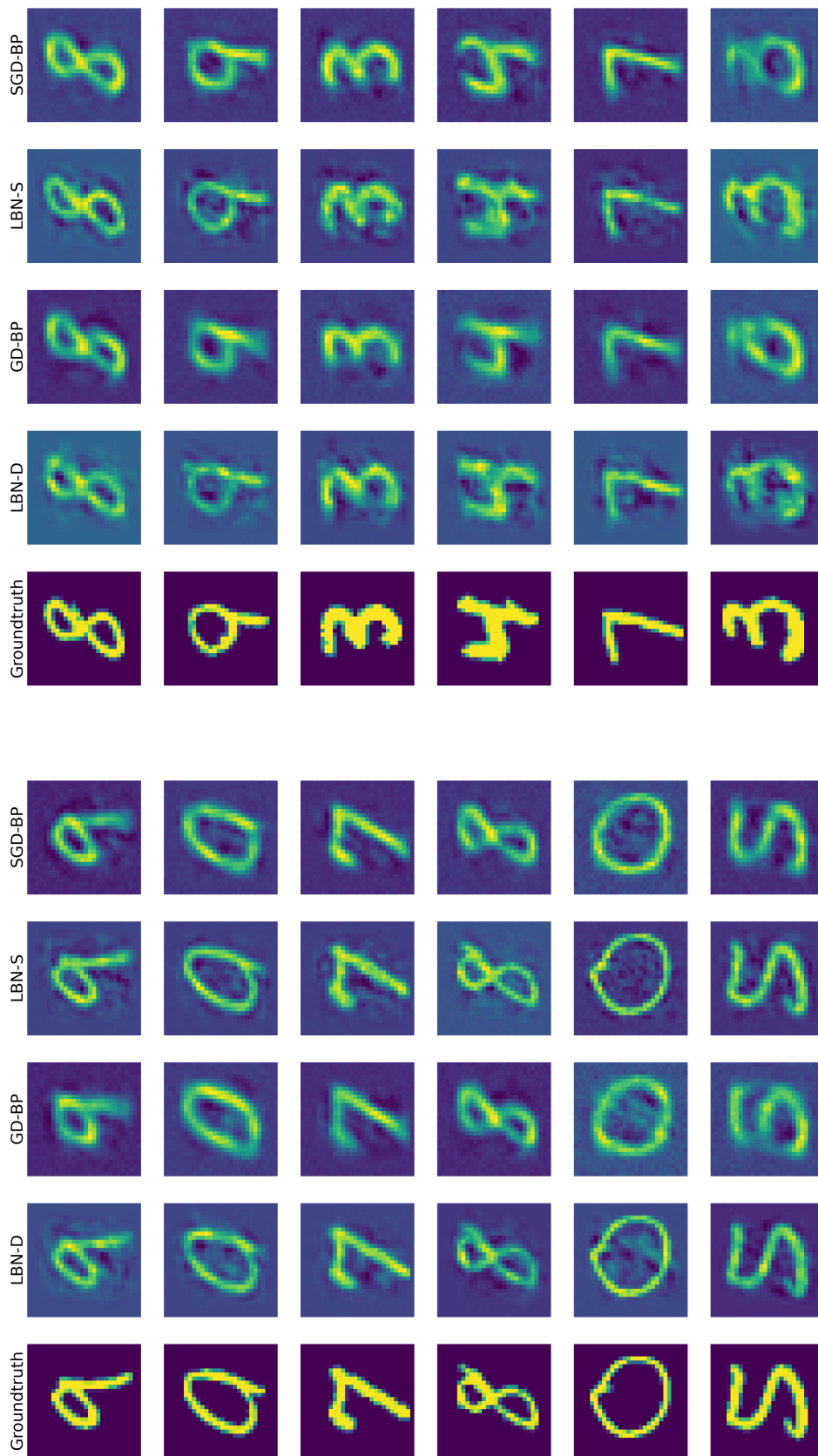


Fig. 4.16 **Left:** This plot shows selected denoised MNIST training images computed with LBN-D, GD-BP, LBN-S and SGD-BP. **Right:** This plot shows selected denoising results on the MNIST validation dataset.

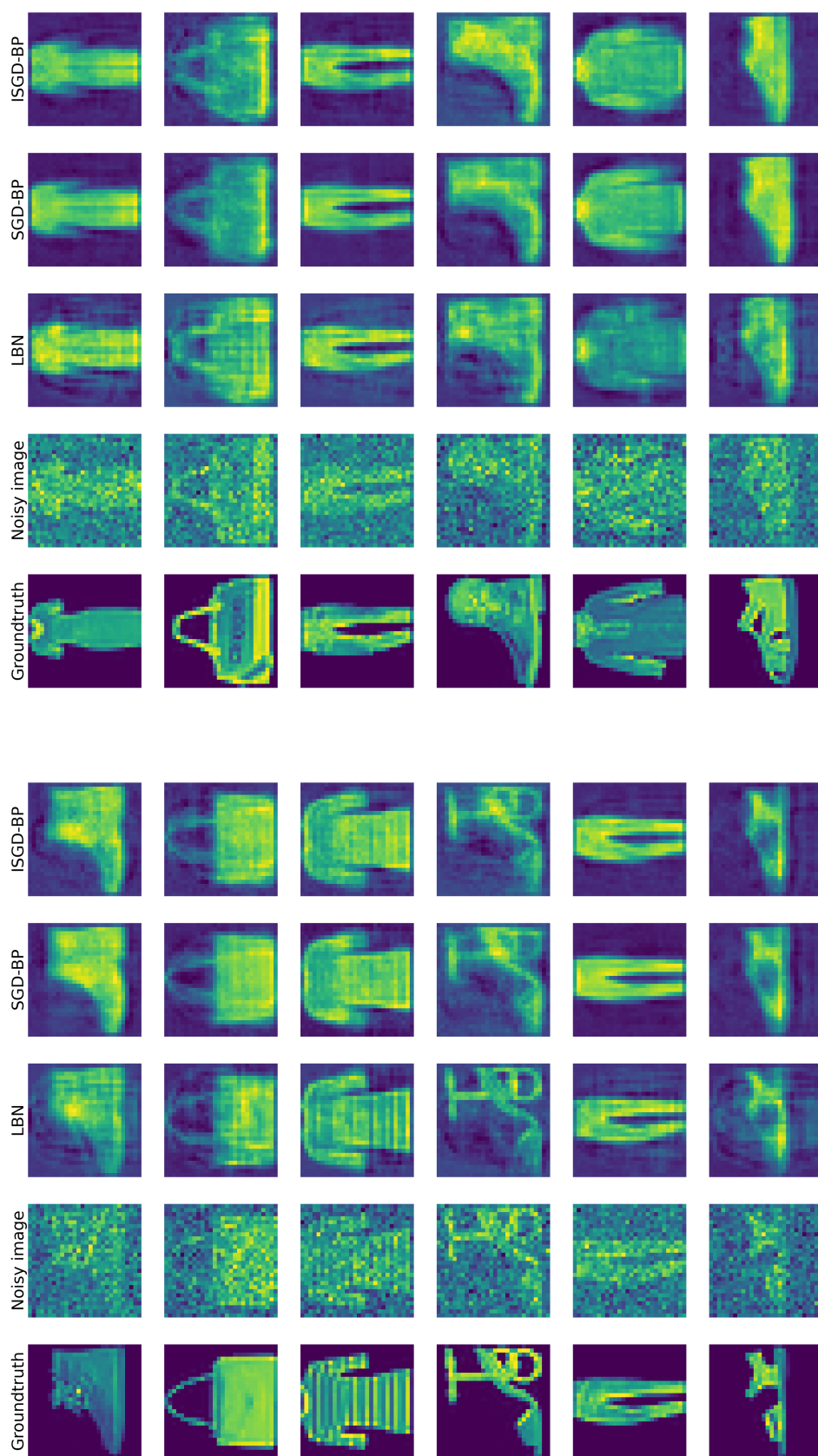


Fig. 4.17 **Left:** Denoised images from the Fashion-MNIST-1K training dataset computed with LBN, SGD-BP and ISGD-BP for 50 epochs, along with the noise-free and noisy images. **Right:** Denoised images from the Fashion-MNIST-1K validation dataset computed with LBN, SGD-BP and ISGD-BP for 50 epochs, along with the noise-free and noisy images.

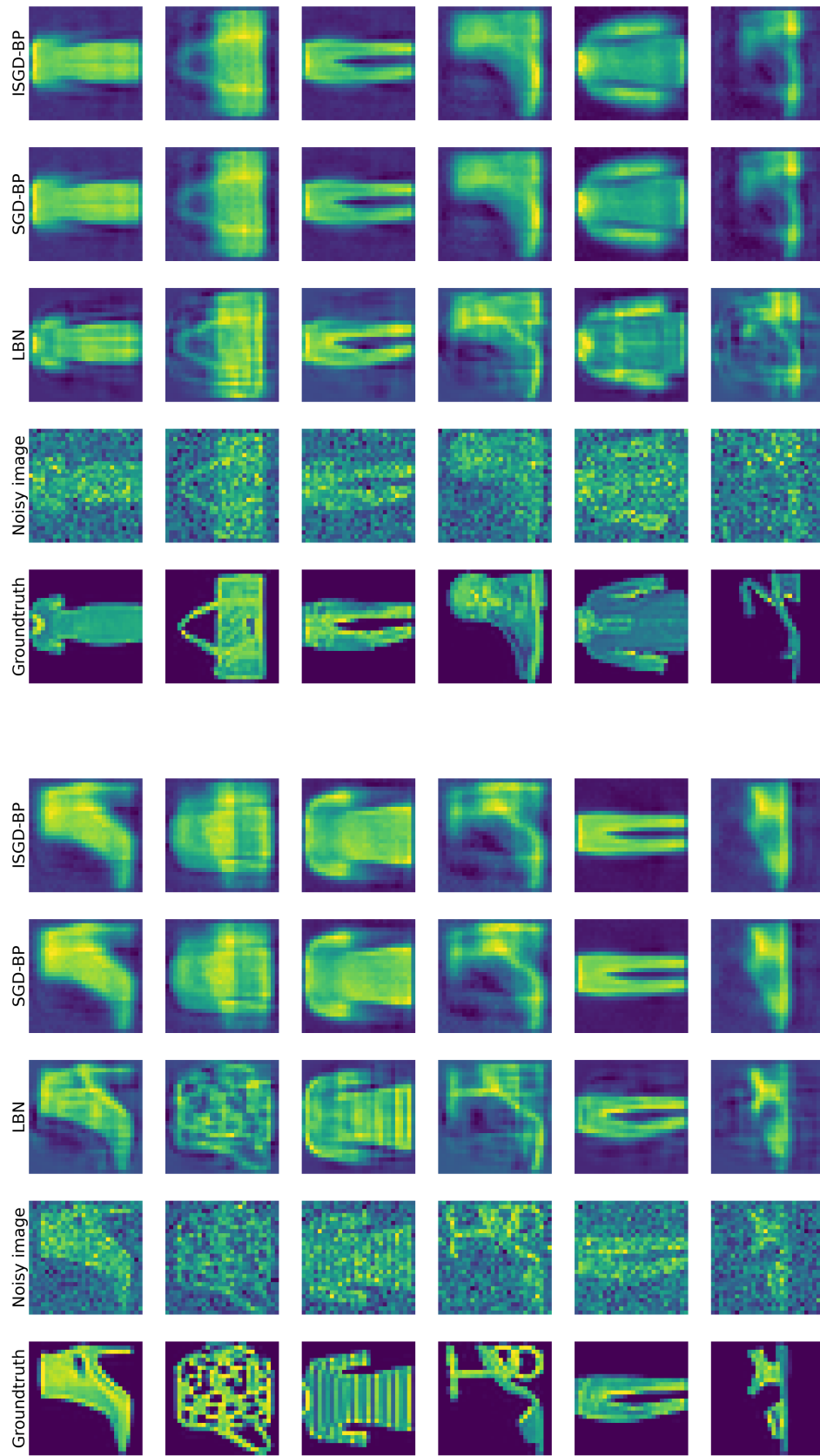


Fig. 4.18 **Left:** Denoised images from the Fashion-MNIST-10K training dataset computed with LBN, SGD-BP (trained for 1500 epochs) and ISGD-BP, along with the noise-free and noisy images. **Right:** Denoised images from the Fashion-MNIST-10K validation dataset computed with LBN, SGD-BP (trained for 1500 epochs) and ISGD-BP, along with the noise-free and noisy images.



## Chapter 5

# Lifted Bregman Inversion of Neural Networks

In the previous chapters, we have established the lifted Bregman framework for the training of neural networks without the differentiation of activation functions. The framework lifts the parameter space into a higher dimensional space by introducing auxiliary variables, and penalises these variables with tailored Bregman distances. Via the introduction of auxiliary variables, in addition to providing an alternative formulation for training neural networks, the proposed framework also opens up new pathways for explicit control and investigation on the network activation variables.

In this chapter, we present a family of variational regularisations based on the Bregman penalty function and introduce the lifted Bregman framework for the regularised inversion of deep neural networks. In particular, we will prove the convergence result for the regularised inversion of a single-layer perceptron as well as formulate computational approaches for solving the regularised inversion problem of multi-layer feedforward neural networks. The main contributions of this chapter are three-fold:

- We establish a lifted Bregman framework for the regularised inversion of multi-layer feedforward neural networks.
- We show that for the single-layer perceptron case, the proposed variational regularisation approach is a provably convergent regularisation without overly restrictive assumptions such as differentiability of the activation function and the presence of a tangential cone condition.

- We present proximal first-order optimisation strategies to solve the proposed variational regularisation method and demonstrate several numerical examples that support the effectiveness of the proposed model-based regularisation approach.

## Organisation of the Chapter

This chapter is structured as follows. In Section 1 we briefly review the inversion problem of feedforward neural networks as well as often neglected aspects of the inversion process. In Section 2 we introduce the lifted Bregman formulation for the model-based inversion of feedforward neural networks. In Section 3 we prove that for the single-layer perceptron case the proposed model is a convergent variational regularisation method and provide general error estimates as well as error estimates for a concrete example of a perceptron with ReLU activation function. In Section 4 we discuss how to implement the proposed variational regularisation computationally for both the single-layer perceptron and the multi-layer neural network setting. Subsequently, we present numerical results that demonstrate empirically that the proposed approach is a model-based regularisation in Section 5, before we conclude this chapter with a brief discussion in Section 6.

# 1 Introduction

Over the past decade, neural networks have emerged as groundbreaking computing systems, revolutionising various research domains and surpassing numerous conventional machine learning approaches (cf. (Goodfellow et al., 2016a; LeCun et al., 2015)). However, this performance comes at the cost of interpretability (or rather a lack thereof) of the outputs that a neural network produces for given inputs.

As a consequence, a lot of research has focused on understanding representations of neural networks and on developing strategies to interpret these representations, predominantly with saliency maps (Chang et al., 2018; Fong et al., 2019; Fong and Vedaldi, 2017; Simonyan et al., 2013). An alternative approach focuses on understanding deep image representations by inverting them (Mahendran and Vedaldi, 2015). The authors propose a total-variation-based variational optimisation method that aims to infer the network input from the network output with regularised inversion.

