Towards a mathematical understanding of biologically plausible learning methods for deep neural networks

Alexander Meulemans

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
wiskundige ingenieurstechnieken

Promotoren:
Prof. Benjamin Grewe
Prof. Johan Suykens

Academiejaar 2018 – 2019

**Master of Science in de ingenieurswetenschappen:
wiskundige ingenieurstechnieken**

KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Towards a mathematical understanding of biologically plausible learning methods for deep neural networks

Alexander Meulemans

Thesis voorgedragen tot het behalen
van de graad van Master of Science
in de ingenieurswetenschappen:
wiskundige ingenieurstechnieken

Promotoren:
Prof. Benjamin Grewe
Prof. Johan Suykens

Assessoren:
Prof. Daan Huybrechs
Dr. Nick Vannieuwenhoven

Begeleider:
Prof. Benjamin Grewe

Academiejaar 2018 – 2019

Abstract

Due to the recent successes of deep artificial neural networks (ANNs) and the resemblance of ANNs to the mammalian brain, the question has arisen whether the learning processes in the mammalian brain are driven by similar deep hierarchical models. This question is investigated in the newly emerged field of biologically plausible deep learning. Answering this question can contribute both to the field of machine learning and neuroscience. On the one hand, investigating biologically plausible deep learning can provide machine learning with new insights on hard challenges such as continuous learning, one-shot learning and general intelligence, which the human brain seems to achieve effortlessly. On the other hand, solving the credit assignment problem –how the strength of synapses in the brain must be changed to improve global behaviour– will help neuroscientists to develop new insights on how our brain (dys)functions in health and disease. Although a lot of progress has already been made, many of the biologically plausible learning methods currently lack a solid mathematical foundation and a well-defined link to the biological properties of neurons. This thesis focusses on target propagation (TP), a promising biologically plausible learning method for ANNs. Its main contributions to the field of TP are (1) a new extensive mathematical framework for TP, (2) various improvements to the TP method based on gained mathematical insights and (3) two biological network models of TP that are closely linked to the properties of pyramidal neurons.

The new mathematical theory of TP has three main results. First, we prove that under well-specified conditions, TP with exact inverses uses Gauss-Newton optimization to compute its local layer targets, after which it performs a gradient descent step on the local layer losses to update the forward weights of the network. Second, we show that in TP with approximate inverses, reconstruction errors interfere with the propagated learning signals. We prove that difference target propagation (DTP), a variant of TP, cancels out these reconstruction errors by adding a correction term to its targets, which explains the better performance of DTP compared to TP. Third, we show that under well-specified conditions, DTP uses Gauss-Newton to compute its local layer targets, even when approximate inverses are used.

Based on the gained mathematical insight on target propagation, we propose two improvements for the TP method. First, we introduce a new parametrization for the backward mapping function of the targets, which ensures that it can learn the inverse of the forward mapping to arbitrary precision when layers of equal dimension are used. Experimental results show that this new form led to significantly better performance of TP. Second, we propose a randomized version of TP which has better theoretical properties. Experimental results indicate that the randomized TP is more stable for deeper architectures.

Finally, we develop two biological network models that exhibit TP-like learning dynamics and are closely linked to the properties of pyramidal neurons. The first model is a mixture model of the apical dendritic spikes and the basal dendritic spikes that occur in the pyramidal neuron. Both a single-phase and a two-phase version of this model are introduced, that exhibit TP-like and DTP-like learning dynamics, respectively. The second model makes use of multiplexing in pyramidal neurons to separate the feed-forward and feedback signals. The separation of signal paths provides this network with cleaner learning signals, compared to the previous mixture models. Both models are worked out in theory, while future research should verify their experimental performance.

This thesis has as significant contributions to the field of biologically plausible deep learning that (1) it creates an extensive mathematical foundation for the TP method that led to improved variants of TP and (2) it develops two biological network models of TP that are closely linked to the biological properties of pyramidal neurons. Future research can use the mathematical framework to further improve TP and other biologically plausible learning methods and can use the biological network models in its quest for solving the credit assignment problem in the mammalian brain.

Samenvatting

Vanwege de recente successen van deep artificial neural networks (ANNs) en de gelijkenis van ANNs met onze hersenen, is de vraag gerezen of de leerprocessen in onze hersenen worden aangedreven door vergelijkbare diepe hiërarchische modellen. Deze vraag wordt onderzocht in het niewe onderzoeksveld van biologically plausible deep learning. Het beantwoorden van deze vraag kan zowel bijdragen aan het onderzoeksveld van machine learning als de neurowetenschappen. Aan de ene kant kan het onderzoeken van biologisch plausibele leermethodes machine learning nieuwe inzichten geven over harde uitdagingen zoals continuous learning, one-shot learning en general artificial intelligence, die het menselijk brein moeiteloos lijkt te kunnen. Aan de andere kant zal het oplossen van het credit assignment problem –hoe de sterkte van synapsen in de hersenen veranderd moet worden om het globale gedrag te verbeteren– neurowetenschappers helpen met nieuwe inzichten te ontwikkelen over hoe onze hersenen (dys)functioneren in gezondheid en ziekte. Hoewel er al veel vooruitgang is geboekt, missen veel van de biologisch plausibele leermethoden momenteel een sterke wiskundige basis en een preciese gedefinieerde link naar de biologische eigenschappen van neuronen. Deze thesis richt zich op target propagation (TP), een veelbelovende biologisch plausibele leermethode voor ANNs. De belangrijkste bijdragen aan het onderzoeksveld van TP zijn (1) een nieuwe uitgebreide wiskundig theorie van TP, (2) verschillende verbeteringen aan de TP-methode op basis van verworven wiskundige inzichten en (3) twee biologische netwerkmodellen van TP die nauw verbonden zijn met de eigenschappen van piramidale neuronen.

De nieuwe wiskundige theorie van TP heeft drie belangrijke resultaten. Ten eerste hebben we bewezen dat TP met exacte inverses Gauss-Newton optimalisatie gebruikt om de lokale targets te berekenen, waarna het een gradiënt stap uitvoert op de lokale kostfuncties om de voorwaartse parameters van het netwerk bij te werken. Ten tweede toonden we aan dat in TP met benaderende inverses, reconstructiefouten de leersignalen vervormen. We hebben bewezen dat difference target propagation (DTP), een variant van TP, deze reconstructiefouten verwijdert door een correctieterm aan zijn target signalen toe te voegen, wat de betere prestaties van DTP in vergelijking met TP verklaart. Ten derde hebben we aangetoond dat DTP Gauss-Newton optimalisatie gebruikt om de lokale targets te berekenen, zelfs wanneer benaderende inverses worden gebruikt.

Op basis van de opgedane wiskundige inzichten in target propagation, hebben we twee verbeteringen voor de TP-methode voorgesteld. Eerst hebben we een nieuwe parametrisatie geïntroduceerd voor de achterwaartse transformatie functie van de targets, die ervoor zorgt dat de inverse van de voorwaartse transformatie functie kan worden geleerd tot willekeurige precisie wanneer de netwerk lagen van gelijke dimensie zijn. Experimentele resultaten toonden dat deze nieuwe vorm tot significant betere prestaties van TP leidde. Ten tweede hebben we een gerandomiseerde versie van TP voorgesteld die betere theoretische eigenschappen heeft. Experimentele resultaten gaven aan dat de gerandomiseerde TP stabieler is voor diepere netwerken.

Ten slotte hebben we twee biologische netwerkmodellen ontwikkeld die TP-achtige leerdynamieken vertonen en nauw verbonden zijn met de eigenschappen van piramidale neuronen. Het eerste model is een mixture model van de apicale dendritische spikes en de basale dendritische spikes die voorkomen in piramidale neuronen. Zowel een enkel-fasige als een twee-fasige versie van dit model worden geïntroduceerd, die respectievelijk TP-achtige en DTP-achtige leerdynamieken vertonen. Het tweede model maakt gebruik van een multiplex techniek in pyramidale neuronen om de feed-forward en feedback signalen te scheiden. De scheiding van de signaalpaden geeft dit netwerk betere leersignalen vergeleken met de eerdere mixture modellen. Beide modellen zijn uitgewerkt in theorie. Toekomstig onderzoek zal hun experimentele prestaties moeten verifiëren.

Deze thesis heeft als belangrijke bijdragen aan het gebied van biologically plausible deep learning dat (1) het een uitgebreide mathematische basis heeft gecreëerd voor de TP-methode die leidde tot verbeterde varianten van TP en (2) het twee biologische netwerkmodellen van TP heeft ontwikkeld die nauw zijn verbonden aan de biologische eigenschappen van piramidale neuronen. Toekomstig onderzoek kan de wiskundige theorie gebruiken om TP en andere biologisch plausibele leermethoden verder te verbeteren en kan de biologische netwerkmodellen gebruiken in haar zoektocht naar het oplossen van het credit assignment problem in onze hersenen.

Preface

Not all those who wander are lost.

J. R. R. Tolkien

This thesis finds itself on the boundary between two very exciting fields: artificial intelligence and neuroscience. The subject of my thesis started very vaguely as 'developing biologically plausible learning methods for spiking neural networks'. Due to this very broad topic, I spend most of the first semester on reading neuroscience papers and deep learning papers, in the search for an exciting learning method where I could focus on during my thesis. After many brainstorms, readings and interesting discussions, I decided to return to my roots as a mathematical engineer and started building a theoretical work around target propagation, a recent developed biologically plausible learning method. That is how the final subject of my thesis was born: 'Towards a mathematical understanding of biologically plausible learning methods for deep neural networks'.

This thesis would not have been possible without the help, dedication and expertise of many inspiring people. First of all, I would like to thank prof. Benjamin Grewe for the countless brainstorms, interesting discussions and the fantastic lab retreats with the whole research group. Without him, I would not have had such an interesting thesis where I could completely throw myself in and such a supporting research group to work with. Also, his detailed feedback on my thesis text is much appreciated. I would also like to thank prof. Johan Suykens, for all the good suggestions and the sharp comments he gave on my work.

Furthermore, I would like to express my gratitude to Christian, for always being his critic self while giving detailed feedback, but also for always being the first one to say congrats when I had some new theoretical result. A big thank you for Joao and Johannes (and also Christian again) for the many hours of brainstorms and discussions, and for generating many interesting ideas. Special thanks to the whole GreweLab in general, for making my time at INI and ETH Zürich such a nice experience.

Surviving a thesis is a lot more fun with friends and family around you. Therefore I would like to thank Adriaan, Wout, Holger, Laura, Lauren, Seba, Rik and Thomas for all the fun Game of Thrones nights, circuit trainings and beach volleyball conquests. Special thanks to my housemates Adriaan and Wout for always being enthusiastic to help me prove some theorems or generate some new ideas during the afternoon coffee. A big thank you to my mom and dad, for, as the cliché goes, always believing in me all those years and simply being the best mom and dad I could have wished for. I would also like to thank Justine and Pieter, for all the fun moments at home. Last but certainly not least, I would like to thank Lauren, for always trying to look interested while I enthusiastically explain my thesis, but most of all for reminding me that the world outside of my thesis is also full of wonders.

Enjoy the reading!

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

AMPA	α-amino-3-hydroxy-5-methyl-4isoxazolepropionic acid
AMPAR	AMPA receptor
ANN	Artificial neural network
AO-SDTP	Auxiliary output difference target propagation
ATP	Adenosine Triphosphate
BDFA	Bidirectional feedback alignment
BFA	Broadcast feedback alignment
BP	Backpropagation, referring to the error backpropagation method
CNN	Convolutional neural network
DFA	Direct feedback alignment
DTP	Difference target propagation
DTP-AI	Difference target propagation with approximate inverses
FA	Feedback alignment
GABA	Gamma-aminobutyric acid
GAN	Generative adversarial network
GN	Gauss-Newton
GPU	Graphics processing unit
IBM	International Business Machines Corporation
Leaky-ReLU	Leaky rectified linear unit
LIF	Leaky-integrate-and-fire
LM	Levenberg-Marquardt
LTD	Long-term depression
LTP	Long-term potentiation
MLP	Multi-layer perceptron
MSE	Mean squared error
MTP	Modified target propagation
NE	Norepinephrine
NMDA	N-methyl-D-aspartate
NMDAR	NMDA receptor
ReLU	Rectified linear unit
RL	Reinforcement learning
RDTP-AI	Randomized difference target propagation with approximate inverses
RMTP	Randomized modified target propagation

RMTP-EI	Randomized modified target propagation with exact inverses
RTP	Randomized target propation
RTP-AI	Randomized target propagation with approximate inverses
RTP-EI	Randomized target propagation with exact inverses
SDTP	Simplified difference target propagation
SGD	Stochastic gradient descent
SL	Supervised learning
SM	Sherman-Morrison
STD	Short-term synaptic depression
STF	Short-term synaptic facilitation
TP	Target propagation
TP-AI	Target propagation with approximate inverses
TP-EI	Target propagation with exact inverses
fixed-BP	Error backpropagation with fixed hidden layer parameters
original-DTP	The original difference target propagation method from literature
original-TP	The original target propagation method from literature

List of symbols

B_i	Random feedback weights of the i-th layer
C	Damped curvature matrix
$\tilde{C}$	Block-diagonal approximation of the damped curvature matrix
D_{s_i}	Jacobian matrix of $s(\boldsymbol{a}_i)$ with respect to $\boldsymbol{a}_i$
E_K	Kalium equilibrium potential of the neuronal membrane
E_{Na}	Natrium equilibrium potential of the neuronal membrane
E_{ion}	Ionic equilibrium potential of the neuronal membrane
E_{rest}	Resting potential of the neuronal membrane
E_{th}	Threshold potential of the neuronal membrane
G	Gauss-Newton curvature matrix
$\tilde{G}$	Block-diagonal approximation of the Gauss-Newton curvature matrix
H	Hessian matrix
I	Input current of a neuron, or the identity matrix, depending on the context
J	Jacobian matrix
J_{f_i}	The Jacobian matrix of f_i with respect to $\boldsymbol{h}_{i-1}$ evaluated at $\boldsymbol{h}_{i-1}$
J_{g_i}	The Jacobian matrix of g_i with respect to $\boldsymbol{h}_{i+1}$ evaluated at $\boldsymbol{h}_{i+1}$
$J_{\boldsymbol{h}_i,\boldsymbol{a}_i}$	Jacobian matrix of $\boldsymbol{h}_i$ with respect to $\boldsymbol{a}_i$
J_i	The Jacobian matrix of $\boldsymbol{h}_L$ with respect to $\boldsymbol{h}_i$
J_i^f	The factorized approximation of the pseudoinverse of J_i
L	Number of layers in a network
L_1	L_1 loss
L_2	Mean squared error, also known as L_2 loss
$L^{(i)}$	Output loss of the i-th mini-batch of the dataset
L_i	Local layer loss function for training the forward weights W_i in the target propagation methods
L_i^{inv}	Inverse loss function of the i-th layer for learning the backward maping g_i
$L_i^{inv,r}$	Regularized inverse loss function of the i-th layer for learning the backward maping g_i
L_{tot}	Total output loss of the whole dataset
M^{3rd}	Time-varying third factor of the three-factor learning rule
Q_i	Matrix containing all the backward weights of the i-th layer
U	The left singular vectors of a singular value decompositon
V	The right singular vectors of a singular value decomposition
W_i	Matrix containing all forward weights of the i-th layer

$\boldsymbol{a}_i$	Vector containing all linear activations a_{ij} of the i-th layer
$\boldsymbol{a}_i^b$	Voltage level of the basal compartments of the i-th layer in the multiplexing model
$\boldsymbol{a}_i^a$	Voltage level of the apical compartments of the i-th layer in the multiplexing model
a_{ij}	Linear activation of neuron j in layer i, representing the internal voltage level a biological neuron
$\boldsymbol{b}_i$	Bias term of the i-th layer
d	Part of the denominator in the Sherman-Morrison formula
$\boldsymbol{e}^{(i)}$	Output residual at the m-th iteration
e_{ij}	Eligibility trace, also called synaptic flag
$\boldsymbol{e}_i^{approx}$	The approximation error in the i-th layer, defined as $\hat{\boldsymbol{h}}_i - \boldsymbol{h}_i - \boldsymbol{e}_i^{TP}$
$\boldsymbol{e}_i^{BP}$	The learning signal propagated to the i-th layer by the error backpropagation method
$\boldsymbol{e}_i^{GN}$	The learning signal propagated to the i-th layer by the Gauss-Newton method
$\boldsymbol{e}_i^{rec}$	The accumulated reconstruction error in the i-th layer
$\boldsymbol{e}_i^{TP}$	The useful learning signal propagated to the i-th layer in the target propagation method and its variants
$\boldsymbol{e}_i^{Taylor}$	The Taylor approximation error when approximating
$\boldsymbol{e}_i^{phase}$	The difference between sum of the second and third term of (6.31) and the sum of the ones of (6.33) $\hat{\boldsymbol{h}}_i$
$\boldsymbol{e}_L$	Gradient of the output loss function L with respect to $\boldsymbol{h}_L$
f_i	Forward mapping function that maps $\boldsymbol{h}_{i-1}$ to $\boldsymbol{h}_i$
g_i	Backward mapping function that maps $\hat{\boldsymbol{h}}_{i+1}$ to $\hat{\boldsymbol{h}}_i$
$\boldsymbol{h}_i$	Vector containing all non-linear activations h_{ij} of the i-th layer
$\boldsymbol{h}_i^*$	Equilibrium point of the dynamical target propagation model with segregated dendrites
$\bar{\boldsymbol{h}}$	Vector containing the concatenated layer activations $\boldsymbol{h}_i$, $i = 1, ..., L-1$
$\hat{\boldsymbol{h}}_i$	Target activation of the i-th layer
h_{ij}	Non-linear activation of neuron j in layer i, representing the output firing rate of a biological neuron
n_i	The dimension of the i-th layer
s_i	Non-linear forward activation function of the i-th layer
t_i	Non-linear backward activation function of the i-th layer
$\boldsymbol{t}$	Output target value, as given by the training dataset
w_{jk}	Forward weight representing the synapse connecting neuron j of the current layer with neuron k of the previous layer
z	Auxiliary output, used in the AO-SDTP method
$\Delta\boldsymbol{\beta}$	Parameter update
$\Delta\boldsymbol{Q}_i$	The update for the backward parameters $\boldsymbol{Q}_i$
ΔW_i	The update for the forward parameters W_i
Γ_i	The covariance matrix of the layer activations $\boldsymbol{h}_i$
Σ	Matrix containing the singular values of a singular value decomposition
α	Additive damping parameter for the curvature matrices $\tilde{G}$ and G
$\boldsymbol{\alpha}_i^f$	Plateau potential of the i-th layer in the free-phase of the deep learning with segragated dendrites model

$\boldsymbol{\alpha}_i^t$	Plateau potential of the i-th layer in the target-phase of the deep learning with segragated dendrites model
β	Scaling value for the learning rate, used in the robust Sherman-Morrison update
$\boldsymbol{\beta}^{(m)}$	Parameters of a model at the m-th iteration
$\boldsymbol{\delta_i}$	Helper variable in the error backpropagation algorithm, defined as $\left(\frac{\partial L}{\partial \boldsymbol{a}_i}\right)^T$
$\boldsymbol{\delta_i}^{BP}$	Helper variable in the error backpropagation algorithm, defined as $\left(\frac{\partial L}{\partial \boldsymbol{a}_i}\right)^T$
$\boldsymbol{\delta_i}^{DFA}$	Helper variable in the direct feedback alignment algorithm, defined as $\left(\frac{\partial L}{\partial \boldsymbol{a_i}}\right)^T$
$\boldsymbol{\delta_i}^{FA}$	Helper variable in the feedback alignment algorithm, defined as $\left(\frac{\partial L}{\partial \boldsymbol{a}_i}\right)^T$
ϵ	Threshold used in the robust Sherman-Morrison update
$\hat{\eta}$	Output step size for computing the output target in the target propagation method
η_{f_i}	Learning rate of the forward parameters W_i
η_{g_i}	Learning rate of the backward parameters $\boldsymbol{Q}_i$
η_i	Learning rate of the forward parameters W_i
η_i^{inv}	Learning rate for the backward weights $\boldsymbol{Q}_i$
γ	Damping parameter for the curvature matrix $\tilde{G}$ or regularizer parameter for the regularized inverse loss $L_i^{inv,r}$, depending on the context
$\theta_W^{i:j}$	Subset of parameters used in the mapping from the i-th layer to the j-th layer
λ	Damping parameter for the Levenberg-Marquardt method
μ	Convex mixing factor in the neural dynamic equations
ρ	Reduction ratio or Pearson's correlation coefficient, depending on the context
$\rho_{j,k}$	Pearson's correlation coefficient between the j-th and k-th neuron of a layer
σ_i	The i-th singular value of a singular value decomposition
σ^2	The variance of a random variable
τ	Time constant of the neural dynamic equations
τ_e	Decay time constant of the Hebbian learning rule
τ_{init}	Threshold for initializing the weight matrices W_i in target propagation with exact inverses
ψ	Damping parameter for the curvature matrix G
$\odot$	Element-wise multiplication
$\dagger$	Operator for the More-Penrose pseudoinverse

Chapter 1

Introduction

During the last decade, the impact of machine learning on our daily life has increased enormously. Examples of applications range from customized web searches to improving medical diagnoses to beating the world champion in the board game GO [1]. Most notably, deep learning –a form of machine learning that allows models to learn data representations with multiple levels of abstraction– has driven the majority of recent developments thanks to the increased computing power of graphics processing units (GPUs) and the large size of available data sets. Artificial neural networks (ANNs) are the most common form of deep learning, and their model architecture is loosely inspired by the neural networks inside the neocortex of the mammalian brain. Multiple layers of artificial neurons are connected with each other, which enables the model to process the data in different levels of abstraction. The model parameters that need to be learned are the connection weights between the different artificial neurons. This resembles the synapses in our own brain, which form the connections between the different neurons, and which have connection strengths that can change over time.

Due to the recent successes of deep learning and the resemblance of ANNs to the mammalian brain, the question has arisen whether the learning processes in the mammalian brain are driven by similar deep hierarchical models. In the newly created research direction of *biologically plausible deep learning*, researchers from the field of machine learning and neuroscience have joined forces to address this question. The motivation for this new research field can be explained from a machine learning point of view and a neuroscience point of view. Despite the numerous successes of deep learning, ANNs are still lacking behind in fields such as continuous learning, one-shot learning and general intelligence, which the human brain seems to achieve effortlessly. In the history of deep learning, researchers have repeatedly drawn inspiration from neuroscience to improve their models. The most famous example is the convolutional neural network (CNN), of which the origins can be traced back to the early neuroscience research of Hubel and Wiesel in the 1960s [2], who first described the hierarchical neural processing of visual inputs in the mammalian brain. Therefore, it is most likely that the field of biologically plausible deep learning will inspire new developments in the field of machine learning to address the current challenges. In neuroscience, the credit assignment problem –how the strength of each individual synapse in the brain must be changed in order to improve global behaviour– remains still an open question. Answering this question will help neuroscientists and psychologists to develop new insights and hypotheses on how biological networks in the brain (dys)function in health and disease. In ANNs, the credit assignment problem is solved by the error backpropagation algorithm [3]. However, this algorithm in its pure form is not likely to be biologically plausible based on what is currently known. Therefore, the field of biologically plausible deep learning can help neuroscience by providing new, more biologically plausible, solutions for the credit assignment problem.

As the field of biologically plausible deep learning is still very young, there exist various areas that are not yet explored in enough detail. First of all, many of the biologically plausible learning methods for deep neural networks are based on intuition or insights from neuroscience, without having a thorough mathematical foundation. Therefore, a better mathematical understanding of those learning methods can lead to significant improvements to the performance of these methods and help the field to focus on the most promising methods. Secondly, most of the biologically plausible learning methods are defined in an abstract form [4, 5, 6] without a clear link to biological neurons, making it hard for neuroscientists to interpret those models. Hence, creating learning methods that are more closely linked towards biological neurons will help neuroscientists to deduce workable hypotheses from those models that can provide new insights on the inner working of our brain. Recent work [7, 8] has already started with bridging this gap, but more research is still needed on this topic.

The focus of this thesis is on *target propagation* [9, 5], a promising biologically plausible learning method that is intuitively very appealing and has already decent experimental results on benchmark datasets such as MNIST [10, 5, 11].

This thesis aims to:

1. create a mathematical foundation for target propagation and its variants, in order to better understand its learning dynamics,
2. improve the existing target propagation method, based on the gained mathematical understanding,
3. create a biologically realistic learning model that is closely linked to the biological properties of neurons and exhibits target-propagation-like learning dynamics.

Based on the above-defined aims of this thesis, the following approach is followed.

- Chapter 2 explores the relevant properties of both biological neural networks and artificial neural networks, in order to provide the needed background for this thesis.
- Chapter 3 provides an overview of the existing target propagation method and other relevant work in the field of biologically plausible deep learning.
- Chapter 4 creates a thorough mathematical foundation for the target propagation method by proposing and proving new theorems that clarify the optimization strategy and learning dynamics of target propagation. Based on the gained mathematical insights, various improvements to the target propagation method are proposed.
- Chapter 5 experimentally verifies the theoretical predictions and assumptions made in chapter 4 and assesses the performance of the new variants of target propagation. As previous work has mainly focussed on the performance of target propagation on large datasets such as MNIST, this chapter focusses more on the fundamental learning properties of target propagation by investigating more easily interpretable toy datasets.
- Chapter 6 proposes a biologically realistic learning model that is closely linked to the biological properties of biological neurons and exhibits target-propagation-like learning dynamics.

This thesis will have two main contributions to the field of biologically plausible deep learning and the scientific community as a whole. First, we will create an extensive mathematical foundation for the target propagation method, which will lead to improved variants of target propagation and which will help future research to further improve the target propagation method and other alternatives to error backpropagation. Hopefully, this work will encourage other researchers and accelerate the mathematical understanding of biologically plausible learning methods. Secondly, a biologically more realistic model of target propagation can give new hypotheses for neuroscience on how credit assignment is done in biological networks.

Chapter 2

Background

In this chapter, the needed background for this thesis is explained, covering both biological and artificial neural networks.

2.1 The biological neuron

Neuroscience is a wide and complex field, and it is not the purpose of this text to give a comprehensive and detailed overview of the field. Instead, we will briefly explain the basics of a biological neuron that are needed to understand the new ideas and experiments performed in this thesis. We refer the interested reader to [12] and [13] for a more detailed and complete introduction to the field of neuroscience and theoretical neuroscience, respectively. This section discusses first the anatomy of the neuron, after which the basic properties of the communication signalling between neurons – the action potential – are explained. Thereafter this section discusses the properties of the synapses in our brain, and it ends with explaining synaptic plasticity, which makes learning possible in the brain.

2.1.1 Anatomy of the neuron

A wide variety of neuron cell types exist that differ based on their structure, chemistry and function. Nonetheless, all neurons have three characteristic compartments: the soma, the axon and the dendrites [12] as visualized in figure 2.1. The synapse, which connects the axon of one neuron with a dendrite of another neuron, is essential for learning in the brain and will be discussed separately from the axon and dendrites in section 2.1.3.

Soma

The soma is the central, roughly spherical, part of the neuron. It supports the neuron in its basic functions such as protein synthesis and cell respiration. The organelles present in the soma can also be found in all other animal cells. The most important ones are the nucleus, the Golgi apparatus and the mitochondria [12]. These organelles are shown in figure 2.2.

Axon

The axon is a cell structure only found in neurons and is highly specialized in transmitting electrical signals to other neurons in the form of action potentials. The axon starts at the axon hillock (figure 2.2) and ends in one or more (up to several thousands) axon terminals (figure 2.7). The length of an axon can vary from less than a millimeter to over a meter, and the axon can split up in multiple branches (figure 2.3), which are called *axon collaterals*. All the branches combined are called the *terminal arbor* [12]. The axon can propagate only one electrical signal at the same time, so the neuron has only one output signal, that is delivered at the synapses of all the connecting neurons.

Dendrites

The dendrites receive the inputs of the connected neurons, and can therefore be seen as the 'antennas' of the neuron. The term 'dendrite' is derived from the Greek word 'dendros' for 'tree, reflecting the charac-

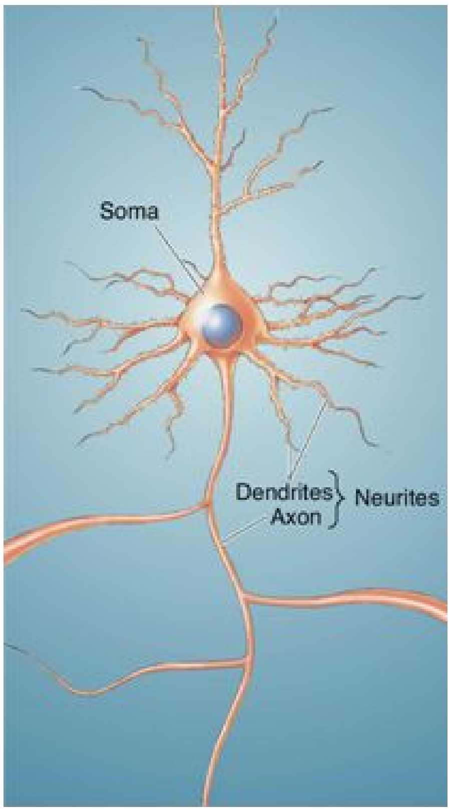


Figure 2.1: **The soma, axon and dendrites of a neuron.** Axons and dendrites are also called neurites. Figure reprinted from [12].

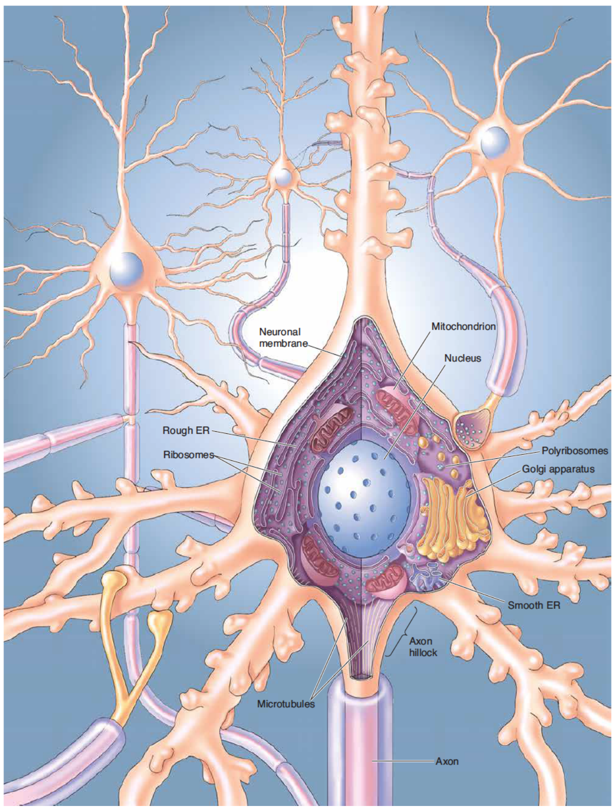


Figure 2.2: **The internal structure and organelles of a typical neuron.** Figure reprinted from [12].

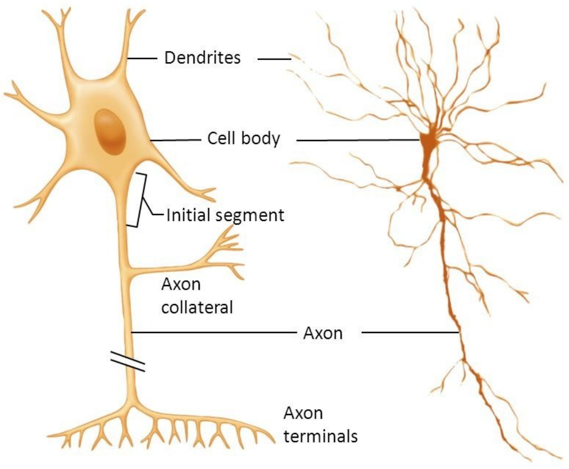


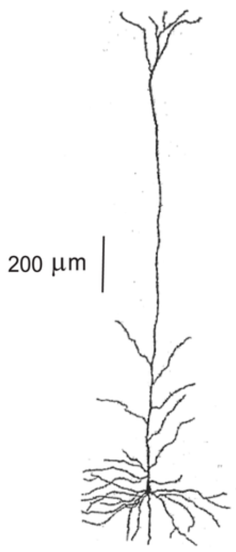
Figure 2.3: **The structure of a neuron and its an axon.** Figure reprinted from [14].

teristic branching of the dendrites in one or more dendritic trees (see figure 2.1). Dendrites are covered by synapses which connect the axons of incoming neurons with the dendrites.

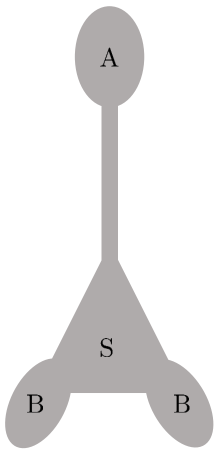
Types of neurons

The human brain consists of roughly 100 billion neurons and even more glial cells (cells that support the neurons in the brain) [15]. All these neurons can be classified in numerous classes based on various criteria such as dendritic structure, axon length and neurotransmitters [12]. This thesis focuses on the neural networks inside the cerebral cortex, as this brain area governs the processing of sensations, perceptions, voluntary movement, learning, speech and cognition, which are of primary interest for the field of machine learning. Furthermore, the cerebral cortex is also believed to be the foundation of human intelligence [16]. $70-85\%$ of the neurons in the cerebral cortex are pyramidal neurons, the other $15-30\%$ are interneurons and aspiny non-pyramidal neurons, which cover a wide variety of neurons [17]. The biological interpretation of the learning rules in this thesis is based on pyramidal neurons, which can be justified by their high prevalence in the cerebral cortex. Furthermore, pyramidal neurons project to other brain areas, whereas interneurons only project locally. This makes pyramidal neurons more suited for investigating hierarchical learning mechanisms.

The pyramidal neuron. The pyramidal neuron can be schematized in 3 compartments as visualized in figure 2.4: (1) The pyramidal-shaped cell body or soma, (2) the basal dendrites, connected to the lower part of the soma and (3) the apical dendrites, connected to the upper part of the soma. Note that also other compartment structures are possible, such as modelling each dendrite by a separate compartment. The three compartment structure is used in this thesis because of its minimal complexity while it still uses segregated dendritic compartments as observed in biology. The axon of the pyramidal neuron is connected to the soma (not shown in the figure) and the generation of action potentials is dependent on the voltage level in the soma. One supported idea is that in the primary sensory areas of the neocortex, the feedback connections from neurons in higher-order areas arrive in the apical dendritic compartment of the pyramidal neuron [18, 19, 20] and the feed-forward sensory information arrives at the basal dendrites [21, 22, 23]. In a pyramidal neuron, the feed-forward and feedback connections thus arrive in electrically distant compartments [8]. This feature of the pyramidal neuron will be used later in this thesis for a biological interpretation of the discussed learning algorithms.



(a)



(b)

Figure 2.4: **The pyramidal neuron.** (a) Anatomic drawing of the pyramidal neuron (figure adapted from [24]). (b) Schematic representation of a pyramidal neuron, with (A) the apical dendrites compartment, (B) the basal dendrites compartment and (S) the soma compartment.

2.1.2 The action potential

Neurons can transmit signals to other neurons via action potentials. An action potential thus can be seen as the information signal of the nervous system. In what follows, first, the resting membrane potential is explained as a balanced state between ion concentrations and overall electric charge. Afterwards, the mechanisms of an action potential are described as an interaction between electrically charged ions and voltage-gated ion channels. To conclude, the neural code and axon multiplexing, a recent hypothesis from theoretical neuroscience, are discussed.

Neuronal membrane at rest

The neuronal membrane forms a boundary between the extracellular fluid and the intracellular cytosol. The extracellular fluid and intracellular cytosol both consist out of electrically charged ions, dissolved in water. However, the concentration of the 4 most prominent ions (Na^+, K^+, Cl^- and Ca^{2+}) is different inside and outside of the cell, as can be seen in table 2.1. This results in a voltage difference, which is called the *membrane potential*. The concentration difference inside and outside of the membrane is controlled by ion pumps and ion channels in the neuronal membrane.

Table 2.1: **The ion concentrations inside and outside of the neuron.** The resulting resting potentials are displayed at the right [12].

Ion	**Concentration outside** [mM]	**Concentration inside** [mM]	**Ratio out:in**	$\boldsymbol{E_{ion}}$ (at 37°C)
K^+	5	100	1 : 20	$-80mV$
Na^+	150	15	10 : 1	$62mV$
Ca^{2+}	2	0.0002	10,000 : 1	$123mV$
Cl^-	150	13	11.5 : 1	$-65mV$

Ion channels. The neuronal membrane is full of ion channels. Each channel is specific to one or more ions (e.g. a potassium channel is only permeable for potassium ions). This property is called *ion selectivity*. Another important property of many ion channels is *gating*. Channels with this property can be opened and closed again by specific conditions in its environment. The most important gating conditions are intracellular voltage level, the binding to neurotransmitters (this gating condition is essential for the working of synapses, as discussed in section 2.1.3) and the binding to internal second messenger proteins such as the g-protein.

Ion pumps. Ion pumps make it possible for the neuron to pump ions against the concentration gradient to maintain a certain concentration difference. The ion pumps use energy from the breakdown of Adenosine Triphosphate (ATP) to pump out ions. Ion pumps have also the property of ion selectivity.

In a passive membrane (membrane without voltage dependent and neurotransmitter dependent ion channels) with different ion concentrations inside and outside of the membrane, the equilibrium concentration of ions inside and outside the membrane is governed by two forces. A diffusion force caused by the concentration difference and an electrical force caused by the voltage difference. When an ion specific channel opens, the membrane becomes permeable for that specific ion. The ions thus start moving through the channel according to the concentration gradient. However, because the ions are electrically charged, this causes a shift in the charge distribution which will create (or change) an electrical gradient which will influence the movement of the ions. When equilibrium is reached between these two forces, the membrane is at its resting potential E_{rest}. When the membrane is only permeable to one ion, the equilibrium potential E_{eq} is equal to the ionic equilibrium potential E_{ion}. The numeric values for E_{ion} can be calculated via the Nernst Equation [25] and are given in table 2.1. When the membrane is permeable to more than one ion, the resting potential can be calculated via the Goldman Equation [26]. The resting potential of a typical neuron is $-65mV$.

The action potential

The action potential is the information signal that makes it possible for neurons to communicate with each other. The action potentials generated by a neuron are all similar in size and duration and their amplitude stays the same while travelling down the axon (no leakage). The action potential is thus a robust communication signal, in which the information is encoded in the frequency and precise timing of the spikes (as will be further discussed in section 2.1.2 on the neural code). The action potential is generated by the interaction between electrically charged ions and voltage-gated ion channels. Its typical shape is shown in figure 2.5. An action potential is generated by the following chain of events:

1. **Resting state.** The neuronal membrane at rest is mostly permeable to K^+ and slightly permeable to $Na+$, resulting in a resting potential of $E_{rest} = -65mV$, close to the equilibrium potential of K^+ (see table 2.1).

2. **Depolarization above threshold.** When the voltage level in the soma is depolarized above the threshold $E_{th} = -40mV$, voltage-gated sodium channels open, making the membrane highly permeable to Na^+, relative to K^+. How the threshold voltage is reached, is dependent on the function of the neuron (e.g. visual stimuli for a neuron in the retina or input action potentials of other connected neurons for a neuron in the cerebral cortex).
3. **Rising phase.** Because of the large electrochemical gradient for Na^+ (compare the resting potential of the neuron with the equilibrium potential of Na^+ in table 2.1), a large influx of sodium ions rushes into the cell. This causes a rapid depolarization of the neuron, as visualized in figure 2.5.
4. **Overshoot.** As the membrane highly favours the permeability of sodium, the membrane potential rises close to E_{Na}, which is above $0mV$.
5. **Falling phase.** The repolarization is caused by the different time constants of the sodium and potassium voltage-gated channels. The sodium channels opened immediately after the voltage was above threshold. Now it closes again after $1ms$ (property of the channel) and cannot open again until the membrane is back again at resting potential. The potassium channel is activated by the change in voltage due to the sodium influx, and opens with a delay of approximate $1ms$ after the opening of the sodium channel. Now the membrane is again more permeable to potassium, and the depolarization gives a high driving force for potassium to enter the cell. This influx causes the membrane potential to become negative again.
6. **Undershoot.** The extra open voltage-gated potassium channels cause the membrane potential to go under its resting potential towards E_K, causing a hyperpolarization. When the voltage-gated potassium channels close again, the membrane goes back to its resting potential.
7. **Absolute refractory period.** The sodium channels cannot be activated again until the membrane potential goes sufficiently negative to de-inactivate the channels.

The action potential originates in the axon hillock (figure 2.2) and propagates forward through the axon, because the depolarization of one piece of the membrane causes the next piece of membrane to reach the threshold value, causing again an action potential. Note that the action potential cannot propagate both forward and backwards, because of the absolute refractory period. The velocity of an action potential varies from $1m/s$ to $100m/s$ and increases with axon diameter. In the mammalian brain, most of the axons are insulated by myelin, which greatly increases the velocity of action potentials. Because the general shape of the action potential is always the same, it can be represented by a binary *spike*, as is frequently done in the field of computational neuroscience. The next subsection discusses how a sequence of action potentials or a *spike train* can encode information.

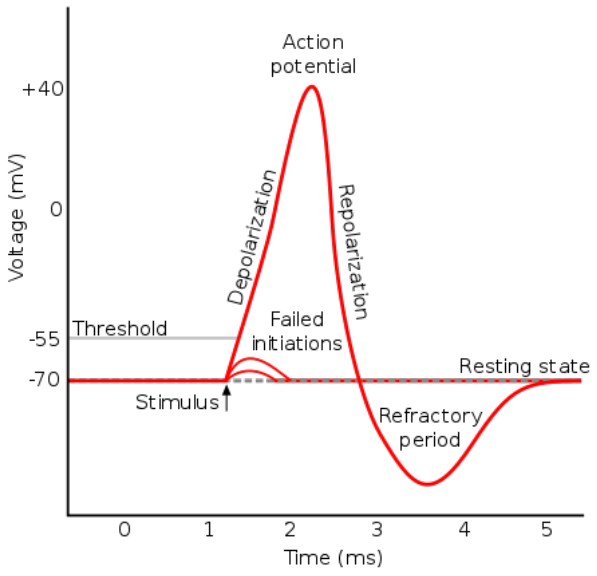


Figure 2.5: **The characteristic shape of an action potential.** Figure reprinted from [27]

Neural code

The *neural code* represents how our state and environment (e.g. emotions and visual input) are encoded by specific neurons in spike trains and how this information can be decoded from those spike trains. There exist two important paradigms on how information is carried in spike trains [13]:

1. **Rate coding.** In this paradigm, the information is carried by the firing rate of the neuron (spiking frequency). It is often assumed that the firing rate captures all the relevant information encoded in the spike train. Most artificial neural networks implicitly use rate coding to represent information, as the scalar activation value in each neuron represents the firing rate or the difference in firing rate from the baseline rate to allow for negative scalars.

2. **Temporal coding.** This paradigm may refer to several different ideas, all related to the specific timing of the spikes with respect to a reference signal.

   (a) **Phase.** The specific timing of a spike, relative to a background oscillation (e.g. the thalamocortical oscillations [28]) can carry information. This information is thus encoded in the phase of the background oscillation at the moment of the spike.

   (b) **Synchrony.** The synchronous or quasi-synchronous firing of neurons within and across ensembles can carry information. The information is thus not encoded in a single spike train, but in the co-firing of multiple spike trains from multiple neurons.

   (c) **Patterns.** Specific patterns in a spike train can also carry information.

Note that the different forms of neural code do not exclude each other. There might well be general information encoded in the firing rate of a neuron, while extra information is encoded in the phase, synchrony and/or pattern of the spike train. In this thesis, we will use rate coding to represent the information in spike trains, as this gives us a straight-forward mathematical scheme of scalar activation values for each neuron.

Multiplexing through bursting spikes

An interesting phenomenon in cortical pyramidal neurons is the occurrence of plateau potentials, resulting in bursting spikes, as illustrated in figure 2.6 [21]. As explained in section 2.1.1, the apical dendrites are electrically distant from the soma. When a small apical input current occurs, the current will leak away before it can reach the soma (figure 2.6b) and thus before it can cause an action potential. If the soma generates an action potential (e.g. because of a basal input above spiking threshold), this action potential propagates back through the apical dendrites, with a decreased amplitude but increased width (figure 2.6c). If a backpropagating spike and a significant apical input at the apical dendrites occur simultaneously, this gives rise to a plateau potential in the apical dendrite, which travels down to the soma (figure 2.6d). This plateau current input in the soma results in a burst firing (fast firing sequence of a small group of spikes). The initial spike of the soma, (e.g. caused by the basal input), is thus transformed into a bursting spike group due to the apical inputs. If a very high apical input occurs, it can cause a bursting spike group without the need for an initial spike in the soma (figure 2.6e).

Multiplexing. The bursting ability of pyramidal neurons gave rise to the hypothesis from theoretical neuroscience that neurons are able to multiplex two distinct information signals through the same axon [29]. Note that this hypothesis is not yet confirmed by experimental results.

1. **Event rate.** A group of bursting spikes, or a single spike if no burst occurs, counts as one event. This *event rate* correlates the most with the basal inputs, as this input causes the initial spike that starts an event (burst or single spike).

2. **Burst probability.** The probability that a single initial spike will result in a bursting spike is the *burst probability*. It is calculated as the ratio between the burst rate (average number of bursting groups) over the event rate. The burst rate correlates most with the apical inputs, as they cause the plateau potential to arise when an initial spike from the soma propagates back to the apical dendrites.

The pyramidal neuron can thus communicate information related to its basal and apical inputs to other neurons through multiplexing.

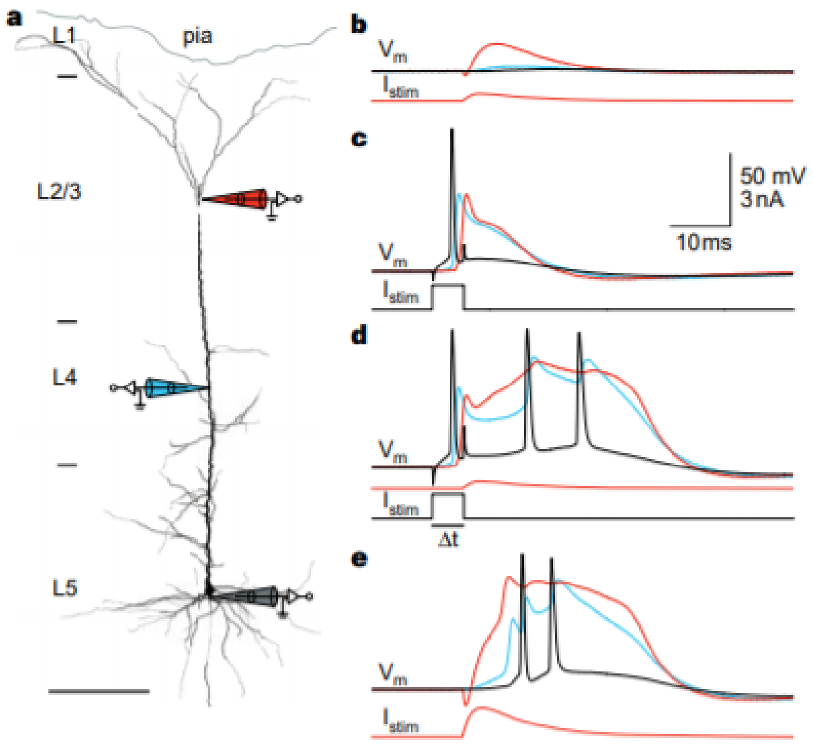


Figure 2.6: **Apical dendritic input can transform a single somatic spike into a burst ensemble of spikes, by evoking a *plateau potential*.** (a) Reconstruction of a pyramidal neuron, with the 3 recording pipettes shown in grey, blue and red, resp. for $0\mu m$, $400\mu m$ and $770\mu m$ from the soma. (b) The apical input (red) did not reach threshold for either a plateau potential in the apical dendrites or an action potential in the soma. (c) an action potential evoked in the soma backpropagates through the apical dendrites, decreasing in amplitude but increasing in width. (d) The combination of the same small apical input of (b) with a backpropagating action potential of (c) gives rise to a plateau potential in the apical dendrites, which results in a bursting action potential in the soma. (e) if the apical input is high enough, it can also generate a plateau potential and resulting bursting spikes in the soma on its own. Figure reprinted from [21].

Decoding. For multiplexing to work, the connected neurons need to have a mechanism in place to decode the multiplexed signal. This role can be fulfilled by short-term synaptic facilitation (STF) and short-term synaptic depression (STD) [29]. Both STF and STD are explained in more detail in section 2.1.4. STD acts as a low-pass filter, making only the first spike of each event contribute significantly to the input current, and thus STD can filter out the event rate. STF acts as a high-pass filter, causing the burst groups to have a much larger effect on the input current compared to a single spike. The burst probability can then be decoded by a combination of STF and a divisive feedforward inhibition from an event rate decoder [29].

Neural pathways. In section 2.1.1 it was mentioned that the apical dendrites receive mostly feedback inputs and the basal dendrites receive mostly feed-forward inputs. If the apical dendrites decode the burst probability input and if the basal dendrites decode the event rate, this can give rise to two separate signalling pathways through the neural networks.

1. **Feed-forward path.** If the basal dendrites only decode the event rates, its inputs will correspond with the event rates of incoming neurons. As the event rate of the current neuron correlates mostly with the basal inputs, the event rate of the current neuron will correspond to the event rates of the incoming neurons. This is the *feed-forward path*, as mostly feed-forward inputs arrive at the basal dendrites.

2. **Feed-backward path.** If the apical dendrites only decode the burst probabilities, its inputs will correspond with the burst probabilities of incoming neurons. As the burst probability of the current

neuron correlates mostly with its apical inputs, the burst probability of the current neuron will correspond to the burst probabilities of the incoming neurons. This is the *feed-backward path*, as mostly feedback inputs arrive at the apical dendrites.

2.1.3 The synapse

A neuron on its own cannot produce the vast complexity of our neural processing, such as our image and audio processing, reflexes and consciousness. For doing such complex processes, the neurons have to be able to communicate with each other, forming networks of neurons. This communication of signals runs through the synapse, a small area of the neuron that connects the axon terminal of one neuron with a dendrite of another neuron.

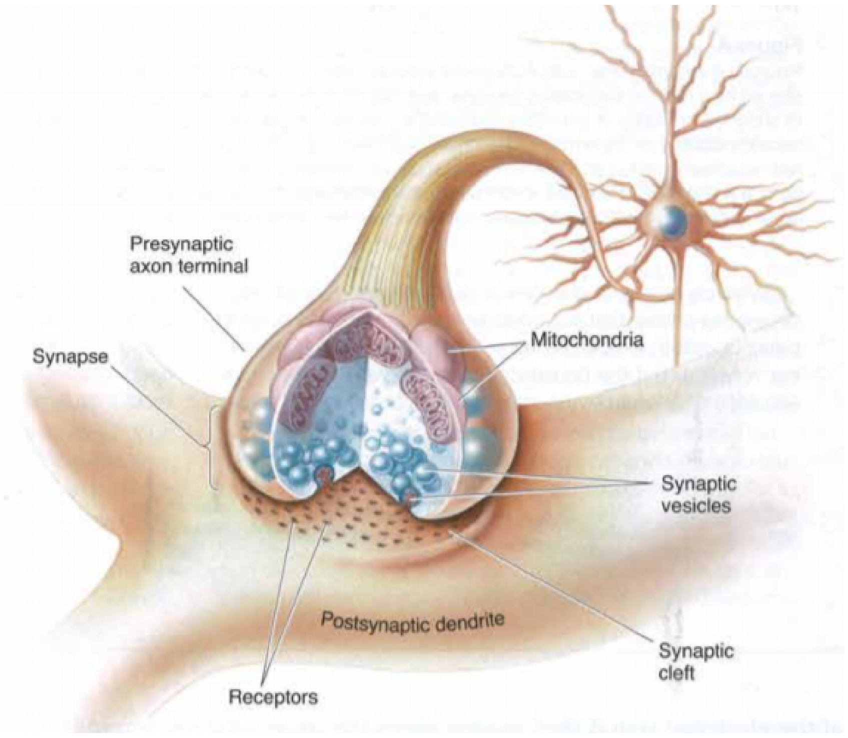


Figure 2.7: **The axon terminal and the chemical synapse.** The axon terminals form synapses with the somata or dendrites of other neurons. After an action potential arrives at the presynaptic axon terminal, neurotransmitters are released from synaptic vesicles into the synaptic cleft. When the neurotransmitters reach the post-synaptic cell, they bind to specific receptors, causing the generation of electrical and chemical signals the cell. Figure reprinted from [12].

The axon terminal connects to the dendrite through a synapse, as shown in figure 2.7. The synapse consists of two sides, *presynaptic* and *postsynaptic*, indicating the information flow through a neuron: from axon terminal (*pre*) to dendrite (*post*). There exist two types of synapses: an electrical synapse, better known as a *gap junction*, and a chemical synapse.

Electrical synapse. In the electrical synapse, the presynaptic axon terminal and the postsynaptic dendrite are electrically coupled, resulting in a fast and reliable transmission of the whole presynaptic action potential to the postsynaptic neuron (if the presynaptic neuron produces an action potential, it will almost always cause an action potential in the postsynaptic neuron). This type of synapse is used for producing reflexes and other signals that must be fail-safe and precise in time such as the heartbeat. The electrical synapse ensures that the connected neurons are strongly synchronised, and evidence suggests that this property is used in early brain development to help to coordinate the growth of the brain [30, 31].

Chemical synapse. The vast majority of the synapses in the adult brain are chemical synapses. As visualized in figure 2.7, the chemical synapse consists of a presynaptic axon terminal, a synaptic cleft and a postsynaptic dendrite. When an action potential arrives in the presynaptic axon terminal, voltage-gated calcium channels open and a strong influx of calcium ions into the axon terminal occurs. This influx causes the synaptic vesicles to fuse with the presynaptic membrane at the synaptic cleft, resulting in a release of their neurotransmitters into the synaptic cleft in a process called *exocytosis* [32]. There exist a wide variety of neurotransmitters in the brain, with glutamate, gamma-aminobutyric acid (GABA), dopamine, serotonin and norepinephrine (NE) the most important ones. The neurotransmitters diffuse through the synaptic cleft and the majority reaches the neurotransmitter-receptors, located in the postsynaptic membrane of the dendrite. There, the neurotransmitters bind to the receptors and depending on the type of receptor, this can have various effects on the postsynaptic neuron:

- **Transmitter-gated ion channels**: When the corresponding neurotransmitter binds to a receptor of this type, the channel opens for one or more types of ions, causing an influx or efflux of ions in/out of the postsynaptic dendrite. When the channel is permeable to Na^+, the postsynaptic voltage will rise. When the channel is permeable to Cl^-, the postsynaptic voltage will lower. These two channels are respectively called excitatory and inhibitory channels. Transmitter-gated ion channels provide a fast chemical synaptic transmission.

- **G-protein-coupled receptors**: Almost all the neurotransmitters can also have a slower, longer-lasting and much more diverse effect on the postsynaptic membrane via G-protein-coupled receptors [33]. This type of transmitter-receptor interaction involves three steps: (1) the neurotransmitter binds to the receptor protein, (2) the receptor activates small proteins, called *G-proteins*, that can now move freely along the postsynaptic membrane and finally (3) the activated G-proteins activate *effector proteins*, also known as *secondary messengers*. These secondary messengers can diffuse into the cytosol and they have widespread metabolic effects, other than influencing the postsynaptic voltage.

- **Voltage-gated ion channels:** As the neurotransmitters influence the permeability of the post-synaptic membrane and thus also the post-synaptic voltage level, voltage-gated ion channels are also influenced indirectly by the neurotransmitters. The NMDA[1] receptor in the post-synaptic membrane, for example, is strongly voltage dependent and plays an important role in synaptic plasticity.

The chemical synapse has two important properties, that give the brain the power to perform complex computations and signal processing:

1. In contrast to the electrical synapse, which transmits the incoming action potential directly to the postsynaptic neuron, the signal through a chemical synapse will only have a small contribution to the postsynaptic somatic voltage. The postsynaptic neuron thus needs a lot of simultaneous inputs from various chemical synapses to induce a somatic voltage above threshold for producing an action potential. On top of that, the strength (the change in postsynaptic voltage resulting from a presynaptic action potential) of the chemical synapses are different among synapses. This feature makes sure that neurons only produce action potentials when specific input patterns occur at their dendrites.

2. The strength of a chemical synapse can be changed over time. This mechanism is called *synaptic plasticity* and will be discussed in more detail in section 2.1.4. This synaptic plasticity is essential for learning and forming memories, as experiences can be encoded in the changes of synaptic strengths in the brain. From a mathematical point of view, the brain can be seen as a model that learns to interpret its inputs (environment and state) by changing its model parameters (synaptic strengths). The human brain contains roughly 100 billion neurons. Each neuron has on average thousands of synapses, making the brain a model with over 10^{14} parameters.

2.1.4 Synaptic plasticity

Synaptic plasticity refers to the activity-dependent modification of the strength of synaptic transmission at pre-existing synapses [34]. It gives a mammalian brain the capacity to modify its neural circuit function depending on the neural activity generated by experience and thereby altering its thoughts, feelings and behaviour. Plasticity can either enhance or depress the synaptic transmission, and the timescale of its effect can be short term or long term. This section provides an overview of the different known forms of synaptic plasticity. Note that almost all synapses in the mammalian brain simultaneously express a number of different forms of synaptic plasticity, making the resulting plasticity of the synapse an even more complex process. The different forms of synaptic plasticity result almost always in one or both of the

[1]N-methyl-D-aspartate

two following outcomes: (1) modification of the efficacy of neurotransmitter release at the presynaptic side or (2) modification in the amount of neurotransmitter-gated ion channels at the postsynaptic side. These two outcomes should be a guideline for the reader through this section.

Short-term synaptic plasticity

Short-term synaptic plasticity acts on a timescale in the order of milliseconds to several minutes. These forms of plasticity are thought to play important roles in the short-term adaptation to sensory inputs, short-lasting forms of memory and transient changes in behaviour [34]. Short-term facilitation (STF) (increased synaptic strength) and short-term depression (STD) (decreased synaptic strength) are mostly linked to a calcium build-up in the presynaptic terminal and the depletion of release-ready neurotransmitter vesicles available [35], but it is likely that additional mechanisms are involved such as the involvement of glial cells

Short-term facilitation. When an action potential arrives shortly after a previous action potential, there will still be some residual calcium ions present in the presynaptic terminal from the previous spike, leading to a higher resulting calcium level after the current spike. This, in turn, leads to higher efficacy of the neurotransmitter release.

Short-term depression. There are only a limited amount of neurotransmitter vesicles available in the presynaptic terminals, thus neurotransmitters need to be recycled or reproduced and encapsulated in new vesicles. This process is referred to as the synaptic vesicle cycle [32]. When a high amount of action potentials arrive at the synapse in a short time, the synaptic vesicle cycle is not fast enough to keep up with the high demand for neurotransmitters, resulting in a decrease in released neurotransmitters.

Short-term synaptic plasticity thus occurs mostly at the presynaptic terminal by modifying the neurotransmitter release efficacy. The function of short-term plasticity can be seen as a filtering function. For example, synapses with a low initial probability to release neurotransmitters, act as a high-pass filter. Due to the calcium build-up with high-frequency spike trains, the neurotransmitter release probability will increase for high-frequency signals. On the other hand, synapses with a high initial release probability will act as a low-pass filter, as low-frequency signals pass without a problem (due to high release probability), but high-frequency signals are attenuated due to the vesicle depletion. Moreover, the filtering characteristics of the synapse can be altered by changing the initial release probability. This can be done by neuromodulators that activate specific presynaptic receptors that reduce the release probability.

Long-term synaptic plasticity

In order to form memories, to learn new skills and to learn from its mistakes, the brain needs long-lasting synaptic changes. This hypothesis was put forward over 100 years ago by the Nobel Laureate Santiago Ramon y Cajal and was further investigated by Donald Hebb, who proposed that associative memories could be formed by a plasticity process nowadays called *Hebbian learning* [36]. Bliss and colleagues gave in the early 1970s the experimental support for the existence of long-lasting synaptic plasticity in the mammalian brain [37, 38]. In long-term synaptic plasticity, a distinction is made between long-term potentiation (LTP) which strengthens the synapse and long-term depression (LTD) which weakens the synapse. It is well established that synapses that exhibit LTP also exhibit LTD, making it possible to modify the synapses in both directions. Besides LTP and LTD, more recently discovered forms of long-term plasticity exist, including homeostatic plasticity [39] and metaplasticity [40]. The most common forms of LTP and LTD are dependent on the NMDA[2] receptor (NMDAR) in the postsynaptic membrane [41] and will be further discussed. Note that there exist other less common mechanisms for LTP and LTD, such as metabotropic glutamate receptor-dependent LTD [42] and endocannabinoid-mediated LTD [43].

Long-term potentiation. Two receptors play an important role for LTP and LTD: the AMPA[3] receptor (AMPAR) and NMDA receptor (NMDAR). They are both receptors for the neurotransmitter glutamate. AMPAR is permeable to Na^+ and K^+ and provides most of the inward current for the excitatory synaptic response when the postsynaptic cell is around resting potential. NMDAR is mostly permeable to Ca^{2+} and Na^+, but is blocked by extracellular magnesium at resting potential (voltage-dependent). NMDAR acts as

[2]N-methyl-D-aspartate

[3]α-amino-3-hydroxy-5-methyl-4-isoxazolepropionic acid

a 'coincidence detector' for presynaptic and postsynaptic firing: If (1) the postsynaptic neuron is strongly depolarized (close to threshold), which removes the magnesium blockade of the NMDAR and if (2) the presynaptic neuron spikes, which releases glutamate in the synaptic cleft, the NMDAR opens and calcium ions can enter the postsynaptic cell. This calcium influx acts as the coincidence signal. If the calcium level in the postsynaptic terminal is above a certain threshold, it initiates a chain of reactions which ultimately results in the creation of extra AMPAR in the postsynaptic membrane and in an increase of the existing AMPARs conductances [34]. This increase in total AMPA receptors and individual conductances leads to a stronger synaptic connection.

Long-term depression. When the calcium level in the postsynaptic terminal is below a certain upper threshold and above a certain bottom threshold –indicating that (1) the postsynaptic neuron is only slightly depolarized, which slightly removes the magnesium blockage and (2) the presynaptic neuron has fired, releasing glutamate in the synaptic cleft–, a chain of reactions is initiated that ultimately results in a decrease of AMPAR in the postsynaptic membrane [44, 34]. This decrease in the amount of AMPA receptors results in the long-term depression of the synapse.

Calcium as correlation signal. The quantitative properties of the postsynaptic calcium signal thus dictate whether LTP or LTD occurs in the synapse, with LTD requiring a modest calcium level and LTP a calcium level above a certain threshold. The calcium level indicates the amount of correlation between the presynaptic spikes and postsynaptic spikes: if there is a presynaptic spike without a postsynaptic spike, there will be a low calcium level, indicating LTD, whereas if there is both a presynaptic and postsynaptic spike at the same moment, there will be a high calcium level, resulting in LTP. This link with the pre- and postsynaptic firing led to the proposition of the *Hebbian learning* paradigm.

Hebbian learning rules. The dependence of LTD and LTP on the co-occurrence of pre-synaptic spiking and postsynaptic voltage level are investigated in theoretical neuroscience under the branch *Hebbian learning rules*. It is hypothesized that a synapse w_{ij} has an internal state e_{ij}, called a *synaptic flag* or an *eligibility trace*. The synaptic plasticity then depends on this synaptic inner state, in most cases linearly [45]:

$$\frac{d}{dt}w_{ij} = c \cdot e_{ij} \tag{2.1}$$

with c a fixed constant. In Hebbian learning, the dynamics of the synaptic flag e_{ij} depend only on two factors: (1) the presynaptic factor x_j (indicating the firing activity of the presynaptic neuron or the amount of neurotransmitters in the cleft) and the postsynaptic factor y_i (indicating the state of the postsynaptic neuron) [45]. This is formulated mathematically as

$$\frac{d}{dt}e_{ij} = \eta f_j(x_j)g_i(y_i) - e_{ij}/\tau_e \tag{2.2}$$

with η a constant learning rate, τ_e a decay time constant and f_j and g_i two arbitrary functions. A specific type of Hebbian learning, *spike-timing dependent plasticity* [46], deserves some more details. It is experimentally shown (reviewed in [47, 48]) that LTP and LTD strongly depend on the time difference between the pre- and postsynaptic spike timing, as shown in figure 2.8. LTP and LTD have 'critical windows' around the postsynaptic spike in which the presynaptic spike must occur. The order of pre- and postsynaptic spike decides whether LTP or LTD is occurring, and the closer they are to each other, the stronger the effect. Although the mechanisms of STDP rely on the same principles as explained in the previous section on LTP and LTD, it's precise shape is more complicated to explain. For a detailed review of STDP, see [47, 48].

Three-factor learning rules. More recent experimental studies, reviewed in [45], support the hypothesis that the eligibility trace e_{ij} does not lead directly to a synaptic change, but only if a third factor is involved. This third factor is most commonly a neuromodulator such as dopamine, indicating reward or surprise, but can also be a local calcium spike from the apical dendrites. These three-factor learning rules, also called *neoHebbian* learning rules, can be mathematically formulated as follows,

$$\frac{d}{dt}w_{ij} = e_{ij}M^{3rd}(t), \tag{2.3}$$

with M^{3rd} a time-varying third factor. This third factor is most commonly a global signal, as reward and surprise are also global signals, but can also be cell specific, such as the calcium spike signal. The

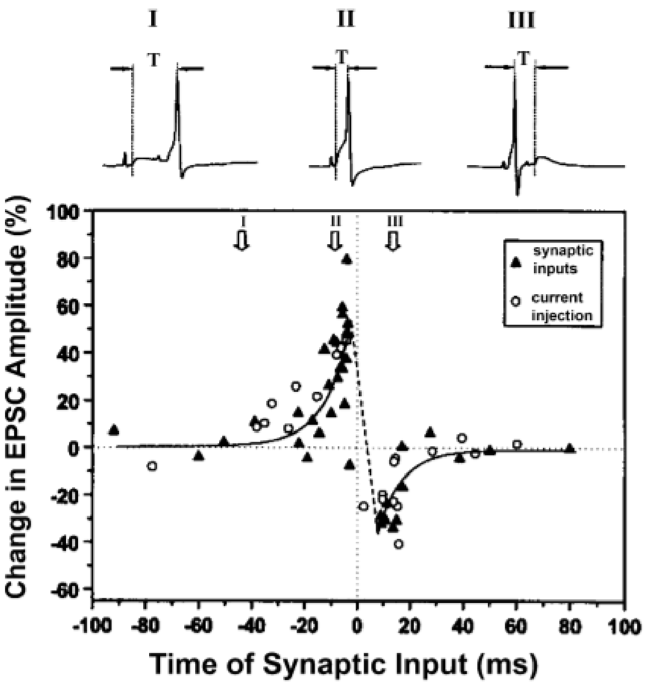


Figure 2.8: **"The critical window for spike-timing-dependent plasticity of developing retinotectal synapses."** "The percentage change in the synaptic strength (excitatory postsynaptic potential (EPSP) amplitude) after repetitive retinal stimulation was plotted against the onset time of the retinal stimulation relative to the peak of the action potential initiated in the tectal cell. Data shown are for experiments in which spiking of the tectal neuron was initiated by either a suprathreshold input or a group of coactive inputs (filled circles) or by injection of a depolarizing current (open circles) in a tectal neuron". Figure cited and reprinted from [47].

use of reward and surprise signals resembles the reinforcement learning paradigm in machine learning. However, the neohebbian learning rules are locally controlled, compared to reinforcement learning where a global policy is optimized. It is therefore not guaranteed that the three-factor learning rules optimize a global policy or cost function, which is the cause of their yet poor performance on machine learning tasks. There is thus still a vast area to discover on how synaptic plasticity is regulated and how this makes us humans able to learn and create memories.