While the concept of inverting neural networks is certainly not new (cf. (Jensen et al., 1999; Kindermann and Linden, 1990; Linden and Kindermann, 1989; Lu et al., 1999)), there has been increasing interest in recent years largely due to developments in nonlinear dimensionality reduction and generative modelling that include (but are not limited to) (variational) Autoencoders (Kingma and Welling, 2013), Normalising Flows (Dinh et al., 2015; Rezende and Mohamed, 2015), Cycle-Consistent Generative Adversarial Networks (Zhu et al., 2017), and Probabilistic Diffusion Models (Ho et al., 2020; Sohl-Dickstein et al., 2015).

Several approaches for the inversion of neural networks have been proposed especially in the context of generative modelling (see for example (Behrmann et al., 2019, 2021) in the context of normalising flows, (Xia et al., 2022) in the context of generative adversarial networks and (Gal et al., 2022) in the context of probabilistic diffusion models), yet an important aspect, which is often overlooked, is that invertible operations alone are not automatically stable with respect to small variations in the data. For example, computing the solution of the heat equation after a fixed termination time is stable with respect to variations in the initial condition, but estimating the initial condition from the terminal condition of the heat equation is not stable with respect to perturbations in the terminal condition. This issue cannot be resolved without approximation of the inverse with a family of continuous operators, also known as regularisation. The research field of Inverse and Ill-posed Problems and its branch Regularisation Theory focus strongly on the stable approximation of ill-posed and ill-conditioned inverses via regularisations (Engl et al., 1996) and so-called variational regularisations (Benning and Burger, 2018; Scherzer et al., 2009) that are a special

class of (nonlinear) regularisations. The optimisation model proposed in (Mahendran and Vedaldi, 2015) can be considered as a variational regularisation method with total variation regularisation; however, the work in (Mahendran and Vedaldi, 2015) is purely empirical, and to the best of our knowledge no works exist that rigorously prove that the proposed approach is a variational regularisation.

In this chapter, we introduce a regularisation framework based on lifting with tailored Bregman distances and prove that the proposed framework is a convergent variational regularisation for the inverse problem of estimating the inputs from single-layer perceptrons or the inverse problem of estimating hidden variables in multi-layer neural networks sequentially. While there has been substantial work in previous years that focuses on utilising neural networks as nonlinear operators in variational regularisation methods (Arridge et al., 2019; Li et al., 2020; Lunz et al., 2018; Mukherjee et al., 2021; Schwab et al., 2019), our proposed variational regularisation framework bridges the gap by providing for the first time theoretical guarantees for the stable, model-based inversion of neural networks.

## 2 Model-based Inversion of Feedforward Neural Networks

Suppose we are given an  $L$ -layer feedforward neural network  $\mathcal{N} : \mathbb{R}^n \times \mathcal{P} \rightarrow \mathbb{R}^m$  of the form

$$\mathcal{N}(x, \Theta) = \sigma_L(f(\sigma_{L-1}(f(\dots \sigma_1(f(x, \Theta_1)) \dots)), \Theta_L)), \quad (5.1)$$

for input data  $x \in \mathbb{R}^n$  and pre-trained parameters  $\Theta \in \mathcal{P}$ . Here,  $\{\sigma_l\}_{l=1}^L$  denotes the collection of nonlinear activation functions and  $f$  denotes a generic function parametrised by parameters  $\{\Theta_l\}_{l=1}^L$ . For ease of notation, we use  $\Theta$  to refer to all parameters  $\{\Theta_l\}_{l=1}^L$ . For a given network output  $y \in \mathbb{R}^m$ , our goal is to solve the inverse problem

$$\mathcal{N}(x, \Theta) = y \quad (5.2)$$

for an unknown input  $x \in \mathbb{R}^n$ . The problem (5.2) is usually ill-posed in the sense that a solution may not exist (especially if  $m \ll n$ ) or is not unique (especially if  $n \ll m$ , or if information is lost through application of nonlinear activation functions). Moreover, even for a network with identity activation functions  $\sigma_l$  and affine linear

transformation  $f$ , solving (5.2) is often ill-conditioned in the sense that errors in  $y$  get heavily amplified when solving for  $x$ . We therefore propose to approximate the inverse of this nonlinear, potentially ill-posed inverse problem via the minimisation of a lifted Bregman formulation of the form

$$\begin{pmatrix} x^\alpha \\ x_1^\alpha \\ \vdots \\ x_{L-1}^\alpha \end{pmatrix} \in \arg \min_{x, x_1, \dots, x_{L-1}} \left\{ \sum_{l=1}^L B_{\Psi_l}(x_l, f(x_{l-1}, \Theta_l)) + \alpha R(x) \right\}, \quad (5.3)$$

where we assume  $x_0 = x$  and  $x_L = y^\delta$  for simplicity of notation. The data  $y^\delta$  is a perturbed version of  $y$ , for which we assume  $B_{\Psi_L}(y^\delta, f(x_{L-1}^\dagger, \Theta_L)) \leq \delta^2$ , for some constant  $\delta \geq 0$ . The functions  $B_{\Psi_l}$  for  $l = 1, \dots, L$  are defined as

$$B_{\Psi_l}(x, z) = \frac{1}{2}\|x\|^2 + \Psi_l(x) + \left( \frac{1}{2}\|\cdot\|^2 + \Psi_l \right)^*(z) - \langle x, z \rangle, \quad (5.4)$$

for a proper, convex and lower semi-continuous function  $\Psi_l: \mathbb{R}^{n_l} \rightarrow \mathbb{R} \cup \{\infty\}$ . The notation  $\left( \frac{1}{2}\|\cdot\|^2 + \Psi_l \right)^*$  refers to the convex or Fenchel conjugate of  $\frac{1}{2}\|\cdot\|^2 + \Psi_l$ , i.e.  $\left( \frac{1}{2}\|\cdot\|^2 + \Psi_l \right)^*(z) = \sup_y \langle z, y \rangle - \frac{1}{2}\|y\|^2 - \Psi_l(y)$ . Last but not least, the function  $R: \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$  is a proper, convex, and lower semi-continuous function that enables us to incorporate a-priori information into the inversion process. The impact of this is controlled by the parameter  $\alpha > 0$ .

Please note that the functions  $B_{\Psi_l}$  have some useful properties and are directly connected to the chosen activation functions  $\{\sigma_l\}_{l=1}^L$ . Following Chapter 4 Corollary 3.1, we observe

$$B_{\Psi_l}(x, z) \geq \frac{1}{2}\|\sigma_l(z) - x\|^2,$$

where  $\sigma_l: \mathbb{R}^{n_l} \rightarrow \mathbb{R}^{n_l}$  is a non-linear proximal activation function with respect to  $\Psi_l$ , for all  $l \in \{1, \dots, L\}$ .

The advantage of using functions  $B_{\Psi_l}$  over more conventional functions such as the squared Euclidean norm of the difference of the network output and the measured output, i.e.  $\frac{1}{2}\|\mathcal{N}(x, \Theta) - y\|^2$ , is that the functions  $B_{\Psi_l}$  are continuously differentiable with respect to their second argument (cf. Chapter 4 Theorem 3.1). If we define  $F_x^l(z) := B_{\Psi_l}(x, z)$ , we observe

$$\nabla F_x^l(z) = \sigma_l(z) - x. \quad (5.5)$$

For the remainder of this chapter, we assume that the parametrised functions  $f$  are affine-linear in the first argument, with parameters  $\Theta_l$ . A concrete example is the affine-linear transformation  $f(x, \Theta_l) = W_l x + b_l$ , for a (weight) matrix  $W_l \in \mathbb{R}^{n_l \times n_{l-1}}$ , a (bias) vector  $b_l \in \mathbb{R}^{n_l}$  and the collection of parameters  $\Theta_l = (W_l, b_l)$ .

In the next section, 1) we show that (5.3) is a variational regularisation method for  $L = 1$  and 2) we prove a convergence rate with which the solution of (5.3) converges towards the true input of a perceptron when  $\delta$  converges to zero.

### 3 Convergence Analysis and Error Estimates

In this section we show that the proposed model (5.3) is a convergent variational regularisation for the specific choice  $L = 1$  and the assumption  $f(x, \Theta) = Wx + b$  for  $\Theta = (W, b)$ , which reduces (5.3) to a variational regularisation model for the perceptron case. Let it be noted that here we are not interested in estimating the perceptron parameters  $W$  and  $b$  but assume that these are pre-trained hence fixed, and that we study the regularisation operator

$$\mathcal{R}_\alpha: \text{dom}(\Psi) \rightrightarrows \mathbb{R}^n, \quad \mathcal{R}_\alpha: y^\delta \rightrightarrows x_\alpha \in \arg \min_{x \in \mathbb{R}^n} \left\{ B_\Psi(y^\delta, Wx + b) + \alpha R(x) \right\}, \quad (5.6)$$

where  $\text{dom}(\Psi)$  is defined as  $\text{dom}(\Psi) := \{y \in \mathbb{R}^m \mid \Psi(y) < \infty\}$ . We first want to establish under which assumptions (5.6) is well-defined for all  $y^\delta$ .

#### 3.1 Well-definedness

For simplicity, we focus on the finite-dimensional setting with network inputs in  $\mathbb{R}^n$  and outputs in  $\text{dom}(\Psi)$ . However, the following analysis also extends to more general Banach space settings with additional assumptions on the operator  $W$ . We refer interested readers to (Benning and Burger, 2018, Section 5.1) for more detailed discussion. Recall from Chapter 2 Assumption 3.1, we will make the following assumptions with respect to a generalised perceptron:

- We assume that  $R$  is non-negative and the polar of a proper function, i.e.  $R = H^*$  for a proper function  $H: \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$ . Note that this automatically implies convexity of  $R$ .
- Moreover, we assume that  $\Psi$  is a proper, non-negative and convex function that is continuous on  $\text{dom}(\Psi)$ , which implies that  $B_\Psi$  is proper, non-negative, convex in

its second argument and continuous in its first argument for every  $y^\delta \in \text{dom}(\Psi)$ . Then, for every  $g \in \text{dom}(\Psi)$  there exists  $x$  with

$$B_\Psi(g, Wx + b) + \alpha R(x) < \infty.$$

- Last but not least, we assume that  $R$  and  $\Psi$  are chosen such that for each  $g \in \text{dom}(\Psi)$  and  $\alpha > 0$  we have

$$\|x\| \leq c(a, b, \|g\|), \quad \text{if } B_\Psi(g, Wx + b) \leq a \quad \text{and} \quad \alpha R(x) \leq d,$$

for constants  $a, d$  and a constant  $c$  that depends monotonically non-decreasing on all arguments.

With these assumptions, we can then verify the following lemma.

**Theorem 3.1.** *Let the assumptions outlined in the previous paragraph be satisfied.*

1. *Then, for every  $y \in \left\{ g \in \text{dom}(\Psi) \mid \arg \min_{x \in \mathbb{R}^n, R(x) < \infty} B_\Psi(g, Wx + b) \neq \emptyset \right\}$  the selection operator (cf. Chapter 2 Definition 3.2)*

$$\mathcal{S}(y) = \arg \min_{x \in \mathbb{R}^n} \left\{ R(x) \mid x \in \arg \min_{\tilde{x} \in \mathbb{R}^n} B_\Psi(y, W\tilde{x} + b) \right\}$$

*is well-defined.*

2. *The regularisation operator  $\mathcal{R}_\alpha$  as defined in (5.6) is well-defined in the sense that for every  $y \in \text{dom}(\Psi)$  there exists  $x_\alpha \in \mathbb{R}^n$  with  $x_\alpha \in \mathcal{R}_\alpha(y)$ . Moreover, the set  $\mathcal{R}_\alpha(y)$  is a convex set.*
3. *For every sequence  $y_n \rightarrow y \in \text{dom}(\Psi)$  there exists a subsequence  $x_{n_k} \in \mathcal{R}_\alpha(y_{n_k})$  converging to an element  $x^* \in \mathcal{R}_\alpha(y)$ .*

*Proof.* A more generic proof can be found in Chapter 2 Theorem 3.1, Theorem 3.2, and Theorem 3.3. The original statement only implies convergence in the weak-star topology; however, since here we consider a finite-dimensional Hilbert space, this automatically implies strong convergence here.  $\square$

### 3.2 Error Estimates

Having established that (5.3) is a regularisation operator, we now want to prove that it is a convergent regularisation operator in the sense of the estimate

$$D_R(x^\dagger, x^\alpha) \leq C\delta, \quad (5.7)$$

such that the regularisation method is  $D$ -convergent (cf. Chapter 2 Definition 3.4). The vector  $x^\alpha$  is a solution of (5.3) with data  $y^\delta$  for which we assume  $B_\Psi(y^\delta, Wx^\dagger + b) \leq \delta^2$ , and  $C \geq 0$  is a constant. The vector  $x^\dagger$  is an element of the selection operator as specified in Lemma 3.1.1, i.e.  $x^\dagger \in \mathcal{S}(y)$  for  $y \in \text{dom}(\Psi)$ . Note that  $x^\dagger \in \mathcal{S}(y)$  is equivalent to  $x^\dagger$  being a  $R$ -minimising vector amongst all vectors that satisfy  $0 = W^*(\sigma(Wx^\dagger + b) - y)$ , where  $\sigma$  denotes the proximal map with respect to  $\Psi$ . This is due to the fact that  $x^\dagger \in \arg \min_{\tilde{x} \in \mathbb{R}^n} B_\Psi(y, W\tilde{x} + b)$  is equivalent to  $0 = \nabla B_\Psi(y, Wx^\dagger + b) = W^*(\sigma(Wx^\dagger + b) - y)$ . Assuming that  $\sigma(Wx^\dagger + b) - y$  does not lie in the nullspace of  $W^*$ , this further implies  $y = \sigma(Wx^\dagger + b)$ .

In order to be able to derive error estimates of the form (5.7), we restrict ourselves to solutions  $x^\dagger$  that are in the range of  $\mathcal{R}_\alpha$  (cf. Chapter 2 Definition 3.5). This means that there exists  $y^\dagger$  such that  $x^\dagger \in \mathcal{R}_\alpha(y^\dagger)$ . Considering the optimality condition of (5.3) for  $y^\dagger$ , this implies

$$W^* \left( \frac{y^\dagger - \sigma(Wx^\dagger + b)}{\alpha} \right) \in \partial R(x^\dagger),$$

which for  $v^\dagger := (y^\dagger - \sigma(Wx^\dagger + b))/\alpha = (y^\dagger - y)/\alpha$  is equivalent to the existence of a source condition element  $v^\dagger$  that satisfies the source condition (cf. Chapter 2 Definition 3.6)

$$W^*v^\dagger \in \partial R(x^\dagger). \quad (\text{SC})$$

In the following, we verify that the symmetric Bregman distance (cf. Chapter 2 Definition 1.13) with respect to  $R$  between a solution of the regularisation operator and the solution of the inverse problem is converging to zero if the error in the data is converging to zero. As the symmetric Bregman distance or Jeffreys distance between two vectors  $x$  and  $\tilde{x}$  simply is the sum of two Bregman distances with interchanged arguments, hence an error estimate in the symmetric Bregman distance also implies an error estimate in the classical Bregman distance.

Before we begin our analysis, we recall the concept of the Jensen-Shannon divergence (Lin, 1991), which for general proper, convex and lower semi-continuous functions  $F : \mathbb{R}^n \rightarrow \mathbb{R} \cup \{\infty\}$  generalises to so-called Burbea-Rao divergences (cf. Chapter 2 Definition 1.14).