Homeostatic Plasticity. Without any form of stabilizing mechanisms, activity-dependent plasticity rules such as LTP and LTD are unstable and would lead to epileptogenic excitation (constant firing) or total neural silence. *Synaptic scaling,* a form of homeostatic plasticity counters this unstable behaviour by globally changing the strength of all synapses of a neuron. When the activity of a neuron is dramatically decreased for a long period of time, synaptic scaling increases the strength of all the synapses of the neuron, whereas when the activity is dramatically increased for a long period of time, the synapses are scaled down [49]. Important to notice is that all the relative strengths of individual synapses stay the same after scaling. The precise molecular mechanisms of synaptic scaling are not yet fully understood [34].

Metaplasticity. Metaplasticity refers to 'plasticity of plasticity': it is the higher-order plasticity, that does not directly modify the synaptic strength, but modifies the magnitude and direction of activity-dependent plasticity that takes place in the synapse [40]. For example, metaplasticity can cause shifts in the calcium thresholds for LTP and LTD in neurons of the visual cortex during development [50].

2.2 Artificial neural networks

In recent years, artificial neural networks (ANNs), the most common form of deep learning, have caused a revolution in machine learning, computer vision, speech recognition and many other fields. Deep learning is a form of machine learning, where the computational models are composed of multiple processing layers, which gives them the ability to learn representations of the data with multiple levels of abstraction [1]. The architecture of an artificial neural network is inspired by biological neural networks. Abstract neurons are connected with each other in multiple organized layers and information can flow through the network. The processing characteristics of the ANN depend on the connection weights (synapses) between the different abstract neurons. This section first discusses the perceptron, an abstract neuron model used in most of the ANNs, after which the most basic form of an ANN, the feedforward multi-layer perceptron (MLP) is explained. Thereafter, the backpropagation algorithm, the method of choice to train multi-layer perceptrons, is discussed. Finally, a more advanced form of ANNs, auto-encoders, are briefly analysed. Note that there are many other deep learning models, such as recurrent neural networks, energy-based neural networks and convolutional neural networks, that will not be discussed in this thesis, as they are not necessary to investigate the fundamental learning properties of networks, which is the topic of this thesis.

2.2.1 The abstract neuron

As seen in the previous section, a biological neuron is a very complex cell with complex input-output relations. In machine learning, this complexity is omitted by representing the neuron as an abstract unit with input weights w_i, an internal voltage level a and an output firing rate h, as shown in figure 2.9. This abstract neuron is more commonly known as the *perceptron* [51, 52], which caused a wave of excitement in the 1960s on the information processing capabilities of neurons. The abstract neuron used in this thesis makes thus use of rate coding as neural code. The internal voltage level a is calculated as

$$a = \sum_i w_i x_i + w_0 \tag{2.4}$$

with w_0 the bias term. In a biological setting, this bias term is equal to the negative threshold of the neuron, such that the neuron fires when $a \geq 0$ and is silent when $a < 0$. In a machine learning setting, the bias can have any value and is learned during training of the network. The output firing rate h of the neuron is a nonlinear function of the internal voltage level:

$$h = s(a) \tag{2.5}$$

with s an arbitrarily nonlinear function. In a biological setting, the relation s between the output firing rate and the inner voltage level is typically as shown in figure 2.10. In a machine learning setting, various nonlinear functions are used for s, most commonly the Rectified Linear Unit (ReLU), the sigmoid or the tangens hyperbolicus.

2.2.2 Feed-forward neural network model

One of the most straightforward forms of artificial neural networks is a feed-forward neural network as shown in figure 2.11a. The network consists of multiple layers of abstract neurons, with only feed-forward connections (connection between a neuron of layer i with a neuron of layer $i+1$). As there are no inner layer connections or recurrent connections, the information can only flow in the forward direction. The following notation is used in this thesis: h_i^j is the output value of the jth neuron of the ith layer. h_0^j are the inputs of the feed-forward network (e.g. pixel values of a picture) and h_L^j, with L the last layer of the network, are the outputs of the feed-forward network (e.g. the probability that the picture contains a cat). All other layers are called *hidden layers*, as they learn representations of the data without direct external supervision. Typically a neuron of layer i is connected to all neurons of the previous layer $i-1$. This network setting is called a *fully connected network*. A feed-forward network can be represented as a vector for each layer, with as entries all the neurons in the specific layer, as visualized in figure 2.11b. The relation between layers in the network can then be written as follows:

$$\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(\boldsymbol{a}_i) = s_i(W_i \boldsymbol{h}_{i-1} + \boldsymbol{b}_i), \tag{2.6}$$

with f_i the umbrella layer function of layer i for ease of notation, s_i the non-linearity of layer i, W_i the weight matrix of the weights connecting layer $i-1$ with layer i, $\boldsymbol{b}_i$ the bias vector and $\boldsymbol{h}_i$ the outputs of layer i. In this thesis, vectors will be written in bold and matrices with a capital letter. As an exception, loss functions L will also be written with a capital letter, although they are a scalar.

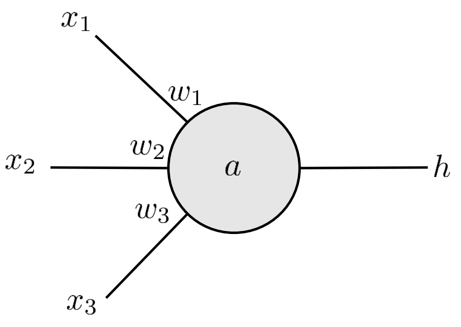


Figure 2.9: **Abstract representation of a neuron, used in most artificial neural networks.** x_i's represent the inputs of the abstract neuron, which can either be other neurons or external inputs to the network. w_i's are the synaptic strengths, a the internal voltage level and h the output firing rate.

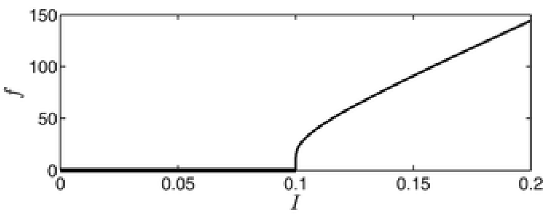
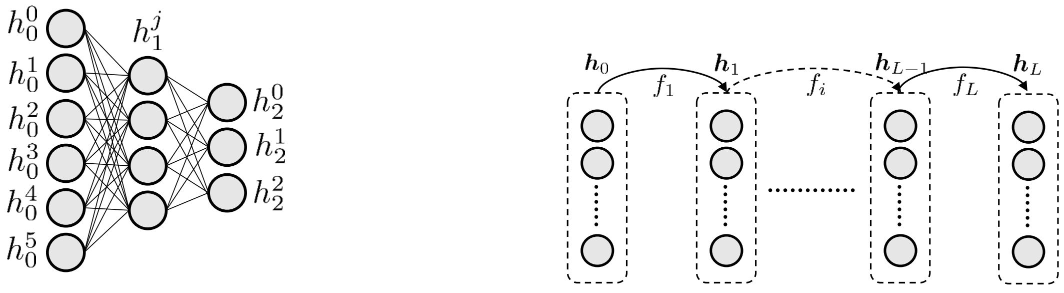


Figure 2.10: **Typical relation between input current I $[nA]$ of a neuron and its output firing rate f $[Hz]$.** Figure reprinted from [53].



(a)

(b)

Figure 2.11: **The feed-forward network.** (a) Structure and notation of a feed-forward network with one hidden layer and (b) vectorized structure and notation of a feed-forward neural network.

2.2.3 Training artificial neural networks

In machine learning, a distinction is made between three kinds of learning methods, depending on the available training data:

- **Supervised learning.** If the dataset used to train the model consists of true input-output pairs $\{(\boldsymbol{h}_0, \hat{\boldsymbol{h}}_L)_i\}$, there is a teaching output $\hat{\boldsymbol{h}}_L$ available to tell the model what the correct output is supposed to be. This type of dataset is commonly called a *labeled* dataset. In supervised learning, all data points and labels are available at the start of the training.

- **Unsupervised learning.** If the dataset only consists of input samples (e.g. a large dataset of unlabeled animal pictures), the model cannot use teaching signals to learn. In unsupervised learning, the model has to learn the underlying structure of the data by evaluating the statistics of the dataset, without knowing what its output is supposed to be. *Self-supervised learning*, a subset of unsupervised learning, uses the input samples for comparison with the outputs of the model. Auto-encoders are a common example of self-supervised learning.

- **Reinforcement learning.** In reinforcement learning, no real dataset is available at the beginning of training, but instead, an environment is given that the agent (machine learning algorithm) can explore to gather its training data. An intuitive example of reinforcement learning is a video game. The agent (player) can explore the virtual world, gathering new information with every action it takes. After a certain sequence of actions, the agent can get a feedback signal (e.g. reach the next level in the game). Reinforcement learning (RL) differs from supervised learning (SL) for two main reasons: (1) in RL, the algorithm can choose which data points it wants to explore, in contrast with SL, where all data is given at the beginning of training and (2) the feedback signal in RL is typically in a very low dimension (e.g. a scalar indicating reward), whereas in SL, the feedback signal can have higher dimensions.

This thesis will mainly focus on supervised and unsupervised learning. The typical workflow of training an ANN is visualized in figure 2.12. The available data should be split into three independent sets before the start of the training: (1) the training data, used for training the ANN, (2) the validation data, used for choosing the optimal model architecture and other hyperparameters and (3) the test data, used for evaluating the final chosen model on data that it has never seen before. After that, the model is designed: a suitable loss function is chosen and the hyperparameters such as the number of layers and neurons and the regularization parameters are determined. Now the designed model can be trained on the training data. This is typically done by an iterative non-linear optimization scheme with the following two steps per iteration: (1) compute the gradient of the loss function with respect to the neuron weights and biases (done by the back-propagation algorithm) and (2) use those gradients to perform an optimization step, most commonly with stochastic gradient descent [1]. The trained model is then validated on unseen data from the validation dataset. In most cases, multiple model designs are trained, and the performance of all the trained models are tested on the validation dataset. The best model is picked out, and finally, this model is evaluated on the test dataset, to check if its performance is consistent with the previous performance on the validation dataset (to prevent overfitting on the validation data). In the following paragraph, three important steps of this workflow are explained in more detail: designing the loss function, computing the gradients with the backpropagation algorithm and performing the optimization step.

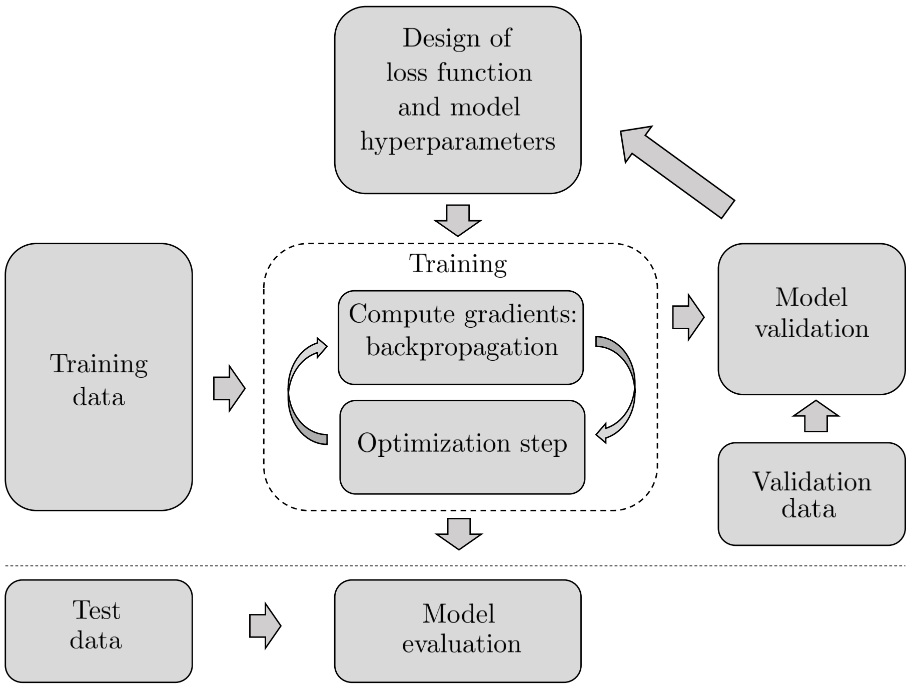


Figure 2.12: **The machine learning work-flow, applied on deep learning.**

Loss functions

A loss function indicates what you want to minimize during training or in other words, what you want your network to learn. In a supervised learning setting, the outputs (or intermediate outputs) $\boldsymbol{h}_L$ of the network are compared to the true labels $\hat{\boldsymbol{h}}_L$ of the data by means of a loss function $L(\boldsymbol{h}_L, \hat{\boldsymbol{h}}_L)$. The most commonly used loss functions for supervised learning are given in table 2.2. In an unsupervised learning setting, defining the cost function is not that straightforward. Typically, custom loss functions are made for each type of unsupervised ANN. For example, autoencoders use the input samples as a teaching signal for the output layer [54] and generative adversarial networks (GANs) [55] use a custom GAN loss e.g. the Wasserstein distance [56]. An ANN is trained on a dataset which consists of many samples (either input-output pairs or only input samples). The total loss of the network is then computed as the sum of all sample losses:

$$L_{tot} = \sum_i L^{(i)} \tag{2.7}$$

Table 2.2: **The most commonly used loss functions for supervised learning with deep neural networks.** $\boldsymbol{h}_L$ is the output of the network and $\hat{\boldsymbol{h}}_L$ is the target of the output.

Loss function	Formula	Usage
L_2	$\|\boldsymbol{h}_L - \hat{\boldsymbol{h}}_L\|_2^2$	Regression with a linear output layer
L_1	$\|\boldsymbol{h}_L - \hat{\boldsymbol{h}}_L\|_1$	Regression with a linear output layer
Cross-entropy loss	$-\sum_c^M \hat{\boldsymbol{h}}_{L,c} \log \boldsymbol{h}_{L,c}$	Classification with a softmax output layer

Backpropagation

Once the loss function is defined, the loss of each input-output pair of the dataset can be computed. To know in which direction the parameters should be tweaked by the optimization algorithm, the gradients of each loss to the model parameters should be calculated. This is typically done by the workhorse of deep learning: *backpropagation of the error*, backpropagation in short, also known as the *generalized delta rule* [57, 3]. The backpropagation algorithm is based on the chain rule of derivatives. First let us define $\boldsymbol{\delta}_i$ as the gradient of the loss function to the linear activation $\boldsymbol{a}_i$ of layer i.

$$\boldsymbol{\delta}_i \triangleq \left(\frac{\partial L}{\partial \boldsymbol{a}_i}\right)^T = \left(\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{a}_i}\right)^T \left(\frac{\partial \boldsymbol{a}_{i+1}}{\partial \boldsymbol{h}_i}\right)^T \left(\frac{\partial L}{\partial \boldsymbol{a}_{i+1}}\right)^T = J^T_{\boldsymbol{h}_i,\boldsymbol{a}_i} J^T_{\boldsymbol{a}_{i+1},\boldsymbol{h}_i} \boldsymbol{\delta}_{i+1} = D^T_{s_i} W^T_{i+1} \boldsymbol{\delta}_{i+1} \tag{2.8}$$

with J the Jacobian matrix and D_{s_i} diagonal matrix containing the partial derivatives of the nonlinear activation function $s(\boldsymbol{a}_i)$ to the linear activation of the layer $\boldsymbol{a}_i$. Note that because s' is a diagonal matrix, it can be replaced by a vectorized form to speed up computations. $\boldsymbol{\delta}_i$ is often called an *error signal*, as it represents the direction in which the output error (loss) increases the most by changing the linear activation $\boldsymbol{a}_i$. The magnitude of $\boldsymbol{\delta}_i$ indicates the magnitude of the increase. However, $\boldsymbol{\delta}_i$ are in general not pure error signals, but gradients, so I will call them simply helper variables to prevent confusion. These helper variables can be cheaply computed recursively from the next layer's $\boldsymbol{\delta}_{i+1}$. For ease of notation, we define the function g_i to perform the above computations:

$$\boldsymbol{\delta}_i = g_i(\boldsymbol{\delta}_{i+1}) = D^T_{s_i} W^T_{i+1} \boldsymbol{\delta}_{i+1} \tag{2.9}$$

The recursive process of computing the helper variables is shown in figure 2.13. Now that the helper variables $\boldsymbol{\delta}_i$ are computed, it is straightforward to compute the gradients of the loss function to the model weights and biases:

$$\frac{\partial L}{\partial W_i} = \frac{\partial L}{\partial \boldsymbol{a}_i} \frac{\partial \boldsymbol{a}_i}{\partial W_i} = \boldsymbol{\delta}_i \boldsymbol{h}^T_{i-1} \tag{2.10}$$

$$\frac{\partial L}{\partial \boldsymbol{b}_i} = \frac{\partial L}{\partial \boldsymbol{a}_i} \frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{b}_i} = \boldsymbol{\delta}_i \tag{2.11}$$

The rightmost equalities hold because of the sparse structure of $\frac{\partial \boldsymbol{a}_i}{\partial W_i}$ and that $\frac{\partial \boldsymbol{a}_i}{\partial \boldsymbol{b}_i}$ is equal to the identity matrix.

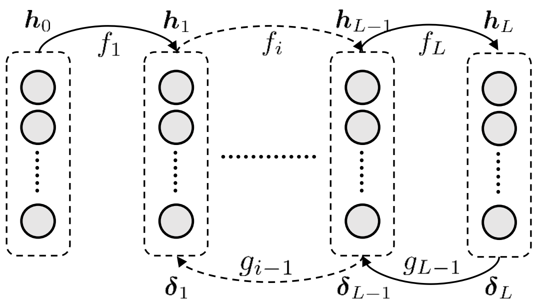


Figure 2.13: **Schematic network representation of backpropagation.**

Optimization

An artificial neural network is trained by minimizing its loss function. The most straightforward method is *gradient descent*. The weight updates are then given by:

$$\Delta W_i = -\eta \frac{\partial L_{tot}}{\partial W_i} \tag{2.12}$$

$$\Delta \boldsymbol{b}_i = -\eta \frac{\partial L_{tot}}{\partial \boldsymbol{b}_i} \tag{2.13}$$

with the gradients computed by backpropagation. In deep learning, usually very large training datasets are used. This makes it very computationally expensive to compute the gradient of all the data samples for each training iteration. Together with the need to overcome local minima, this problem gave rise to *stochastic gradient descent* (SGD). In this optimization scheme, the gradient of the loss function for a single sample is used as an estimate of the gradient of the total loss function in each training iteration. SGD with mini-batches, a variant of normal SGD, uses a small subset of the dataset –called a *mini-batch*– to estimate the gradient. When the optimization algorithm has gone through all the data once, it is called an *epoch*. Usually, several epochs are needed to train an ANN. Note that also more elaborated optimization algorithms can be used to train ANNs, such as second order (or approximate second order) optimization strategies. However, variants of stochastic gradient descent are almost always used, due to their computational simplicity and the large used datasets [58].

The biological problems of pure backpropagation

The great successes of artificial neural networks trained with backpropagation [1] raised the question of whether our brain could also be doing backpropagation to learn. It is highly unlikely that pure backpropagation is used in a biological setting due to the following three main reasons [5, 4]:

1. **Weight transport.** As can be seen in equation (2.9), backpropagation requires that the weights of the feedback path to compute $\boldsymbol{\delta}_i$ are the transpose of the feed-forward weights. It is highly unlikely that the feedback weights are coupled to the feed-forward weights in order to be always equal to each other's transposes, as they are separate synapses with no known direct communication. This issue of weight transport is also often referred to as *weight symmetry*

2. **Distinct plasticity of feedback weights.** The feedback weights –known to provide attention mechanisms and perceptual acuity enhancement in sensory cortices– are likely plastic [7, 59, 18]. This argument seems closely intertwined with the previous one, but a distinction is necessary as recent research has shown that backpropagation-like algorithms can work with fixed random feedback weights [6].

3. **Distinct phases.** The computations would need to be precisely clocked to alternate between the feed-forward phase (needed to compute the output of the network) and the feedback phase (needed to compute the gradients), as a neuron can only represent one voltage level. Arguments are made that the known background oscillations in the brain [28] could provide a clocking mechanism for those distinct phases [8], but with Occam's razor [60] in mind and the fact that very simple organisms are also able to learn, simpler solutions to this problem should be investigated.

In recent years, the field of deep learning and neuroscience has made progress in addressing these cavities in the biological plausibility of backpropagation and in searching alternatives to backpropagation. This progress will be discussed in detail in chapter 3.

The credit assignment problem

In neuroscience, it is still an open question of how the brain can correctly change the local synaptic strengths to achieve an improved global behaviour. This fundamental question is often referred to as the *credit assignment problem* [3, 4]. In machine learning, the backpropagation algorithm is used for credit assignment, as each synapse is changed along the direction and proportional to its negative gradient. The core research question of this thesis is finding an algorithm for credit assignment in ANNs that could also be implemented by our own brain.

2.2.4 Auto-encoder neural network models

Auto-encoder neural networks were first proposed as an unsupervised learning method to find a low dimensional representation of the data which can be used as a better alternative to principal component analysis for data compression [61]. An auto-encoder consists of an encoder network coupled to a decoder network, as shown in figure 2.14b. The encoder network is used to compress the input data to a low dimensional *bottleneck* layer. The decoder network is used to decode the low dimensional representation back to the original input data. The loss of the network is a measure of the difference between the reconstructed data sample $\tilde{\boldsymbol{x}}$ and the original data sample $\boldsymbol{x}$, such as the L_2 loss. As an alternative to the low dimensional bottleneck layer for data compression, a sparse regularizer on the middle layer activations can be used to enforce a low dimensional representation, such as the L1 loss [54].

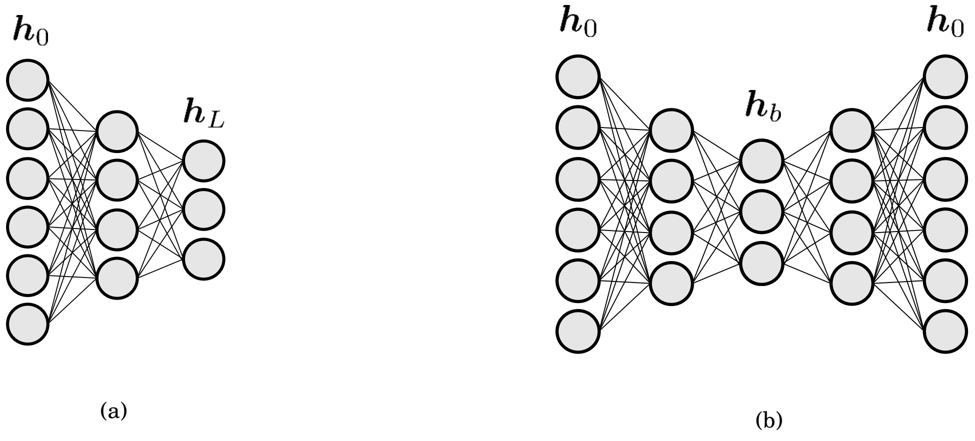


Figure 2.14: **The auto-encoder.**(a) A standard feedforward ANN with $\boldsymbol{h}_0$ the input and $\boldsymbol{h}_L$ the output. (b) An auto-encoder ANN with a bottleneck layer $\boldsymbol{h}_b$, input layer $\boldsymbol{h}_0$ and the auto-encoded input $\tilde{\boldsymbol{h}}_0$ as output. The network to the left from the bottleneck layer functions as encoder, while the network to the right of the bottleneck layer functions as decoder to reproduce the input.

Denoising auto-encoders. To make auto-encoders find a more robust lower dimensional representation of the data, the input data can be corrupted with noise. The decoded output data is then compared with the original uncorrupted data. This training method makes the auto-encoder less sensitive to noise. Auto-encoders trained by this method are called *denoising auto-encoders* [62]. The denoising auto-encoder can also be used to denoise corrupted data samples that are similar to the training data.

2.2.5 Spiking neural networks

All previous discussed deep learning models made use of scalar-valued signals, which represented the average firing rate of the neurons. In biology, however, the signals are transmitted via spikes. A different group of artificial neural networks, the *spiking neural networks*, simulates neurons communicating by spikes in time [63, 64]. One of the most simple and widely used models of a spiking neuron is the *leaky integrate and fire neuron* [65]. This neuron model simply integrates the incoming spike trains to obtain its voltage level, while having a small exponential leakage. Spiking networks have the following benefits in nature.

- **Robust signalling.** From electrical communication theory, it is known that digital signals can be transmitted more robustly over long distances compared to analogue signals. The action potential, which can be seen as a digital signal, can propagate through the axon without diminishing in amplitude, due to the specific action potential generating properties of the axon (section 2.1.2). If analogue (voltage) signals would be used to communicate, a large part of the information would be lost due to leakage in the axons and dendrites. In some small organisms, such as the C. elegans worm, graded voltage signals instead of action potentials are observed [66]. This could be explained by the small distance the signals in the axons have to travel, causing no voltage leakage issues.
- **Energy saving.** If the soma voltage level of a neuron is not above threshold, no spike is generated so no energy is used. This property is called *event based communication*. If analogue voltage levels would be used, also sub-threshold voltages would be transmitted, leading to larger energy usage.
- **Neural code.** Communicating with spikes opens the ability to use multiple different forms of neural coding, such as phase coding (see section 2.1.2).

In machine learning, however, spiking neural networks have also disadvantages:

- **Training.** It is not yet discovered how to efficiently train spiking neural networks to perform tasks such as classification, without first training a rate encoded network and afterwards converting it to a spiking neural network.
- **Computational complexity.** To use spiking neural networks, the continuous time dynamics have to be simulated. While nature has optimized our brain for doing this, regular computers are not, making it very hard to simulate large spiking neural networks. Note that this is a hardware problem and that efforts are made to make specialized hardware for spiking networks, such as the IBM's True North chip [67].

While spiking neural networks are more biologically realistic, this thesis does not use them, due to the lack of knowledge on how to train them. It is hypothesized that spikes are mainly used in our brain for energy saving and robust signalling, whereas the fundamental learning properties of the network can be investigated by rate encoding networks.

2.3 Conclusion

This chapter gave the necessary background on both biological and artificial neural networks. In the section on the biological neuron, it explained that biological neurons communicate through spike trains along their axon and that biological neural networks learn from their environment by adapting the strength of their synapses through various mechanisms of synaptic plasticity. We decided to use rate-coding to represent the information in the spike trains of neurons in the models that we develop in future chapters. Furthermore, we showed a recent hypothesis from theoretical neuroscience that proposes a multiplexing mechanism for neurons, by which a neuron can send two separate signals along its axon.

In the section on artificial neural networks, this chapter introduced the abstract neuron, better known as the perceptron, and showed how neurons can be structured in multiple layers to create the multi-layer perceptron, also known as a feed-forward neural network. After explaining the training framework of artificial neural networks, this chapter introduced error backpropagation, the algorithm by which artificial neural networks perform credit assignment. We discussed that error backpropagation is highly unlikely to be biologically plausible due to three main reasons: (1) the weight transport problem, (2) the coupled plasticity of the feedback weights and (3) the need for distinct clocked phases during training. The following chapter on related work will discuss the efforts from the field of biologically plausible deep learning to create alternatives to error backpropagation that do not cope with the above three biological problems.

Chapter 3

Related work

Almost immediately after the introduction of the backpropagation algorithm for artificial neural networks in 1986 [3], the question emerged whether our brain could be doing something similar to tackle the credit assignment problem. Neuroscientists in the 1980s agreed that this backpropagation algorithm for credit assignment was not biologically plausible, most notably due to (1) the occurring weight transport, (2) the lack of feedback weight plasticity and (3) the need for distinct phases (see section 2.2.3). In recent years, the fields of neuroscience and deep learning have joined forces to tackle the open problem of how our brain performs credit assignment in a biologically plausible manner. This chapter reviews the most important progress made in this field: target propagation, feedback alignment and segregated compartment neural models. Target propagation is a new alternative to backpropagation for performing credit assignment in neural networks. Instead of propagating gradient signals backwards through the network, the target signal is propagated back. Each layer thus receives its own local target which it can use for local credit assignment. Target propagation solves the weight transport issue and the feedback plasticity issue. Feedback alignment is closely related to backpropagation and solves the weight transport issue. Instead of using the feed-forward weights to propagate the derivatives backwards through the network, it uses fixed random feedback weights to propagate error-like signals back through the network and shows that networks learn to interpret these feedback signals. Finally, the segregated compartment neural models combine aspects of target propagation and feedback alignment with biological properties of pyramidal neurons to provide more biologically plausible algorithms. This section first explains target propagation and its variants, after which it discusses feedback alignment and its variants. Lastly, this section closes with analysing the segregated compartment neural network models.

3.1 Target propagation and its variants

The main idea of target propagation is to propagate target values for each layer backwards in the network, based on the supervised output target [9, 5]. These layer specific targets are meant to provide a smaller global loss while staying close to the current activation of the layers. Target propagation differs from back propagation, where the partial derivatives of the output layers cost function are back-propagated through the layers. From a machine learning point of view, target propagation has as a benefit that it can handle discrete networks in a more natural way. From a biological point of view, target propagation solves the weight transport and feedback plasticity issues of section 2.2.3 that back-propagation encounters. In the following, the original target propagation algorithm is explained based on [5], after which some variants on target propagation are elaborated.

3.1.1 Target propagation

Model setup. Let us consider a supervised deep learning setting, with inputs x and outputs y sampled from an unknown distribution $p(x,y)$. The network structure is then defined by:

$$\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(W_i\boldsymbol{h}_{i-1}), \quad i = 1,...,L \tag{3.1}$$

where $\boldsymbol{h}_i$ represents the activation of the i-th layer (with $\boldsymbol{h}_0 = \boldsymbol{x}$ and $\boldsymbol{h}_L$ the output of the network), f_i is the i-th layer feed-forward mapping, with s_i a nonlinearity (e.g. ReLU) and W_i the weight matrix of the feed-forward weights of the i-th layer, representing the synaptic strenghts. Note that the bias term is absorbed into W_i for simplicity of notation. This model setup is visualized in figure 3.1

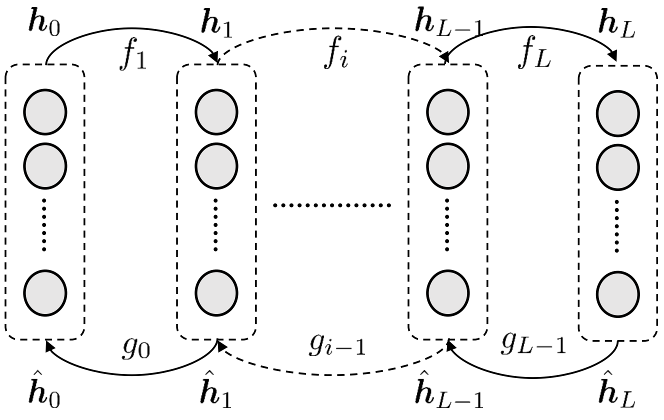


Figure 3.1: **Schematic network representation of target propagation.**

Cost functions. Let $\theta_W^{i:j}$ denote the subset of parameters (weights) defining the mapping from layer i to layer j. Now $\boldsymbol{h}_j$ can be written in function of an arbitrarily lower layer i: $\boldsymbol{h}_j = \boldsymbol{h}_j(\boldsymbol{h}_i;\theta_W^{i:j})$. Now an arbitrary cost function can be written in a form to emphasize the dependence on a particular layer i:

$$L\Big(\boldsymbol{h}_L(\boldsymbol{x};\theta_W^{0:L}),\boldsymbol{y}\Big) = L\Big(\boldsymbol{h}_L\big(\boldsymbol{h}_i(\boldsymbol{x};\theta_W^{0:i});\theta_W^{i:L}\big),\boldsymbol{y}\Big) \tag{3.2}$$

The purpose of target propagation is to assign for each layer a nearby target $\hat{\boldsymbol{h}}_i$ that leads to a lower global loss. The targets $\hat{\boldsymbol{h}}_i$ need thus to fulfill the following inequality:

$$L\Big(\boldsymbol{h}_L\big(\hat{\boldsymbol{h}}_i;\theta_W^{i:L}\big),\boldsymbol{y}\Big) < L\Big(\boldsymbol{h}_L\big(\boldsymbol{h}_i(\boldsymbol{x};\theta_W^{0:i});\theta_W^{i:L}\big),\boldsymbol{y}\Big) \tag{3.3}$$

Now that each layer has a target (the specific assignments of proper targets will be discussed later), we want to update the parameters of the network to let $\boldsymbol{h}_i$ make a small move towards $\hat{\boldsymbol{h}}_i$. This can be done with the help of a local layer cost function $L_i(\hat{\boldsymbol{h}}_i,\boldsymbol{h}_i)$, for example the Mean Squared Error (MSE) loss function or other loss function in table 2.2. Now W_i can be updated locally with the help of this loss function if $\boldsymbol{h}_{i-1}$ and $\hat{\boldsymbol{h}}_i$ are considered constant:

$$\Delta W_i = -\eta_{f_i}\frac{\partial L_i(\hat{\boldsymbol{h}}_i,\boldsymbol{h}_i)}{\partial W_i} = -\eta_{f_i}\frac{\partial L_i(\hat{\boldsymbol{h}}_i,\boldsymbol{h}_i)}{\partial \theta_W^{i-1:i}} = -\eta_{f_i}\frac{\partial L_i(\hat{\boldsymbol{h}}_i,\boldsymbol{h}_i)}{\partial \boldsymbol{h}_i}\frac{\partial \boldsymbol{h}_i\big(\boldsymbol{h}_{i-1};\theta_W^{i-1:i}\big)}{\partial \theta_W^{i-1:i}} \tag{3.4}$$

with η_{f_i} the local layer learning rate. Note that the use of derivatives in this local setting is less likely to cause problems like vanishing or exploding gradients, because the computations are done in a single layer, which lowers the chance on strong non-linearities.