Another important concept that we need in order to establish error estimates is that of Fenchel conjugates (cf. Chapter 2 Definition 1.10). The Fenchel conjugate that is of particular interest to us is the conjugate of the function  $B_\Psi(y, z)$  with respect to the second argument, which we characterise with the following lemma.

**Lemma 3.1.** *The Fenchel conjugate of  $F_y(z) := B_\Psi(y, z)$  with respect to the second argument  $z$  reads*

$$F_y^*(w) = \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right) (y + w) - \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right) (y).$$

*Proof.* From the definition of the Fenchel conjugate we observe

$$\begin{aligned} F_y^*(w) &= \sup_{z \in \mathbb{R}^m} \langle z, w \rangle - F_y(z) \\ &= \sup_{z \in \mathbb{R}^m} \langle z, w \rangle - \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right) (y) - \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right)^* (z) + \langle y, z \rangle \\ &= - \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right) (y) + \sup_{z \in \mathbb{R}^m} \langle z, w + y \rangle - \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right)^* (z) \\ &= - \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right) (y) + \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right) (w + y), \end{aligned}$$

which concludes the proof.  $\square$

Having defined the Burbea-Rao divergence (cf. Chapter 2 Definition 1.14) and having established the Fenchel conjugate of  $B_\Psi(y, z)$  with respect to the second argument  $z$  for fixed  $y$ , we can now present and verify our main result on error estimation on the basis of Chapter 2 Theorem 3.4.

**Theorem 3.2.** *Suppose  $R$  and  $\Psi$  satisfy the assumptions outlined in Section 3.1. Then, for data  $y^\delta$  and  $y$  that satisfy  $B_\Psi(y^\delta, Wx^\dagger + b) \leq \delta^2$  with  $\delta \geq 0$ , a solution  $x^\alpha \in \mathcal{R}_\alpha(y^\delta)$  of the variational regularisation problem (5.3), and a solution  $x^\dagger$  of the perceptron problem  $y = \sigma(Wx^\dagger + b)$  that satisfies  $x^\dagger \in \mathcal{S}(y)$  and (SC), we observe the error estimate*

$$\begin{aligned} (1 - c)B_\Psi(y^\delta, Wx_\alpha + b) + \alpha D_R^{symm}(x_\alpha, x^\dagger) &\leq (1 + c)\delta^2 + \frac{\alpha^2}{c} \|v^\dagger\|^2 \\ &\quad + 2cJ_\Psi \left( y^\delta + \frac{\alpha}{c} v^\dagger, y^\delta - \frac{\alpha}{c} v^\dagger \right), \end{aligned} \quad (5.8)$$

for a constant  $c \in (0, 1]$  and  $J_\Psi$  is the Burbea-Rao divergence of  $\Psi$ .

*Proof.* Every solution  $x_\alpha$  that satisfies  $x_\alpha \in \mathcal{R}_\alpha(y^\delta)$  can equivalently be characterised by the optimality condition

$$W^* (\sigma(Wx_\alpha + b) - y^\delta) + \alpha p_\alpha = 0,$$

for some subgradient  $p_\alpha \in \partial R(x_\alpha)$ . Subtracting  $\alpha p^\dagger \in \partial R(x^\dagger)$  from both sides of the equation and taking a dual product with  $x_\alpha - x^\dagger$  then yields

$$\langle \sigma(Wx_\alpha + b) - y^\delta, Wx_\alpha - Wx^\dagger \rangle + \alpha D_R^{\text{symm}}(x_\alpha, x^\dagger) = -\alpha \langle p^\dagger, x_\alpha - x^\dagger \rangle. \quad (5.9)$$

We easily verify

$$D_{B_\Psi(y^\delta, W \cdot + b)}(x^\dagger, x_\alpha) = B_\Psi(y^\delta, Wx^\dagger + b) - B_\Psi(y^\delta, Wx_\alpha + b) - \langle \sigma(Wx_\alpha + b) - y^\delta, Wx^\dagger - Wx_\alpha \rangle;$$

hence, we can replace  $\langle \sigma(Wx_\alpha + b) - y^\delta, Wx_\alpha - Wx^\dagger \rangle$  with  $D_{B_\Psi(y^\delta, W \cdot + b)}(x^\dagger, x_\alpha) + B_\Psi(y^\delta, Wx_\alpha + b) - B_\Psi(y^\delta, Wx^\dagger + b)$  in (5.9) to obtain

$$D_{B_\Psi(y^\delta, W \cdot + b)}(x^\dagger, x_\alpha) + B_\Psi(y^\delta, Wx_\alpha + b) + \alpha D_R^{\text{symm}}(x_\alpha, x^\dagger) = B_\Psi(y^\delta, Wx^\dagger + b) - \alpha \langle p^\dagger, x_\alpha - x^\dagger \rangle.$$

We know  $0 \leq D_{B_\Psi(y^\delta, W \cdot + b)}(x^\dagger, x_\alpha)$  due to the convexity of  $B_\Psi(y^\delta, W \cdot + b)$ , and we also know that (SC) enables us to choose  $p^\dagger = W^* v^\dagger$ . Hence, we can estimate

$$B_\Psi(y^\delta, Wx_\alpha + b) + \alpha D_R^{\text{symm}}(x_\alpha, x^\dagger) \leq B_\Psi(y^\delta, Wx^\dagger + b) - \alpha \langle v^\dagger, Wx_\alpha - Wx^\dagger \rangle.$$

Next, we introduce the constant  $c \in (0, 1]$  to split the loss functions  $B_\Psi(y^\delta, Wx_\alpha + b)$  and  $B_\Psi(y^\delta, Wx^\dagger + b)$  into  $(1-c)B_\Psi(y^\delta, Wx_\alpha + b) + cB_\Psi(y^\delta, Wx_\alpha + b)$  and  $(1+c)B_\Psi(y^\delta, Wx^\dagger + b) - cB_\Psi(y^\delta, Wx^\dagger + b)$ , respectively. This means we estimate

$$\begin{aligned} (1-c)B_\Psi(y^\delta, Wx_\alpha + b) + \alpha D_R^{\text{symm}}(x_\alpha, x^\dagger) &\leq (1+c)B_\Psi(y^\delta, Wx^\dagger + b) \\ &\quad + \langle \alpha v^\dagger, Wx^\dagger + b \rangle - cB_\Psi(y^\delta, Wx^\dagger + b) \\ &\quad - \langle \alpha v^\dagger, Wx_\alpha + b \rangle - cB_\Psi(y^\delta, Wx_\alpha + b). \end{aligned}$$

Next, we make use of Lemma 3.1 to estimate

$$\langle \alpha v^\dagger, Wx^\dagger + b \rangle - cB_\Psi(y^\delta, Wx^\dagger + b) \leq c \left( \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right) \left( y^\delta + \frac{\alpha}{c} v^\dagger \right) - \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right) (y^\delta) \right),$$

and

$$-\langle \alpha v^\dagger, Wx_\alpha + b \rangle - cB_\Psi(y^\delta, Wx_\alpha + b) \leq c \left( \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right) \left( y^\delta - \frac{\alpha}{c} v^\dagger \right) - \left( \frac{1}{2} \|\cdot\|^2 + \Psi \right) (y^\delta) \right).$$

Adding both estimates together yields

$$\begin{aligned} & \langle \alpha v^\dagger, Wx^\dagger + b \rangle - cB_\Psi(y^\delta, Wx^\dagger + b) - \langle \alpha v^\dagger, Wx_\alpha + b \rangle - cB_\Psi(y^\delta, Wx_\alpha + b), \\ & \leq \frac{\alpha^2}{c} \|v^\dagger\|^2 + c \left( \Psi \left( y^\delta + \frac{\alpha}{c} v^\dagger \right) + \Psi \left( y^\delta - \frac{\alpha}{c} v^\dagger \right) - 2\Psi(y^\delta) \right), \\ & = \frac{\alpha^2}{c} \|v^\dagger\|^2 + 2cJ_\Psi \left( y^\delta + \frac{\alpha}{c} v^\dagger, y^\delta - \frac{\alpha}{c} v^\dagger \right), \end{aligned}$$

which together with the error bound  $B_\Psi(y^\delta, Wx_\alpha + b) \leq \delta^2$  concludes the proof.  $\square$

**Remark 3.1.** We want to emphasise that for continuous  $\Psi$  and  $c > 0$  we automatically observe

$$\lim_{\alpha \rightarrow 0} J_\Psi \left( y^\delta + \frac{\alpha}{c} v^\dagger, y^\delta - \frac{\alpha}{c} v^\dagger \right) = 0,$$

in which case the important question from an error estimate point-of-view is if the term converges quicker to zero than  $\alpha$ , as we would need to guarantee

$$\lim_{\alpha \rightarrow 0} J_\Psi \left( y^\delta + \frac{\alpha}{c} v^\dagger, y^\delta - \frac{\alpha}{c} v^\dagger \right) / \alpha = 0$$

in order to guarantee that the symmetric Bregman distance in (5.8) converges to zero for  $\alpha \rightarrow 0$ .

**Example 3.1** (ReLU perceptron). Let us consider a concrete example to demonstrate that (5.6) is a convergent regularisation with respect to the symmetric Bregman distance of  $R$ . We know that for  $\sigma(z) = \text{prox}_\Psi(z) = \max(0, z)$  to hold true we have to choose

$$\Psi(z) = \begin{cases} 0 & z \in [0, \infty)^m \\ \infty & \text{else} \end{cases}.$$

This means that for  $B_\Psi(y^\delta, z)$  to be well-defined for any  $z$  we require  $y_i^\delta \geq 0$  for all  $i \in \{1, \dots, m\}$ . In order for the Burbea-Rao divergence to be well-defined, we further require

$$-\frac{c}{\alpha} y_i^\delta \leq v_i^\dagger \leq \frac{c}{\alpha} y_i^\delta,$$

for all  $i \in \{1, \dots, m\}$ , or  $\|v^\dagger\|_\infty \leq (c\|y^\delta\|_\infty/\alpha)$  in more compact notation. If  $\|v^\dagger\|_\infty \leq (c\|y^\delta\|_\infty/\alpha)$  is guaranteed, we observe  $J_\Psi(y^\delta + \frac{\alpha}{c}v^\dagger, y^\delta - \frac{\alpha}{c}v^\dagger) = 0$ . Hence, we can simplify the estimate (5.8) to

$$\frac{1-c}{\alpha}B_\Psi(y^\delta, Wx_\alpha + b) + D_R^{symm}(x_\alpha, x^\dagger) \leq \frac{1+c}{\alpha}\delta^2 + \frac{\alpha}{c}\|v^\dagger\|^2,$$

where we have also divided by  $\alpha$  on both sides of the inequality. If we choose  $\alpha(\delta) = \sqrt{c(1+c)}\delta/\|v^\dagger\|$ , we obtain the estimate

$$\frac{(1-c)\|v^\dagger\|}{\delta\sqrt{c(1+c)}}B_\Psi(y^\delta, Wx_{\alpha(\delta)} + b) + D_R^{symm}(x_{\alpha(\delta)}, x^\dagger) \leq 2\sqrt{\frac{1+c}{c}}\|v^\dagger\|\delta,$$

as long as we can ensure

$$\left\| \frac{v^\dagger}{\|v^\dagger\|} \right\|_\infty \leq \sqrt{\frac{c}{1+c}} \left\| \frac{y^\delta}{\delta} \right\|_\infty.$$

Together with  $D_R^{symm}(x_{\alpha(\delta)}, x^\dagger) \geq D_R(x^\dagger, x_{\alpha(\delta)})$  we have established an estimate of the form (5.7), with constant  $C = 2\sqrt{\frac{1+c}{c}}\|v^\dagger\|$ . Hence, we have verified that the variational regularisation method (5.3) is not only a regularisation method but even a convergent regularisation method in this specific example.

We want to briefly comment on the extension of the convergence analysis to the general case  $L > 1$  with the following remark.

**Remark 3.2.** *The presented convergence analysis easily extends to a sequential, layer-wise inversion approach. Suppose we have  $L$  layers and begin with the final layer, then we can formulate the variational problem*

$$x_{L-1}^\alpha \in \arg \min_{x_{L-1}} \left\{ B_{\Psi_L}(y^\delta, W_L x_{L-1} + b_L) + \alpha_{L-1} \Psi_{L-1}(x_{L-1}) \right\},$$

which is also of the form of (5.6), but where  $R$  has been replaced with  $\Psi_{L-1}$ . Alternatively, one can also replace  $\Psi_{L-1}$  with another function  $R_{L-1}$  if good prior knowledge for the auxiliary variable  $x_{L-1}$  exists. Once we have estimated  $x_{L-1}^\alpha$ , we can recursively estimate

$$x_{l-1}^\alpha \in \arg \min_{x_{l-1}} \{ B_{\Psi_l}(x_l^\alpha, W_l x_{l-1} + b_l) + \alpha_{l-1} \Psi_{l-1}(x_{l-1}) \},$$

for  $l = L - 1, \dots, 2$  and subsequently compute  $x_\alpha$  as a solution of (5.3) but with data  $x_1^\alpha$  instead of  $y^\delta$ .

The advantage of such a sequential approach is that every individual regularisation problem is convex and the previously presented theorems and guarantees still apply. The disadvantage is that for this approach to work in theory, we require bounds for every auxiliary variable of the form  $B_{\Psi_l}(x_l^\alpha, W_l x_{l-1}^\alpha + b_l) \leq \delta_l^2$ , which is a rather unrealistic assumption. Moreover, it is also not realistic to assume that good prior knowledge for the auxiliary variables exist.

Please note that showing that the simultaneous approach (5.3) is a (convergent) variational regularisation is beyond the scope of this chapter as it is harder and potentially requires additional assumptions for the following reason. The overall objective function in (5.3) is no longer guaranteed to be convex with respect to all variables simultaneously, which means that we cannot simply carry over the analysis of the single-layer to the multi-layer perceptron case.

**Remark 3.3** (Infinite-dimensional setting). Please note that almost all theoretical results presented in this section also apply to neural networks that map functions between Banach spaces instead of finite-dimensional vectors. The only result that changes is Theorem 3.1, Item 3, where the statement in an infinite-dimensional setting only implies convergence in the weak-star topology.

This concludes the theoretical analysis of the perceptron inversion model. In the following section we focus on how to implement (5.6) and its more general counterpart (5.3).

## 4 Computational Implementation

In this section, we describe how to computationally implement the proposed variational regularisation for both the single-layer perceptron and the multi-layer neural network setting. More specifically, we show that the proposed variational regularisation can be efficiently solved via a generalised primal-dual hybrid gradient method (cf. Chapter 2 Algorithm 9) and a coordinate descent approach (cf. Chapter 2 Algorithm 5).

### 4.1 Inverting Perceptrons

To begin with, we first consider the example of inverting a (single-layer) perceptron. For  $L = 1$ , Problem (5.3) reduces to (5.6), which for a composite regularisation function

$R \circ K$  reads

$$x^\alpha \in \arg \min_x \left\{ B_\Psi(y^\delta, f(x, \Theta)) + \alpha R(Kx) \right\}. \quad (5.10)$$

Here  $K$  is a matrix and  $\alpha R(Kx)$  denotes the regularisation function acting on the argument  $x$ . The above Problem (5.10) can be reformulated to the saddle-point problem

$$\min_x \max_z B_\Psi(y^\delta, f(x, \Theta)) + \langle z, Kx \rangle - \alpha R^*(z), \quad (5.11)$$

where  $R^*$  denotes the convex conjugate of  $R$ . Computationally, we can then solve the saddle-point problem with a generalised version (Chambolle and Pock, 2016a) of the popular primal-dual hybrid gradient (PDHG) method:

$$x^{k+1} = x^k - \tau_x \left( \left( \text{prox}_\Psi(f(x^k, \Theta)) - y^\delta \right) \mathcal{J}_f^x(x^k, \Theta) + \alpha K^\top z^k \right) \quad (5.12a)$$

$$z^{k+1} = \text{prox}_{\tau_z R^*} \left( z^k + \tau_z \alpha K \left( 2x^{k+1} - x^k \right) \right). \quad (5.12b)$$

where  $\mathcal{J}_f^x$  denotes the Jacobian of  $f$  with respect to argument  $x$  and we alternate between a descent step in the  $x$  variable and an ascent step in the dual variable  $z$ . Since (5.10) is a convex minimisation problem, (5.12) is guaranteed to converge globally for arbitrary starting point, given that  $\tau_x$  and  $\tau_z$  are chosen such that  $\tau_x \tau_z < 1/\|K\|^2$  and such that (5.12a) is contractive (Chambolle and Pock, 2016a, Theorem 5.1).

In this section, we will focus on the discrete total variation  $\|\nabla x\|_{p,1}$  (cf. Chapter 2 Definition 2.53), as our regularisation function  $R(Kx)$ , but other choices are certainly possible. If we consider a two-dimensional scalar-valued image  $x \in \mathbb{R}^{H \times W}$ , using the finite forward difference discretisation of the gradient operator  $\nabla : \mathbb{R}^{H \times W} \rightarrow \mathbb{R}^{H \times W \times 2}$ , the discrete total variation is defined as the  $\ell_1$  norm of the  $p$ -norm of the pixel-wise image gradients, i.e.

$$\|\nabla x\|_{p,1} = \sum_{i=1}^H \sum_{j=1}^W |(\nabla x)_{i,j}|_p = \sum_{i=1}^H \sum_{j=1}^W \left( (\nabla x)_{i,j,1}^p + (\nabla x)_{i,j,2}^p \right)^{1/p}.$$

For our numerical results we consider the isotropic total variation and consequently choose  $p = 2$ . Hence for a perceptron with affine-linear transformation  $f(x, \Theta) = Wx + b$ , and with  $\sigma = \text{prox}$  denoting the activation function, the PDHG approach (5.12) of

solving the perceptron inversion problem (5.6) can be summarised as

$$x^{k+1} = x^k - \tau_x \left( W^\top \left( \sigma(Wx^k + b) - y \right) - \alpha \operatorname{div} z^k \right), \quad (5.13a)$$

$$z^{k+1} = \operatorname{prox}_{\tau_z \|\cdot\|_{2,1}^*} \left( z^k + \tau_z \left( 2\alpha \nabla x^{k+1} - \alpha \nabla x^k \right) \right). \quad (5.13b)$$

Please note that we define the discrete approximation of the divergence  $\operatorname{div}$  such that it satisfies  $\operatorname{div} = -\nabla^\top$  in order to be the negative transpose of the discretised finite difference approximation of the gradient in analogy to the continuous case, which is why the sign in (5.13a) is flipped in comparison to (5.12a). The proximal map with regards to the convex conjugate of  $\|\cdot\|_{2,1}^*$  is simply the argument itself if the maximum of the Euclidean vector-norm per pixel is bounded by one or the projection onto this unit ball.

## 4.2 Inverting Feedforward Neural Networks

We now discuss the implementation of the inversion of multi-layer perceptrons with  $L$  layers as described in (5.3). Note that in this case in order to minimise for  $x$ , we also need to optimise with respect to the auxiliary variables  $x_1, \dots, x_{L-1}$ .

For the minimisation of (5.3) we consider an alternating minimisation approach, also known as coordinate descent (Beck and Tetrushvili, 2013; Wright, 2015; Wright and Recht, 2022). In this approach we minimise the objective with respect to one variable at a time. In particular, we focus on a semi-explicit coordinate descent algorithm, where we linearise with respect to the smooth functions of the overall objective function. This breaks down the overall minimisation problem into  $L$  sub-problems, where for  $x_0$  and each  $x_l$  variable for  $l \in \{1, \dots, L-1\}$ , we have individual minimisation problems of the following form:

$$x_0^{t+1} = \arg \min_{x_0} \left\{ \left( \frac{1}{2} \|\cdot\|^2 + \Psi_1 \right)^* (f(x_0, \Theta_1)) - \langle x_1^t, f(x_0, \Theta_1) \rangle + \alpha R(Kx_0) \right\}, \quad (5.14a)$$

$$x_l^{t+1} = \arg \min_{x_l} \left\{ \left( \frac{1}{2} \|\cdot\|^2 + \Psi_l \right) (x_l) - \langle x_l, f(x_{l-1}^{t+1}, \Theta_l) \rangle + \frac{1}{2\tau_{x_l}} \|x_l - x_l^t\|^2 \right. \\ \left. + \langle x_l, \left( \operatorname{prox}_{\Psi_{l+1}} \left( f(x_l^t, \Theta_{l+1}) \right) - x_{l+1}^t \right) \mathcal{J}_f^x(x_l^t, \Theta_{l+1}) \rangle \right\}. \quad (5.14b)$$

Note that one advantage for adopting this approach is that we exploit that the overall objective function is convex in each individual variable when all other variables are

kept fixed. In the following, we will discuss different strategies to computationally solve each sub-problem.

When optimising with respect to the input variable  $x_0$ , the structure of sub-problem (5.14a) is identical to the perceptron inversion problem that we have discussed in Section 4.1. Hence, we can approximate  $x_0^{t+1}$  with (5.11), but now with respect to  $x_1^t$  instead of  $y^\delta$ , which yields the iteration

$$x_0^{k+1} = x_0^k - \tau_{x_0} \left( \left( \text{prox}_{\Psi} \left( f(x_0^k, \Theta_1) \right) - x_1^t \right) \mathcal{J}_f^x(x_0^k, \Theta_1) + \alpha K^\top z^k \right), \quad (5.15a)$$

$$z^{k+1} = \text{prox}_{\tau_z R^*} \left( z^k + \tau_z \alpha K \left( 2x_0^{k+1} - x_0^k \right) \right). \quad (5.15b)$$

For each auxiliary variable  $x_l$  with  $l \in \{1, \dots, L-1\}$ , the sub-problem associated with (5.14b) amounts to solving a proximal gradient step with suitable step-size  $\tau_{x_l}$ , which we can rewrite to

$$x_l^{t+1} = \text{prox}_{\frac{\tau_{x_l}}{1+\tau_{x_l}} \Psi_l} \left( \frac{1}{1+\tau_{x_l}} \left( x_l^t - \tau_{x_l} \left( \left( \text{prox}_{\Psi_l} \left( f(x_l^t, \Theta_{l+1}) \right) - x_{l+1}^t \right) \mathcal{J}_f^x(x_l^t, \Theta_{l+1}) - f(x_{l-1}^{t+1}, \Theta_l) \right) \right) \right). \quad (5.16)$$

This concludes the discussion on the implementation of the regularised single-layer and multi-layer perceptron inversion. In the next section, we present some numerical results to demonstrate the effectiveness of the proposed approaches empirically.

## 5 Numerical Results

In this section, we present numerical results for the perceptron inversion problem implemented with the PDHG algorithm as outlined in (5.13), and for the multi-layer perceptron inversion problem implemented with the coordinate descent approach as described in (5.15) and (5.16). In the following, we first demonstrate that we can invert a perceptron with random weights and bias terms and ReLU activation function via (5.10) with total variation regularisation and the algorithm described in (5.13). We then proceed to a more realistic example of inverting the code of a simple autoencoder with perceptron encoder, before we extend the results to the total variation-based inversion of encodings from multi-layer convolutional autoencoders. All results have been computed using PyTorch 3.7 on an Intel Xeon CPU E5-2630 v4.

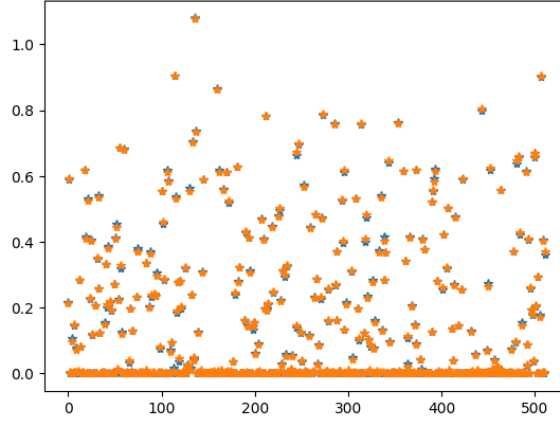


Fig. 5.1 A visual comparison between the groundtruth  $y$  (blue stars) and the noisy data  $y^\delta$  (orange stars).

## 5.1 Perceptrons

We present results for two experiments: the first one is the perceptron inversion of the image of a circle from the noisy output of the perceptron, where we compare the Landweber iteration regularisation and the total-variation-based variational regularisation (5.6). For the second experiment, we perform perceptron inversion for samples from the MNIST dataset (LeCun et al., 1998), where we compare the performance of the proposed inversion strategy with the performance of linear and nonlinear decoders on the collection of (approximately) piecewise constant images of hand-written digits.

**Image of a Circle** We begin with the toy example of recovering the image of a circle from noisy measurements of a ReLU perceptron. To prepare the experiment, we generate a circle image  $x^\dagger \in \mathbb{R}^{64 \times 64}$ , as shown in Figure 5.2. We construct a perceptron with ReLU activation function using random weights and biases where  $W \in \mathbb{R}^{512 \times 4096}$ ,  $b \in \mathbb{R}^{512 \times 1}$ . The weights operates on the column-vector representation of  $x$ , where  $x \in \mathbb{R}^{4096 \times 1}$ . The noise-free data is generated via the forward operation of the model, i.e.  $y = \sigma(Wx^\dagger + b)$ . We generate noisy data  $y^\delta$  by adding Gaussian noise with mean 0 and standard deviation 0.005. Note that we clip all the negative values of  $y^\delta$  to ensure  $y^\delta \in \text{dom}(\Psi)$ . Figure 5.1 shows an illustration where we plot both the groundtruth data  $y$  and the noisy data  $y^\delta$ .

A first attempt to solve this ill-posed perceptron inversion problem is via Landweber iteration regularisation (Landweber, 1951). In Figure 5.3 we see the reconstructed image obtained with Landweber iteration regularisation in combination with early stopping following Morozov's discrepancy principle Engl et al. (1996); Morozov (2012).

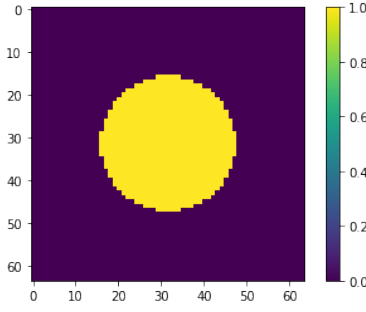


Fig. 5.2 Groundtruth image  $x^\dagger$  of a circle.

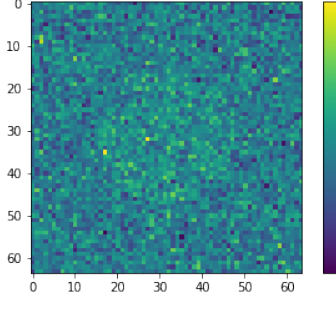


Fig. 5.3 Reconstructed image via Landweber iterative regularisation.

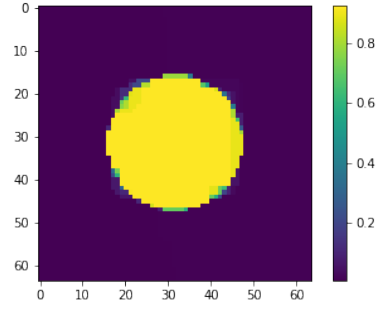


Fig. 5.4 Reconstructed image via TV regularisation.

Even though the Landweber iteration regularised reconstruction matches the data up to the discrepancy value  $\|\sigma(Wx^K + b) - y^\delta\|$ , the recovered image does not resemble the image  $x^\dagger$ . We will discuss shortly the reason for this visually poor inversion. In comparison, we see a regularised inversion via the total variation regularisation approach following (5.13) in Figure 5.4. The regularisation parameter for this reconstruction is chosen as  $\alpha = 1.5 \times 10^{-2}$ . Both  $x_0$  and  $z$  are initialised with zero vectors. The stepsize-parameters are chosen as  $\tau_x = 1.99/\|W\|_2^2$  and  $\tau_z = 1/(8\alpha)$ , where the operator norm of the discrete gradient operator is estimated to be  $\|\nabla\| \leq \sqrt{8}$  (see Chambolle (2004)). We stop the iterations when changes in  $x_0$  and  $z$  in norm are less than a threshold of  $10^{-5}$  or when we reach the maximum number of iterations, which we set to 10000. As shown in Figure 5.4, the TV-regularisation approach is capable of finding a (visually) more meaningful solution.

To explain why the Landweber iteration regularisation performs worse compared to the total variation regularisation for this specific example, we compare the  $\ell_2$  norms of each two solutions and the groundtruth image  $x^\dagger$ . The  $\ell_2$  norm of the Landweber solution in Figure 5.3 measures 6.58 while the TV-regularised solution as in Figure 5.4 and the groundtruth image  $x^\dagger$  measure 25.69 and 28.07 respectively. This is not surprising, as the Landweber iteration is known to converge to a minimal Euclidean norm solution if the noise level converges to zero. On the other hand, when we compare the TV semi-norm of each solution, the groundtruth image in measures 128.0, while the Landweber solution in Figure 5.3 and TV-regularised solution in Figure 5.4 measure 707.02 and 114.93 respectively, suggesting that the TV-semi-norm is a more suitable regularisation function for the inversion of cartoon-like images such as  $x^\dagger$ .