Assigning proper targets. The question remains how to find proper targets $\hat{\boldsymbol{h}}_i$ that fulfill (3.3). In general, it is hard to track the influence of a local target $\hat{\boldsymbol{h}}_i$ on the global cost function, so we reduce the requirement for a target $\hat{\boldsymbol{h}}_i$ to be a proper target to a condition on the local layer loss:

$$L_i\big(\hat{\boldsymbol{h}}_i, f_i(\hat{\boldsymbol{h}}_{i-1})\big) < L_i\big(\hat{\boldsymbol{h}}_i, f_i(\boldsymbol{h}_{i-1})\big). \tag{3.5}$$

In a supervised setting, the target of the top layer should be tweaked to a lower global loss via its gradient:

$$\hat{\boldsymbol{h}}_L = \boldsymbol{h}_L - \hat{\eta}\frac{\partial L(\boldsymbol{h}_L,\boldsymbol{y})}{\partial \boldsymbol{h}_L}, \tag{3.6}$$

with $\hat{\eta}$ a small step size. In order to propagate this upper layer target to the lower layers, we can use an "approximate inverse", as proposed in [9]. So for each mapping function f_{i+1}, we try to find a function g_i for which

$$f_{i+1}\big(g_i(\boldsymbol{h}_{i+1})\big) \approx \boldsymbol{h}_{i+1} \quad \text{or} \quad g_i\big(f_{i+1}(\boldsymbol{h}_i)\big) \approx \boldsymbol{h}_i. \tag{3.7}$$

Now we can propagate the targets through the layers via

$$\hat{\boldsymbol{h}}_i = g_i(\hat{\boldsymbol{h}}_{i+1}). \tag{3.8}$$

Figure 3.1 visualizes this process. If g_i is the perfect inverse of f_i, the left side of (3.5) is zero and the inequality holds trivially. Furthermore, in [5] they prove that if g_i is a perfect inverse of f_i and if f_i has a certain structure, the update direction of target propagation deviates no more than 90 degrees from the gradient direction (obtained by back-propagation). However, in general, it is not possible or computationally infeasible to compute a perfect inverse. We can try to approximate the inverse in an *auto-encoder* setting of f_i and g_i, with g_i parametrized as follows:

$$g_i(\hat{\boldsymbol{h}}_{i+1}) = t_i(Q_i \hat{\boldsymbol{h}}_{i+1}) \tag{3.9}$$

with t_i a non-linear activation function (can be different from the forward activation function s_i), $\boldsymbol{Q}_i$ the backward weight matrix and the bias term included in $\boldsymbol{Q}_i$ for simplicity of notation. In this setting, f_{i+1} can be viewed as the encoder and g_i as the decoder. g_i can then be trained via a reconstruction loss:

$$L_i^{inv}\Big(g_i\big(f_{i+1}(\boldsymbol{h}_i)\big), \boldsymbol{h}_i\Big) = \big\|g_i\big(f_{i+1}(\boldsymbol{h}_i)\big) - \boldsymbol{h}_i\big\|_2^2. \tag{3.10}$$

$\boldsymbol{Q}_i$ can then be trained via this local loss:

$$\Delta \boldsymbol{Q}_i = -\eta_{g_i} \frac{\partial L_i^{inv}}{\partial \boldsymbol{Q}_i} = -\eta_{g_i} \frac{\partial L_i^{inv}}{\partial g_i\big(f_{i+1}(\boldsymbol{h}_i)\big)} \frac{\partial g_i\big(f_{i+1}(\boldsymbol{h}_i)\big)}{\partial \boldsymbol{Q}_i} \tag{3.11}$$

$$= 2\eta_{g_i} \cdot t_i'\big(\boldsymbol{Q}_i f_{i+1}(\boldsymbol{h}_i)\big)^T \Big(\boldsymbol{h}_i - g_i\big(f_{i+1}(\boldsymbol{h}_i)\big)\Big) f_{i+1}(\boldsymbol{h}_i)^T \tag{3.12}$$

With t_i' the Jacobian of the nonlinear activation function, which is a diagonal matrix in the case of element wise nonlinearities. Note that also loss functions other than the MSE could be used. This is however not yet explored in literature for target propagation. In order to obtain an inverse mapping of the neighbouring region around the training points instead of only the training points themselves, the authors of [5] opted for noise injection in the local inverse loss function

$$L_i^{inv}\Big(g_i\big(f_{i+1}(\boldsymbol{h}_i)\big), \boldsymbol{h}_i\Big) = \big\|g_i\big(f_{i+1}(\boldsymbol{h}_i + \epsilon)\big) - (\boldsymbol{h}_i + \epsilon)\big\|_2^2, \quad \epsilon \sim \mathcal{N}(0, \sigma) \tag{3.13}$$

with σ a chosen hyper parameter. Similarly to denoising auto-encoders, this increases the robustness of the auto-encoding couple f_{i+1} and g_i. Now that g_i is an approximate inverse of f_i instead of the exact inverse, there is no guaranty anymore that the computed targets will lead the network layers to a decrease in local cost (3.5) and the network itself to a decrease in global cost (3.3). This is a central problem in target propagation and is the major cause of the poor performance of "vanilla" target propagation so far in literature. Several variants of target propagation have been developed to cope with this problem, and these algorithms will be discussed in the next section.

Major future challenges

- Signals will travel up and down through different possible loops, arriving at a neuron after different time delays due to the recurrence introduced in the network. Ideally, the continuous time dynamics should be taken into account, thus care has to be taken when the network is discretized, for example in which order the neurons and synapses in the network are updated.

- The inverse mapping function f_i^{-1} is known to be hard to approximate by g_i. As a simple intuition: the vanilla target propagation will have difficulties when different instances of the same class have varying appearances, as the back-propagated targets will always be the same for the same class, if no information other than the class labels is used in the upper layer to generate the targets. This problem arises due to the decoupling of the forward and backward signals, as they don't influence each other directly. In error back-propagation, the propagated error is directly influenced by the feed-forward activations, which makes the back-propagated error sample specific. The inverse mapping will cause two issues: (1) there is no certainty that the inverses that we try to approximate even exist and (2) even if the inverses exist, it will not be trivial to let the network converge, as the inverses are hard to approximate. In order to make a working version of target propagation on challenging datasets, first, a solution for these two problems should be found.

3.1.2 Variants on Target propagation

Currently, there exist 3 closely related variations on target propagation to the best of our knowledge: (1) Difference Target Propagation (DTP) [5], (2) Simplified difference target propagation (SDTP) [11] and (3) Auxiliary output SDTP (AO-SDTP) [11].

Difference target propagation

A logical requirement for the training process of target propagation to converge is that if the activation matches the target in a certain layer, the activation of the lower layers also should match their targets:

$$\boldsymbol{h}_i = \hat{\boldsymbol{h}}_i \Rightarrow \boldsymbol{h}_{i-1} = \hat{\boldsymbol{h}}_{i-1}. \tag{3.14}$$

Otherwise, the lower weights will still be adjusted during training, even if the output layer matches perfectly its target, leading to an increase in global loss. If $g_i = f_i^{-1}$, condition (3.14) holds trivially, but if g_i is not a perfect inverse, a correction term is needed to make the condition hold. This correction term was introduced in [5] under the name *"difference target propagation"*

$$\hat{\boldsymbol{h}}_{i-1} = g_i(\hat{\boldsymbol{h}}_i) + \boldsymbol{h}_{i-1} - g_i(\boldsymbol{h}_i) \tag{3.15}$$

Now condition (3.14) holds by definition. This correction makes a lot of sense in a machine learning point of view (for above reason), and in [5] it is proved that condition (3.5) holds for DTP under mild conditions for g_i and f_i. In a biologically point of view however, difference target propagation is harder to motivate. To compute the target $\hat{\boldsymbol{h}}_{i-1}$, two distinct signals ($\boldsymbol{h}_i$ and $\hat{\boldsymbol{h}}_i$) need to go through the same channel g_i, which rises the need for separate phases. These phases could be coordinated by background occilations [28], as discussed in section 2.2.3, but there is not yet experimental evidence proving that this is possible.

In classification problems, the output layer is usually of much smaller dimension than the second last layer, and this causes severe problems for the inverse mapping between these layers. To cope with this problem in DTP, they set $\hat{\boldsymbol{h}}_{L-1} = \boldsymbol{h}_{L-1} - \tilde{\eta}\frac{\partial L(\boldsymbol{h}_L,\boldsymbol{y})}{\partial \boldsymbol{h}_{L-1}}$. In other words, the gradient is back-propagated into the second last layer. This is of course not biologically motivated and was only done to improve the performance of DTP.

Simplified Difference Target Propagation

Simplified Difference Target Propagation (SDTP) [11] removes the gradient backpropagation between the last two layers to make it more biologically plausible. It is thus literally a simplified version of DTP. Of course, the performance of the algorithm suffers from this simplification, and that is why Auxiliary output SDTP was developed.

Auxiliary output SDTP

In Auxiliary output SDTP (AO-SDTP) [11], a different solution for the inverse mapping problem between the two last layers is proposed. To give the inverse mapping auxiliary information about the activation of the second last layer $\hat{\boldsymbol{h}}_{L-1}$, a composite structure for the output layer is introduced: $\boldsymbol{h}_L = [o,z]$, with o the predicted class distribution and z an auxiliary output vector. This auxiliary output z can then be used to generate more varying targets for the second last layer:

$$\hat{\boldsymbol{h}}_{L-1} = g_{L-1}(\hat{o},z;Q_{L-1}) + \boldsymbol{h}_{L-1} - g_{L-1}(o,z;Q_{L-1}), \tag{3.16}$$

with $\hat{o}$ the correct class distribution, z computed from $z = s_L\big(\tilde{W}_L\boldsymbol{h}_{L-1}\big)$ and Q_{L-1} the parameters of the inverse mapping. z does not need to match some external data, and is only used to improve the inverse mapping. Their forward weights $\tilde{W}_L$ can thus be kept constant to simplify the training process (as the inverse weights still can change).

3.2 Feedback Alignment and its variants

In contrast to target propagation, which is a new alternative to backpropagation, feedback alignment [68, 6] is closely related to backpropagation. In feedback alignment, they discovered that if the feedback weights in backpropagation –normally equal to the transpose of the feed-forward weights– were instead random and fixed, the network was still able to learn. In the following, the setup of feedback alignment is explained, after which the current variants on feedback alignment are discussed. For a deeper understanding of why feedback alignment can work, the reader is referred to [6].

3.2.1 Feedback Alignment

Feedback alignment (FA) shows that a deep neural network does not need the exact transpose of the feedforward weights as feedback weights to propagate useful feedback signals through their layers, thereby resolving the biological implausibility of weight transport in backpropagation, mentioned in section 2.2.3. FA uses a fixed random feedback matrix B_i instead of the symmetric weight matrix W_i^T to propagate error signals $\boldsymbol{\delta}_i$ backwards through the network. Figure 3.2 shows a schematic representation of the propagating signals in feedback alignment. The forward mapping function f_i is the conventional mapping used in feedforward networks:

$$\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(W_i\boldsymbol{h}_{i-1}), \quad i = 1,...,L, \tag{3.17}$$

with the bias term included in W_i for simplicity of notation. The feedback mapping function g_i for the error signals $\boldsymbol{\delta}_i$ is defined by:

$$\boldsymbol{\delta}_i^{FA} = g_i(\boldsymbol{\delta}_{i+1}^{FA}) = D_{s_i}B_i\boldsymbol{\delta}_{i+1}^{FA}, \quad i = 1,...,L-1, \tag{3.18}$$

with D_{s_i} the Jacobian of $s_i(\boldsymbol{a}_i)$ with respect to $\boldsymbol{a}_i$. This backward mapping is based on back propagation, where g_i is defined by:

$$\boldsymbol{\delta}_i^{BP} = g_i(\boldsymbol{\delta}_{i+1}^{BP}) = D_{s_i}W_{i+1}^T\boldsymbol{\delta}_{i+1}^{BP}, \quad i = 1,...,L-1, \tag{3.19}$$

with $\boldsymbol{\delta}_{i-1}^{BP} = \frac{\partial L}{\partial \boldsymbol{a}_i}$, and $\boldsymbol{a}_i = W_i\boldsymbol{h}_{i-1}$ the linear activation of the layer. The forward weights in FA are then updated according to this propagated error signal:

$$\Delta W_i = \boldsymbol{\delta}_i^{FA}\boldsymbol{h}_{i-1}^T \tag{3.20}$$

In [68, 6] it is shown that during learning, the propagated error of feedback alignment starts to align with the real back-propagated error, more precisely that angle between the backpropagation update and feedback alignment update is smaller than 90 degrees ($\boldsymbol{\delta}_i^{FA}\angle\boldsymbol{\delta}_i^{BP} < 90°$). The feed-forward weights thus learn to use the random backward mappings of the output error with as a result that the backwards propagated errors point in a useful direction for the network to learn.

Performance In [11], the authors show that the performance of FA is almost equal to back-propagation on simple datasets like MNIST [10] and CIFAR-10 [69], but that it performs much worse than back-propagation on difficult data sets like ImageNet when specialized architectures are needed. In [70] the performance of FA on ImageNet is slightly improved, but remains still much worse compared to back-propagation. It is also mentioned in [6] that for FA to be able to perform well, there has to be some redundancy available in the representations in the hidden layers, otherwise the network does not have the flexibility to learn their weights W_i to both align with the random feedback matrix B_i and to learn useful hidden representations for the supervised task. It is shown for example that in auto-encoders with a bottleneck, FA performs badly.

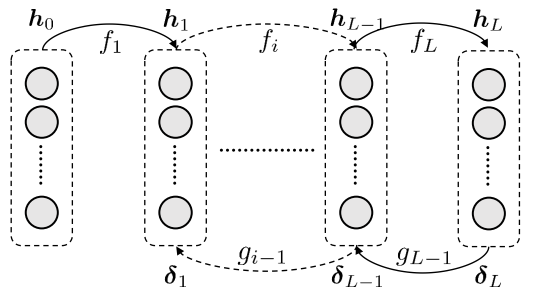


Figure 3.2: **Schematic network representation of Feedback Alignment (FA).**

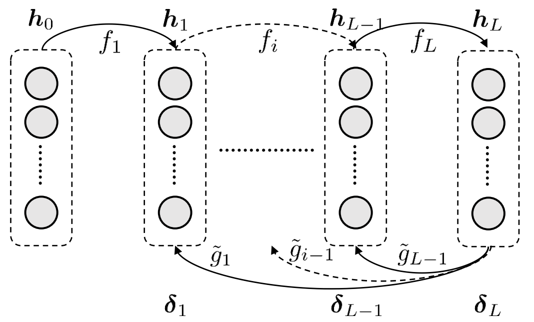


Figure 3.3: **Schematic network representation of direct feedback alignment (DFA).**

3.2.2 Direct feedback alignment

In [71] it is shown experimentally that the random feedback mapping of the error does not have to be layer-wise, but can also be in a direct connection from the last layer error to the wished hidden layer, as visualized in 3.3. This extension of feedback alignment is called 'Direct Feedback Alignment (DFA)'. Here the feedback mapping function $\tilde{g}_i$ is defined as follows:

$$\boldsymbol{\delta}_i^{DFA} = g_i(\boldsymbol{\delta}_L^{DFA}) = D_{s_i}\tilde{B}_i\boldsymbol{\delta}_L^{DFA}, \quad i = 1,...,L-1 \tag{3.21}$$

This direct connection between the outer layer error and the hidden layers has as benefit that no vanishing gradient or exploding gradient problems occur when used in very deep networks, but this learning method performs a lot worse in comparing to FA on MNIST and CIFAR-10 [11].

3.2.3 Broadcast Feedback Alignment

Feedback alignment solves one of the major biological implausibilities of back-propagation: weight transport between the feed-forward weights and the feed-backwards weights. Therefore, a logical extension of FA is to test if it works in a more biological setting, such as with spiking neurons. This extension is explored in Broadcast Feedback Alignment (BFA) [72], where the authors train a neural network consisting of leaky-integrate-and-fire (LIF) spiking neurons on MNIST. BFA largely resembles DFA but uses a couple of extensions to make it work on a LIF spiking neural network. These extensions will not be discussed in this thesis, as we will not work with spiking networks. The authors of [72] show that feedback alignment also works on spiking neural networks. It achieves state-of-the-art performance on the MNIST dataset for spiking neural networks and thus shows that biological inspired spiking neural networks can learn without weight transport between feed-forward and feed-backwards weights.

3.2.4 Bidirectional Feedback Alignment

A major biological shortcoming of feedback alignment is that the random feedback weights are fixed, while in the brain all the synapses, both feed-forward and feed-backwards, are plastic. One attempt to fix this shortcoming is Bidirectional Feedback Alignment (BDFA) [73]. In this learning scheme, two phases are used to update respectively the feed-forward and feed-backwards weights, as illustrated in figure 3.4. Phase one (indicated by superscript f) is almost exactly the same as in normal feedback alignment: the input is fed through the network by f_i^f in the forward direction, and the error is propagated from the last layer to the hidden layers via g_i^f (figure 3.4a) to update the feed-forward weights W_i.

$$\boldsymbol{h}_i^f = f_i^f(\boldsymbol{h}_{i-1}^f) = s_i(W_i\boldsymbol{h}_{i-1}^f), \quad i = 1,...,L, \tag{3.22}$$

$$\boldsymbol{\delta}_i^f = g_i^f(\boldsymbol{\delta}_{i+1}^f) = D_{s_i}B_i\boldsymbol{\delta}_{i+1}^f, \quad i = 1,...,L-1, \tag{3.23}$$

$$\Delta W_i = \boldsymbol{\delta}_i^f(\boldsymbol{h}_{i-1}^f)^T. \tag{3.24}$$

In phase two (indicated by superscript b), the feedback weights are updated to let the sequence of backward mappings g_i^b approximate the inverse of the sequence of forward mappings f_i^f as follows: the true target $\boldsymbol{h}_L^b = \boldsymbol{y}$ is propagated backward through the network via g_i^b and the reconstructed input $\boldsymbol{h}_0^b = \hat{\boldsymbol{x}}$ is compared to the original input $\boldsymbol{h}_0^f = \boldsymbol{x}$ via the auto-encoder loss $L^b = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2^2$. Then the derivative of this loss with respect to the reconstructed input (more precisely the linear activation $\boldsymbol{a}_0^b$ of the reconstructed input) $\boldsymbol{\delta}_0^b$ is propagated forward through the network via f_i^b to provide a teaching signal for the feedback weights B_i (as visualized in figure 3.4b).

$$\boldsymbol{h}_i^b = g_i^b(\boldsymbol{h}_{i+1}^b) = s_i(B_i\boldsymbol{h}_{i+1}^b), \quad i = 1,...,L, \tag{3.25}$$

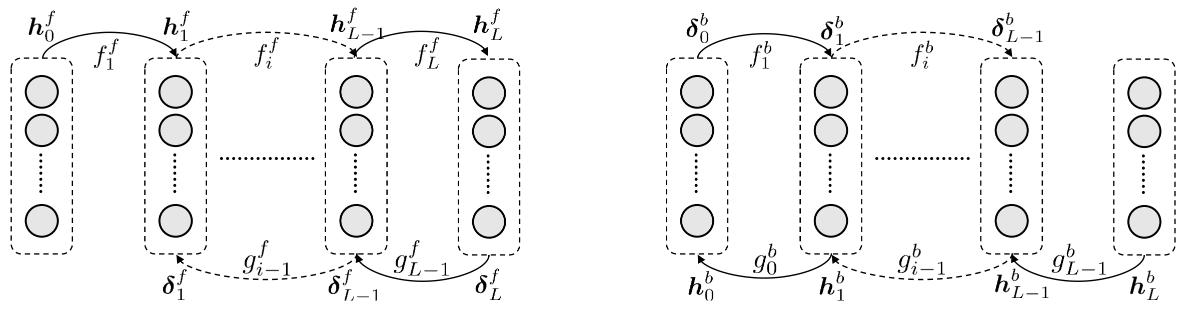
$$\boldsymbol{\delta}_i^b = f_i^b(\boldsymbol{\delta}_{i-1}^b) = D_{s_i}^b\left(W_i\boldsymbol{\delta}_{i-1}^b\right), \quad i = 1,...,L-1, \tag{3.26}$$

$$\Delta B_i = \boldsymbol{\delta}_i^b(\boldsymbol{h}_{i+1}^b)^T, \tag{3.27}$$

with $D_{s_i}^b$ the Jacobian of $s_i(B_i\boldsymbol{h}_{i+1}^b)$ with respect to $B_i\boldsymbol{h}_{i+1}^b$. The learning of the inverse mapping g_i^b reminds us of target propagation, leading one to think of BDFA as a combination of FA and TP. There are however two important differences in comparing to TP:

1. BDFA propagates a pure error signal through the layers, derived from the global loss function (in both forward and backward phase). In Target Propagation, a target signal for each layer is propagated, after which a local loss in each layer is computed with that local target.

2. BDFA tries to learn the inverse mapping g_i^b by propagating the output target through the whole network and afterwards propagating the auto-encoder error through the network with FA. In TP, however, the inverse mapping is learned layer by layer individually. BDFA also does not use a pure auto-encoder, as it feeds the output target through the network instead of the forward output of the network. There is no clear benefit of why they do this, and this can be expected to deteriorate the performance.



(a) Forward weights update phase (b) Backward weights update phase

Figure 3.4: **Schematic network representation of Bidirectional Feedback Alignment (BDFA).** (a) The feed-forward weight update phase. (b) The feedback weight update phase.

Performance In [73] they show that BDFA performs better than normal FA on simple tasks (e.g. MNIST), however, when compared to the results of a more recent study on FA [11], BDFA underperforms normal FA. On more complex tasks (e.g. CIFAR) BDFA performs even worse, because the inverse mapping is much harder to learn, making it harder for the network to interpret the error signals than in the case of random fixed feedback matrices. In pure FA, the feed-forward weights align with the fixed feedback weights in order to receive useful feedback signals. If the feedback weights are not fixed anymore, without giving additional benefits, the feed-forward weights will have more difficulties with aligning to the feedback weights.

Biological Plausibility Whereas the authors of [73] proposed BFA to be more biologically plausible than FA because of the plastic feedback weights, they introduce other methods to the learning scheme that are biologically highly unlikely:

- There are now four distinct phases that need to be coordinated precisely after each other: (1) propagate the input forward through the network, (2) propagate the output error backward through the network to update the forward weights, (3) propagate the output target backward through the network to reconstruct the input and (4) propagate the auto-encoder error forward through the network to update the backward weights.
- There are now 4 distinct channels that share weights (W_i and B_i), but have different mapping functions and process different signals (making it highly unlikely that they are represented by the same biological channel), whereas in biological networks you only have two distinct channels (feed-forward and feed-backwards)

3.3 Learning with segregated dendrites

Feedback alignment and target propagation were both developed from a deep learning point of view and received a biological interpretation later on. This section discusses 2 models that are greatly inspired by the specific properties of pyramidal neurons in our brain to solve the credit assignment problem.

3.3.1 Deep learning with segregated dendrites

In order to solve the credit assignment problem in a more biologically plausible way, the authors of the segregated dendrites deep learning model [8] were inspired by the following four observations:

1. Current solutions to the credit assignment problem in deep learning without weight transport (target propagation and feedback alignment) need segregated feed-forward and feedback signal pathways or distinct phases to transmit the two signals [6, 5].
2. In the neocortex, feedforward sensory information and higher-order feedback signals are largely received in respectively the basal dendrites and apical dendrites (see section 2.1.1).
3. The apical dendritic compartments are electrically distant from the soma of the neuron. As communication between the two, plateau potentials are generated (see section 2.1.2). These plateau potentials can generate burst spiking groups at the soma.
4. Plateau potentials can guide synaptic plasticity in pyramidal neurons in vivo [74, 75].

With their proposed model, the authors manage to solve the weight transport issue in a biological setting.

Model. For a detailed view on the model dynamics and the plasticity rules, the reader is referred to [8]. In what follows, an intuitive approach to the model is explained. Figure 3.5 shows the network structure of the segregated dendrite deep learning model. Each compartment has its own voltage level. The voltage level of the basal and apical compartments depends respectively on the feed-forward inputs and the feedback inputs. The authors use a spiking neural network model, in which the voltage levels of the dendritic compartments are simply the weighted sums of the incoming spike trains with the synaptic strengths as weights, and the spike trains are generated by a Poisson process. To understand the model intuitively, however, it is sufficient to assume a rate encoded network. The voltage level of the soma is largely dependent on the voltage level of the basal compartment, and only weakly connected to the apical voltage level. At the apical dendritic compartment, plateau potentials are generated, which are modelled as a time window average of the apical compartment voltage level.

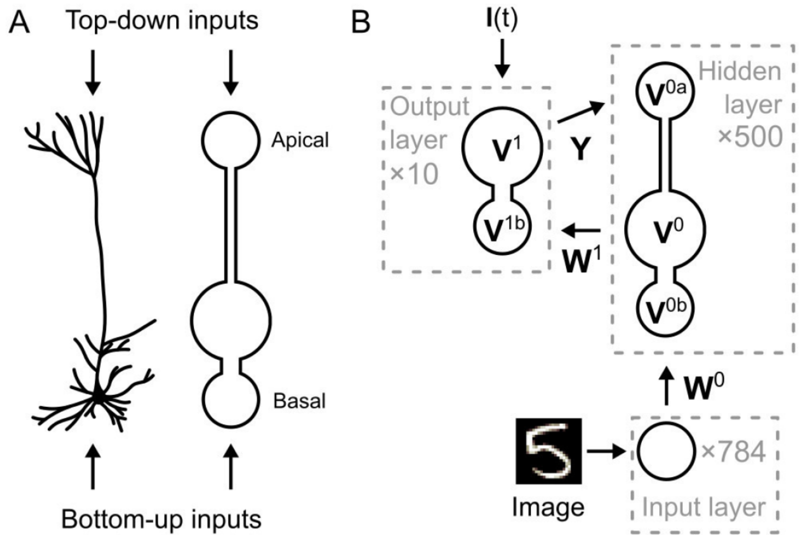


Figure 3.5: **Structure of the deep learning model with segregated dendrites.** (A) the three-compartment model of the pyramidal neuron. (B) The network structure of a segregated dendrite network with one hidden layer. Figure adapted from [8].

Learning mechanism. The learning method of this model is a mixture between difference target propagation [5] and feedback alignment [6], structured in two distinct learning phases.

1. **Free phase.** During the free phase, no teaching signals are imposed on the output layer. Sensory signals are propagating forward through the network, and feedback signals (originating from the feed-forward activation of the output layer) are propagated back to the hidden layer of the network, which result in plateau potentials in this hidden layer. These plateau potentials are similar to $g_i(\boldsymbol{h}_i)$ in equation (3.15) of difference target propagation.

2. **Target phase.** During the target phase, the output layer is tweaked to the target value by a 'teaching current', similar to target propagation. Now the output layer will send different feedback inputs to the hidden layer, causing different plateau potentials to occur in this layer. These plateau potentials are similar to $g_i(\hat{\boldsymbol{h}}_i)$ in equation (3.15) of difference target propagation.

The feedback weights to the apical dendrites are kept fixed, as in feedback alignment. Similar to difference target propagation, a local L_2 loss is computed in the hidden layer:

$$L_h = \left\| \hat{\boldsymbol{h}}_h - \boldsymbol{h}_h \right\|_2^2 \tag{3.28}$$

$$= \left\| \boldsymbol{h}_h + \boldsymbol{\alpha}_h^t - \boldsymbol{\alpha}_h^f - \boldsymbol{h}_h \right\|_2^2 \tag{3.29}$$

$$= \left\| \boldsymbol{\alpha}_h^t - \boldsymbol{\alpha}_h^f \right\|_2^2 \tag{3.30}$$

with $\boldsymbol{h}_h$ the hidden layer activation, $\hat{\boldsymbol{h}}_h$ the target of the hidden layer, computed as in (3.15), and $\boldsymbol{\alpha}_h^t$ and $\boldsymbol{\alpha}_h^f$ the plateau potentials in the hidden layer of respectively the target phase and free phase. The forward weights of the hidden layer are then updated via the gradient of the local loss function.

Discussion. The contribution of this work is the introduction of the segregated compartment model as a more biologically correct version of the abstract pyramidal neuron. Mathematically, their model is not innovative, as it is a plain combination of difference target propagation and feedback alignment. The biological arguments the authors provide for this model are a step in a good direction, however, there are still issues that need to be further investigated:

- The plateau potentials in this model are only used to regulate the plasticity of the basal synapses. The somatic voltage level is only very weakly coupled to the plateau potentials. However, it is experimentally observed that the plateau potentials interact greatly with the soma, generating bursting spikes (see section 2.1.2 and [21]).
- There is no mechanism in place in the model to propagate the plateau potentials through multiple hidden layers in the network. The authors now solve this issue by having direct connections from the output layer to each hidden layer, similar to direct feedback alignment [71]. This is no optimal solution, as it breaks the layered structure of the feed-forward path, which makes it harder for the network to learn representations of the data in multiple layers of abstraction. This issue is reflected in their results, that state that the network does not improve further when modelling more than 2 hidden layers. The multiplexing properties of a neuron, discussed in section 2.1.2, could provide a biological solution for this problem.
- Two separate phases are needed for their model: free phase and target phase. This could be coordinated in the brain with the help of background oscillations as discussed before, but this is a complicated solution with so far no experimental evidence supporting it.

3.3.2 Dendritic error backpropagation

The dendritic error backpropagation model [7] consists of cortical microcircuits of pyramidal neurons and inhibitory interneurons as shown in figure 3.6. The modelled pyramidal neurons consist of the same three compartments as in the segregated dendrites model. The used interneurons consist out of two compartments: the basal dendritic compartment and the soma. In this model, each compartment has its own voltage level. The dynamics of the dendritic voltage levels depend on their inputs and connection weights, the dynamics of the soma voltage level depends on the voltage level of the dendritic compartments. The output firing rates of both types of neurons are a non-linear function of their soma voltage level. The specific wiring of the cortical microcircuit can be seen in figure 3.6. The feed-forward connections arrive at the basal dendritic compartment, the feedback connections and the connections from the inhibitory interneuron arrive at the apical compartment of the pyramidal neuron. The interneurons receive connections from the basal compartments of all pyramidal the neurons in the same layer and its output connects to the apical compartments of all pyramidal neurons in the same layer. On top of that, each interneuron receives a small teaching current of one neuron of the next layer (one-on-one pairing). With this proposed model, the authors manage to solve the weight transport issue of backpropagation and have a learning method without phases, all in a biological setup.

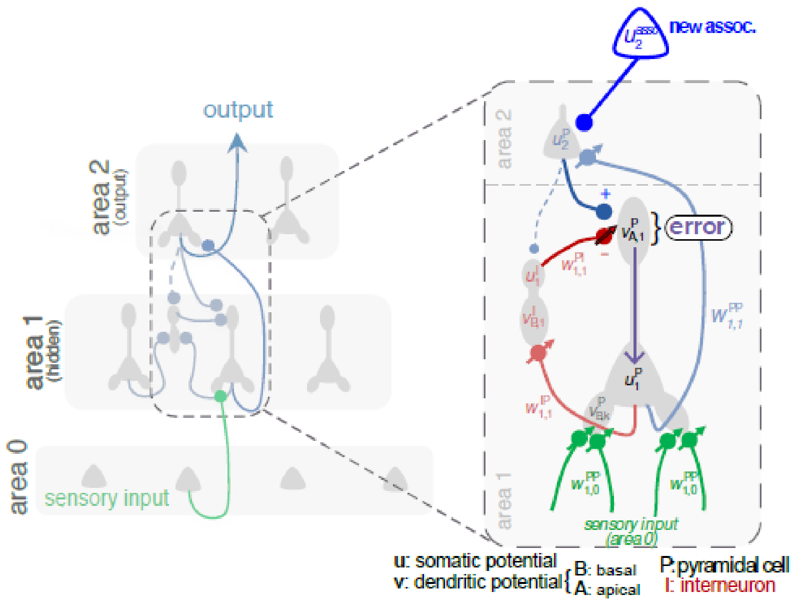


Figure 3.6: **Structure of the cortical microcircuits used in the dendritic backpropagation model.** Figure adapted from [7].

Learning mechanism. For the detailed description of the dynamics of the microcircuits and their plasticity, the reader is referred to [7]. In what follows, an intuitive explanation of the learning mechanism is proposed.

- **Local dendritic prediction.** Before the actual training of the network begins, the interneurons learn to predict the feedback input of its coupled pyramidal neuron, given the feedforward activation of the pyramidal neurons in its layer. This local prediction can be achieved by letting the plasticity of the synapses from the interneuron to the apical dendrites be driven by the negative of the apical dendrite voltage level. This plasticity rule drives those synapses to always counterbalance the feedback input that the apical compartment receives, as then the total voltage level of the apical compartment is zero. After that the network is initialized in its self-predicting state, the plasticity of the synapses from the interneuron to the pyramidal neuron is kept on, so the synapses can adapt to the changing network during training.