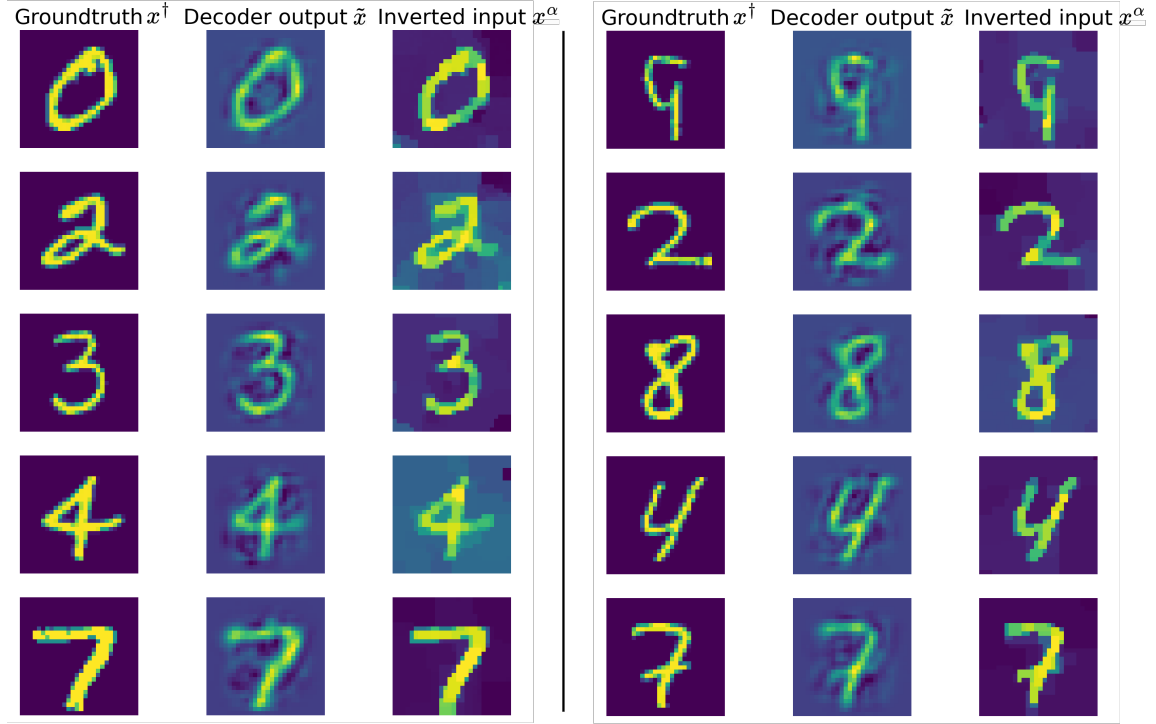


Fig. 5.5 Groundtruth input images from the MNIST training dataset (**Left**) and validation dataset (**Right**), together with the corresponding autoencoder output images  $\tilde{x}$  and inversions  $x^\alpha$  of the encoding via (5.10).

**Images from MNIST dataset** In this second example, we perform perceptron inversion on the MNIST dataset (LeCun et al., 1998). In particular, we consider the following experimental setup. We first train an autoencoder  $\mathcal{A}(x) = \mathcal{D}(\mathcal{E}(x, \Theta_{\mathcal{E}}), \Theta_{\mathcal{D}})$ , where  $\mathcal{D}(\cdot, \Theta_{\mathcal{D}})$  and  $\mathcal{E}(\cdot, \Theta_{\mathcal{E}})$  denotes the decoder and the encoder, parametrised by parameters  $\Theta_{\mathcal{D}}$  and  $\Theta_{\mathcal{E}}$  respectively. We pre-train the autoencoder  $\mathcal{A}$ , compute the code  $\mathcal{E}(x, \Theta_{\mathcal{E}})$  and assign it to the noise-free data variable  $y$ , and solve the inverse problem for the input  $x$  from the perturbed code  $y^\delta$

$$\mathcal{E}(x, \Theta_{\mathcal{E}}) = y^\delta.$$

To be more precise, we first train a two-layer fully connected autoencoder  $\tilde{x} = W_2(\sigma(W_1x + b_1)) + b_2$  using the vanilla stochastic gradient method (SGM) by minimising the mean squared error (MSE) on the MNIST training dataset. We set the code dimension to 100 and use ReLU as the activation function. Hence  $\Theta_{\mathcal{E}} = (W_1, b_1)$  where  $W_1 \in \mathbb{R}^{784 \times 100}$  and  $b_1 \in \mathbb{R}^{100 \times 1}$ .

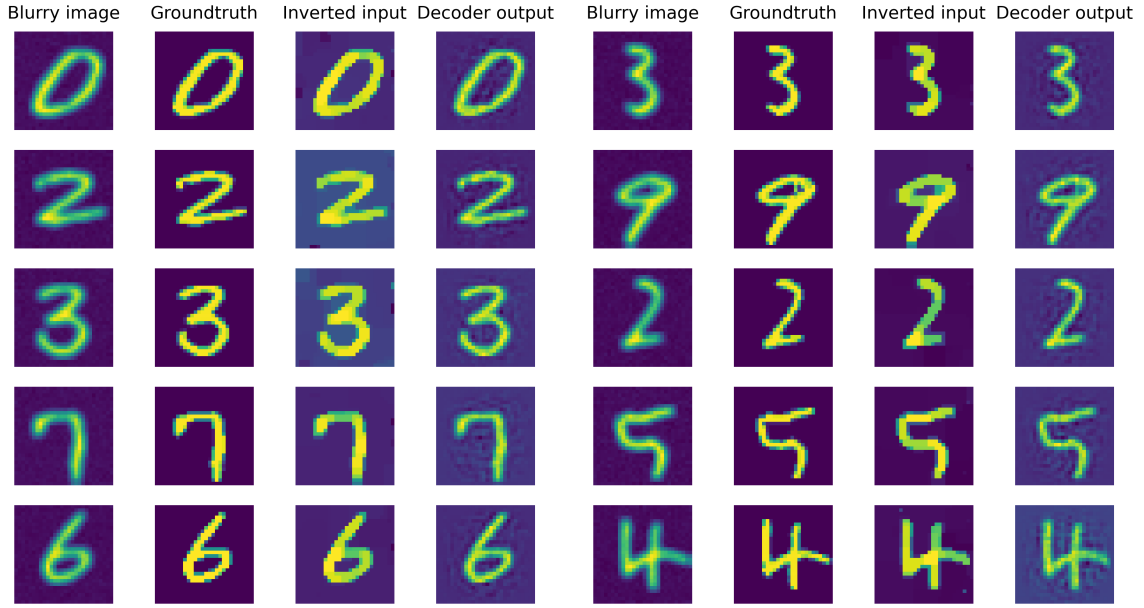


Fig. 5.6 Groundtruth solution images from the MNIST training dataset, together with the corresponding noisy blurry images, the reconstructed images from inversion and output images of the well-trained deblurring autoencoder.

Fig. 5.7 Groundtruth solution images from the MNIST validation dataset, together with the corresponding noisy blurry images, the reconstructed images from inversion and output images of the well-trained deblurring autoencoder.

We then invert the code  $y^\delta = \sigma(W_1x + b_1)$  via (5.10) with Algorithm (5.13). All MNIST images are pre-processed to be zero-centred by subtracting the mean. Algorithmically, we follow (5.13) to computationally solve (5.10). The stepsize-parameters are chosen at  $\tau_x = 1.99/\|W_1\|_2^2$  and  $\tau_z = 1/(8\alpha)$ . We choose the regularisation parameter  $\alpha$  in the range  $[10^{-4}, 10^{-2}]$  and set to  $5 \times 10^{-3}$  for all sample images from the training set, and set to  $\alpha = 5 \times 10^{-2}$  for all sample images from the validation set. These choices work well with regards to the visual quality of the reconstructed images.

In Figure 5.5, we show visualisations of five sample images from the training set, and from the validation set respectively. In comparison, we have also visualised the decoder output. As can be seen, using the code that contains the same compressed information, the reconstructed images show more clearly defined edges and better visual quality than the decoded outputs. This is to be expected as we compare a nonlinear regularised inversion method with a linear decoder.

## 5.2 Feedforward Neural Networks

In this section, we present numerical results for inverting multi-layer neural networks. We present results for two experiments. In the first experiment we show inversion of MNIST sample images from the noisy blurry outputs of a neural network and compare with the performance of a deblurring autoencoder. In the second experiment, we perform inversion of MNIST sample images of a convolutional neural network.

**Blind Deblurring of Images from MNIST** In the first example, we consider the inversion of a pre-trained fully connected neural network. We generate noisy blurry data  $y^\delta$  by first blurring all images with Gaussian blur, where we choose the kernel size to be 3 and standard deviation to be 5, then add Gaussian noise with mean 0 and standard deviation 0.01. More precisely, we set up the experiment by first train a two-layer fully connected autoencoder  $y = W_2(\sigma(W_1x + b_1)) + b_2$  using the vanilla stochastic gradient method (SGM) by minimising the mean squared error (MSE) on the MNIST training dataset. Notice that unlike conventional denoising or deblurring autoencoders, instead of trying to reconstruct clean outputs, we revert this process by feeding clean input images to reproduce noisy blurry output images. We set the hidden dimension at 392 and use ReLU as the activation function.

The goal is to first train a "blurring" neural network to capture the forward model in the deblurring problem, then perform stable neural network inversion for the deblurring task using the proposed variational regularisation framework. Note that the inversion problem in this experiment is therefore to invert the entire autoencoder. In comparison, we also train a conventional deblurring autoencoder by feeding in noisy blurry images for the reconstruction of clean deblurred outputs, minimised using the vanilla stochastic gradient method (SGM) by minimising the mean squared error (MSE).

we follow the implementation details outlined in Section 4.2 and iteratively compute the update steps (5.15) and (5.16) to solve the inversion problem. For the PDHG method, we choose the stepsize-parameters as  $\tau_x = 1.99/\|W_1\|_2^2$  and  $\tau_z = 1/(8\alpha)$ . The initial values  $x_0$  and  $z$  are both zero. The update steps stop either after reaching the maximum iterations of 500 or when the improvements on  $x_0$  and  $z$  are less than  $10^{-5}$  in norm. For the coordinate descent algorithm, the stepsize-parameters are set to  $\tau_{x_l} = 1.99/\|W_{l+1}\|_2^2$  for each layer.

In Figure 5.6 and Figure 5.7, we visualise five samples of the reconstructed images from inversion, the deblurred output images from the conventional deblurring autoencoder, along with the groundtruth images, and blurry images from the training dataset and validation dataset respectively. For each image,  $\alpha$  is chosen in the range

$[5 \times 10^{-3}, 5 \times 10^{-2}]$  and set at  $1 \times 10^{-2}$  for both training sample images and validation sample images for best visual reconstruction quality.

**Inverting a Convolutional Autoencoder** In the second experiment, we consider feedforward neural networks with convolutional layers (CNN), where in the network architecture two-dimensional convolution operations are used to represent the linear operations in the affine-linear functions  $f(x, \Theta)$ . Similar to the experimental design described in Section 5.1, we consider a multi-layer neural network inversion problem where we infer input image  $x$  from a noise perturbed code  $y^\delta$ .

More specifically, we first train a six-layer convolutional autoencoder on the MNIST training dataset via stochastic gradient method to minimise the MSE. The encoder  $\mathcal{E}(x, \Theta_{\mathcal{E}})$  consists of two successive convolutional layers, both with  $4 \times 4$  convolution kernels with stride 2, each followed by the application of a ReLU activation function. As image spatial dimension reduce by half, we double the number of feature channels from 8 to 16. We use a fully-connected layer with weights  $W_3 \in \mathbb{R}^{300 \times 784}$  and bias  $b_3 \in \mathbb{R}^{300 \times 1}$  to generate the code. The decoder network first expands the code with an affine-linear transformation with weights  $W_4 \in \mathbb{R}^{784 \times 300}$  and bias  $b_4 \in \mathbb{R}^{784 \times 1}$ . This is followed by two successive transpose convolutional layers with kernel size  $4 \times 4$ , where each is followed by a ReLU activation function. The number of feature channels halves each time as we double the spatial dimension.

Following the implementation details outlined in Section 4.2, we iteratively compute the update steps (5.15) and (5.16) to recover  $x$  from  $y = \mathcal{E}(x, \Theta)$ . For the PDHG method, we choose the stepsize-parameters as  $\tau_x = 1.99/\|W_1\|_2^2$  and  $\tau_z = 1/(8\alpha)$ . The initial values  $x_0$  and  $z$  are both zero. The update steps stop either after reaching the maximum iterations of 1500 or when the improvements on  $x_0$  and  $z$  are less than  $10^{-5}$  in norm. For the coordinate descent algorithm, the stepsize-parameters are set to  $\tau_{x_l} = 1.99/\|W_{l+1}\|_2^2$  for each layer, where  $\|\cdot\|$  denotes the spectral norm.

In Figure 5.8, we visualise the reconstructed images, the decoder output images, along with the groundtruth images, from the training dataset and validation dataset respectively. For each image,  $\alpha$  is chosen in the range  $[10^{-4}, 10^{-2}]$  and set at  $9 \times 10^{-3}$  for both training sample images and validation sample images for best visual reconstruction quality.

In Figure 5.9 we further compare how total variation regularisation and decoder respond to different levels of data noise. The noisy data is produced by adding Gaussian noise to perturb the code of each image. We start with zero mean Gaussian noise

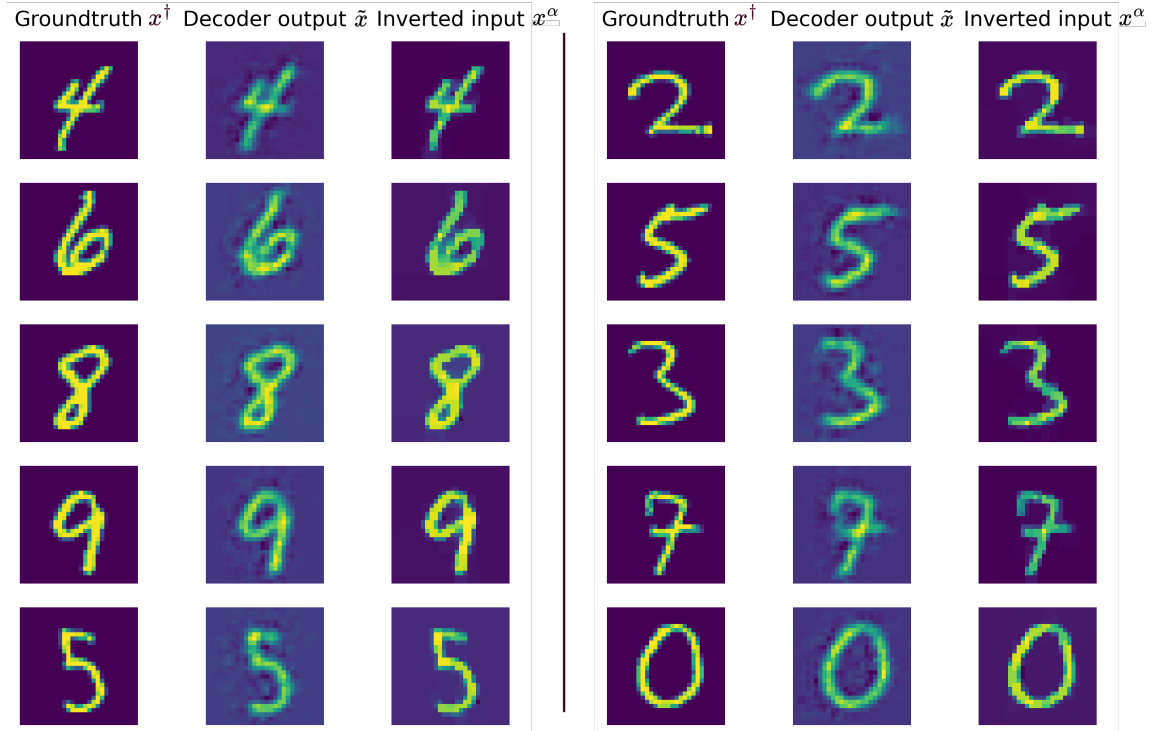


Fig. 5.8 Groundtruth input images from the MNIST training dataset (**Left**) and validation dataset (**Right**), together with the corresponding CNN autoencoder output images  $\tilde{x}$  and inversions  $x^\alpha$  of the encoding via (5.3).

with standard deviation 0.33 and gradually reduce the noise level, this translates to decreasing  $\delta^2$  from 6.80 down to 0.00.

Please note that for each noise level the regularisation factor  $\alpha$  is manually selected in the range  $[10^{-4}, 10^{-2}]$  for the best PSNR value. As we can see, for the noise level with standard deviation 0.33 where  $\delta^2$  is at 6.80, the decoder is only capable of producing a blurry distorted output, while the reconstructed image shows the structure of the digit more clearly. When we decrease the noise level down to 0.00, the reconstructed image becomes more clean-cut while the decoded image is still less sharply defined.

Figure 5.10 plots the PSNR value of the decoded and reconstructed image against decreasing noise level. We want to emphasise that it would be more rigorous to compute and compare  $D_R^{\text{symm}}(x_{\alpha(\delta)}, x^\dagger)$  as suggested in the error estimate bound in (5.8), but empirically the PSNR value does also support the notion of a convergent regularisation.

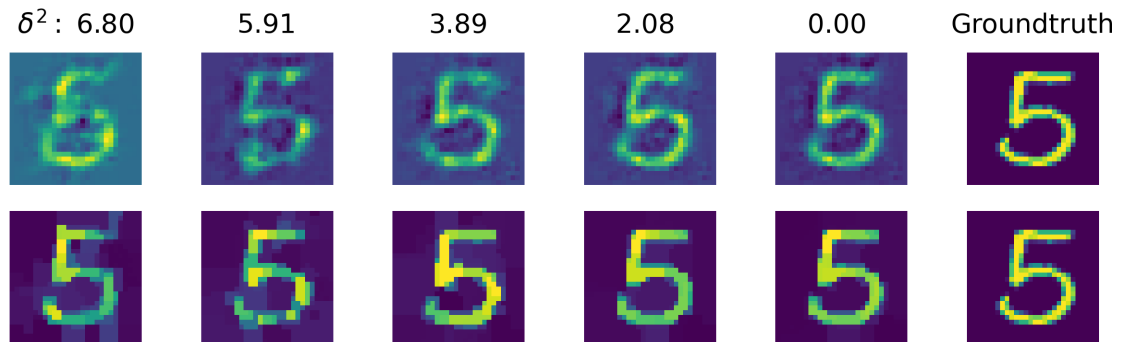


Fig. 5.9 Visualisation of the comparison between reconstructed image and decoded image against various levels of noise. **Top:** Decoded output image from the trained convolutional autoencoder. **Bottom:** Reconstructed input image from the CNN with total variation regularisation.

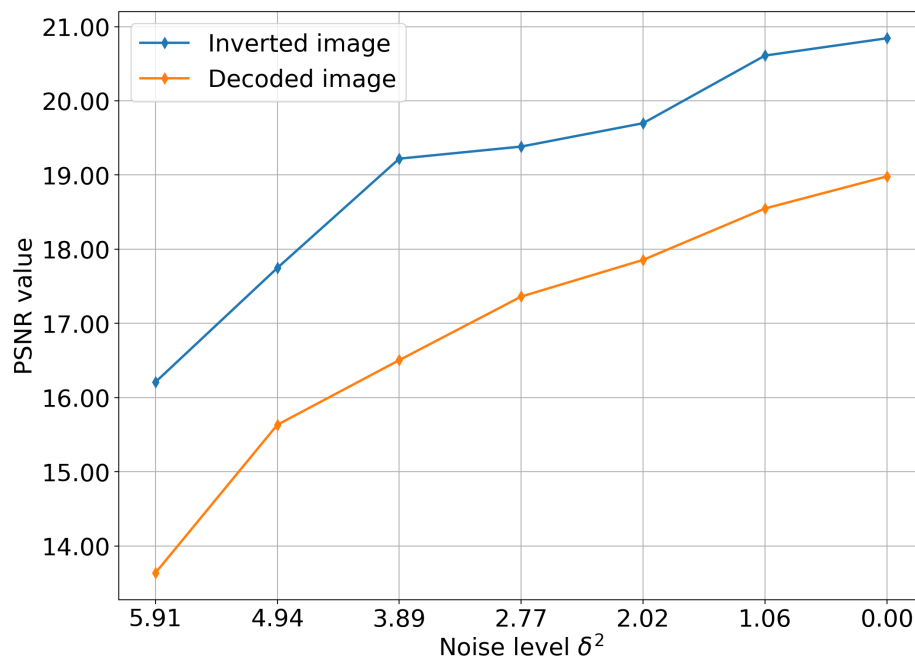


Fig. 5.10 Comparison of the PSNR value of total variation-based reconstruction and decoder output per noise level. Each curve reports the change of PSNR value over gradually decreasing levels of Gaussian noise, with  $\delta^2$  ranging from 0.00 to 6.80.

## 6 Discussion

In this chapter, we have introduced a variational regularisation framework based on the lifted Bregman formulation for the stable inversion of feedforward neural networks. Under the mild assumption that the inverse problem solution has to be in the range of the regularisation operator, we have proven that the proposed framework is a convergent regularisation for the single-layer perceptron case. Furthermore, we have derived a general error estimate as well as a specific error estimate for the case of ReLU activation function. The extension of the theory to multi-layer neural network cases can be carried out sequentially, albeit under some strict assumptions. We have also discussed implementation strategies to solve the proposed scheme computationally, and presented numerical results. In particular, we presented results for the regularised inversion for simple geometric shapes such as the image of a circle and piecewise constant images of hand-written digits from single-layer perceptrons. For inverting multi-layer neural networks, we demonstrated the application of blind deblurring, as well as results from the inversion of convolutional neural networks.



# Chapter 6

## Conclusion and Outlook

In this thesis, we have presented a Bregman learning framework for both the training problem and the inversion problem of neural networks. Our research in the three main chapters has demonstrated that the Bregman learning framework can perform comparably or even outperform back-propagation based training approaches in different learning scenarios. Our results show that this framework can provide an alternative solution for its adaptivity and stable performance compared to the standard training schemes.

Through the introduction of the novel Bregman loss (penalty) function, we formalised a generalised perceptron learning scheme and then extended it rigorously to construct distributed optimisation approaches for training multi-layer neural networks. Building upon the proposed lifted Bregman training framework for forward-direction learning problems, we formulated a variational regularisation framework for the inversion of neural networks. Supported by extensive theoretical analysis and evaluated through various numerical experiments, we demonstrated that the proposed Bregman learning framework is an effective and powerful solution that overcomes potential issues and limitations of the classical back-propagation based learning approach.

### 1 Summary

To conclude this chapter, we will begin by summarising major highlights and outcomes in each of the three research chapters. Following this, we will discuss possible future work in the next section.

## 1.1 Generalised Perceptron Learning

In **Chapter 3**, based on the broad family of proximal activation functions, we discussed the generalised perceptron learning problem and proposed a surrogate Bregman loss function. We also closely examined one of its key properties that the computation of partial derivatives of the network parameters does not require differentiation of the activation functions.

This new formulation permits an energy minimisation perspective to interpret the historical Rosenblatt's Perceptron learning algorithm. In particular, we confirmed for the first time that the original Rosenblatt's Perceptron learning algorithm can be understood as an incremental gradient method with respect to a data discrepancy term based on the proposed tailored Bregman distance. In addition, we showed that the proposed scheme also allows for the generalisation of the classical ADALINE delta rule. This interpretation was then adapted to apply on an example problem of learning sparse perceptrons using the proposed Rosenblatt-ISTA algorithm for classification tasks. We demonstrated through numerical results that the proposed learning approach outperformed all three other standard subgradient-based methods, capable of achieving both higher accuracy results and faster convergence.

## 1.2 Lifted Bregman Training of Neural Networks

In **Chapter 4**, we introduced the lifted Bregman framework for training feedforward and residual neural networks. We proposed the Bregman penalty functions and provided detailed and extensive mathematical analysis of their properties. We unified the MAC-QP and the lifted training approach into a single lifted Bregman training framework and established its rigorous mathematical properties. This framework can overcome potential drawbacks such as the vanishing gradient problem and limitations of non-parallelisability that are present in standard training approaches that use the back-propagation algorithm.

One key aspect of the proposed framework is its ability to break down the nested training objective function into layer-wise Bregman (penalty) loss functions, each of which has several significant advantages, including but not limited to non-expansiveness, bi-convexity, and 1-Lipschitz gradient. The decomposition enables the use of parallelisable or distributed optimisation methods and we presented a variety of deterministic and stochastic minimisation algorithms through which this Bregman learning framework can be computationally realised.

Our numerical experiments demonstrated that the proposed framework performed comparably or outperformed stochastic gradient methods in classification, compression, and denoising tasks. In the classification task, our trained network retained non-linearity, in contrast to the classical lifted network approach. In compression and denoising tasks, we demonstrated the convenience of incorporating regularisation on activation variables into the training process, which led to regularised autoencoders with sparse codes that produced more adaptive compressions compared to conventional autoencoders.

### 1.3 Lifted Bregman Inversion of Neural Networks

In **Chapter 5**, we established the lifted Bregman framework for inverting neural networks. Specifically speaking, we introduced a novel variational regularisation framework based on the lifted Bregman formulation for stable inversion of feedforward neural networks.

Under mild assumptions, we proved for the first time that the proposed framework is a convergent regularisation for the single-layer perceptron case. Notably, we have derived a general error estimate as well as a specific error estimate for the case of using the ReLU activation function. The extension of the theory to the multi-layer perceptron case was also addressed.

We presented implementation strategies to computationally solve the proposed scheme for both the perceptron and the multi-layer neural network case. Empirical results from the regularised inversion of the single-layer perceptron and multi-layer neural network outputs with total variation regularisation further supported the effectiveness of the proposed model-based variational regularisation approach.

## 2 Outlook

This section highlights potential areas for future extensions and open problems related to the work presented in this thesis.

### Extensions to More Architectures

Despite all the positive achievements presented in the chapters, the proposed framework also has some limitations. Currently, the proposed framework is limited to neural network architectures with proximal activation functions. Although a scheme has been presented to extend the framework to neural networks with residual blocks or skip connections, such as ResNets (He et al., 2016) or U-Nets (Ronneberger et al., 2015), it would be interesting to further expand this framework to Recurrent Neural Networks (RNNs) (Chung et al., 2014; Hochreiter and Schmidhuber, 1997) and Graph Neural Networks (GNNs) architecture (Bronstein et al., 2017; Wu et al., 2020).

Another limitation is that it is not yet straight-forward to include nonlinear operations that cannot be expressed as proximal maps of convex functions, such as max-pooling. However, for many examples there exist remedies, for instance using average pooling instead of max-pooling. Recent work also shows that an architecture based solely of multi-layer perceptrons can attain competitive scores on image classification benchmarks (Tolstikhin et al., 2021). Similarly, Touvron et al. (2022) presents a similar architecture and replaces the layer normalisation (Ba et al., 2016; Ioffe and Szegedy, 2015) with a simpler affine transformation. Therefore another interesting direction is to extend the Bregman learning framework to these pure multi-layer perceptron architectures that are mainly stacked by fully-connected layers. Nevertheless one potential avenue of research involves investigating the feasibility of incorporating other popular layer functions, such as transformer layers (Vaswani et al., 2017), and additional regularisation techniques like dropout (Hinton et al., 2012) directly into the proposed framework.

### Utilising Parallelism

While we did not demonstrate through numerical experiments, the proposed Bregman learning framework benefits greatly from parallel computation that accelerates the learning process. Parallel processing can be further utilised as a potential solution to mitigate the additional run time introduced in the lifted approaches. In particular, it would be worthwhile to explore the use of PyTorch’s distributed training framework,

including the distributed Autograd design, for implementing layer-wise updates of the network parameters in parallel.

### **Alternative Optimisation Methods**

Our proposed framework has opened more pathways to use other possible combinations of optimisation strategies, such as stochastic coordinate descent, stochastic ADMM, or stochastic primal-dual hybrid gradient methods (Chambolle et al., 2018) as well as tailored non-convex optimisation methods. Hence one possible future work is to implement alternative optimisation strategies for the proposed framework.

As we have also demonstrated that in contrary to wide-spread belief, expressive feed-forward networks can indeed overfit to training data without generalising well when non-standard optimisation methods are being used, it would be of particular interest to investigate more thoroughly if good generalisation properties of a deep neural network are the result of the chosen optimisation strategy or the chosen network architecture itself.

### **Explicit Control on Auxiliary Variables**

The Bregman learning framework offers explicit control and investigation of network activation variables, which is a key advantage. This capability allows for regularisation of activation variables to be easily incorporated during the forward training and inversion of neural networks, as shown in the chapters. For instance, the scheme used for learning sparse autoencoders can be extended to learning low-rank autoencoders by regularising the nuclear norm and comparing with implicit rank-minimising autoencoders (Jing et al., 2020). In the context of inversion, this capability can be useful for understanding the robustness and generalisation properties of neural networks. In particular, it would be beneficial to explore generating adversarial examples and adversarial attacks using the proposed Bregman learning framework. Another application is in data augmentation for image recognition tasks, where new input data can be created with different variations of features to augment the training dataset.

### **Connections to Bio-plausible Neural Networks**

The line of work on target propagation algorithm and recent extensions (Bartunov et al., 2018; Lee et al., 2015) are considered to be one of the more bio-plausible

alternatives to back-propagation algorithms which are extended formulations of the work in Le Cun (1986); LeCun et al. (1988).

One notable feature of the Bregman learning framework is that it decomposes the nested function objective into several loosely coupled, simple and layer-wise local sub-problems. Explicit auxiliary variable based method (Askari et al., 2018; Carreira-Perpinan and Wang, 2014; Zhang and Brand, 2017), as well as our proposed framework can be viewed as alternative bio-plausible solutions due to the explicit propagation of noisy or nondeterministic neuronal activity and layer-wise local synaptic updates (Bartunov et al., 2018).

It would be of particular interest to investigate potential connections with neuroscience-inspired learning algorithms, such as target propagation algorithm and predictive coding algorithm (Millidge et al., 2022) that are biologically plausible and utilise local learning, or with neuroscience-inspired neural networks such as the Spiking Neural Network (Lee et al., 2016; Maass, 1997) that model the behavior of neurons more directly.

### **Convergent Inversion of Feedforward Neural Networks**

For the lifted Bregman inversion of neural networks, an open question is how a convergence theory without too restrictive assumptions can be established for the multi-layer neural network. One issue is the non-convexity of the proposed formulation. A remedy could be the use of different architectures that lead to lifted Bregman formulations that are jointly convex in all auxiliary variables.

Last but not least, we would also like to consider other forms of regularisation, such as iterative regularisation, data-driven regularisations (Kabri et al., 2022), or even combinations of both (Aspri et al., 2020). However, the convergence analysis for such approaches is still an open problem and requires further investigation.

## **3 Concluding Remarks**

Despite the ongoing challenges, the potential applications and future extensions of the Bregman learning framework are vast and promising. Through this thesis, we have demonstrated that the forward-direction learning and the backward-direction learning are indeed two sides of the same coin within the unified Bregman learning framework. Particularly, our proposed framework has the capacity to provide a more efficient and effective training approach that is crucial to unlocking the full potential of Artificial Neural Networks.