- **Local prediction error.** During the training phase, a teaching current is imposed on the output layer of the network (new associative input in figure 3.6). Now the interneurons of the hidden layers cannot exactly predict the feedback inputs anymore, as the output layer is tweaked to another value by the teaching current, which cannot be explained by the feed-forward activity of the network alone. This results in a non-zero voltage level in the apical dendritic compartment, which can be interpreted as an error signal. This error signal influences the voltage level of the soma. The plasticity of the basal synapses (connecting the previous layer pyramidal neurons to the current one) is driven by the difference of the soma voltage level and the basal voltage level. If there is no error signal, the soma voltage level is solely driven by the basal voltage level, thus there occurs no plasticity. If there is an error signal, the difference will be non-zero and plasticity occurs.

- **Local teaching currents.** To improve learning, a weak feedback coupling from higher layer pyramidal neurons to the interneurons is introduced (shown in striped lines in figure 3.6). This makes the interneuron able to mimic the higher layer activity, by adapting its basal weights.

- **Feedback weights.** The synapses connecting the higher layer pyramidal neurons to the apical dendritic compartment of the current neuron are random and fixed. As shown in the feedback alignment algorithms, the network is still able to deduce useful teaching errors with random fixed feedback weights. Sacramento et al. [7] show that the network can also learn with plastic feedback weights, through a mechanism similar to difference target propagation.

Discussion. The dendritic error learning rule is an elegant interpretation of the properties of the observed microcircuits in the human brain. The used neural dynamics are more realistic compared to the dynamics of the previously discussed segregated dendrites model. The performance results of this learning rule are close to the ones of backpropagation on MNIST. The authors also prove that their model is an approximation of backpropagation [7]. From a biological point of view, this model solves all three issues of error backpropagation (no weight transport, feedback plasticity and no distinct phases). However, it introduces two new issues that prevent the model from being an all-comprehensive biologically plausible learning method:

1. The network makes use of a dedicated micro-circuit, in which each pyramidal cell needs a corresponding interneuron. Hence, there must be at least as many interneurons as pyramidal neurons in the next layer for the self-prediction to work. However, $70-85\%$ of the neurons in the cerebral cortex are pyramidal neurons, the other $15-30\%$ are interneurons and aspiny non-pyramidal neurons, which cover a wide variety of other neurons [17].
2. The local teaching current from higher layer pyramidal neurons to the interneurons is a direct coupled current proportional to the voltage levels, instead of spike trains that travel through connecting synapses.

3.4 Conclusion

In this chapter, the relevant learning models that try to merge deep learning with neuroscience were discussed. The field of biologically plausible deep learning was only recently developed, although already a lot of progress has been made. The two major advances in abstract biologically plausible learning methods are target propagation and feedback alignment. In the literature was mentioned that the pure form of target propagation performs poorly, but that difference target propagation, a variant of target propagation, can reach good performance on benchmark datasets such as MNIST. The deep learning model with segregated dendrites [8] and the dendritic error backpropagation model [7] use both difference target propagation and feedback alignment to create a learning model based on the properties of pyramidal neurons. The dendritic error backpropagation model of Sacramento et al. [7], succeeds in tackling all the three biological issues of backpropagation mentioned in section 2.2.3, however, it needs a very specific cortical microcircuit structure that is unlikely to be found in large numbers in the brain. Table 3.1 gives an overview of the performance of the discussed learning algorithms on the MNIST and CIFAR-10 dataset [10, 69].

Table 3.1: **Comparison of the discussed learning methods.** The test error (%) is given for fully connected networks on the MNIST and CIFAR-10 dataset. For MNIST, 5 fully connected layers of 256 neurons were used, with a softmax layer on top. For CIFAR, 3 fully connected layers of 1024 neurons were used, with a softmax layer on top. For the segregated dendrites algorithm, a fully connected network of 2 hidden layers was used, resp. with 500 and 100 neurons. For the dendritic error algorithm, a fully connected network of 2 hidden layers was used, both with 500 neurons. As a summary, the solved biological issues are also displayed per algorithm. [11, 7, 8, 73]

Algorithm	Test error [%]		Solved issues		
	MNIST	CIFAR	No weight transport	Feedback plasticity	No phases
DTP	1.83	42.32	✓	✓	✗
SDTP	2.28	54.27	✓	✓	✗
AO-SDTP	1.86	45.40	✓	✓	✗
FA	1.85	41.97	✓	✗	✗
DFA	2.75	47.80	✓	✗	✗
BFA	2.37	/	✓	✗	✗
BDFA	2.84	48.01	✓	✗	✗
Segr. dendrites	3.20	/	✓	✗	✗
Dendritic error	1.96	/	✓	✓	✓
BP	1.48	41.32	✗	✗	✗

In the next chapter, we delve deeper into target propagation by developing an extensive mathematical foundation of the method and by showing why the pure form of target propagation performs poorly, whereas its variant difference target propagation reaches good results. In chapter 6, we go further on the work of Sacramento et al. [7] and Guerguiev et al. [8] by developing a new segregated dendrite model of pyramidal neurons that exhibits target-propagation-like learning dynamics.

Chapter 4

A theoretical analysis of Target Propagation

In this chapter, a thorough theoretical analysis of target propagation and its variants is done to gain a deeper understanding of the network learning mechanisms. This makes a significant contribution to the field of target propagation and biologically plausible training methods for deep learning more generally, as most of these methods do not yet have a solid mathematical foundation. This chapter starts with investigating the ideal form of target propagation: target propagation with exact invertible layers. The characteristics of the propagated targets are investigated by linear Taylor approximations and an efficient implementation of target propagation with exact inverses is explored. The next section handles the main result of this chapter: target propagation with exact inverses uses Gauss-Newton optimization to compute its local layer targets. After proving this main theorem, we discuss the implications of this insight on designing suitable training schemes for target propagation. The last section investigates the target propagation method in a more general setting, as now the backward mapping functions learn to approximate the inverses of the forward mapping. First, the implications of the occurring reconstruction errors on the learning signals are investigated, after which we propose new approximation methods for learning the inverses. The section ends with the second main result of this chapter: under well-specified conditions, difference target propagation uses Gauss-Newton optimization to compute its local layer targets, without the need for the existence of perfect inverses.

4.1 Target propagation with exact inverses

This section starts with investigating how the targets are backpropagated through the layers, after which it proves that for sufficient small step sizes, the update direction computed with target propagation is always a descending direction. Finally, an efficient implementation of the exact inverses is discussed.

4.1.1 A linear Taylor approximation of the local layer targets

Figure 4.1 shows the schematic network representation of pure target propagation. The forward propagation of layer activations is defined by

$$\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(W_i \boldsymbol{h}_{i-1}), \quad i = 1, ..., L \tag{4.1}$$

and the backward propagation of target activations is defined by

$$\hat{\boldsymbol{h}}_i = g_i(\hat{\boldsymbol{h}}_{i+1}), \quad i = L-1, ..., 0. \tag{4.2}$$

Throughout the rest of the thesis, no biases will be used in the analyses, for ease of notation. The created mathematical framework can be extended to include biases by using homogeneous coordinates. This extention is left for future research. Let us now consider the ideal case, in which g_i is the perfect inverse of f_{i+1}:

$$g_i(\hat{\boldsymbol{h}}_{i+1}) = f_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1}) = W_{i+1}^{-1} s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1}). \tag{4.3}$$

Note that for this perfect inverse to exist, the forward nonlinearity s_i has to be invertible (e.g. a leaky ReLU) and the matrix W_i has to be square and of full rank. This implies that the network layers all need

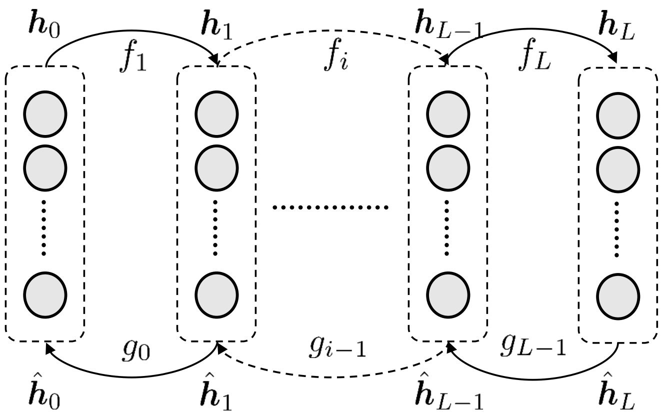


Figure 4.1: **Schematic network representation of target propagation.**

to have the same dimensions. Later in this thesis, we will propose methods to relax this strict condition on the layer dimensions. Lets now define the target of the output layer as the forward output, tweaked in the direction of lower loss:

$$\hat{\boldsymbol{h}}_L = \boldsymbol{h}_L - \hat{\eta}\boldsymbol{e}_L \tag{4.4}$$

$$\boldsymbol{e}_L \triangleq \frac{\partial L}{\partial \boldsymbol{h}_L}, \tag{4.5}$$

with $\hat{\eta}$ the output target step size, L the loss function of the network and $\boldsymbol{e}_L$ the gradient of the loss function with respect to the output, which can be interpreted as an error signal if for example the L_2 loss is used. To investigate how the output target gets propagated backwards through the network, we do a Taylor approximation around $\boldsymbol{h}_i$ for each local layer target $\hat{\boldsymbol{h}}_i$. For clarity, we start with $\hat{\boldsymbol{h}}_{L-1}$.

$$\hat{\boldsymbol{h}}_{L-1} = g_{L-1}(\hat{\boldsymbol{h}}_L) = g_{L-1}(\boldsymbol{h}_L - \hat{\eta}\boldsymbol{e}_L) \tag{4.6}$$

$$= g_{L-1}(\boldsymbol{h}_L) - \hat{\eta}J_{g_{L-1}}\boldsymbol{e}_L + \mathcal{O}(\hat{\eta}^2) \tag{4.7}$$

$$= \boldsymbol{h}_{L-1} - \hat{\eta}J_{g_{L-1}}\boldsymbol{e}_L + \mathcal{O}(\hat{\eta}^2) \tag{4.8}$$

with J_{g_i} the Jacobian of g_i with respect to $\boldsymbol{h}_{i+1}$, evaluated at $\boldsymbol{h}_{i+1}$. For the last step, we used that g_i is the perfect inverse of f_{i+1}. If we continue doing Taylor expansions for each layer, we reach a general expression for $\hat{\boldsymbol{h}}_i$:

$$\hat{\boldsymbol{h}}_i = \boldsymbol{h}_i - \hat{\eta}\left[\prod_{k=i}^{L-1} J_{g_k}\right]\boldsymbol{e}_L + \left[\prod_{k=i}^{L-2} J_{g_k}\right]\mathcal{O}(\hat{\eta}^2) + \mathcal{O}(\hat{\eta}^2) \tag{4.9}$$

$$\approx \boldsymbol{h}_i - \hat{\eta}\left[\prod_{k=i}^{L-1} J_{g_k}\right]\boldsymbol{e}_L \tag{4.10}$$

From the penultimate term in equation (4.9), we can see that the Taylor approximation becomes inaccurate when J_{g_i} has large eigenvalues or when $\hat{\eta}$ is not taken small enough. From the inverse function theorem, we know that $J_{g_i} = J_{f_{i+1}}^{-1}$, with $J_{f_{i+1}}^{-1}$ evaluated at $\boldsymbol{h}_i$. Note that $J_{f_{i+1}} = D_{s_{i+1}}W_{i+1}$, with $D_{s_{i+1}}$ the diagonal matrix with the derivatives of the activation function evaluated at $W_{i+1}\boldsymbol{h}_i$. Now the following expression can be written for $\hat{\boldsymbol{h}}_i$

$$\hat{\boldsymbol{h}}_i \approx \boldsymbol{h}_i - \hat{\eta}\left[\prod_{k=i}^{L-1} W_{k+1}^{-1}D_{s_{k+1}}^{-1}\right]\boldsymbol{e}_L \tag{4.11}$$

When an L_2 loss $\frac{1}{2}\|\boldsymbol{h}_i - \hat{\boldsymbol{h}}_i\|_2^2$ is taken as local layer loss, the forward weight updates (see (3.4)) have the following form:

$$\Delta W_i = -\eta_i D_{s_i}(\boldsymbol{h}_i - \hat{\boldsymbol{h}}_i)\boldsymbol{h}_{i-1}^T \tag{4.12}$$

$$\approx -\eta_i D_{s_i}\left(\hat{\eta}\left[\prod_{k=i}^{L-1} W_{k+1}^{-1}D_{s_{k+1}}^{-1}\right]\boldsymbol{e}_L\right)\boldsymbol{h}_{i-1}^T, \tag{4.13}$$

with η_i the local learning rate. The difference $\boldsymbol{h}_i - \hat{\boldsymbol{h}}_i$ can be seen as the learning signal of the target propagation method, similar to the backpropagated error in the backpropagation method. The weight update resulting from the classic backpropagation of the error algorithm can be written in a similar form:

$$\Delta W_i = -\eta_i D_{s_i}\left(\left[\prod_{k=i}^{L-1} W_{k+1}^T D_{s_{k+1}}\right]\boldsymbol{e}_L\right)\boldsymbol{h}_{i-1}^T. \tag{4.14}$$

We thus see that target propagation with exact inverses is similar to backpropagation, but that W_{i+1}^T and $D_{s_{i+1}}$ are replaced by resp. W_{i+1}^{-1} and $D_{s_{i+1}}^{-1}$. Note that $\hat{\eta}$ can be absorbed in the local learning rate η_i after the Taylor approximation.

4.1.2 Descent direction of the target propagation update

In what follows, we prove that the weight update computed with the target propagation method points always in a descent direction, if the output target step size $\hat{\eta}$ is taken sufficiently small. A similar proof already exists in [5], but there the authors used an uncommon form of the feed-forward mapping $f_i = W_i s_i(\boldsymbol{h}_{i-1})$ instead of $f_i = s_i(W_i \boldsymbol{h}_{i-1})$, thus a proof for the common form of target propagation is still lacking in the field.

Theorem 4.1. *Consider a feed-forward neural network with forward mapping function $\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(W_i\boldsymbol{h}_{i-1})$, $i = 1,...,L$ where s_i can be any differentiable, monotonically increasing and invertible element-wise function. Assume that the backward mapping functions g_i, used for propagating the target activations, are the exact inverses of f_{i+1}. Let ΔW_i^{tp} and ΔW_i^{bp} be the target propagation update and the back-propagation update in the i-th layer, respectively. If $\hat{\eta}$ in equation* (4.4) *is taken in limit to zero ($\hat{\eta} \to 0+$), then the angle α between ΔW_i^{tp} and ΔW_i^{bp} is bounded by*

$$0 < cos(\alpha) \leq 1, \tag{4.15}$$

implying a descent direction of ΔW_i^{tp}, as ΔW_i^{bp} points in the opposite direction of the gradient.

The proof of this theorem can be found in appendix B.

4.1.3 Efficient implementation of target propagation with exact inverses

In target propagation with exact inverses, g_i is defined by

$$g_i(\hat{\boldsymbol{h}}_{i+1}) = W_{i+1}^{-1} s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1}). \tag{4.16}$$

The inverse nonlinearity s_{i+1}^{-1} is still element-wise, thus can be computed in $\mathcal{O}(n)$ time, with n the dimension of the layer. For W_{i+1}^{-1} however, either a linear set of equations has to be solved or a matrix inversion has to be computed, which both have an expensive time complexity of $\mathcal{O}(n^3)$ in general. This makes it unfeasible with current hard-ware to scale a naive implementation of TP with exact inverses to large networks, emphasizing the need for an efficient implementation of target propagation with exact inverses.

Sherman-Morisson formula. From equation (4.12) we see that the forward weights are updated by rank-1 matrix additions. This makes it possible to compute the inverse of the updated forward weights based on the inverse of the forward weights of the previous iteration with the Sherman-Morrison formula[1] [76]:

$$\boldsymbol{u}_i = -\eta_i D_{s_i}\left(\boldsymbol{h}_i - \hat{\boldsymbol{h}}_i\right) \tag{4.17}$$

$$\boldsymbol{v}_i = \boldsymbol{h}_{i-1} \tag{4.18}$$

$$\left(W_i + \Delta W_i\right)^{-1} = W_i^{-1} - \frac{W_i^{-1}\boldsymbol{u}_i\boldsymbol{v}_i^T W_i^{-1}}{1+\boldsymbol{v}_i^T W_i^{-1}\boldsymbol{u}_i}. \tag{4.19}$$

If ΔW_i is averaged over multiple samples, like is done in mini-batch stochastic gradient descent, this update is not of rank 1 anymore. Hence, the Sherman-Morisson formula can only be used when mini-batch sizes of 1 are used. In [77], the authors extend the Sherman-Morrison formula for pseudo inverses, which could be used for extending the target propagation framework to non-uniform layer dimensions.

[1]Sherman-Morisson formula: $(A + \boldsymbol{u}_i\boldsymbol{v}_i^T)^{-1} = A^{-1} - \frac{A^{-1}\boldsymbol{u}_i\boldsymbol{v}_i^T A^{-1}}{1+\boldsymbol{v}_i^T A^{-1}\boldsymbol{u}_i}$.

Computational cost. The inverse W_i^{-1} only needs to be computed once at the beginning of training, after which only matrix-vector products are needed to compute the updated inverses. These updates have the same time complexity as backpropagation, which is $\mathcal{O}(n^2)$. In what follows, we compare the computational cost of target propagation with exact inverses to the computational cost of back-propagation. Table 4.1 summarizes the computational cost of the operations used in both target propagation and error back-propagation. The total cost for one training iteration per layer is summarized in tables 4.2 and 4.3 for error backpropagation and target propagation, respectively. A leaky-ReLU was chosen as non-linearity, but other non-linearities will give similar results. We see that target propagation with exact inverses has roughly the double amount of computations compared to error backpropagation, but stays in the same time-complexity $\mathcal{O}(n^2)$.

Table 4.1: **Computational cost of the most common operations in training feed-forward neural networks.** n represents the layer dimension

Operation	**# additions**	**# multiplications**
matrix-vector product	$n*(n-1)$	n^2
vector outer product	0	n^2
vector inner product	$n-1$	n
element-wise vector product	0	n
matrix sum	n^2	0
leaky-ReLU non-linearity	0	n or less

Table 4.2: **Total computational cost per layer of the error-backpropagation method.** M stands for multiplication, A for addition and n for the layer dimension.

Training phase	**Operations**	**Cost**
Forward pass	1 matrix-vector product	$n(n-1)A+n^2M$
	1 Leaky-ReLU non-linearity	nM
Backpropagation of error	1 matrix-vector product	$n(n-1)A+n^2M$
	element-wise vector product	nM
Updating weights	vector outer product	n^2M
	matrix sum	n^2A
Total:		$(3n^2-2n)A+(3n^2+2n)M$

Table 4.3: **Total computational cost per layer of the target propagation method with exact inverses.** M stands for multiplication, A for addition and n for the layer dimension.

Training phase	**Operations**	**Cost**
Forward pass	1 matrix-vector product	$n(n-1)A+n^2M$
	1 Leaky-ReLU non-linearity	nM
Update inverse weights	3 matrix-vector products	$3n(n-1)A+3n^2M$
	vector outer product	n^2M
	vector inner product	$(n-1)A+nM$
	matrix sum	n^2A
Backpropagation of target	1 matrix-vector product	$n(n-1)A+n^2M$
	inverse leaky-ReLU non-linearity	nM
Updating weights	vector outer product	n^2M
	matrix sum	n^2A
Total:		$(7n^2-4n-1)A+(7n^2+3n)M$

4.2 Target propagation as an approximation of Gauss-Newton optimization

In this section, we will show that target propagation with exact inverses is an approximation of the Gauss-Newton optimization algorithm [78, 79]. More specific, Target Propagation computes the local layer targets conform with Gauss-Newton optimization and then updates the layer weights by gradient descent in order to push the layer activation towards the target layer activation. This section starts by explaining the Gauss-Newton optimization scheme after which it applies the Gauss-Newton scheme on feed-forward neural networks to show that the target propagation equations appear. This section ends with a discussion on choosing suitable step sizes for target propagation compatible with the Gauss-Newton optimization scheme.

4.2.1 The Gauss-Newton optimization algorithm

The Gauss-Newton (GN) algorithm is an iterative optimization method that is used for non-linear regression problems, defined as follows:

$$\min_{\boldsymbol{\beta}} \quad L = \frac{1}{2}\sum_{i=1}^{B} e_{(i)}^2 \tag{4.20}$$

$$e_{(i)} \triangleq y_{(i)} - t_{(i)}, \tag{4.21}$$

with L the regression loss, B the mini-batch size, $e_{(i)}$ the regression residual of the i^{th} sample, $y_{(i)}$ the model output and $t_{(i)}$ the target output. The one-dimensional output y is a nonlinear function of the inputs $\boldsymbol{x}$, parameterized by $\boldsymbol{\beta}$. At the end of this section, the Gauss-Newton method will be extended for models with multiple outputs. The Gauss-Newton algorithm is an approximation of Newton's method, more precisely, it approximates the Hessian matrix used in Newton's method by a positive semidefinite matrix approximation. As the GN method is related to Newton's method, it can approximate second-order convergence, however, there are no convergence guaranties [80]. Especially when the initialization of the parameters is far from the optimum, Gauss-Newton is known to have convergence problems. There exist variations on the Gauss-Newton algorithm to cope with this convergence problem, most notably the Levenberg-Marquardt algorithm [81]. The Gauss-Newton algorithm can be derived in two different ways: (1) via a linear Taylor expansion around the current parameter values $\boldsymbol{\beta}$ and (2) via an approximation of Newton's method. Both derivations will be discussed, after which we elaborate on Gauss-Newton optimization for multiple output models and on the Levenberg-Marquardt variant of Gauss-Newton optimization.

Derivation of the Gauss-Newton method via a Taylor expansion

The goal of a Gauss-Newton iteration step is to find a parameter update $\Delta\boldsymbol{\beta}$ that leads to a lower regression loss:

$$\boldsymbol{\beta}^{(m+1)} \leftarrow \boldsymbol{\beta}^{(m)} + \Delta\boldsymbol{\beta}. \tag{4.22}$$

Ideally, we want to minimize the regression loss L with respect to the parameters $\boldsymbol{\beta}$:

$$0 \overset{!}{=} \frac{\partial L}{\partial \boldsymbol{\beta}} = J^T \boldsymbol{e} \tag{4.23}$$

$$J \triangleq \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{\beta}}, \tag{4.24}$$

with $\boldsymbol{e}$ a vector containing all the B residuals $e_{(i)}$ and $\boldsymbol{y}$ a vector containing all the outputs. $\boldsymbol{y}$ and $\boldsymbol{e}$ can be approximated by a first order Taylor expansion around the current parameter values $\boldsymbol{\beta}^{(m)}$:

$$\boldsymbol{y}^{(m+1)} \approx \boldsymbol{y}^{(m)} + J\Delta\boldsymbol{\beta} \tag{4.25}$$

$$\boldsymbol{e}^{(m+1)} = \boldsymbol{y}^{(m+1)} - \boldsymbol{t} \approx \boldsymbol{e}^{(m)} + J\Delta\boldsymbol{\beta} \tag{4.26}$$

Now this approximation of $\boldsymbol{e}$ can be filled in equation (4.23), which results in

$$\frac{\partial L}{\partial \boldsymbol{\beta}} \approx J^T\big(\boldsymbol{e}^{(m)} + J\Delta\boldsymbol{\beta}\big) = 0 \tag{4.27}$$

$$\Leftrightarrow \quad J^T J\Delta\boldsymbol{\beta} = -J^T\boldsymbol{e}^{(m)}. \tag{4.28}$$

If $J^T J$ is invertible, this leads to:

$$\Delta\boldsymbol{\beta} = -\big(J^T J\big)^{-1} J^T \boldsymbol{e}^{(m)} \tag{4.29}$$

$$\Delta\boldsymbol{\beta} = -J^{\dagger}\boldsymbol{e}^{(m)}, \tag{4.30}$$

With $J^\dagger$ the left Moore-Penrose pseudo inverse of J. Note that if J^TJ is not invertible, $-J^\dagger \boldsymbol{e}^{(m)}$ leads to the solution $\Delta\boldsymbol{\beta}$ with the smallest norm, thus $J^\dagger$ is still the best choice. If J is square and invertible, the pseudo inverse is equal to the real inverse, leading to the following expression:

$$\Delta\boldsymbol{\beta} = -J^{-1}\boldsymbol{e}^{(m)}. \tag{4.31}$$

Note the similarity between the above equations (4.28)-(4.31) and linear least squares: the design matrix X is replaced by the Jacobian J and the residuals and parameter increments are used instead of the output values and the parameters respectively.

Derivation of the Gauss-Newton method via Newton's method

Newton's method updates the parameters in each iteration as follows:

$$\boldsymbol{\beta}^{(m+1)} \leftarrow \boldsymbol{\beta}^{(m)} - H^{-1}\boldsymbol{g}, \tag{4.32}$$

with H and $\boldsymbol{g}$ the Hessian and gradient respectively of the loss function L with respect to parameters $\boldsymbol{\beta}$. The gradient is given by equation (4.23) and the elements of the Hessian are given by:

$$H_{jk} = \sum_{i=1}^{B}\Big(\frac{\partial e_{(i)}}{\partial \beta_j}\frac{\partial e_{(i)}}{\partial \beta_k} + e_{(i)}\frac{\partial^2 e_{(i)}}{\partial \beta_j \partial \beta_k}\Big). \tag{4.33}$$

Note that the Hessian H is not necessary positive semi definite (PSD), which can lead to a solution of equation (4.32) with a higher loss. The Gauss-Newton algorithm approximates the Hessian by ignoring the second term in the above equation, as in many cases, the first term dominates the Hessian. This leads to the following approximation of the Hessian:

$$H \approx G \triangleq J^TJ, \tag{4.34}$$

with G the curvature matrix of the Gauss-Newton optimization method. This Gauss-Newton approximation G of the Hessian is always a PSD matrix. By adjusting equation (4.32), the update from Newton's method with the approximated Hessian gives rise to the Gauss-Newton update:

$$\boldsymbol{\beta}^{(m+1)} \leftarrow \boldsymbol{\beta}^{(m)} - \big(J^TJ\big)^{-1}J^T\boldsymbol{e}^{(m)} \tag{4.35}$$

The Gauss-Newton method for multiple-output models

In the previous paragraphs, the Gauss-Newton method was derived for regression models with a one-dimensional output. This can easily be extended to regression models with multi-dimensional outputs, such as most feed-forward neural networks. The regression loss is now given by:

$$L = \frac{1}{2}\sum_{i=1}^{B}\|\boldsymbol{e}_{(i)}\|_2^2 \tag{4.36}$$

$$\boldsymbol{e}_{(i)} \triangleq \boldsymbol{y}_{(i)} - \boldsymbol{t}_{(i)}, \tag{4.37}$$

The Jacobian J of the model outputs and samples with respect to the model parameters can be structured in a 3 dimensional tensor, with its first axis d_s equal to the number of samples, its second axis d_o equal to the number of outputs and its third axis d_β equal to the number of parameters. The derivations of the previous paragraph stay exactly the same, only the matrix products with J^T should be replaced by tensor inner products along d_s and d_o. $\boldsymbol{y}$ and $\boldsymbol{e}$ are now two-dimensional tensors with axes d_s and d_o. In the following sections, we will assume a mini-batch size of 1, such that dimension d_s disappears and we can work again with matrices and vectors for clarity of the reasoning.

The Levenberg-Marquardt method

The linear system (4.28) can sometimes be poorly conditioned, leading to very large step sizes of the Gauss-Newton (GN) method. The Levenberg-Marquardt (LM) method solves this issue by adding a reguralizer matrix to the curvature matrix $G = J^TJ$:

$$\big(J^TJ + \lambda I\big)\Delta\boldsymbol{\beta} = -J^T\boldsymbol{e}^{(m)}, \tag{4.38}$$

with λ the damping parameter. λ is typically updated during each training iteration by a heuristic based on trust regions. The added damping prevents the Levenberg-Marquardt method from taking too large steps and thereby greatly stabilizes the optimization process. Intuitively, the Levenberg-Marquardt method can be seen as an interpolation between Gauss-Newton optimization and gradient descent, as for $\lambda \to 0$ the LM method is equal to the GN method, and for $\lambda \to \infty$ the LM method is equal to gradient descent with a very small step size.

4.2.2 Target Propagation as an approximation of the Gauss-Newton method

In the following derivations, we will show that target propagation with exact inverses uses an approximation of the Gauss-Newton optimization method to compute its local layer targets after which it does gradient descent to update the network parameters in order to move the layer activations closer to the layer targets. We first start with proving a lemma, which will later be used in the main theorem.

Lemma 4.2. *Consider a feed-forward neural network with as forward mapping function $\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(W_i\boldsymbol{h}_{i-1})$, $i = 1,...,L$ where s_i can be any differentiable element-wise function. Furthermore assume a mini-batch size of 1 and a L_2 output loss function. Under these conditions, the Gauss-Newton optimization step for the layer activations, with a block-diagonal approximation of the Gauss-Newton curvature matrix with blocks equal to the layer sizes, is given by:*

$$\Delta\boldsymbol{h}_i = -J_i^{\dagger}\boldsymbol{e}_L, \quad i = 1,...,L-1, \tag{4.39}$$

with $J_i = \frac{\partial \boldsymbol{h}_L}{\partial \boldsymbol{h}_i} = \prod_{k=i+1}^{L} D_{s_k} W_k$, $J_i^{\dagger}$ its Moore-Penrose pseudo-inverse [82, 83] and $\boldsymbol{e}_L = \boldsymbol{h}_L - \boldsymbol{t}$ the output error, with $\boldsymbol{t}$ the output target.

Proof. The output loss function L can be written as:

$$L = \frac{1}{2}\|\boldsymbol{e}_L\|_2^2 \tag{4.40}$$

$$\boldsymbol{e}_L \triangleq \frac{\partial L}{\partial \boldsymbol{h}_L} = \boldsymbol{h}_L - \boldsymbol{t}, \tag{4.41}$$

with $\boldsymbol{h}_L$ the output layer activation and $\boldsymbol{t}$ the output target. As an experiment of thought, imagine that the parameters of our network are the layer activations $\boldsymbol{h}_i$, $i = 1,...,L-1$, concatenated in the total activation vector $\bar{\boldsymbol{h}}$, instead of the weights W_i. The weights can be seen as fixed values for now. If we now want to update the activation values $\bar{\boldsymbol{h}}$ according to the Gauss-Newton method we get the following approximation of the Hessian of the output loss with respect to $\bar{\boldsymbol{h}}$:

$$H \approx G = \bar{J}^T \bar{J} \tag{4.42}$$

with $\bar{J}$ the Jacobian of $\boldsymbol{h}_L$ with respect to $\bar{\boldsymbol{h}}$. $\bar{J}$ can be structured in blocks along the column dimension:

$$\text{Block}_i(\bar{J}) = J_i = \frac{\partial \boldsymbol{h}_L}{\partial \boldsymbol{h}_i}, \quad i = 1,...,L-1 \tag{4.43}$$

Consequently, G can also be structured in blocks of the form:

$$\text{Block}_{i,j}(G) = J_i^T J_j, \quad i,j = 1,...,L-1 \tag{4.44}$$

In the field of Gauss-Newton optimization for deep learning, it is common to approximate G by a block-diagonal matrix $\tilde{G}$ [84, 85, 86], as the authors of [84] show that the GN Hessian matrix is block diagonal dominant for feed-forward neural networks. $\tilde{G}$ then consists of the following diagonal blocks:

$$\text{Block}_{i,i}(\tilde{G}) = J_i^T J_i, \quad i = 1,...,L-1 \tag{4.45}$$

Now the following linear system has to be solved to compute the activations update $\Delta\bar{\boldsymbol{h}}$:

$$\tilde{G}\Delta\bar{\boldsymbol{h}} = -\bar{J}^T\boldsymbol{e}_L, \tag{4.46}$$

As $\tilde{G}$ is block-diagonal, this system factorizes naturally in $L-1$ linear systems of the form

$$J_i^T J_i \Delta\boldsymbol{h}_i = -J_i^T\boldsymbol{e}_L, \quad i = 1,...,L-1 \tag{4.47}$$

$$\Leftrightarrow \Delta\boldsymbol{h}_i = -J_i^{\dagger}\boldsymbol{e}_L, \quad i = 1,...,L-1 \tag{4.48}$$

If $J_i^T J_i$ is not invertible, the Moore-Penrose pseudo-inverse gives the solution $\Delta\boldsymbol{h}_i$ with the smallest norm, so is still the best choice in practice. Due to the layered structure of the network, J_i can be decomposed:

$$J_i = \prod_{k=L}^{i+1} \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{a}_k} \frac{\partial \boldsymbol{a}_k}{\partial \boldsymbol{h}_{k-1}} \tag{4.49}$$

$$J_i = \prod_{k=L}^{i+1} D_{s_k} W_k, \tag{4.50}$$

thereby concluding the proof. □

With this lemma proven, we are now ready to prove the main result of this section.

Theorem 4.3. *Consider a feed-forward neural network with as forward mapping function $\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(W_i \boldsymbol{h}_{i-1})$, $i = 1,...,L$ where s_i can be any differentiable, monotonically increasing and invertible element-wise function. Assume that the inverse of the forward mappings f_i exist and that the backward mapping functions, used for propagating the target activations, are equal to $g_i(\hat{\boldsymbol{h}}_{i+1}) = f_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1}) = W_{i+1}^{-1} s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1})$. Furthermore assume a mini-batch size of 1, a sufficiently small output step size $\hat{\eta}$ and an L_2 output loss function. Under these conditions, target propagation approximately uses Gauss-Newton optimization with a block-diagonal approximation of the Gauss-Newton Hessian, with block sizes equal to the layer size, to compute the local layer targets $\hat{\boldsymbol{h}}_i$.*

Proof. Under the conditions assumed in this theorem, the Gauss-Newton optimization step for the layer activations, with a block-diagonal approximation of the Gauss-Newton Hessian matrix with blocks equal to the layer sizes, is given by (a result of lemma 4.2):

$$\Delta\boldsymbol{h}_i = -J_i^{\dagger} \boldsymbol{e}_L, \quad i = 1,...,L-1, \tag{4.51}$$

with J_i defined as:

$$J_i = \prod_{k=L}^{i+1} \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{a}_k} \frac{\partial \boldsymbol{a}_k}{\partial \boldsymbol{h}_{k-1}} \tag{4.52}$$

$$J_i = \prod_{k=L}^{i+1} D_{s_k} W_k, \tag{4.53}$$

As the forward mappings are assumed to be invertible, the weight matrices W_i are square and of full rank and the diagonal matrices D_{s_k} have non-zero entries. A product of full rank, square matrices is also square and of full rank, indicating that the pseudo inverse $J_i^{\dagger}$ is equal to the real inverse J_i^{-1}. The inverse can be factorized over the decomposed J_i, leading to the following update for $\Delta\boldsymbol{h}_i$:

$$\Delta\boldsymbol{h}_i = -\Big(\prod_{k=i+1}^{L} W_k^{-1} D_{s_k}^{-1}\Big)\boldsymbol{e}_L. \tag{4.54}$$

Now define the layer target $\hat{\boldsymbol{h}}_i^{GN}$ as the updated layer activation:

$$\hat{\boldsymbol{h}}_i^{GN} = \boldsymbol{h}_i + \Delta\boldsymbol{h}_i. \tag{4.55}$$

Due to the block-diagonal approximation, it is common practice in the field to use an optimal step size $\hat{\eta}$ for the parameter update, leading to:

$$\hat{\boldsymbol{h}}_i^{GN} = \boldsymbol{h}_i + \hat{\eta}\Delta\boldsymbol{h}_i. \tag{4.56}$$

$$= \boldsymbol{h}_i - \hat{\eta}\Big(\prod_{k=i+1}^{L} W_k^{-1} D_{s_k}^{-1}\Big)\boldsymbol{e}_L \tag{4.57}$$

This optimal step size will be further discussed in the next section. To get an expression for the targets propagated by the target propagation method, we can do a sequence of first order Taylor expansions around $\boldsymbol{h}_i$, similar to equation (4.9):

$$\hat{\boldsymbol{h}}_i^{TP} = \boldsymbol{h}_i - \hat{\eta}\Big[\prod_{k=i}^{L-1} J_{g_k}\Big]\boldsymbol{e}_L + \mathcal{O}(\hat{\eta}^2) \tag{4.58}$$

$$= \boldsymbol{h}_i - \hat{\eta}\Big[\prod_{k=i+1}^{L} W_k^{-1} D_{s_k}^{-1}\Big]\boldsymbol{e}_L + \mathcal{O}(\hat{\eta}^2). \tag{4.59}$$

We see that $\hat{\boldsymbol{h}}_i^{GN}$ and $\hat{\boldsymbol{h}}_i^{TP}$ are approximately equal with an error of $\mathcal{O}(\hat{\eta}^2)$ in equations (4.57) and (4.59) respectively, thereby proving the theorem.

□

Discussion of the theorem In theorem 4.3, we showed that target propagation uses Gauss-Newton optimization to compute its local layer targets. This is an important result for the research field of biologically plausible learning mechanisms, as there were no theoretical results showing that target propagation does any kind of useful optimization, before this result. Now we do not only have a proof that target propagation does optimize a cost function, but we also proved that it consists of a mix of approximate second-order optimization (for computing the targets) and first-order optimization (for computing the weight updates). In the following, we discuss the important implications of this theorem for the interpretation of target propagation.

- In the theorem, the layer activations were treated as parameters that could be optimized with the Gauss-Newton method. However, the final outcome of the target propagation method is of course to optimize the layer weights. In the original target propagation method [9, 5], this is done by gradient descent on the local layer L_2 loss for updating the network parameters in order to move the layer activations closer to the layer targets:

$$L_i = \|\boldsymbol{h}_i - \hat{\boldsymbol{h}}_i\|_2^2 = \|-\Delta \boldsymbol{h}_i\|_2^2 \tag{4.60}$$

$$\Delta W_i = -\eta_i \frac{\partial L_i}{\partial W_i}. \tag{4.61}$$

  So target propagation mixes 2 optimization methods, with very different characteristics and heuristics to choose the training parameters (such as step sizes and damping). Hence, for obtaining good performance, care has to be taken with choosing suitable training parameters for this mixture of optimization methods. The next section will discuss this topic in more detail.

- The Gauss-Newton method can be extended to other loss functions besides the L_2 loss [87]. We hypothesize that our theorem also works for other output losses, but future work should elaborate in more detail whether target propagation can be made compatible with other output losses for Gauss-Newton.

4.2.3 Choosing training parameters for target propagation

In the deep learning field, there exist a wide variety of training schemes to improve the performance of plain stochastic gradient descent, most notably the momentum method [88], Nesterov momentum [89, 90], Adagrad [91] and Adam [92]. The last two methods have adaptive step sizes, which greatly improves the robustness of the training methods to the chosen values of the training hyperparameters. On the other hand, the Gauss-Newton optimization method has also an extensive variety of training schemes to improve its performance, convergence and robustness, most notably the Levenberg-Marquardt method [93, 94] and the modified Gauss-Newton method [95]. This section first briefly discusses the existing Gauss-Newton training methods for deep neural networks, after which it will explore training schemes for the target propagate method that merges knowledge of both first and second order optimization methods for deep neural networks.

Training parameters for GN optimization in deep neural networks

In the practical Gauss-Newton optimisation for deep learning [85], the authors introduce four extra parameters for training the network parameters θ:

- A damping parameter ψ added to the full Gauss-Newton matrix G,
- A separate damping parameter γ added to the block-diagonal approximation matrix $\tilde{G}$,
- A common additive damping parameter to both G and $\tilde{G}$, with weight $\frac{\alpha}{2}$
- An optimal step size $\hat{\eta}$, resulting from a line search.

The damping parameters can be interpreted in the Levenberg-Marquardt optimization framework [93]. The added damping parameters result in the following full curvature matrix C and diagonal approximated curvature matrix $\tilde{C}$:

$$C = G + (\psi + \alpha)I \tag{4.62}$$

$$\tilde{C} = \tilde{G} + (\gamma + \alpha)I \tag{4.63}$$

The optimal step size $\hat{\eta}$ is introduced to compensate for the diagonal approximation of G. Based on the full curvature matrix C, a quadratic approximation is made of the objective function L:

$$L(\theta + \Delta\theta) \approx \hat{L}(\theta + \Delta\theta; C) \triangleq L(\theta) + \Delta\theta^T \nabla_\theta L + \frac{1}{2}\Delta\theta^T C \Delta\theta \tag{4.64}$$

This quadratic approximation is then used to find an optimal step size $\hat{\eta}$ for the parameter update $\Delta\tilde{\theta}$, that was found with the block-diagonal curvature matrix $\tilde{C}$:

$$\hat{\eta} = \arg\min_\eta \hat{L}(\theta + \eta\Delta\tilde{\theta}; C) = \arg\min_\eta L(\theta) + \eta\Delta\tilde{\theta}^T \nabla_\theta L + \frac{1}{2}\eta^2 \Delta\tilde{\theta}^T C \Delta\tilde{\theta} = -\frac{\Delta\tilde{\theta}^T \nabla_\theta L}{\Delta\tilde{\theta}^T C \Delta\tilde{\theta}} \tag{4.65}$$

Note that the product $\Delta\tilde{\theta}^T C \Delta\tilde{\theta}$ can be computed efficiently both in computations and in memory [85], explaining why it can be used to improve the GN optimization, without the need of explicitly storing C. For adapting the damping parameter ψ throughout training, the authors use a Levenberg-Marquardt heuristic based on the reduction ratio ρ, which is defined as:

$$\rho = \frac{L(\theta + \hat{\eta}\Delta\tilde{\theta}) - L(\theta)}{\hat{L}(\theta + \hat{\eta}\Delta\tilde{\theta}; C) - \hat{L}(\theta; C)}. \tag{4.66}$$

ρ expresses how well the quadratic approximation matches with the real objective function. When $\rho < 1$, the quadratic approximation underestimates the objective curvature, while in the other case it overestimates the curvature. The Levenberg-Marquardt heuristic introduces a parameter $\omega_\psi < 1$. When $\rho < 0.75$, ψ is divided by ω_ψ and in the other case, ψ is multiplied by ω_ψ.

The parameter γ is used to regularize $\tilde{C}$, in order to better match with the quadratic approximation with the full C. γ is updated greedily: the algorithm computes the update $\hat{\eta}\Delta\tilde{\theta}$ for each of $\{\omega_\gamma\gamma, \gamma, \omega_\gamma^{-1}\gamma\}$ with scaling factor ω_γ, and the one with the lowest $\hat{L}$ is selected.

Training parameters for target propagation

The target propagation method was first proposed as a more biologically plausible method for training neural networks, compared to error backpropagation. The main focus of this thesis is also on biologically plausible training methods for neural networks, so the training parameters for target propagation should also have a certain abstract biologically plausible implementation. This implies that neurons should have all training information locally available. This imposes a major restriction on possible adaptive training parameters. If one is interested to exploit the newly discovered quasi-Newton characteristics of target propagation in a non-biological setting, the training parameters of the previous section can be tailored towards target propagation without restrictions, but in the following, we will limit the discussion to possible local implementations of training parameters for target propagation. We first focus on finding an alternative for the damping parameters γ and α, that only needs information which is locally available to each layer in the network. Afterwards, we investigate which step sizes and learning rates should be used for target propagation.

An alternative for damping parameters in target propagation. We start by investigating the influence of the damping parameters introduced by the Levenberg-Marquardt (LM) modification of the Gauss-Newton method. LM adds an extra diagonal term λI to the GN hessian matrix $\tilde{G}$. We have reduced the two damping parameters γ and α of the above paragraph into one damping parameter λ for simplifying the discussion. Adding a damping parameter increases the curvature of the quadratic approximation and consequently decreases the step size of the parameter update. This can be seen more clearly by making use of the singular value decomposition of $J_i = U_i \Sigma_i V_i^T$. The i-th diagonal block of $\tilde{C}$ can then be written as:

$$\text{Block}_i(\tilde{C}) \triangleq \tilde{C}_i = J_i^T J_i + \lambda I \tag{4.67}$$
$$= V_i \Sigma_i^T \Sigma_i V_i^T + \lambda V_i V_i^T \tag{4.68}$$
$$= V_i \big(\Sigma_i^T \Sigma_i + \lambda I\big) V_i^T \tag{4.69}$$

There is thus a constant term λ added to each singular value of $\tilde{C}_i$, indicating an increase in curvature. λ will have the biggest effect on the smallest singular values, corresponding to the directions of smallest curvature. This will greatly decrease the step size in those directions of small curvature, as the minimum of the quadratic approximation will now be much closer to the current iteration point in those directions. If J_i is square, $\tilde{C}_i$ can be interpreted as the Gauss-Newton matrix computed with an adjusted Jacobian J_i^*:

$$\tilde{C}_i = J_i^{*^T} J_i^* \tag{4.70}$$
$$J_i^* = V_i \sqrt{\Sigma_i^2 + \lambda I} V_i^T. \tag{4.71}$$

In the LM method, the layer activation update would be computed as

$$\Delta \boldsymbol{h}_i^{LM} = -\hat{\eta}(J_i^T J_i + \lambda I)^{-1} J_i^T \boldsymbol{e}, \tag{4.72}$$

which is not equal anymore to $\Delta \boldsymbol{h}_i = -\hat{\eta} J_i^{\dagger} \boldsymbol{e}$. In target propagation, the targets are computed following equation (4.58). By comparing equation (4.58) and (4.72), the following equality must hold for target propagation to incorporate the LM method:

$$\prod_{k=i}^{L-1} J_{g_k} = (J_i^T J_i + \lambda I)^{-1} J_i^T \tag{4.73}$$

In general, it is not possible to find a sequence of g_i for which this equality always holds based on only local information, as the influence of λ on the singular values of $J_i^T J_i$ cannot be factored over the different g_k as is needed for the left-hand expression, without having access to global information (all the J_{g_k}'s). In what follows, we will explore a training scheme for target propagation with exact inverses that incorporates the same principles as LM (ensuring that there are no directions with too low curvature), but with only local information. First, we approximate the LM update (4.72) by the pseudo-inverse of J_i^*:

$$\Delta \boldsymbol{h}_i^* = -\hat{\eta}(J_i^{*^T} J_i^*)^{-1} J_i^{*^T} \boldsymbol{e} \tag{4.74}$$
$$= -\hat{\eta}(J_i^T J_i + \lambda I)^{-1} J_i^{*^T} \boldsymbol{e} \tag{4.75}$$
$$= -\hat{\eta}\Big((J_i^T J_i + \lambda I)^{-1} J_i^T \boldsymbol{e} + (J_i^T J_i + \lambda I)^{-1} U(\sqrt{\Sigma_i^2 + \lambda I} - \Sigma_i) V^T \boldsymbol{e}\Big) \tag{4.76}$$

The first term on the right-hand side is equal to the LM update (4.72), while the second term is the error of the approximation. This error is small comparing to the LM update, as

$$\|U(\sqrt{\Sigma_i^2 + \lambda I} - \Sigma_i) V^T\|_2 \leq \lambda, \tag{4.77}$$

and λ is small compared to the average singular values of J_i. Note that if $\boldsymbol{e}$ lies entirely along a direction corresponding to one of the smallest singular values, the relative approximation error can be significant, but in general there will likely be parts of $\boldsymbol{e}$ that lay alongside higher singular values.

If now the Jacobian of the sequence of forward mappings f_i would match J_i^* and g_i be the inverse of the forward mappings, the target propagation update would be equal to the LM approximation (4.74). However, as the Jacobian of the sequence of forward mappings f_i is also factored by the product of J_{f_i}, the influence of λ on the singular values of J_i should be factored over all J_{f_i}. This is not possible for target propagation, as it would need global information of all $J_{f_{j\neq i}}$ to adjust one particular J_{f_i}. Therefore, we make one last approximation to the LM update: instead of adding a diagonal term λ to $J_i^T J_i$ in order to make the smallest singular values bigger, we ensure that all singular values of $J_i^T J_i$ are big enough. In this way, there are no small singular values causing small curvature in the GN Hessian matrix and there is thus no need to adjust them by λ. Note that this approximation is very crude, but that it still has the same goal as the LM-update: ensure that there are no directions with too low curvature to prevent too large step sizes.

As the smallest singular value of J_i for a network as specified in lemma 4.8 is bounded by below by the product of the smallest singular values of J_{f_k}, $k = i+1,...,L$, it suffices to ensure that the singular values of J_{f_k} are sufficiently big. J_{f_k} can be written as:

$$J_{f_k} = D_{s_k} W_k, \tag{4.78}$$

with D_{s_k} a diagonal matrix with the derivatives of the element-wise non-linearity and W_k the k-th layer weight matrix. If we assume that the network has invertible non-linearities without saturation (e.g. leaky-ReLU), the diagonal terms of D_{s_k} (and consequently its singular values) will not be close to zero. To ensure that the singular values of J_{f_k} are big enough, it thus suffices to make sure that the smallest singular value σ_{min} of W_k is never too close to zero. For this, two actions need to be taken: (1) when initializing the network, the random weight matrices W_i should have a large enough σ_{min} and (2) during training, we have to prevent that the weight updates (4.12) cause W_i to become close to singular.

Initialization of weight matrices. Two approaches can be taken in order to ensure initial weight matrices with σ_{min} big enough.

1. One can keep generating random matrices until σ_{min} is above a user-specified threshold τ_{init}. A better measure could be that $\sigma_{min} \cdot \sigma_{max}$ should be above a user-specified threshold, such that the norm of W_i^{-1} is roughly smaller than the norm of W_i. Note that the expected magnitude of σ_{min} of random Gaussian matrices is of order $\mathcal{O}(n^{-0.5})$ [96] with n the size of the matrix, which means that this brute-force technique will work well for small networks, but will have trouble to find good weight matrix initializations for big networks.
2. One can perform a singular value decomposition of the random initialized weight matrix, change all singular values s_i below a user-specified threshold to a larger value and take the resulting matrix as weight matrix initialization. Note however that this method can interfere with the random properties of the matrix, as the singular values of the random matrix are changed.

For this thesis, we use method 1 to initialize the weight matrices, as we use only small networks and it works well in practice.

Robust weight updates. Even when we start with far-from-singular weight initializations, the weight matrices W_i can still become singular during training. From equation (4.19) we can see that the inverse update becomes unstable when the denominator $1 + d = 1 + \boldsymbol{v}_i^T W_i^{-1} \boldsymbol{u}_i$ of the right-hand side has a value close to zero. In chapter 5, we will see that $1 + d$ experimentally correlates very well with σ_{min} of the updated forward matrix W_i. A straight forward method to keep the updated matrices W_i far away from singularity is to clip the denominator when its absolute value is below a user-specified threshold ϵ. However, to ensure that the updated inverse remains the exact inverse of the forward weight matrix, the forward weight update ΔW_i should be scaled by a corresponding β, which can be interpreted as a scaling of the learning rate of the specific layer. The resulting robust Sherman-Morrison update is depicted in algorithm 4.1.

The scaling value β is derived from the condition that the clipped inverse update must be the perfect inverse of the scaled forward update:

$$I = \left(W_i + \beta \boldsymbol{u}_i \boldsymbol{v}_i^T\right)\left(W_i^{-1} - \frac{W_i^{-1}\boldsymbol{u}_i\boldsymbol{v}_i^T W_i^{-1}}{\epsilon}\right) \tag{4.79}$$

$$\Rightarrow \quad \beta = \frac{1}{\epsilon - \boldsymbol{v}_i^T W_i^{-1} \boldsymbol{u}_i} \tag{4.80}$$

Optimal output target step size $\hat{\eta}$. The previous paragraphs focused on finding a substitute method for the LM damping parameters used in conventional GN optimization. However, the discussed Gauss-Newton method for deep learning has also an optimal step size parameter $\hat{\eta}$ resulting from a line search. From equation (4.65), we see that the full curvature matrix C is used for computing the optimal step size. C consists of non-local matrix products $J_i^T J_j$, making it not feasible for target propagation to use a similar line search. This can be explained intuitively, as the line search is used to compensate the block-diagonal approximation of the curvature matrix. However, the structure of target propagation itself

Algorithm 4.1: Robust Sherman-Morrison update

```
Result: $W_{i,(m+1)}$ and $W^{-1}_{i,(m+1)}$: robust update of the forward weight matrix and its inverse with the
        adjusted Sherman-Morrison update.
$\boldsymbol{u}_i = -\eta_i D_{s_i}(\boldsymbol{h}_i - \hat{\boldsymbol{h}}_i)$;
$\boldsymbol{v}_i = \boldsymbol{h}_{i-1}$;
$d = \boldsymbol{v}_i^T W^{-1}_{i,(m)} \boldsymbol{u}_i$;
if $|1+d| \geq \epsilon$ then
    $W_{i,(m+1)} = W_{i,(m)} + \boldsymbol{u}_i \boldsymbol{v}_i^T$;
    $W^{-1}_{i,(m+1)} = W^{-1}_{i,(m)} - \frac{W^{-1}_{i,(m)} \boldsymbol{u}_i \boldsymbol{v}_i^T W^{-1}_{i,(m)}}{1+d}$;
else
    $\beta = \frac{1}{\epsilon - d}$;
    $W_{i,(m+1)} = W_{i,(m)} + \beta \boldsymbol{u}_i \boldsymbol{v}_i^T$;
    $W^{-1}_{i,(m+1)} = W^{-1}_{i,(m)} - \frac{W^{-1}_{i,(m)} \boldsymbol{u}_i \boldsymbol{v}_i^T W^{-1}_{i,(m)}}{\epsilon}$;
```

limits its available information to the block-diagonal approximation, thus it cannot compensate for it due to this lack of global information. For the simulations done in this thesis, a decreasing step-size scheme is used for $\hat{\eta}$, similar to the decreasing step-size schemes used in many first-order optimization training schemes. However, this approach is prone to a lot of hyperparameter tuning and is far from ideal (as it was developed for first order optimization methods), so more thorough theoretical and experimental research on this matter is encouraged for future work.

Step-size for gradient descent on layer loss. The local layer targets were all computed by an approximation of Gauss-Newton optimization, whereas the layer weights are updated by gradient descent on the local layer loss L_i. As this is a form of first-order optimization, a step-size η is needed. We take all layer step-sizes η_i equal to η, to reduce the number of training hyperparameters, but in future work, layer specific step sizes can be explored. In state-of-the-art deep learning, adaptive training schemes are used such as Adagrad [91] or Adam [92]. However, in order to not mix too many properties of first and second order optimization, we choose here for a simple step-size decay scheme as is used in stochastic gradient descent. As can be seen in equation (4.12), both $\hat{\eta}$ and η have the same influence on the weight update ΔW_i after the first order Taylor expansion. Consequently, we opt for implementing the decaying step-size scheme in $\hat{\eta}$ instead of η, as the Taylor approximation is more accurate for small $\hat{\eta}$ and thus target propagation will more accurately approximate Gauss-Newton optimization. η can than be taken a constant.

Discussion on the training parameters of target propagation. In this section, we derived a local training scheme for target propagation with exact inverses that only uses local information. To summarize, 4 training hyperparameters are used: τ_{init} for the initialization of the weight matrices, ϵ for ensuring that the weight matrices do not become close to singular during training, the output target step size $\hat{\eta}$ and the step size η used for gradient descent on the local layer loss. We end this section with some comments on the obtained training method for target propagation.

- Although the robust weight updates have the same purpose as the LM update, there are also some fundamental differences. With the robust weight updates, the forward weight matrices are forced to remain far from singularity, while with the LM update, the weight matrices can be singular, as the LM update compensates for this. Furthermore, the LM update can change both the direction and length of the update step (it can be interpreted as an interpolation between the Gauss-Newton step and the gradient descent step), while our robust weight update scheme can only adjust the length of the update step with β.

- Note that β is always smaller than one because ϵ is always taken positive (instead of e.g. $\text{sign}(1+d)\cdot\epsilon$), resulting in an adaptive decrease of the step-size. This was done to improve the stability of training.

4.3 Target propagation with approximate inverses

In a biological setting, it is not very likely that the exact inverse of the forward mapping can be computed with a method such as the Sherman-Morrison algorithm. Therefore, this section explores the target propagation method where g_i *learns* to approximate the inverse of f_{i+1}, instead of computing the exact inverse during training. This section starts with investigating the influence of the occurring reconstruction errors on the training of the network, after which it discusses methods to learn the approximate inverses of f_i. The section ends with a brief analysis of Difference Target Propagation [5] in this mathematical framework.

4.3.1 The influence of reconstruction errors on the network training

We repeat the previous Taylor analysis of the backpropagated targets in section 4.1.1, but now with g_i not necessarily equal to the exact inverse of f_{i+1}. For clarity of the derivation, we start with the penultimate layer. The target of the second to last layer can be approximated by a first order Taylor expansion around $\boldsymbol{h}_{L-1}$ as follows.

$$\begin{aligned}\hat{\boldsymbol{h}}_{L-1} &= g_{L-1}(\hat{\boldsymbol{h}}_L) = g_{L-1}(\boldsymbol{h}_L - \hat{\eta}\boldsymbol{e}_L) &(4.81)\\ &\approx g_{L-1}(\boldsymbol{h}_L) - \hat{\eta} J_{g_{L-1}}\boldsymbol{e}_L &(4.82)\\ &= \boldsymbol{h}_{L-1} + \big(g_{L-1}(\boldsymbol{h}_L) - \boldsymbol{h}_{L-1}\big) - \hat{\eta} J_{g_{L-1}}\boldsymbol{e}_L &(4.83)\end{aligned}$$

By repeating the first order Taylor expansions for the other layers, the following general approximation for $\hat{\boldsymbol{h}}_i$ is obtained.

$$\hat{\boldsymbol{h}}_i \approx \boldsymbol{h}_i - \hat{\eta}\left[\prod_{k=i}^{L-1} J_{g_k}\right]\boldsymbol{e}_L + \big(g_i(\boldsymbol{h}_{i+1}) - \boldsymbol{h}_i\big) + \sum_{j=i+1}^{L-1}\left[\left(\prod_{k=i}^{j-1} J_{g_k}\right)\big(g_j(\boldsymbol{h}_{j+1}) - \boldsymbol{h}_j\big)\right]. \qquad (4.84)$$

When comparing this approximation of $\hat{\boldsymbol{h}}_i$ to the one obtained in (4.84) with exact inverses, we see that two extra error terms appeared. The third term on the right-hand side of (4.84) represents the reconstruction error of g_i in the current layer, while the fourth term represents the backpropagated reconstruction errors of the layers on top. Note that these backpropagated reconstruction errors can either be amplified or attenuated by J_{g_k}, depending on the singular values of these Jacobians. The weight updates are derived from a gradient step on the local L_2 layer loss $L_i = \|\hat{\boldsymbol{h}}_i - \boldsymbol{h}_i\|_2^2$:

$$\begin{aligned}\Delta W_i &= -\eta_i D_{s_i}(\boldsymbol{h} - \hat{\boldsymbol{h}}_i)\boldsymbol{h}_{i-1}^T &(4.85)\\ &\approx -\eta_i D_{s_i}\left(\hat{\eta}\left[\prod_{k=i}^{L-1} J_{g_k}\right]\boldsymbol{e}_L\right)\boldsymbol{h}_{i-1}^T + \eta_i D_{s_i}\left(\big(g_i(\boldsymbol{h}_{i+1}) - \boldsymbol{h}_i\big) + \sum_{j=i+1}^{L-1}\left[\left(\prod_{k=i}^{j-1} J_{g_k}\right)\big(g_j(\boldsymbol{h}_{j+1}) - \boldsymbol{h}_j\big)\right]\right)\boldsymbol{h}_{i-1}^T, &(4.86)\end{aligned}$$

By comparing the above equation to (4.12), we see that the two reconstruction error terms of equation (4.84) cause an error term in the weight update ΔW_i. If the approximations g_i of the inverses of f_{i+1} are not accurate enough, the reconstruction error term will dominate the weight update, resulting in weight updates that do not contribute to the objective of minimizing the output loss. Therefore, it is of uttermost importance that either the approximation of the inverses are accurate enough, or that an adjustment to the target propagation method is made to get rid of the occurring reconstruction errors. The next section will explore which forms and training methods of g_i can provide the network with good approximations of the inverses of f_{i+1}, while the last section will interpret Difference Target Propagation [5] as a method to get rid of the reconstruction errors.

4.3.2 Approximate the inverses

The goal of this section is to explore how to find a suitable form and training method for g_i in order to approximate the inverse of f_{i+1}:

$$f_{i+1}^{-1}(\boldsymbol{h}_{i+1}) = W_{i+1}^{-1} s_{i+1}^{-1}(\boldsymbol{h}_{i+1}) \qquad (4.87)$$

Note that this inverse only can exist between layers of equal size. For layers of unequal size, we want to approximate the unexisting inverse by its pseudo-inverse defined as:

$$f_{i+1}^{\dagger}(\boldsymbol{h}_{i+1}) = W_{i+1}^{\dagger} s_{i+1}^{-1}(\boldsymbol{h}_{i+1}), \qquad (4.88)$$

with $W_{i+1}^{\dagger}$ the Moore-Penrose pseudo-inverse [82, 83] of W_{i+1}. In the following derivations and discussions, we assume that the element-wise non-linearity s_i is always invertible. The section starts with defining an appropriate form for g_i, after which it discusses learning methods to train the parameters of g_i.

Defining an appropriate form of g_i

In the original papers on target propagation [9, 5], the authors use the following form of g_i:

$$g_i(\boldsymbol{h}_{i+1}) = t_i(Q_i \boldsymbol{h}_{i+1}), \tag{4.89}$$

with t_i an element-wise non-linearity and Q_i the backward weight matrix. The authors based this form on the theory of auto-encoders, as the f_{i+1} - g_i pair can be seen as an auto-encoder. However, it is important to notice that the f_{i+1} - g_i pair is a shallow auto-encoder (it has no hidden layers in both its encoder f_{i+1} and decoder g_i). Therefore, the universal approximation theorem [97] does not hold for g_i and in general, it will not be able to approximate the inverse of f_{i+1} to arbitrary precision. This means that the reconstruction errors of equation (4.85) will keep interfering with the weight updates ΔW_i during training. Either a hidden layer should be introduced to g_i, or another form of g_i should be used. We propose the following form of g_i:

$$g_i(\boldsymbol{h}_{i+1}) = Q_i t_i(\boldsymbol{h}_{i+1}) \tag{4.90}$$

When t_i is taken equal to s_{i+1}^{-1} and if the inverse of W_{i+1} exists, this form of g_i can approximate f_{i+1}^{-1} to arbitrary precision, as it is of the same form as f_{i+1}^{-1}. From a biological perspective, this form of g_i is less appealing than the one of equation (4.89), but from a mathematical point of view this form makes more sense. For future work, other forms of g_i should be explored that can both approximate f_{i+1}^{-1} to a sufficiently high precision and that are more biologically plausible.

The training of g_i

In the original papers on target propagation [9, 5], the authors propose the following loss function to train the backward parameters Q_i:

$$L_i^{inv}\Big(g_i\big(f_{i+1}(\boldsymbol{h}_i)\big), \boldsymbol{h}_i\Big) = \big\| g_i\big(f_{i+1}(\boldsymbol{h}_i)\big) - \boldsymbol{h}_i \big\|_2^2 \tag{4.91}$$

In the following, we will analyse the training behavior of the backward weights Q_i when using gradient descent on the inverse loss function defined in equation (4.91) with the form of g_i as specified in equation (4.90). We start with proving two lemmas on the gradient of L_i^{inv} and its corresponding energy function, after which a theorem that under certain conditions Q_i approximates $W_{i+1}^{\dagger}$ is proposed and proved.

Lemma 4.4. *Consider a feed-forward neural network with forward mapping $f_i(\boldsymbol{h}_{i-1}) = s_i(W_i \boldsymbol{h}_{i-1})$ and backward mapping $g_i(\boldsymbol{h}_{i+1}) = Q_i s_{i+1}^{-1}(\boldsymbol{h}_{i+1})$. In this network setting, the expected gradient $\mathbb{E}\left[\nabla_{Q_i} L_i^{inv}\right]$ with L_i^{inv} as specified in equation* (4.91) *is equal to*

$$\mathbb{E}\left[\nabla_{Q_i} L_i^{inv}\right] = 2(Q_i W_{i+1} - I)\Gamma_i W_{i+1}^T, \tag{4.92}$$

with Γ_i the covariance matrix of $\boldsymbol{h}_i$.

The proof of this lemma can be found in appendix B.

Lemma 4.5. *Consider a feed-forward neural network with forward mapping $f_i(\boldsymbol{h}_{i-1}) = s_i(W_i \boldsymbol{h}_{i-1})$ and backward mapping $g_i(\boldsymbol{h}_{i+1}) = Q_i s_{i+1}^{-1}(\boldsymbol{h}_{i+1})$. If $\boldsymbol{h}_i$ is uncorrelated and has equal variance σ^2, the expectation of the inverse loss function L_i^{inv}, defined in equation* (4.91) *can be written as the following decoupled energy function $E(Q_i)$ depending on w_j and q_k, the columns of W_{i+1} and the rows of Q_i respectively:*

$$E(Q_i) = \sigma^2 \sum_j \Big[\big(q_j^T w_j - 1\big)^2 + \sum_{k \neq j} \big(q_k^T w_j\big)^2 \Big]. \tag{4.93}$$

The proof of the lemma can be found in appendix B.

This lemma shows that during the training of the backward weights Q_i, the rows q_j of Q_i start to couple with the columns of the same index w_j of W_{i+1} towards an inner product of 1, while they start to decouple from all other columns $w_{k\neq j}$. If W_{i+1} is square and of full rank, there exist precisely one Q_i for which $q_j^T w_j = 1$ and $q_j^T w_{k\neq j} = 0$ holds, as there are as many equations as variables. If W_{i+1} has more rows than columns, in general there exist multiple solutions for Q_i for which $q_j^T w_j = 1$ and $q_j^T w_{k\neq j} = 0$ holds, whereas if W_{i+1} has less rows than columns, there are more equations than parameters, thus in general no solution Q_i exist for which $q_j^T w_j = 1$ and $q_j^T w_{k\neq j} = 0$ holds, and a minimum of $E(Q_i)$ different from zero is found. These insights lead to the following theorem.

Theorem 4.6. *Consider a feed-forward neural network with forward mapping $f_i(\boldsymbol{h}_{i-1}) = s_i(W_i\boldsymbol{h}_{i-1})$ and backward mapping $g_i(\boldsymbol{h}_{i+1}) = Q_i s_{i+1}^{-1}(\boldsymbol{h}_{i+1})$. If W_{i+1} is square and of full rank, the minimization of the inverse loss L_i^{inv}, defined in equation* (4.91) *leads in expectation towards the unique solution*

$$Q_i^* = W_{i+1}^{-1}, \tag{4.94}$$

if and only if Γ_i, the covariance matrix of $\boldsymbol{h}_i$, is of full rank.
If W_{i+1} is not square, the minimization of the inverse loss L_i^{inv}, defined in equation (4.91) *leads in expectation towards the unique solution*

$$Q_i^* = W_{i+1}^{\dagger}, \tag{4.95}$$

with $W_{i+1}^{\dagger}$ the Moore-Penrose pseudo-inverse of W_{i+1}[82, 83], if and only if W_{i+1} has linearly independent rows and if h_i is uncorrelated and has equal variances different from zero.

The proof of this theorem can be found in appendix B.

Discussion on theorem 4.6. The first result of this theorem is that if W_{i+1} is of full rank, the backward weights Q_i approximate W_{i+1}^{-1} when they optimize the loss function L_i^{inv} of equation (4.91), if and only if the covariance matrix of $\boldsymbol{h}_i$ is of full rank. The second result of this theorem shows that if W_{i+1} is not of full rank, the backward weights Q_i approximate $W_{i+1}^{\dagger}$ when they optimize the loss function L_i^{inv}, if and only if W_{i+1} has linearly independent rows and $\boldsymbol{h}_i$ is uncorrelated and has equal variances. In the following, the implications of the conditions of the theorem are discussed.