# References

- Abi-Chahla, F. (2008). Nvidia’s cuda: The end of the cpu?’. *Tom’s Hardware*, pages 1954–7.
- Aghasi, A., Abdi, A., Nguyen, N., and Romberg, J. (2017). Net-trim: Convex pruning of deep neural networks with performance guarantee. *Advances in neural information processing systems*, 30.
- Alain, G. and Bengio, Y. (2014). What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research*, 15(1):3563–3593.
- Antoniou, A., Storkey, A., and Edwards, H. (2017). Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*.
- Arridge, S., Maass, P., Öktem, O., and Schönlieb, C.-B. (2019). Solving inverse problems using data-driven models. *Acta Numerica*, 28:1–174.
- Askari, A., Negiar, G., Sambharya, R., and Ghaoui, L. E. (2018). Lifted neural networks. *arXiv preprint arXiv:1805.01532*.
- Aspri, A., Banert, S., Öktem, O., and Scherzer, O. (2020). A data-driven iteratively regularized landweber iteration. *Numerical Functional Analysis and Optimization*, 41(10):1190–1227.
- Azarian, K., Bhalgat, Y., Lee, J., and Blankevoort, T. (2020). Learned threshold pruning. *arXiv preprint arXiv:2003.00075*.
- Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.
- Bartlett, P. L., Long, P. M., Lugosi, G., and Tsigler, A. (2020). Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*, 117(48):30063–30070.
- Bartunov, S., Santoro, A., Richards, B., Marris, L., Hinton, G. E., and Lillicrap, T. (2018). Assessing the scalability of biologically-motivated deep learning algorithms and architectures. *Advances in neural information processing systems*, 31.
- Bauschke, H. H., Bolte, J., and Teboulle, M. (2017). A descent lemma beyond lipschitz gradient continuity: first-order methods revisited and applications. *Mathematics of Operations Research*, 42(2):330–348.

- Bauschke, H. H., Combettes, P. L., et al. (2011). *Convex analysis and monotone operator theory in Hilbert spaces*, volume 408. Springer.
- Beck, A. (2017). *First-order methods in optimization*. SIAM.
- Beck, A. and Tetruashvili, L. (2013). On the convergence of block coordinate descent type methods. *SIAM journal on Optimization*, 23(4):2037–2060.
- Behrmann, J., Grathwohl, W., Chen, R. T., Duvenaud, D., and Jacobsen, J.-H. (2019). Invertible residual networks. In *International Conference on Machine Learning*, pages 573–582. PMLR.
- Behrmann, J., Vicol, P., Wang, K.-C., Grosse, R., and Jacobsen, J.-H. (2021). Understanding and mitigating exploding inverses in invertible neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 1792–1800. PMLR.
- Belkin, M. (2021). Fit without fear: remarkable mathematical phenomena of deep learning through the prism of interpolation. *Acta Numerica*, 30:203–248.
- Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854.
- Bellman, R. (1957). Dynamic programming.
- Bengio, Y., Courville, A., and Vincent, P. (2013a). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.
- Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.
- Bengio, Y., Yao, L., Alain, G., and Vincent, P. (2013b). Generalized denoising auto-encoders as generative models. *Advances in neural information processing systems*, 26.
- Benning, M., Betcke, M. M., Ehrhardt, M. J., and Schönlieb, C.-B. (2021). Choose your path wisely: gradient descent in a bregman distance framework. *SIAM Journal on Imaging Sciences*, 14(2):814–843.
- Benning, M. and Burger, M. (2011). Error estimates for general fidelities. *Electronic Transactions on Numerical Analysis*, 38(44-68):77.
- Benning, M. and Burger, M. (2018). Modern regularization methods for inverse problems. *Acta Numerica*, 27:1–111.
- Benning, M. and Ehrhardt, M. J. (2016). Lecture notes on inverse problems in imaging.
- Benning, M. and Riis, E. S. (2021). Bregman methods for large-scale optimisation with applications in imaging. *Handbook of Mathematical Models and Algorithms in Computer Vision and Imaging: Mathematical Imaging and Vision*, pages 1–42.

- Berner, J., Grohs, P., Kutyniok, G., and Petersen, P. (2021). The modern mathematics of deep learning.
- Bertsekas, D. P. (1999). *Nonlinear Programming*. Athena Scientific, second edition.
- Bertsekas, D. P. (2011). Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning*, 2010(1-38):3.
- Bishop, C. M. and Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*, volume 4. Springer.
- Bolte, J. and Pauwels, E. (2020). A mathematical model for automatic differentiation in machine learning. *Advances in Neural Information Processing Systems*, 33:10809–10819.
- Bolte, J., Sabach, S., Teboulle, M., and Vaisbourd, Y. (2018). First order methods beyond convexity and lipschitz gradient continuity with applications to quadratic inverse problems. *SIAM Journal on Optimization*, 28(3):2131–2151.
- Bottou, L. (1998). Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142.
- Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational Statistics Paris France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer.
- Boyd, S., Boyd, S. P., and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.
- Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122.
- Bregman, L. M. (1967). The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR computational mathematics and mathematical physics*, 7(3):200–217.
- Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42.
- Bryson, A. E. (1961). A gradient method for optimizing multi-stage allocation processes. In *Proc. Harvard Univ. Symposium on digital computers and their applications*, volume 72, page 22.
- Bungert, L., Roith, T., Tenbrinck, D., and Burger, M. (2021). A bregman learning framework for sparse neural networks. *arXiv preprint arXiv:2105.04319*.
- Burbea, J. and Rao, C. (1982a). On the convexity of higher order jensen differences based on entropy functions (corresp.). *IEEE Transactions on Information Theory*, 28(6):961–963.

- Burbea, J. and Rao, C. (1982b). On the convexity of some divergence measures based on entropy functions. *IEEE Transactions on Information Theory*, 28(3):489–495.
- Burger, M. (2016). Bregman distances in inverse problems and partial differential equations. In *Advances in mathematical modeling, optimization and optimal control*, pages 3–33. Springer.
- Cai, J.-F., Osher, S., and Shen, Z. (2009). Linearized bregman iterations for compressed sensing. *Mathematics of computation*, 78(267):1515–1536.
- Candès, E. J., Romberg, J., and Tao, T. (2006). Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 52(2):489–509.
- Carreira-Perpinan, M. and Wang, W. (2014). Distributed optimization of deeply nested systems. In *Artificial Intelligence and Statistics*, pages 10–19. PMLR.
- Censor, Y. and Lent, A. (1981). An iterative row-action method for interval convex programming. *Journal of Optimization theory and Applications*, 34(3):321–353.
- Censor, Y. and Zenios, S. A. (1992). Proximal minimization algorithm with  $d$ -functions. *Journal of Optimization Theory and Applications*, 73(3):451–464.
- Chambolle, A. (2004). An algorithm for total variation minimization and applications. *Journal of Mathematical imaging and vision*, 20:89–97.
- Chambolle, A., Caselles, V., Cremers, D., Novaga, M., and Pock, T. (2010). An introduction to total variation for image analysis. *Theoretical foundations and numerical methods for sparse recovery*, 9(263-340):227.
- Chambolle, A., Ehrhardt, M. J., Richtárik, P., and Schonlieb, C.-B. (2018). Stochastic primal-dual hybrid gradient algorithm with arbitrary sampling and imaging applications. *SIAM Journal on Optimization*, 28(4):2783–2808.
- Chambolle, A. and Lions, P.-L. (1997). Image recovery via total variation minimization and related problems. *Numerische Mathematik*, 76:167–188.
- Chambolle, A. and Pock, T. (2011). A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision*, 40:120–145.
- Chambolle, A. and Pock, T. (2016a). An introduction to continuous optimization for imaging. *Acta Numerica*, 25:161–319.
- Chambolle, A. and Pock, T. (2016b). On the ergodic convergence rates of a first-order primal-dual algorithm. *Mathematical Programming*, 159(1-2):253–287.
- Chang, C.-H., Creager, E., Goldenberg, A., and Duvenaud, D. (2018). Explaining image classifiers by counterfactual generation. *arXiv preprint arXiv:1807.08024*.
- Chen, G. and Teboulle, M. (1993). Convergence analysis of a proximal-like minimization algorithm using bregman functions. *SIAM Journal on Optimization*, 3(3):538–543.

- Choromanska, A., Cowen, B., Kumaravel, S., Luss, R., Rigotti, M., Rish, I., Diachille, P., Gurev, V., Kingsbury, B., Tejwani, R., et al. (2019). Beyond backprop: Online alternating minimization with auxiliary variables. In *International Conference on Machine Learning*, pages 1193–1202. PMLR.
- Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.
- Collins, M. (2012). Convergence proof for the perceptron algorithm.
- Collins, M. D. and Kohli, P. (2014). Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*.
- Combettes, P. L. and Pesquet, J.-C. (2020a). Deep neural network structures solving variational inequalities. *Set-Valued and Variational Analysis*, pages 1–28.
- Combettes, P. L. and Pesquet, J.-C. (2020b). Lipschitz certificates for layered network structures driven by averaged activation operators. *SIAM Journal on Mathematics of Data Science*, 2(2):529–557.
- Cornell Chronicle (2022). Rosenblatt’s perceptron. [Online; accessed November 21, 2022].
- Courant, R. (1943). Variational methods for the solution of problems of equilibrium and vibrations.
- Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.
- Daubechies, I., Defrise, M., and De Mol, C. (2004). An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 57(11):1413–1457.
- De Vito, E., Rosasco, L., Caponnetto, A., De Giovannini, U., Odone, F., and Bartlett, P. (2005). Learning from examples as an inverse problem. *Journal of Machine Learning Research*, 6(5).
- De Vito, E., Rosasco, L., and Rudi, A. (2021). Regularization: From inverse problems to large-scale machine learning. In *Harmonic and Applied Analysis*, pages 245–296. Springer.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- Dinh, L., Krueger, D., and Bengio, Y. (2015). Nice: Non-linear independent components estimation. In *International Conference on Learning Representations*.
- Donoho, D. L. (2006). Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306.

- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.
- Dreyfus, S. (1962). The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30–45.
- Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).
- Duchi, J. and Singer, Y. (2009). Efficient online and batch learning using forward backward splitting. *The Journal of Machine Learning Research*, 10:2899–2934.
- Eckstein, J. (1993). Nonlinear proximal point algorithms using bregman functions, with applications to convex programming. *Mathematics of Operations Research*, 18(1):202–226.
- Ekeland, I. and Temam, R. (1999). *Convex analysis and variational problems*. SIAM.
- Engl, H. W., Hanke, M., and Neubauer, A. (1996). *Regularization of inverse problems*, volume 375. Springer Science & Business Media.
- Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S., and Vincent, P. (2009). The difficulty of training deep architectures and the effect of unsupervised pre-training. In *Artificial Intelligence and Statistics*, pages 153–160. PMLR.
- Esser, E., Zhang, X., and Chan, T. F. (2010). A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science. *SIAM Journal on Imaging Sciences*, 3(4):1015–1046.
- Fercoq, O. and Richtárik, P. (2015). Accelerated, parallel, and proximal coordinate descent. *SIAM Journal on Optimization*, 25(4):1997–2023.
- Fong, R., Patrick, M., and Vedaldi, A. (2019). Understanding deep networks via extremal perturbations and smooth masks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2950–2958.
- Fong, R. C. and Vedaldi, A. (2017). Interpretable explanations of black boxes by meaningful perturbation. In *Proceedings of the IEEE international conference on computer vision*, pages 3429–3437.
- Fortman-Roe, S. (2012). Understanding the bias-variance tradeoff. <http://scott.fortmann-roe.com/docs/BiasVariance.html>. Accessed: 2023-03-31.
- Franca, G., Robinson, D., and Vidal, R. (2018). Admm and accelerated admm as continuous dynamical systems. In *International Conference on Machine Learning*, pages 1559–1567. PMLR.
- Frerix, T., Möllenhoff, T., Moeller, M., and Cremers, D. (2017). Proximal backpropagation. *arXiv preprint arXiv:1706.04638*.

- Frid-Adar, M., Diamant, I., Klang, E., Amitai, M., Goldberger, J., and Greenspan, H. (2018). Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331.
- Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.
- Funahashi, K.-I. (1989). On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192.
- Gabay, D. (1983). Applications of the method of multipliers to variational inequalities. In *Studies in mathematics and its applications*, volume 15, pages 299–331. Elsevier.
- Gal, R., Alaluf, Y., Atzmon, Y., Patashnik, O., Bermano, A. H., Chechik, G., and Cohen-Or, D. (2022). An image is worth one word: Personalizing text-to-image generation using textual inversion. *arXiv preprint arXiv:2208.01618*.
- Geiping, J. and Moeller, M. (2019). Parametric majorization for data-driven energy minimization methods. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 10262–10273.
- Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.
- Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings.
- Goldstein, T., O’Donoghue, B., Setzer, S., and Baraniuk, R. (2014). Fast alternating direction optimization methods. *SIAM Journal on Imaging Sciences*, 7(3):1588–1623.
- Goodfellow, I., Bengio, Y., and Courville, A. (2016a). *Deep learning*. MIT press.
- Goodfellow, I., Bengio, Y., and Courville, A. (2016b). *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.
- Griewank, A. (2012). Who invented the reverse mode of differentiation. *Documenta Mathematica, Extra Volume ISMP*, 389400.
- Gu, F., Askari, A., and El Ghaoui, L. (2020). Fenchel lifted networks: A lagrange relaxation of neural network training. In *International Conference on Artificial Intelligence and Statistics*, pages 3362–3371. PMLR.
- Guerguiev, J., Lillicrap, T. P., and Richards, B. A. (2017). Towards deep learning with segregated dendrites. *Elife*, 6:e22901.

- Hadamard, J. (1902). Sur les prolemes aux derives partielles et leur signification physique. *Bulletin of Princeton University*, 13:1–20.
- Hadamard, J. (1908). *Mémoire sur le problème d'analyse relatif à l'équilibre des plaques élastiques encastrées*, volume 33. Imprimerie nationale.
- Hasannasab, M., Hertrich, J., Neumayer, S., Plonka, G., Setzer, S., and Steidl, G. (2020). Parseval proximal neural networks. *Journal of Fourier Analysis and Applications*, 26(4):1–31.
- Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015a). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015b). Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- Hertrich, J., Neumayer, S., and Steidl, G. (2021). Convolutional proximal neural networks and plug-and-play algorithms. *Linear Algebra and its Applications*, 631:203–234.
- Higham, C. F. and Higham, D. J. (2019). Deep learning: An introduction for applied mathematicians. *Siam review*, 61(4):860–891.
- Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.
- Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851.
- Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1).
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- Hoeffler, T., Alistarh, D., Ben-Nun, T., Dryden, N., and Peste, A. (2021). Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005.
- Høier, R. K. and Zach, C. (2020). Lifted regression/reconstruction networks. *arXiv preprint arXiv:2005.03452*.