- If W_{i+1} is of full rank, the only condition is that Γ_i has to be of full rank. During the training phase of Q_i, this condition on $\boldsymbol{h}_i$ should thus be fulfilled.
- The condition that $\boldsymbol{h}_i$ is uncorrelated and has equal variances implies that is should represent white noise during the training phase of Q_i. This is biologically speaking a bit awkward, as $\boldsymbol{h}_i$ represent firing rates of neurons, which are always positive. However, as discussed in Akrout et al. [98], this inconsistency can be solved by treating $\boldsymbol{h}_i$ as the deviation of the firing rates from a specified base firing rate, which could be determined genetically in the neurons or by a moving average.
- As $\boldsymbol{h}_i$ has to represent white noise, Q_i has to be trained for each layer with a separate white noise injection of the neurons. If only a white noise injection at the bottom layer would be done, the propagated $\boldsymbol{h}_i$ is in general not uncorrelated anymore in the other network layers. The layer activations $\boldsymbol{h}_i$ used for training the backward weights are thus different from the layer activations $\boldsymbol{h}_i$ during the forward phase of the network.
- The condition regarding the linear independence of the rows of W_{i+1} implies that in the network, only equal or diminishing layer sizes are allowed ($n_{i+1} \leq n_i$, with n_i the dimension of the i-th layer). Therefore, we should explore if L_i^{inv} of equation (4.91) could be adjusted with regularizing terms to ensure that Q_i approximates $W_{i+1}^{\dagger}$, also for expanding layers. In the next theorem, we will show that this can be done by regularizing the Frobenius norm of Q_i. Another interesting observation is that the expected gradient of L_i^{inv} with $\Gamma_i = \sigma^2 I$, shown in equation (4.92) has a related form to the iterative method of Ben-Israel and Cohen [99], used for iterative approximating the pseudo-inverse. This iterative method has the following iteration step for letting Q_i approximate W_{i+1}^{-1}:

$$Q_i^{(m+1)} = 2Q_i^{(m)} - Q_i^{(m)} W_{i+1} Q_i^{(m)}, \tag{4.96}$$

  which can be interpreted as a gradient descent step with gradient $\nabla_{Q_i} L_i^{inv} = \left(Q_i W_{i+1} - I\right) Q_i$ and a step size equal to one. This gradient has the same form as in equation (4.92), only with W_{i+1}^T replaced by Q_i. In future research, it should thus be investigated if L_i^{inv} can be adapted in order to incorporate the Ben-Israel and Cohen method, as this method approximates the pseudo-inverse without restrictions on the layer sizes and has a quadratic convergence rate.

Improved inverse loss function. We now propose a new regularized inverse loss function $L_i^{inv,r}$ for which the minimum will approximate $W_{i+1}^{\dagger}$ more generally.

$$L_i^{inv,r}\Big(g_i\big(f_{i+1}(\boldsymbol{h}_i)\big),\boldsymbol{h}_i\Big) = \|g_i\big(f_{i+1}(\boldsymbol{h}_i)\big) - \boldsymbol{h}_i\|_2^2 + \lambda\|Q_i\|_F^2. \tag{4.97}$$

The square of the Frobenius norm is defined as:

$$\|Q_i\|_F^2 = \sum_{j,k}\big(Q_i^{(j,k)}\big)^2, \tag{4.98}$$

with $Q_i^{(j,k)}$ the element on the j-th row and k-th column of Q_i. This new loss function uses *weight decay*, a regularizing method widely used in the deep learning community [100].

Theorem 4.7. *Consider a feed-forward neural network with forward mapping $f_i(\boldsymbol{h}_{i-1}) = s_i(W_i\boldsymbol{h}_{i-1})$ and backward mapping $g_i(\boldsymbol{h}_{i+1}) = Q_i s_{i+1}^{-1}(\boldsymbol{h}_{i+1})$. If and only if h_i is uncorrelated and has equal variances σ^2, the minimization of the inverse loss $L_i^{inv,r}$, defined in equation* (4.97) *leads in expectation towards the unique solution*

$$Q_i^* = W_{i+1}^T\big(W_{i+1}W_{i+1}^T + \frac{\lambda}{\sigma^2}I\big)^{-1}. \tag{4.99}$$

When the weight-decay parameter λ is driven in limit to zero, this results in the unique solution

$$\lim_{\lambda\to 0} Q_i^* = W_{i+1}^{\dagger} \tag{4.100}$$

with $W_{i+1}^{\dagger}$ the Moore-Penrose pseudo-inverse of W_{i+1} [82, 83].

The proof of this theorem can be found in appendix B.

Now we have an inverse loss function that lets the backward weights Q_i converge in expectation towards the pseudo-inverse of the forward weights W_{i+1}, independently of the layer sizes. $\boldsymbol{h}_i$ still needs to be white noise during the training phase of the backward weights, as was discussed after theorem 4.6. If W_{i+1} is of full rank, Q_i will approximate $W_{i+1}^{\dagger}$ by minimizing $L_i^{inv,r}$ for $\lambda \to 0$, if and only if Γ_i is of full rank, thereby relaxing the condition on $\boldsymbol{h}_i$ (same proof as in theorem 4.6). In the following section, we will analyse difference target propagation [5], and with the help of the above theorems and lemmas, we will show that under well-specified conditions, difference target propagation uses Gauss-Newton optimization to define its local layer targets, even when it only has approximate inverses.

4.3.3 A theoretical analysis of difference target propagation

In difference target propagation (DTP) [5], the authors introduce a difference correction in the propagated target $\hat{\boldsymbol{h}}_i$, in order to make sure that if one layer matches its target, all lower layers should also match their target:

$$\hat{\boldsymbol{h}}_i = g_i(\hat{\boldsymbol{h}}_{i+1}) + \big(\boldsymbol{h}_i - g_i(\boldsymbol{h}_{i+1})\big) \tag{4.101}$$

With the new mathematical framework of target propagation, we see that this correction exactly cancels out the reconstruction error terms of (4.84), as there are no reconstruction errors to be propagated backwards anymore. The local layer targets can now again be approximated as in (4.10):

$$\hat{\boldsymbol{h}}_i \approx \boldsymbol{h}_i - \hat{\eta}\Big[\prod_{k=i}^{L-1} J_{g_k}\Big]\boldsymbol{e}_L \tag{4.102}$$

Note however that the Jacobians J_{g_k} are not equal anymore to $J_{f_{k+1}}^{-1}$, as no exact inverses are used. The success of difference target propagation can be explained by the disappearance of these reconstruction error terms in the backward propagation of the targets. Each layer still receives a clean error signal from the output, only the transformations of the target error via g_k are not completely equal anymore to the ones obtained in target propagation with exact inverses. When an L_2 loss function is used for the local layer loss L_i, the forward weight update ΔW_i (resulting from a gradient step on the local loss function) are now given by:

$$\Delta W_i = -\eta_i D_{s_i}(\boldsymbol{h} - \hat{\boldsymbol{h}}_i)\boldsymbol{h}_{i-1}^T \tag{4.103}$$

$$\approx -\eta_i D_{s_i}\Big(\hat{\eta}\Big[\prod_{k=i}^{L-1} J_{g_k}\Big]\boldsymbol{e}_L\Big)\boldsymbol{h}_{i-1}^T, \tag{4.104}$$

In what follows, we will propose and prove the final theorem of this chapter, showing that under well-specified conditions, difference target propagation uses Gauss-Newton optimization to compute its target values. We will first prove a lemma, that is needed later on for proving the theorem.

Lemma 4.8. *Consider a feed-forward network with L layers and with as forward mapping function* $\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(W_i\boldsymbol{h}_{i-1})$, $i = 1,...,L$ *where* s_i *can be any differentiable, monotonically increasing and invertible element-wise function. The Moore-Penrose pseudo-inverse of all Jacobians* $J_i = \frac{\partial \boldsymbol{h}_L}{\partial \boldsymbol{h}_i} = \prod_{k=L}^{i+1} D_{s_k} W_k$, $i = 1,...,L-1$, *can be factorized as* $J_i^\dagger = \prod_{k=i+1}^{L} W_k^\dagger D_{s_k}^\dagger$ *if and only if* $n_L = n_{L-1} = ... = n_2$ *and* $n_2 \leq n_1$, *with* n_i *the dimension of the i-th layer,* W_i *is of full rank for* $i = 3,...,L-1$ *and* W_2 *is of full row rank.*

The proof of this lemma can be found in appendix B.

With this lemma proven, we are now ready to state the main result of this section.

Theorem 4.9. *Consider a feed-forward neural network with as forward mapping function* $\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(W_i\boldsymbol{h}_{i-1})$, $i = 1,...,L$ *where* s_i *can be any differentiable, monotonically increasing and invertible element-wise function. Take the backward mapping functions , used for propagating the target activations, equal to* $g_i(\hat{\boldsymbol{h}}_{i+1}) = Q_i s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1})$ *and assume that after each forward weight update, they are trained until optimality with white noise* $\boldsymbol{h}_i$ *and loss function* $L_i^{inv,r}$ *as defined in equation* (4.97) *with* $\lambda \to 0$ *in the limit. Furthermore assume a mini-batch size of 1, a sufficiently small output step size* $\hat{\eta}$ *and an* L_2 *output loss function. Finally, assume that* $n_L = n_{L-1} = ... = n_2$ *and* $n_2 \leq n_1$, *with* n_i *the dimension of the i-th layer, that* W_i *is of full rank for* $i = 3,...,L-1$ *and* W_2 *is of full row rank. Under these conditions, difference target propagation approximately uses Gauss-Newton optimization with a block-diagonal approximation of the Gauss-Newton Hessian to compute the local layer targets* $\hat{\boldsymbol{h}}_i$.

As the proof of this main theorem is similar to the proof of theorem 4.3, it is moved to appendix B.

Discussion of theorem 4.9. This last theorem of the chapter proves that target propagation can approximate Gauss-Newton optimization in a more general network setting, with plastic feedback weights and more general architectures.

- The property of difference target propagation that it eliminates the reconstruction error of equation (4.84) is crucial for the theorem to hold for different layer sizes, because even when the g_i is equal to the pseudo-inverse of f_{i+1}, there will still be reconstruction errors in pure target propagation when different layer sizes are used. Only when equal layer sizes are used, pure target propagation with plastic feedback functions g_i can be used to approximate Gauss-Newton optimization, without having reconstruction errors that interfere with the learning signal.

- Lemma 4.8 implicates that the theorem only holds for networks with constant layer sizes except from the first two hidden layers (the first hidden layer can be any dimension, and the second hidden layer must be smaller than the first one, after which all further layers need to be of the same size as the second hidden layer). This is a serious restriction on the possible network architectures. It should thus be experimentally investigated whether the theorem still approximately holds for other network architectures.

- Similarly to target propagation with exact inverses, the weights are trained by performing a gradient step on a local loss function in function of the layer target and the layer activation. This makes difference target propagation a mix between Gauss-Newton optimization (for the targets) and gradient descent (for the weights), which makes it not trivial to find good training parameters. For the output step size $\hat{\eta}$ and the gradient step size η, the same comments hold as in section 4.2.3. The regularizer parameter λ of the inverse loss function $L_i^{inv,r}$ (4.97) can also be seen as a training parameter. If equation (4.99) is compared with the Levenberg-Marquardt optimizing step

$$\Delta\boldsymbol{h}_i^{LM} = -\hat{\eta}(J_i^T J_i + \lambda I)^{-1} J_i^T \boldsymbol{e}, \tag{4.105}$$

$$J_i = \prod_{k=i+1}^{L} D_{s_k} W_k, \tag{4.106}$$

a certain similarity is clear. With difference target propagation, the optimizing step has the following form:

$$\Delta \boldsymbol{h}_i^{DTP} = -\hat{\eta}\left[\prod_{k=i}^{L-1} Q_i D_{s_{k+1}}^{-1}\right]\boldsymbol{e}_L \tag{4.107}$$

$$= \hat{\eta}\left[\prod_{k=i}^{L-1}(W_{i+1}^T(W_{i+1}W_{i+1}^T + \lambda I)^{-1}D_{s_{k+1}}^{-1}\right]\boldsymbol{e}_L. \tag{4.108}$$

Although equations (4.105) and (4.108) are not equal, both strive to make sure that the matrices that need to be inverted are far enough away from singularity, thereby limiting the step size with a sort of thrust region. The regularizer parameter should consequently be treated as a Levenberg-Marquardt-like training parameter. In classical Levenberg-Marquardt optimization, this is done by a heuristic based on the reduction ratio ρ (see section 4.2.3). As this heuristic is not likely to be biologically computable, other more biologically plausible heuristics for λ should be investigated in future research. For the simulations in this thesis, we treat λ as a fixed training hyperparameter.

- Even if the backward weights Q_i are not trained to approximate $W_{i+1}^{\dagger}$, the network can still learn from the backpropagated errors if the weights of g_k change slower in time compared to the forward weights. This was shown in feedback alignment, which is the limit case of a total fixation of the backward weights to random values [6]. Guerguiev et al. [8] confirmed this hypothesis. In thit case, however, theorem 4.9 does not hold anymore and most likely no Gauss-Newton optimization is used anymore.

4.4 Conclusion

This chapter performed a thorough theoretical analysis of target propagation and its variants and created a well-founded mathematical framework around target propagation to better understand its learning mechanisms. This provides a significant contribution to the field of biologically plausible deep learning, as the target propagation method did not yet have a solid mathematical foundation.

The chapter started with investigating the target propagation method with perfect inverses. By doing first order Taylor approximations around the layer activations $\boldsymbol{h}_i$, it was discovered that the learning signal $\boldsymbol{h}_i - \hat{\boldsymbol{h}}_i$ of target propagation has a form very similar to the backpropagated error in the backpropagation method. Based on this Taylor approximation of the learning signal, theorem 4.3 showed that target propagation uses Gauss-Newton optimization to compute its local layer targets $\hat{\boldsymbol{h}}_i$. After computing the local layer targets, target propagation performs a gradient descent step on the local layer loss L_i to update its parameters W_i. Therefore, target propagation can be interpreted as a mixture between first and second order optimization. Based on this insight, we investigated which training hyperparameters should be used for target propagation. We developed a robust Sherman-Morrison update for the forward weights and its inverse, that has similar stabilizing characteristics as the Levenberg Marquardt method. We highlighted that the target propagation method from literature uses a constant step size, whereas the Gauss-Newton optimization methods for deep learning use an adaptive step size. Hence, future research should investigate whether target propagation can be improved by using adaptive step sizes. Furthermore, we showed that the expensive computations of the weight inverses W_i^{-1} can be implemented efficiently by using the Sherman-Morrison formula that operates in $\mathcal{O}(n^2)$. Consequently, target propagation performs a mixture of first and second order optimization with approximately the cost of first-order optimization.

The chapter ended with investigating the target propagation method with approximate inverses. By doing first order Taylor approximations around the layer activations $\boldsymbol{h}_i$, it was discovered that reconstruction errors interfere with the useful learning signal of target propagation. This explains the poor performance of pure target propagation described in the literature. We proposed an improved form of the backward mapping g_i, that is able to learn the inverse of f_{i+1} more accurately. Therefore, we hypothesise that the performance of target propagation with this form of g_i will suffer less from the reconstruction errors and perform better. Furthermore, we showed that difference target propagation cancels out the reconstruction errors by adding a correction term to its targets. Therefore, difference target propagation does not suffer from reconstruction errors, which explains its good performance. Finally, theorem 4.9 shows that difference target propagation uses Gauss-Newton optimization to compute its local layer targets, when our improved form of the backward mapping g_i is used to learn the inverses.

The next chapter will experimentally verify the theoretical assumptions and predictions made in this chapter, and will further investigate the learning behaviour of target propagation in an experimental manner. Chapter 6 will use the results of this chapter to develop a network model more closely linked to the biological properties of neurons.

Chapter 5

An experimental analysis of target propagation

In this chapter, we perform an experimental analysis of the target propagation method in order to experimentally verify the theoretical results of the previous section and to get a more intuitive understanding of the learning dynamics of the target propagation method. This chapter starts with investigating target propagation with exact inverses, after which it analyses target propagation with approximate inverses.

5.1 Experiments on target propagation with exact inverses

This section starts with an easy interpretable toy example to highlight the similarities and differences of target propagation compared to both error back-propagation and Gauss-Newton optimization. Based on the obtained results, we propose modified target propagation, a new variant of target propagation that is more closely linked to Gauss-Newton optimization. Subsequently, various theoretical assumptions, made in the previous chapter, are experimentally verified. These results reveal that the block-diagonal approximation of the GN curvature matrix used for lemma 4.2 is not valid. We propose a randomized target propagation variant that solves this issue. This section ends with a more challenging task to test the performance of the target propagation method and its new variants: a student-teacher non-linear network training in higher dimensions.

5.1.1 Toy example

For this toy example, we use a network with one hidden layer as visualized in figure 5.1. The network has one input neuron h_0, two hidden neurons $\boldsymbol{h}_1$ with two corresponding weights W_1 and two output neurons $\boldsymbol{h}_2$ with four corresponding weights W_2. No biases are used in this toy network. As we want to visualize the learning dynamics of this network in 2D plots, only the weights W_1 of the hidden layer are trained, while W_2 is kept fixed. This *student network* needs to learn to imitate a *teacher network*, which has the same architecture and output weights W_2, but has a different set of hidden weights W_1^*. An L_2 output loss is used to train the network. We first start with a purely linear network setting, after which we repeat the analysis in a more general non-linear setting.

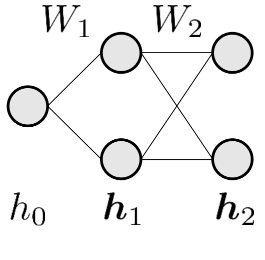


Figure 5.1: **A visualization of the toy example network.**

Linear case

The forward propagation in the linear network is given by the following set of equations:

$$\boldsymbol{h}_1 = W_1 h_0 \tag{5.1}$$

$$\boldsymbol{h}_2 = W_2 \boldsymbol{h}_1. \tag{5.2}$$

The backward propagation in the target propagation method with exact inverses is expressed by:

$$\hat{\boldsymbol{h}}_1 = W_2^{-1}\hat{\boldsymbol{h}}_2 \tag{5.3}$$

We assume that W_2 is always invertible. We first derive analytical expressions for the learning dynamics (update steps) of this linear toy example when using error back-propagation, target propagation and Gauss-Newton optimization, after which we discuss the simulated results. The expected L_2 loss in this student-teacher network setting can be expressed as:

$$\mathbb{E}[L] = \mathbb{E}\Big[\frac{1}{2}(\boldsymbol{h}_2 - \boldsymbol{t})^T(\boldsymbol{h}_2 - \boldsymbol{t})\Big] \tag{5.4}$$

$$= \mathbb{E}\Big[\frac{1}{2}(W_2W_1h_0 - W_2W_1^*h_0)^T(W_2W_1h_0 - W_2W_1^*h_0)\Big] \tag{5.5}$$

$$= \frac{1}{2}\mathbb{E}[h_0^2](W_1 - W_1^*)^T W_2^T W_2(W_1 - W_1^*) \tag{5.6}$$

This is a convex quadratic loss function, as $\mathbb{E}[h_0^2]W_2^TW_2$ is always positive semidefinite. From this loss function, we can derive the expectation of the error backpropagation update step ΔW_1^{BP}, the target propagation update step ΔW_1^{TP} and the Gauss-Newton update step ΔW_1^{GN}:

$$\Delta W_1^{BP} = -\eta\mathbb{E}\Big[\frac{\partial L}{\partial W_1}\Big] = -\eta\mathbb{E}[h_0^2]W_2^TW_2(W_1 - W_1^*) \tag{5.7}$$

$$\Delta W_1^{TP} = \mathbb{E}\Big[(\hat{\boldsymbol{h}}_1 - \boldsymbol{h}_1)h_0\Big] = \mathbb{E}\Big[\big(W_2^{-1}(\boldsymbol{h}_2 - \hat{\eta}\frac{\partial L}{\partial \boldsymbol{h}_2}) - \boldsymbol{h}_1\big)h_0\Big] \tag{5.8}$$

$$= -\hat{\eta}\mathbb{E}[h_0^2](W_1 - W_1^*) \tag{5.9}$$

$$\Delta W_1^{GN} = -\Big(\mathbb{E}\Big[\big(\frac{\partial \boldsymbol{h}_2}{\partial W_1^T}\big)^T\big(\frac{\partial \boldsymbol{h}_2}{\partial W_1^T}\big)\Big]\Big)^{-1}\mathbb{E}\Big[\frac{\partial L}{\partial W_1}\Big] \tag{5.10}$$

$$= -\Big(\mathbb{E}[h_0^2]W_2^TW_2\Big)^{-1}\mathbb{E}[h_0^2]W_2^TW_2(W_1 - W_1^*) = W_1^* - W_1 \tag{5.11}$$

In practice, the expectations $\mathbb{E}$ are estimated by using mini-batches, resulting in a stochastic training scheme. We see that the Gauss-Newton update step reaches always directly the optimum. This could have been expected, as Gauss-Newton optimization is equal to Newton optimization for linear models and Newton optimization reaches the optimum in one step if the loss function is quadratic. In this linear setting with a scalar input h_0, the target propagation update lies in the same direction as the Gauss-Newton update, only the scaling factor $\hat{\eta}\mathbb{E}[h_0^2]$ is different. As we will see later, this will not always be true anymore for the non-linear case or when the network has multiple inputs. The direction of error back-propagation update (gradient descent) is not equal anymore to the Gauss-Newton update, but is instead weighted by $\eta\mathbb{E}[h_0^2]W_2^TW_2$, resulting in the direction of steepest descent. With the help of the singular value decomposition of $W_2 = U\Sigma V^T$, ΔW_1^{BP} becomes more intuitive:

$$\Delta W_1^{BP} = -\eta\mathbb{E}[h_0^2]V\Sigma^2V^T(W_1 - W_1^*) \tag{5.12}$$

The parameter error $(W_1 - W_1^*)$ gets first projected to the principal directions V of W_2, after which they are scaled by the squared singular values σ_i^2 and projected back to the original space. The gradient step thus weights the principal directions of the parameter error $(W_1 - W_1^*)$ with σ_i^2, indicating the importance of that principal direction on the output loss. Hence, it prioritizes first the most important directions to decrease the loss, whereas Gauss-Newton cancels out the influence of W_2 on the output loss and directly follows the direction of the parameter error $(W_1 - W_1^*)$. Target propagation follows the same intuitive interpretation as Gauss-Newton for this toy example. Figure 5.2 shows the experimental results of this toy example, which match nicely with the above analytic derivation. Note that the step size $\hat{\eta}$ of the target propagation method could have been cherry-picked to reach the optimum in one single step, but in realistic problems, this is not possible, so a more modest step size was used.

Non-linear case

Now the analysis of the toy example is repeated for the non-linear case: a Leaky-ReLU element-wise activation function with a negative slope of 0.1 is applied on the hidden layer h_1. Equation (5.1) is now replaced by:

$$\boldsymbol{h}_1 = \text{Leaky-ReLU}(W_1h_0) \triangleq s(W_1h_0), \tag{5.13}$$

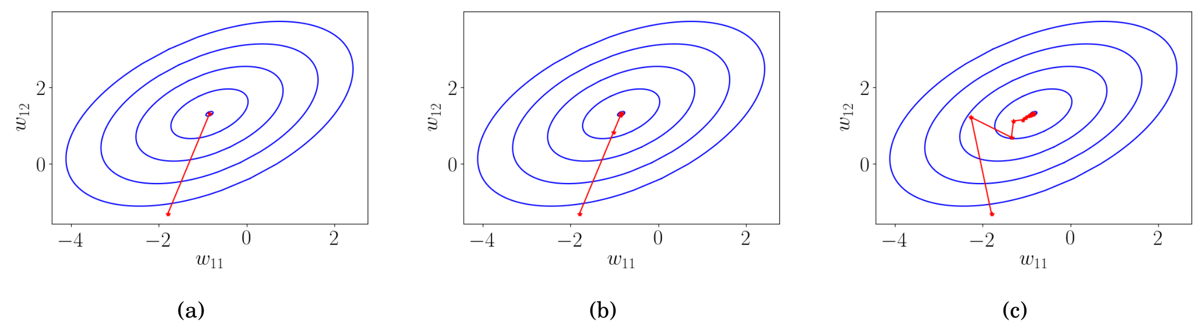


Figure 5.2: **Simulated training results of the linear toy example.** In all subfigures, the elliptic contours of the loss function are visualized in blue. The training iterations are visualized by connected red stars. The inputs h_0 were Gaussian distributed with $\sigma^2 = 1$. (a) The Gauss-Newton training method reaches the optimum in a single step. (b) The target propagation training method has updates in the same direction as the Gauss-Newton method, but needs multiple steps to reach the optimum. The output step size was set as follows: $\hat{\eta} = 0.8$. (c) the error back-propagation method follows the gradient direction, resulting in a typical zig-zag trajectory towards the optimum. The learning was set as follows: $\eta = 0.3$.

with Leaky-ReLU$(x) = x$ if $x \geq 0$ else $0.1x$, and s an abbreviation for clarity of notation. Equation (5.2) and (5.3) remain the same. The expected L_2 loss is now given by:

$$\mathbb{E}[L] = \mathbb{E}\left[\frac{1}{2}(\boldsymbol{h}_2 - \boldsymbol{t})^T(\boldsymbol{h}_2 - \boldsymbol{t})\right] \tag{5.14}$$

$$= \frac{1}{2}\mathbb{E}\left[\big(s(W_1h_0) - s(W_1^*h_0)\big)^T W_2^T W_2 \big(s(W_1h_0) - s(W_1^*h_0)\big)\right] \tag{5.15}$$

The resulting update steps for W_1 can be expressed as:

$$\Delta W_1^{BP} = -\eta\mathbb{E}\left[\frac{\partial L}{\partial W_1}\right] = -\eta\mathbb{E}\left[h_0 D_s W_2^T W_2\big(s(W_1h_0) - s(W_1^*h_0)\big)\right] \tag{5.16}$$

$$\Delta W_1^{TP} = \mathbb{E}\left[D_s(\hat{\boldsymbol{h}}_1 - \boldsymbol{h}_1)h_0\right] = -\hat{\eta}\mathbb{E}\left[h_0 D_s\big(s(W_1h_0) - s(W_1^*h_0)\big)\right] \tag{5.17}$$

$$\Delta W_1^{GN} = -\left(\mathbb{E}\left[\left(\frac{\partial \boldsymbol{h}_2}{\partial W_1^T}\right)^T\left(\frac{\partial \boldsymbol{h}_2}{\partial W_1^T}\right)\right]\right)^{-1}\mathbb{E}\left[\frac{\partial L}{\partial W_1}\right] \tag{5.18}$$

$$= -\left(\mathbb{E}\left[h_0^2 D_s W_2^T W_2 D_s\right]\right)^{-1}\mathbb{E}\left[h_0 D_s W_2^T W_2\big(s(W_1h_0) - s(W_1^*h_0)\big)\right] \tag{5.19}$$

with D_s a diagonal matrix with as entries the derivatives of s, evaluated at W_1h_0. For a mini-batch size of 1, the last expression can be simplified to:

$$\Delta W_1^{GN} = \frac{1}{h_0}D_s^{-1}\big(s(W_1h_0) - s(W_1^*h_0)\big) \tag{5.20}$$

Figure 5.3 shows the experimental results of this non-linear toy example. The target propagation training scheme has update steps in the same direction as the Gauss-Newton training scheme. However, after the first update step, it suffers from the same slow convergence as gradient descent, whereas Gauss-Newton only needs two steps to reach the optimum.

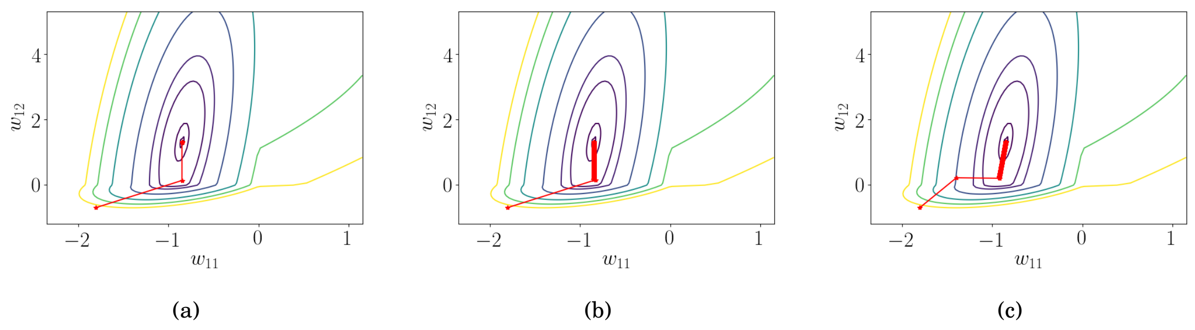


Figure 5.3: **Simulated training results of the nonlinear toy example.** In all subfigures, the contours of the loss function are visualized. The training iterations are visualized by connected red stars. The inputs h_0 were Gaussian distributed with $\sigma^2 = 1$ and a mini-batch size of 1 was used. (a) The Gauss-Newton training method reaches the optimum in two steps. (b) The target propagation training method has updates in the same direction as the Gauss-Newton method, but needs multiple steps to reach the optimum. The training hyper-parameter was set as follows: $\hat{\eta} = 1.0$. (c) the error back-propagation method follows the gradient direction. The training hyper-parameter was set as follows: $\hat{\eta} = 0.5$.

To better understand the experimental results, we further investigate the analytical expressions for the update steps. If W_1 is close to W_1^*, ΔW_1^{TP} and ΔW_1^{GN} can be approximated by a first order Taylor expansion (with a mini-batch size of 1):

$$\Delta W_1^{TP} \approx -\hat{\eta} h_0^2 D_s D_s^* (W_1 - W_1^*) \tag{5.21}$$

$$\Delta W_1^{GN} \approx D_s^{-1} D_s^* (W_1 - W_1^*) \tag{5.22}$$

with D_s^* a diagonal matrix with as entries the derivatives of s, evaluated at $W_1^* h_0$. We see that ΔW_1^{GN} is scaled by $D_s^{-1} D_s^*$, which is close to the identity matrix if W_1 is close to W_1^* and s is smooth in the neighbourhood. ΔW_1^{TP}, on the other hand, is scaled by $h_0^2 D_s D_s^*$, which can change both the direction and magnitude of the update step compared to the parameter error $(W_1^* - W_1)$. As the leaky-ReLU nonlinear activation function results in a piece-wise linear network model, the L_2 loss function is piece-wise quadratic as can be seen in figure 5.3. The starting point of the training is located in the positive region of both leaky-ReLU nonlinearities, resulting in $D_s = I$. For a mini-batch size of 1, this implies that ΔW_1^{TP} lies in the same direction as ΔW_1^{GN} for the first update step (see equations (5.17) and (5.20)). Because $\mathbb{E}\left[h_0^2\right] = 1$ and $\hat{\eta} = 1$, also the magnitudes are roughly equal. After this first update step, the iteration step is located at the negative region of both leaky-ReLU nonlinearities, resulting in $D_s = 0.1I$. From equations (5.21) and (5.22), we now see that $\Delta W_1^{TP} \approx 0.1^2 \hat{\eta} h_0^2 \Delta W_1^{GN} \approx 0.1^2 \Delta W_1^{GN}$, explaining the many small steps target propagation needs to take after the first update step.

Generalization of the toy example

In the previous chapter on the theoretical analysis of target propagation, we showed that target propagation uses an approximation of Gauss-Newton optimization to compute its local layer targets. In this toy example, we highlighted that target propagation not only uses Gauss-Newton for its layer targets, but that it is also closely related to Gauss-Newton optimization for the layer parameters W_i if the other layer parameters $W_{j \neq i}$ remain fixed. As the toy-example used a very specific network architecture, we now briefly analyse a more general network setting in order to reach a more general interpretation of target propagation. Assume a network of L equally sized layers with perfect inverses. Now assume that we want to train the parameters W_i while keeping all the other parameters $W_{j \neq i}$ fixed. Using equation (4.12) and assuming W_i is close to the optimum W_i^*, ΔW_i^{TP} can be approximated as:

$$\Delta W_i^{TP} = \mathbb{E}\Big[-\eta_i D_{s_i}(\boldsymbol{h}_i - \hat{\boldsymbol{h}}_i)\boldsymbol{h}_{i-1}^T\Big] \tag{5.23}$$

$$\approx \mathbb{E}\Big[-\eta_i D_{s_i}\Big(\hat{\eta}\Big[\prod_{k=i}^{L-1} W_{k+1}^{-1} D_{s_{k+1}}^{-1}\Big]\boldsymbol{e}_L\Big)\boldsymbol{h}_{i-1}^T\Big] \tag{5.24}$$

$$\approx \mathbb{E}\Big[-\eta_i \hat{\eta} D_{s_i}\Big[\prod_{k=i}^{L-1} W_{k+1}^{-1} D_{s_{k+1}}^{-1}\Big]\Big[\prod_{k=L-1}^{i} D_{s_{k+1}}^{*} W_{k+1}\Big] D_{s_i}^{*}(W_i - W_i^{*})\boldsymbol{h}_{i-1}\boldsymbol{h}_{i-1}^T\Big] \tag{5.25}$$

$$\approx \mathbb{E}\Big[-\eta_i \hat{\eta} D_{s_i} D_{s_i}^{*}(W_i - W_i^{*})\boldsymbol{h}_{i-1}\boldsymbol{h}_{i-1}^T\Big] \tag{5.26}$$

For the third equation, we used a first order Taylor expansion for $\boldsymbol{e}_L = \boldsymbol{h}_L - \boldsymbol{t}$ around W_i^*, assuming that with W_i^*, the exact output targets are reached. For the fourth equation, we assumed that $D_{s_k} \approx D_{s_k}^*$ for W_i close to W_i^*.