- Huang, B., Ma, S., and Goldfarb, D. (2013). Accelerated linearized bregman method. *Journal of Scientific Computing*, 54(2):428–453.
- Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, D., Chen, M., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., et al. (2019). Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32.
- Huang, Z. and Wang, N. (2018). Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*.
- Hubel, D. H. and Wiesel, T. N. (1959). Receptive fields of single neurones in the cat’s striate cortex. *The Journal of physiology*, 148(3):574.
- Huo, Z., Gu, B., Huang, H., et al. (2018). Decoupled parallel backpropagation with convergence guarantee. In *International Conference on Machine Learning*, pages 2098–2106. PMLR.
- Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR.
- Jackson, J. C. and Craven, M. (1996). Learning sparse perceptrons. In *Advances in neural information processing systems*, pages 654–660.
- Jeffreys, H. (1946). An invariant form for the prior probability in estimation problems. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 186(1007):453–461.
- Jensen, C. A., Reed, R. D., Marks, R. J., El-Sharkawi, M. A., Jung, J.-B., Miyamoto, R. T., Anderson, G. M., and Eggen, C. J. (1999). Inversion of feedforward neural networks: algorithms and applications. *Proceedings of the IEEE*, 87(9):1536–1549.
- Jing, L., Zbontar, J., et al. (2020). Implicit rank-minimizing autoencoder. *Advances in Neural Information Processing Systems*, 33:14736–14746.
- Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. (2021). Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589.
- Kabri, S., Auras, A., Riccio, D., Bauermeister, H., Benning, M., Moeller, M., and Burger, M. (2022). Convergent data-driven regularizations for ct reconstruction. *arXiv preprint arXiv:2212.07786*.
- Kalyanakrishnan, S. (2017). The perceptron learning algorithm and its convergence.
- Kantorovich, L. V. and Akilov, G. P. (2016). *Functional analysis*. Elsevier.
- Kawaguchi, K., Kaelbling, L. P., and Bengio, Y. (2017). Generalization in deep learning. *arXiv preprint arXiv:1710.05468*.
- Kelley, H. J. (1960). Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954.

- Kiefer, J. and Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, pages 462–466.
- Kindermann, J. and Linden, A. (1990). Inversion of neural networks by gradient descent. *Parallel computing*, 14(3):277–286.
- Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.
- Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.
- Kiwiel, K. C. (1997). Proximal minimization methods with generalized bregman functions. *SIAM journal on control and optimization*, 35(4):1142–1168.
- Korpelevich, G. M. (1976). The extragradient method for finding saddle points and other problems. *Matecon*, 12:747–756.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.
- Krotov, D. and Hopfield, J. J. (2019). Unsupervised learning by competing hidden units. *Proceedings of the National Academy of Sciences*, 116(16):7723–7731.
- Lacoste-Julien, S., Schmidt, M., and Bach, F. (2012). A simpler approach to obtaining an  $o(1/t)$  convergence rate for the projected stochastic subgradient method. *arXiv preprint arXiv:1212.2002*.
- Landweber, L. (1951). An iteration formula for fredholm integral equations of the first kind. *American journal of mathematics*, 73(3):615–624.
- Lau, T. T.-K., Zeng, J., Wu, B., and Yao, Y. (2018). A proximal block coordinate descent algorithm for deep neural network training. *arXiv preprint arXiv:1803.09082*.
- Le Cun, Y. (1986). Learning process in an asymmetric threshold network. In *Disordered systems and biological organization*, pages 233–240. Springer.
- LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- LeCun, Y., Touresky, D., Hinton, G., and Sejnowski, T. (1988). A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28.

- Lee, D.-H., Zhang, S., Fischer, A., and Bengio, Y. (2015). Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part I 15*, pages 498–515. Springer.
- Lee, J. H., Delbruck, T., and Pfeiffer, M. (2016). Training deep spiking neural networks using backpropagation. *Frontiers in neuroscience*, 10:508.
- Lemaréchal, C. (2012). Cauchy and the gradient method. *Doc Math Extra*, 251(254):10.
- Li, H., Schwab, J., Antholzer, S., and Haltmeier, M. (2020). Nett: Solving inverse problems with deep neural networks. *Inverse Problems*, 36(6):065005.
- Li, J., Fang, C., and Lin, Z. (2019). Lifted proximal operator machines. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33 No. 01.
- Lin, J. (1991). Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151.
- Linden, A. and Kindermann, J. (1989). Inversion of multilayer nets. In *Proc. Int. Joint Conf. Neural Networks*, volume 2, pages 425–430.
- Linnainmaa, S. (1970). *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. PhD thesis, Master’s Thesis (in Finnish), Univ. Helsinki.
- Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160.
- Lions, P.-L. and Mercier, B. (1979). Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979.
- Liu, B., Wang, M., Foroosh, H., Tappen, M., and Pensky, M. (2015). Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.
- Loog, M., Viering, T., Mey, A., Krijthe, J. H., and Tax, D. M. (2020). A brief prehistory of double descent. *Proceedings of the National Academy of Sciences*, 117(20):10625–10626.
- Lu, B.-L., Kita, H., and Nishikawa, Y. (1999). Inverting feedforward neural networks using linear and nonlinear programming. *IEEE Transactions on Neural networks*, 10(6):1271–1290.
- Lu, H., Freund, R., and Mirrokni, V. (2018). Accelerating greedy coordinate descent methods. In *International Conference on Machine Learning*, pages 3257–3266. PMLR.

- Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. (2017). The expressive power of neural networks: A view from the width. *Advances in neural information processing systems*, 30.
- Lu, Z. and Xiao, L. (2015). On the complexity analysis of randomized block-coordinate descent methods. *Mathematical Programming*, 152:615–642.
- Lunz, S., Öktem, O., and Schönlieb, C.-B. (2018). Adversarial regularizers in inverse problems. *Advances in neural information processing systems*, 31.
- Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671.
- Mahendran, A. and Vedaldi, A. (2015). Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5188–5196.
- Martinet, B. (1970). Brève communication. régularisation d’inéquations variationnelles par approximations successives. *Revue française d’informatique et de recherche opérationnelle. Série rouge*, 4(R3):154–158.
- Martinet, B. (1972). *Algorithmes pour la résolution de problèmes d’optimisation et de minimax*. PhD thesis, Université Joseph-Fourier-Grenoble I.
- McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.
- McDonald, R., Hall, K., and Mann, G. (2010). Distributed training strategies for the structured perceptron. In *Human language technologies: The 2010 annual conference of the North American chapter of the association for computational linguistics*, pages 456–464.
- Meulemans, A., Carzaniga, F., Suykens, J., Sacramento, J., and Grewe, B. F. (2020). A theoretical framework for target propagation. *Advances in Neural Information Processing Systems*, 33:20024–20036.
- Millidge, B., Salvatori, T., Song, Y., Bogacz, R., and Lukasiewicz, T. (2022). Predictive coding: towards a future of deep learning beyond backpropagation? *arXiv preprint arXiv:2202.09467*.
- Minsky, M. and Papert, S. (1969). An introduction to computational geometry. *Cambridge tiass., HIT*, 479:480.
- Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2016). Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582.
- Moreau, J. J. (1962). Fonctions convexes duales et points proximaux dans un espace hilbertien. *Comptes rendus hebdomadaires des séances de l’Académie des sciences*, 255:2897–2899.

- Moreau, J.-J. (1965). Proximité et dualité dans un espace hilbertien. *Bulletin de la Société mathématique de France*, 93:273–299.
- Morozov, V. A. (2012). *Methods for solving incorrectly posed problems*. Springer Science & Business Media.
- Mukherjee, S., Dittmer, S., Shumaylov, Z., Lunz, S., Öktem, O., and Schönlieb, C.-B. (2021). Learned convex regularizers for inverse problems. *arXiv preprint arXiv:2008.02839v2*.
- Mukkamala, M. C., Ochs, P., Pock, T., and Sabach, S. (2020). Convex-concave backtracking for inertial bregman proximal gradient algorithms in nonconvex optimization. *SIAM Journal on Mathematics of Data Science*, 2(3):658–682.
- Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.
- Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Icml*.
- Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., and Zaharia, M. (2019). Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 1–15.
- Nemirovski, A., Juditsky, A., Lan, G., and Shapiro, A. (2009). Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609.
- Nesterov, Y. (2003). *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media.
- Nesterov, Y. (2012). Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362.
- Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate  $\mathcal{O}(1/k^2)$ . In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547.
- Ng, A. et al. (2011). Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19.
- Nielsen, F. and Boltz, S. (2011). The burbea-rao and bhattacharyya centroids. *IEEE Transactions on Information Theory*, 57(8):5455–5466.
- Ogawa, T., Kosugi, Y., and Kanada, H. (1998). Neural network based solution to inverse problems. In *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98CH36227)*, volume 3, pages 2471–2476. IEEE.
- OpenAI (2023). Gpt-4 technical report.
- Osher, S., Burger, M., Goldfarb, D., Xu, J., and Yin, W. (2005). An iterative regularization method for total variation-based image restoration. *Multiscale Modeling & Simulation*, 4(2):460–489.

- Parikh, N. and Boyd, S. (2014). Proximal algorithms. *Foundations and Trends in optimization*, 1(3):127–239.
- Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr.
- Pock, T., Cremers, D., Bischof, H., and Chambolle, A. (2009). An algorithm for minimizing the mumford-shah functional. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1133–1140. IEEE.
- Polyak, B. (1987). *Introduction to Optimization*. Optimization Software, Inc.
- Poon, C. and Liang, J. (2019). Trajectory of alternating direction method of multipliers and adaptive acceleration. *Advances in Neural Information Processing Systems*, 32.
- Redmon, J. and Farhadi, A. (2018). Yolo3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.
- Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28.
- Rezende, D. and Mohamed, S. (2015). Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR.
- Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE international conference on neural networks*, pages 586–591. IEEE.
- Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.
- Rockafellar, R. T. (1970). *Convex Analysis*, volume 36. Princeton University Press.
- Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5–9, 2015, Proceedings, Part III 18*, pages 234–241. Springer.
- Rosasco, L., Villa, S., and Vũ, B. C. (2019). Convergence of stochastic proximal gradient algorithm. *Applied Mathematics & Optimization*, pages 1–27.
- Rosenblatt, F. (1957). The perceptron, a perceiving and recognizing automaton. *Project Para. Cornell Aeronautical Laboratory*.
- Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY.
- Rudin, L. I., Osher, S., and Fatemi, E. (1992). Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268.

- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252.
- Sacramento, J., Ponte Costa, R., Bengio, Y., and Senn, W. (2018). Dendritic cortical microcircuits approximate the backpropagation algorithm. *Advances in neural information processing systems*, 31.
- Scardapane, S., Comminiello, D., Hussain, A., and Uncini, A. (2017). Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89.
- Scherzer, O., Grasmair, M., Grossauer, H., Haltmeier, M., and Lenzen, F. (2009). Variational methods in imaging.
- Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61:85–117.
- Schwab, J., Antholzer, S., and Haltmeier, M. (2019). Deep null space learning for inverse problems: convergence analysis and rates. *Inverse Problems*, 35(2):025008.
- Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.
- Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.
- Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- Sion, M. (1958). On general minimax theorems.
- Sjöström, J., Gerstner, W., et al. (2010). Spike-timing dependent plasticity. *Spike-timing dependent plasticity*, 35(0):0–0.
- Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR.
- Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*.
- Stuart, A. M. (2010). Inverse problems: a bayesian perspective. *Acta numerica*, 19:451–559.

- Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.
- Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., and Goldstein, T. (2016). Training neural networks without gradients: A scalable admm approach. In *International conference on machine learning*, pages 2722–2731. PMLR.
- Teboulle, M. (1992). Entropic proximal mappings with applications to nonlinear programming. *Mathematics of Operations Research*, 17(3):670–690.
- Teboulle, M. (2018). A simplified view of first order methods for optimization. *Mathematical Programming*, 170(1):67–96.
- Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.
- Tieleman, T., Hinton, G., et al. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.
- Tolstikhin, I. O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., et al. (2021). Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34:24261–24272.
- Toulis, P. and Airolidi, E. M. (2017). Asymptotic and finite-sample properties of estimators based on stochastic gradients. *The Annals of Statistics*, 45(4):1694–1727.
- Touvron, H., Bojanowski, P., Caron, M., Cord, M., El-Nouby, A., Grave, E., Izacard, G., Joulin, A., Synnaeve, G., Verbeek, J., et al. (2022). Resmlp: Feedforward networks for image classification with data-efficient training. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Tsyppkin, Y. Z. and Nikolic, Z. J. (1971). *Adaptation and learning in automatic systems*, volume 73. Academic Press New York.
- v. Neumann, J. (1928). Zur theorie der gesellschaftsspiele. *Mathematische annalen*, 100(1):295–320.
- Vapnik, V. (1982). Estimation of dependences based on empirical data: Springer series in statistics (springer series in statistics).
- Vapnik, V. (1999). *The nature of statistical learning theory*. Springer science & business media.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.
- Vidal, R., Bruna, J., Giryes, R., and Soatto, S. (2017). Mathematics of deep learning. *arXiv preprint arXiv:1712.04741*.

- Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., and Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12).
- Wang, H. and Banerjee, A. (2014). Bregman alternating direction method of multipliers. *Advances in Neural Information Processing Systems*, 27.
- Wang, H. and Raj, B. (2017). On the origin of deep learning. *arXiv preprint arXiv:1702.07800*.
- Wang, X. and Benning, M. (2020). Generalised perceptron learning. *12th OPT Workshop on Optimization for Machine Learning at 34th Conference on Neural Information Processing Systems (NeurIPS)*.
- Wang, X. and Benning, M. (2022). Lifted bregman training of neural networks. *arXiv preprint arXiv:2208.08772*.
- Wang, X. and Benning, M. (2023). A lifted bregman formulation for the inversion of deep neural networks. *arXiv preprint arXiv:2303.01965*.
- Wang, Y.-X., Girshick, R., Hebert, M., and Hariharan, B. (2018). Low-shot learning from imaginary data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7278–7286.
- Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. (2016). Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29.
- Werbos, P. J. (2005). Applications of advances in nonlinear sensitivity analysis. In *System Modeling and Optimization: Proceedings of the 10th IFIP Conference New York City, USA, August 31–September 4, 1981*, pages 762–770. Springer.
- Widrow, B. (1960). An adaptive 'adaline' neuron using chemical 'memistors', 1553-1552. *Stanford Electronics Laboratories*.
- Wright, S. J. (2015). Coordinate descent algorithms. *Mathematical programming*, 151(1):3–34.
- Wright, S. J. and Recht, B. (2022). *Optimization for data analysis*. Cambridge University Press.
- Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24.
- Xia, W., Zhang, Y., Yang, Y., Xue, J.-H., Zhou, B., and Yang, M.-H. (2022). Gan inversion: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

- Xie, J., Xu, L., and Chen, E. (2012). Image denoising and inpainting with deep neural networks. *Advances in neural information processing systems*, 25.
- Yosida, K. (1964). *Functional analysis*, volume 123. Springer.
- Zach, C. and Estellers, V. (2019). Contrastive learning for lifted networks. British Machine Vision Conference (BMVC).
- Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.
- Zeng, J., Lau, T. T.-K., Lin, S., and Yao, Y. (2019). Global convergence of block coordinate descent in deep learning. In *International Conference on Machine Learning*, pages 7313–7323. PMLR.
- Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. (2021a). Dive into deep learning. *arXiv preprint arXiv:2106.11342*.
- Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2021b). Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115.
- Zhang, T. and Shen, Z. (2019). A fundamental proof of convergence of alternating direction method of multipliers for weakly convex optimization. *Journal of Inequalities and Applications*, 2019(1):1–21.
- Zhang, Z. and Brand, M. (2017). On the convergence of block coordinate descent in training dnns with tikhonov regularization. In *Advances in Neural Information Processing Systems*, pages 1719–1728.
- Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232.
- Zhu, M. and Chan, T. (2008). An efficient primal-dual hybrid gradient algorithm for total variation image restoration. *Ucla Cam Report*, 34:8–34.
- Zhuang, H., Wang, Y., Liu, Q., and Lin, Z. (2021). Fully decoupled neural network learning using delayed gradients. *IEEE transactions on neural networks and learning systems*, 33(10):6013–6020.
- Zinkevich, M., Weimer, M., Li, L., and Smola, A. (2010). Parallelized stochastic gradient descent. *Advances in neural information processing systems*, 23.