We can do a similar analysis for the Gauss-Newton update step ΔW_i^{GN}. First let us define H_i^T as:

$$H_i^T \triangleq \begin{bmatrix} \boldsymbol{h}_i^T & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & & \vdots \\ \vdots & & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{h}_i^T \end{bmatrix} \tag{5.27}$$

Now take $\text{vec}(W_i)$ the vector consisting of the concatenated rows of W_i. Then it holds that $W_i\boldsymbol{h}_{i-1} = H_{i-1}^T \text{vec}(W_i)$. ΔW_i^{GN} is now given by:

$$\Delta W_i^{GN} = -\mathbb{E}\Big[J_i^T J_i\Big]^{-1}\mathbb{E}\big[J_i^T \boldsymbol{e}_L\big] \tag{5.28}$$

$$J_i \triangleq \frac{\partial \boldsymbol{h}_L}{\partial \text{vec}(W_i)} = \Big[\prod_{k=L-1}^{i} D_{s_{k+1}} W_{k+1}\Big]\frac{\partial \boldsymbol{h}_i}{\partial \text{vec}(W_i)} \tag{5.29}$$

$$= \Big[\prod_{k=L-1}^{i} D_{s_{k+1}}^{*} W_{k+1}\Big] D_{s_i} H_{i-1}^T \tag{5.30}$$

For W_i close to W_i^*, $\boldsymbol{e}_L$ can again be approximated as

$$\boldsymbol{e}_L \approx \Big[\prod_{k=L-1}^{i} D_{s_{k+1}}^{*} W_{k+1}\Big] D_{s_i}^{*}(W_i - W_i^{*})\boldsymbol{h}_{i-1} \tag{5.31}$$

$$= \Big[\prod_{k=L-1}^{i} D_{s_{k+1}}^{*} W_{k+1}\Big] D_{s_i}^{*} H_{i-1}^T\big(\text{vec}(W_i) - \text{vec}(W_i^*)\big) = J_i\big(\text{vec}(W_i) - \text{vec}(W_i^*)\big). \tag{5.32}$$

This leads to

$$\Delta W_i^{GN} \approx -\mathbb{E}\Big[J_i^T J_i\Big]^{-1}\mathbb{E}\Big[J_i^T J_i\Big]\big(\text{vec}(W_i) - \text{vec}(W_i^*)\big) = \text{vec}(W_i^*) - \text{vec}(W_i), \tag{5.33}$$

when W_i is close to the optimum. In the last expression, $\text{vec}(W_i)$ can be transformed again to W_i, as no matrix multiplications are present anymore. When the approximate Gauss-Newton step (5.33) is compared to the approximate target propagation step (5.26), we observe that ΔW_i^{TP} is very similar to ΔW_i^{GN}, apart from a left multiplication with $\eta_i \hat{\eta} D_{s_i} D_{s_i}^*$ and a right multiplication with $\boldsymbol{h}_{i-1}\boldsymbol{h}_{i-1}^T$. In the following, we propose a modified target propagation method, in order to make ΔW_i^{TP} more similar to ΔW_i^{GN} such that it better approximates second-order optimization.

Modified target propagation.

The diagonal matrix D_{s_i} in equation (5.26) is due to the gradient step on the local layer L_2 loss function. If we modify this gradient step towards a more Gauss-Newton inspired update step by replacing D_{s_i} with $D_{s_i}^{-1}$ and if we assume $D_{s_i} \approx D_{s_i}^*$, we get the following approximation of the modified target propagation update step:

$$\Delta W_i^{MTP} \approx -\eta_i \hat{\eta}(W_i - W_i^*)\mathbb{E}\Big[\boldsymbol{h}_{i-1}\boldsymbol{h}_{i-1}^T\Big] \tag{5.34}$$

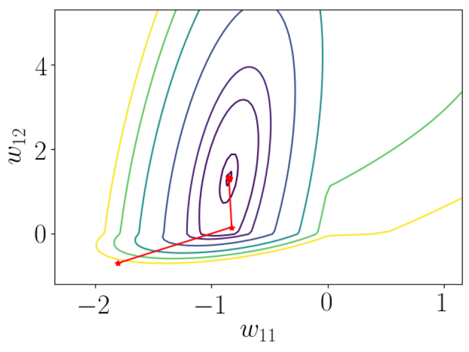


Figure 5.4: **The modified target propagation training scheme applied on the nonlinear toy example.** Only two training steps are needed to reach the minimum.

If we take $\eta_i \hat{\eta} = 1$, the only difference between the approximation of ΔW_i^{MTP} and ΔW_i^{GN} is the right multiplication with $\mathbb{E}\left[\boldsymbol{h}_{i-1}\boldsymbol{h}_{i-1}^T\right]$. In the deep learning community, a common technique for improving the training of neural networks is batch normalization [101]. In this technique, each neuron of the network is independently whitened for each mini-batch. Intuitively speaking, this can be interpreted as preprocessing not only the input layer, but also all hidden layers of the network during each mini-batch. It also introduces a scaling parameter and a bias parameter for each neuron, so that the network can learn how to preprocess its hidden layers, but those parameters are omitted here. If each neuron of $\boldsymbol{h}_i$ is whitened independently, the entries of $\mathbb{E}\left[\boldsymbol{h}_{i-1}\boldsymbol{h}_{i-1}^T\right]$ are given by $\mathbb{E}\left[\boldsymbol{h}_{i-1}\boldsymbol{h}_{i-1}^T\right]_{j,k} = \rho_{j,k}$, with $\rho_{j,k}$ the Pearson's correlation coefficient between the j-th and k-th neuron of $\boldsymbol{h}_{i-1}$. This matrix has diagonal entries equal to 1, but in general it has also non-diagonal entries with values in the interval $[-1,1]$. Hence, in general $\mathbb{E}\left[\boldsymbol{h}_{i-1}\boldsymbol{h}_{i-1}^T\right]$ is not equal to the identity matrix when using batch normalization, but it will be closer to the identity matrix compared to a network without batch normalization, so ΔW_i^{MTP} will also be closer to ΔW_i^{GN}. Therefore, we also include batch normalization into the modified target propagation training scheme.

To summarize, modified target propagation differs from normal target propagation by replacing D_{s_i} with $D_{s_i}^{-1}$ and by using batch normalization for its hidden layers. Algorithm A.1 gives an overview of the modified target propagation method. Figure 5.4 shows the results of applying the modified target propagation training scheme on the nonlinear toy example. As can be seen, the optimum is reached in two steps, similar to the Gauss-Newton optimization in figure 5.3a. During further experiments in section 5.1.3, the modified target propagation scheme will be compared to the normal target propagation scheme to see if it improves the performance.

Conclusion toy example

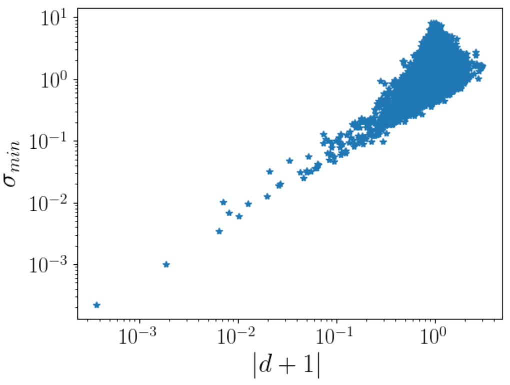
In this toy example, we showed that target propagation not only uses Gauss-Newton optimization to compute its local layer targets, but that it is also closely related to Gauss-Newton optimization for the layer parameters W_i if the other layer parameters $W_{j\neq i}$ remain fixed. Intuitively, when updating the parameters W_i, target propagation cancels out the curvature in the loss function introduced by all higher layers $j > i$ (as W_j and D_{s_j} disappear in equation (5.21)), as also is done by Gauss-Newton optimization. However, due to the gradient step on the local layer loss function, the curvature D_{s_i} of the layer itself still influences the target propagation update, resulting in a gradient-descent-like behaviour of target propagation in the nonlinear toy example shown in figure 5.3b. In order to further eliminate this gradient-descent-like behaviour, we introduced *modified target propagation*, which also counters the curvature D_{s_i} of the layer itself and uses batch-normalization to reduce the influence of the covariance matrix $\mathbb{E}\left[\boldsymbol{h}_{i-1}\boldsymbol{h}_{i-1}^T\right]$ on the parameter update.

5.1.2 Experimental verification of theoretical assumptions

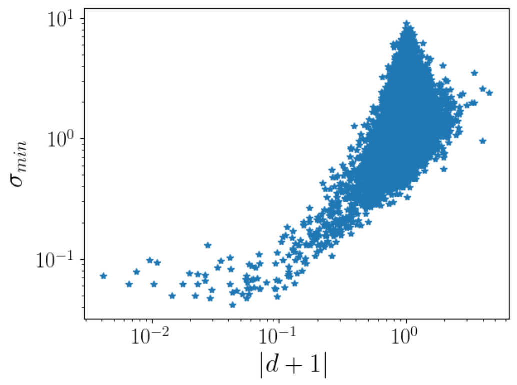
In this section, we verify two important assumptions made in chapter 4: (1) the smallest singular value of the update weight matrix $W_i + \Delta W_i$ has a significant positive correlation with the denominator of the Sherman-Morrison update $1 + d = 1 + \boldsymbol{v}_i^T W_i^{-1} \boldsymbol{u}_i$ and (2) the Gauss-Newton approximation of the Hessian is block-diagonal dominant, so can be approximated by a block-diagonal matrix.

Correlation of σ_{min} with the Sherman-Morrison update

In section 4.2.3 we developed a robust Sherman-Morrison update, based on the assumption that the denominator $1+d = 1+\boldsymbol{v}_i^T W_i^{-1}\boldsymbol{u}_i$ of the Sherman-Morrison update (4.19) correlates well with the smallest singular value σ_{min} of $W_i + \Delta W_i$. Algorithm 4.1 gave an overview of this robust Sherman-Morrison update. Figure 5.5 shows the scatter plot of the absolute value of $1+d$ versus σ_{min}, resulting from a series of random trails where W_i is updated with $\Delta W_i = \boldsymbol{u}_i\boldsymbol{v}_i^T$ and $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ taken as Gaussian random vectors. We see that $|1+d|$ correlates well with σ_{min} of the updated matrix. For samples with $|1+d| < 0.1$ (the samples for which the robust Sherman-Morrison update would threshold the denominator), a Pearson correlation coefficient of $\rho = 0.80$ was measured with σ_{min}. The assumption that the denominator $1+d = 1+\boldsymbol{v}_i^T W_i^{-1}\boldsymbol{u}_i$ of the Sherman-Morrison update (4.19) correlates well with the smallest singular value σ_{min} of $W_i + \Delta W_i$ is thus well justified. As a verification for the robust Sherman-Morisson procedure, figure 5.5b shows the scatter plot of the absolute value of $1+d$ versus σ_{min}, when W_i is updated with the robust Sherman-Morrison update with threshold $\epsilon = 0.1$. We see that σ_{min} of the updated matrix is lower bounded, ensuring that the updated matrix $W_i + \Delta W_i$ is robust invertible.



(a)



(b)

Figure 5.5: **Scatter plots of the denominator $d+1$ of the Sherman-Morrison update versus the smallest singular value σ_{min} of the updated weight matrix $W_i + \Delta W_i$.** 20000 trails were used to produce the scatter plots. During each sample trail, W_i was taken as a Gaussian random robust invertible matrix with $\sigma_{min}(W_i)\cdot\sigma_{max}(W_i) > 0.4$. $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ were taken as Gaussian random vectors. In both figures, σ_{min} represents the smallest singular value of the updated matrix $W_i + \Delta W_i$ and $|d+1| = |1+\boldsymbol{v}_i^T W_i^{-1}\boldsymbol{u}_i|$ the absolute value of the denominator of the Sherman-Morrison update, as defined in equation (4.19). (a) W_i is updated with $\Delta W_i = \boldsymbol{u}_i\boldsymbol{v}_i^T$. (b) W_i is updated with the robust Sherman-Morrison update as summarized in algorithm 4.1, with threshold $\epsilon = 0.1$.

Block-diagonal approximation

In the field of Gauss-Newton optimization for deep learning, it is common to approximate the Gauss-Newton curvature matrix $G = J^T J$ by a block-diagonal matrix $\tilde{G}$ [84, 85, 86], as the authors of [84] show that the GN Hessian matrix is block diagonal dominant for feed-forward neural networks. As this block-diagonal approximation is an important building block of the main theorem of this thesis (theorem 4.3), we verify experimentally whether this approximation holds for the setting of the theorem: a mini-batch size of 1. For the experiment, we use Gaussian random layer Jacobians $J_{\boldsymbol{h}_i} = \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{h}_{i-1}} = D_{s_i} W_i$ and compute the blocks J_i of the total Jacobian J_{tot} as follows:

$$J_i = \text{block}\big(J_{tot}\big)_i = \frac{\partial \boldsymbol{h}_L}{\partial \boldsymbol{h}_i} = \prod_{k=L}^{i+1} J_{\boldsymbol{h}_k} \quad i = 1,\dots,L-1 \tag{5.35}$$

Each block of J_{tot} represents the Jacobian of the output layer $\boldsymbol{h}_L$ towards a hidden layer $\boldsymbol{h}_i$. We then compute two different updates $\Delta\bar{\boldsymbol{h}}$, with $\Delta\bar{\boldsymbol{h}}$ the concatenation of all hidden layer updates $\{\Delta\boldsymbol{h}_i\}_{i=1}^{L-1}$ as defined in theorem 4.3: (1) $\Delta\bar{\boldsymbol{h}}_{full}$, computed with the full Gauss-Newton curvature matrix G and (2) $\Delta\bar{\boldsymbol{h}}_{approx}$, computed with the block diagonal approximation $\tilde{G}$ of the curvature matrix, as defined in lemma

4.2. Figure 5.6 shows the result of the experiment. For two hidden layers and a mini-batch size of 1, figure 5.6a shows that the angle α between $\Delta\bar{\boldsymbol{h}}_{full}$ and $\Delta\bar{\boldsymbol{h}}_{approx}$ can take almost any value within the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$, which implicates that G is not always block-diagonal dominant for a mini-batch size of 1. If the network consist of more hidden layers, figure 5.6b indicates that the situation is even worse: in general, the updates $\Delta\bar{\boldsymbol{h}}_{full}$ and $\Delta\bar{\boldsymbol{h}}_{approx}$ are approximately orthogonal with each other. In the previous papers on Gauss-Newton optimization for deep learning [84, 85, 86], the authors used mini-batches of at least 100 samples to average the curvature matrix G. Figure 5.6c shows that the block-diagonal approximation is valid in this case of large mini-batches.

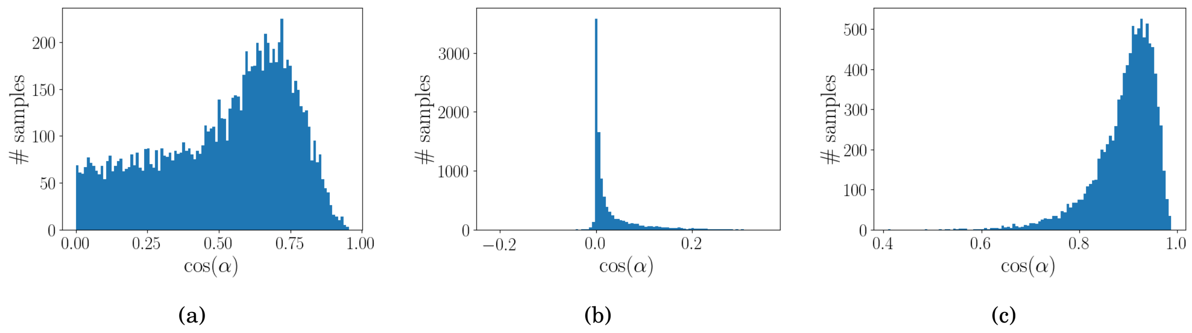


Figure 5.6: **Histogram of the angle α between the layer targets computed with full Gauss-Newton and with the block-diagonal approximation of Gauss-Newton.** For each figure, 10000 random trails were taken. (a) a network with 2 hidden layers and a mini-batch size of 1. (b) a network with 10 hidden layers and a mini-batch size of 1. (c) a network with 10 hidden layers and a mini-batch size of 100.

The results of this experiment can be explained by the phenomenon that the higher the dimensionality of the vector space, the higher the chance that random vectors are approximately orthogonal to each other [102]. For mini-batches greater than one, J_{tot} consists of concatenated Jacobians along the row dimension, thus increasing the row dimension of J_{tot}. Consequently, the curvature matrix $G = J_{tot}^T J_{tot}$ consists of inner products of high dimensional vectors. The diagonal has the squared magnitude of these vectors as entries, and the rest of G has inner products of different vectors as entries. Hence, for large mini-batches, this makes G diagonal dominant. For small mini-batches, however, this argument does not hold anymore and the block-diagonal approximation of G is no longer valid.

Implications of the experimental verifications

From the experimental results, it is clear that the assumptions made for the robust Sherman-Morrison update were correct and that the method works. However, the experimental results on the block-diagonal approximation of the Gauss-Newton curvature matrix showed us that this approximation is not accurate for a mini-batch size of 1. Note that the mini-batch size of the Gauss-Newton algorithm cannot be increased, as target propagation uses directly the (pseudo-)inverses of the forward weight matrices W_i to compute its layer targets, instead of averaging the curvature matrix G over the mini-batch and afterwards solving the linear system, as is done in classical Gauss-Newton. This implicates that the block-diagonal approximation used in theorems 4.3 and 4.9 needs to be adjusted in order to be useful in practice.

Luckily, this issue can be resolved in an intuitive way. If during each training iteration, only one set of layer weights W_i is updated, while all other layer weights $W_{j\neq i}$ are considered fixed, theorems 4.3 and 4.9 still hold. Now, the GN curvature matrix G consists of only one block and is by default equal to its block-diagonal approximation. Intuitively, this makes also more sense. During each training iteration, the output target is now transformed to a local target for one of the layers and those layer parameters are updated. This is in contrast with the original target propagation method, where the output target is transformed to a local layer target for all layers simultaneously and where all the layer parameters are updated at once. In the last case, the output target would be overused by all layers, and each layer updates its parameters assuming that the other parameters stay fixed. In reality, however, all layer parameters change simultaneously.

Consequently, we hypothesize that the target propagation method can be improved by making asynchronous updates for the layer parameters: during each training iteration, one layer is randomly chosen from all layers $\{\boldsymbol{h}_i\}_{i=1}^{L}$ to compute its local layer target and update its parameters. This can be extended to a mini-batch setting by choosing randomly a layer to update for each mini-batch. Note that the mini-batch setting of target propagation differs from the one of Gauss-Newton optimization. In Gauss-Newton optimization the curvature matrix G and the gradient of the loss function are averaged over the mini-batch, after which one single linear system is solved to compute the parameter updates. In target propagation, however, a parameter update is computed for each sample in the mini-batch, and all the parameter updates are averaged to obtain the final update. Algorithm A.2 in appendix A summarizes this *randomized target propagation* method. The same reasoning can also be applied to the modified target propagation method, giving rise to the *randomized modified target propagation method*, as shown in algorithm A.3.

To conclude, in chapter 4, we investigated the mathematical properties of target propagation and which assumptions need to be made for the theorems in order to fit them with the literature on target propagation. In this section, we experimentally investigated those assumptions and proposed modifications and improvements to the target propagation method in order to match the mathematical insights with the experimental results. In the next sections, we will compare the modifications of target propagation with the original target propagation method and with error back-propagation as a control.

5.1.3 Non-linear regression on a student-teacher network

In this section, we investigate the training behaviour of target propagation with exact inverses and its variants on a student-teacher non-linear regression problem. In a student-teacher regression problem, first a random 'teacher network' is initialized to create a training dataset of input-output data pairs by feeding random inputs to the teacher network and recording its outputs. Afterwards, a 'student network' of the same architecture is randomly initialized and it is trained on the previously created dataset in order to mimic the teacher network. In this setting, we investigate 5 different training methods: (1) target propagation with exact inverses, (2) randomized target propagation with exact inverses, (3) randomized modified target propagation with exact inverses, (4) error backpropagation and finally (5) error backpropagation with fixed hidden layer parameters as a control. First, we explain the used methods for this series of experiments, after which we discuss the obtained results.

Methods

We structure the methods in (1) the network setting, (2) the loss functions, (3-8) the five above mentioned training methods, (9) the expression for reconstruction errors and finally (10) the implementation in Python.

Network setting. For all experiments, we use for both the teacher as the student network a network architecture with one input layer, one hidden layer and one output layer, all of equal dimension $n_1 = n_2 = n_3 = 6$. For the hidden layer, a Leaky-ReLU non-linearity with a negative slope of 0.35 was used as the forward non-linearity s_1. A linear layer was used as the output layer (s_2 is the identity function). As with random weight initializations, some weight initializations might start close to the optimum while others start far away from it, we removed this random 'luckiness' by initializing each student network with weight matrices that have a Frobenius distance of 8.0 from the corresponding teacher weight matrices. This Frobenius distance of 8.0 is approximately the mean of the Frobenius distance between two random Gaussian square matrices of dimension 6.

Loss function. For our series of non-linear regression experiments, we chose an L_2 output loss, which is a widely used loss for regression problems. This L_2 loss is expressed as:

$$L(\boldsymbol{h}_L, \boldsymbol{t}_L) = \sum_{b=1}^{B} \|\boldsymbol{h}_L^{(b)} - \boldsymbol{t}_L^{(b)}\|_2^2, \tag{5.36}$$

with B the size of the mini-batch and $\boldsymbol{t}_L$ the output value of the used dataset. For the local layer losses L_i to train the forward weights, we also used a L_2 loss based on the layer activation and target.

Target propagation with exact inverses. In target propagation with exact inverses (TP-EI), the exact inverse of the forward mapping of the output layer is used to propagate the output target to the hidden layer. This results in a linear mapping with the inverse of the output feed-forward weights, as the output non-linearity is the identity function:

$$\hat{\boldsymbol{h}}_1 = W_2^{-1}\hat{\boldsymbol{h}}_2. \tag{5.37}$$

In order to minimize computational cost, W_2^{-1} is updated with the Sherman-Morrison formula (4.19) during each iteration. This Sherman-Morrison formula requires that the forward weight update ΔW_2 is of rank 1, thereby making it necessary to use a mini-batch size of 1 during training. For all methods discussed in this section, a mini-batch size of 1 is used, with 2000 mini-batches per epoch. When testing the target propagation training method with exact inverses, it turned out that in order to stabilize the training, it was absolutely necessary to ensure that the singular values of W_2 remained far away from zero. Therefore, we used two theoretical results of section 4.2.3: (1) for the initialization of W_2, it is ensured that $\sigma_{min} \cdot \sigma_{max} > 0.4$, with σ_i the singular values of W_2, by generating random Gaussian matrices until the condition is satisfied and (2) the forward and backward weights are updated with the robust Sherman-Morrison procedure, explained in algorithm 4.1, in order to prevent W_2 from becoming close to singular. Algorithm A.4 summarizes the target propagation method with exact inverses used in our experiments. Note that for the experiments, $L = 2$ and s_2 is the identity function, while s_1 is the leaky-ReLU nonlinearity. During the initialization of the network, the exact inverses W_i^{-1} are computed for all layers $i = 2,...,L$.

Randomized target propagation with exact inverses. The randomized target propagation training scheme with exact inverses (RTP-EI) uses the same method as the previously explained TP-EI, except that it updates only one randomly chosen pair of layer parameters W_k and W_k^{-1} during each training iteration. Algorithm A.5 provides an overview of the used randomized target propagation method with exact inverses

Randomized modified target propagation with exact inverses. The randomized modified target propagation method (RMTP), proposed in the previous section (algorithm A.3), makes use of batch-normalization. However, for target propagation with exact inverses and its variants, the Sherman-Morrison formula is used to update the inverses of the weight matrices, which requires a mini-batch size of 1. Batch-normalization is not possible for a mini-batch size of 1, thus in the randomized modified target propagation method with exact inverses (RMTP-EI), we do not use batch-normalization. The only remaining difference with the RTP-EI method is that RMTP-EI uses $D_{s_i}^{-1}$ instead of D_{s_i} for the parameter updates. Algorithm A.6 summarizes the RMTP-EI method.

Error backpropagation. Error backpropagation (BP) is the training method of choice in the deep learning community. In our experiments, we use a combination of error backpropagation with plain stochastic gradient descent with mini-batches. For completeness, we provide the pseudo-code for this training method in algorithm A.7. As all previously discussed variants on target propagation with exact inverses use a mini-batch size of 1, we also use a mini-batch size of 1 for this training method in order to remove the influence of the mini-batch sizes when comparing the results of the training methods.

Error backpropagation with fixed hidden layer parameters. In the error backpropagation method with fixed hidden layer parameters (BP-fixed), only the forward weights of the output layer are trained with the error backpropagation method. This training method serves as a benchmark for models that cannot propagate useful learning signals to their hidden layers. If a training method X performs better on average than this error backpropagation method with fixed hidden layer parameters, it indicates that this training method X can propagate useful learning signals towards its hidden layers.

Reconstruction errors. As the methods TP-EI, RTP-EI and RMTP-EI use exact inverses for g_i, we would suspect that no reconstruction errors occur during training. However, due to the limited numerical accuracy of the inverse computations, we notice that reconstruction errors do appear during training with these methods. From section 4.3 on target propagation with approximate inverses, we restate the first order Taylor approximation of the layer targets $\hat{\boldsymbol{h}}_i$:

$$\hat{\boldsymbol{h}}_i = \boldsymbol{h}_i - \hat{\eta}\left[\prod_{k=i}^{L-1} J_{g_k}\right]\boldsymbol{e}_L + \big(g_i(\boldsymbol{h}_{i+1}) - \boldsymbol{h}_i\big) + \sum_{j=i+1}^{L-1}\left[\left(\prod_{k=i}^{j-1} J_{g_k}\right)\big(g_j(\boldsymbol{h}_{j+1}) - \boldsymbol{h}_j\big)\right] + \mathcal{O}(\hat{\eta}^2). \tag{5.38}$$

The second term of the right-hand side represents the useful learning signal $\boldsymbol{e}_i^{TP}$:

$$\boldsymbol{e}_i^{TP} \triangleq -\hat{\eta}\left[\prod_{k=i}^{L-1} J_{g_k}\right]\boldsymbol{e}_L \tag{5.39}$$

The third and fourth term represent the reconstruction errors and backpropagated reconstruction errors, respectively. We assign those two error terms to the symbol $\boldsymbol{e}_i^{rec}$:

$$\boldsymbol{e}_i^{rec} \triangleq \left(g_i(\boldsymbol{h}_{i+1}) - \boldsymbol{h}_i\right) + \sum_{j=i+1}^{L-1}\left[\left(\prod_{k=i}^{j-1} J_{g_k}\right)\left(g_j(\boldsymbol{h}_{j+1}) - \boldsymbol{h}_j\right)\right]. \tag{5.40}$$

The fifth term on the righ-hand side of equation (5.38) represents the Taylor approximation error and is referred to as $\boldsymbol{e}_i^{Taylor}$. Finally, we define the total approximation error $\boldsymbol{e}_i^{approx}$:

$$\boldsymbol{e}_i^{approx} \triangleq \hat{\boldsymbol{h}}_i - \boldsymbol{h}_i - \boldsymbol{e}_i^{TP} = \boldsymbol{e}_i^{rec} + \boldsymbol{e}_i^{Taylor}. \tag{5.41}$$

Note that in our experiments, we use piece-wise linear functions as non-linearities. This implies that $\boldsymbol{e}_i^{Taylor}$ will almost always be equal to zero, except when $\boldsymbol{h}_i$ finds itself on a borderline of the piece-wise linear function. Consequently, $\boldsymbol{e}_i^{approx} = \boldsymbol{e}_i^{rec}$ in most cases. As $\boldsymbol{e}_i^{approx}$ is more straight-forward to compute during training, we use this error to investigate the occurring reconstruction errors in the network.

Implementation. All experiments were done making use of our newly developed PyProp framework. This framework makes use of the PyTorch framework [103] in Python, but replaces its automatic differentiation framework with a new framework that allows for variants of error backpropagation, such as target propagation. The framework is built as flexible as possible, such that future work can continue to use this framework for exploring new alternatives to error backpropagation. Appendix C provides a short documentation on the PyProp framework.

Results and discussion

Figure 5.7 shows the training progress of the student-teacher regression problem trained by the target propagation method with exact inverses and its variants. The results were averaged over 7 random generated network initializations and teacher datasets. During post-processing, we needed to remove 3 outlier runs, because the training of either TP-EI (twice) or RMTP-EI (once) became unstable and the loss exploded to very high values. The fact that the RTP-EI method remained stable during all 10 runs, indicates that the randomization method adds stability to the target propagation method. However, as only 10 runs were examined, no significant conclusions can be drawn and future work (with more time and computing power) should perform an experiment containing more runs to obtain more precise conclusions.

The performance of BP and TP-EI are similar, while RTP-EI is stuck in a local minimum from around epoch 30. This highlights that different optima are reached by TP-EI and RTP-EI, while the methods only slightly differ from each other. Chapter 4 and section 5.1.1 showed that both TP-EI and RTP-EI are related to Gauss-Newton optimization, which is an approximate second-order optimization method. Therefore, we would hope to see a fast initial decay in the output loss during the training process of TP-EI and RTP-EI, which is a typical second order optimization characteristic. However, on figure 5.7 we see that both TP-EI and RTP-EI do not converge faster than the error backpropagation method, which is a first order optimization method. We hypothesize that this is due to the step-sizes used in target propagation and its variants. During each iteration in the Gauss-Newton method, the update step reaches the minimum of a parabola based on the curvature in the current iteration point. Hence, the Gauss-Newton method has implicitly an adaptive step size that always reaches exactly the minimum of the fitted parabola. Furthermore, a linesearch is done in Gauss-Newton optimization for deep learning [85] to find explicitly the optimal step size. Target-propagation-like methods, however, only approximate the direction of the Gauss-Newton update step, not its step size. In our implementation, a linear decaying scheme for the step size was used, which does not take information on the local curvature into account. Hence, the non-adaptive step size must be stable for all iterations, and consequently, it has to be taken very small, similar to first order optimization methods. This explains why both TP-EI and RTP-EI are more similar to first order gradient descent instead of second order Gauss-Newton optimization.

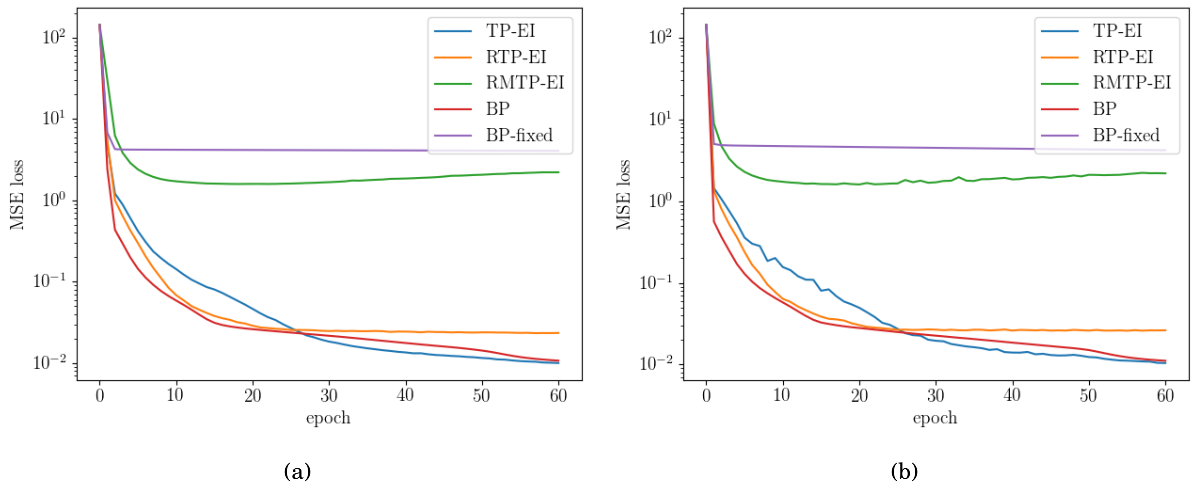
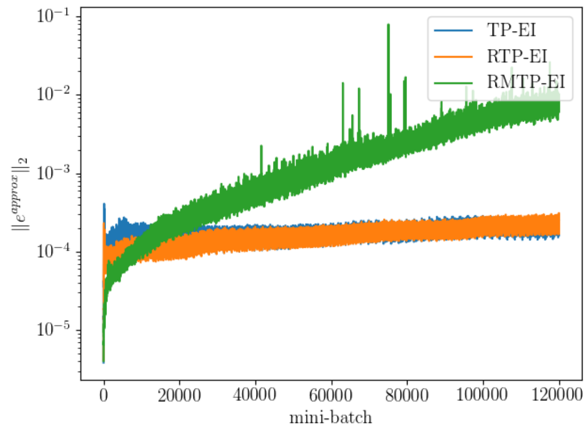


Figure 5.7: **Training progress of the student-teacher regression problem trained by the target propagation method with exact inverses and its variants.** The output mean-squared-error (MSE) loss, also known as L_2 loss, is given in function of the training epoch for all the discussed variants of target propagation. (a) The training loss, evaluated on the training dataset. (b) The test loss, evaluated on the test dataset. The results are averaged over 7 random generated network initializations and teacher datasets. For all experiments, a mini-batch size of 1 was used and a network setting as explained in the methods of section 5.1.3. The output step size was taken equal to $\hat{\eta} = 0.1$ and the threshold parameter for the robust Sherman-Morrison update equal to $\epsilon = 0.5$, for all target propagation-like methods. A limited grid-search was done to find the best learning rates for each method: $\eta_{init}^{TP-EI} = \eta_{init}^{RTP-EI} = 0.01$, $\eta_{init}^{RMTP-EI} = 0.0005$ and $\eta_{init}^{BP} = \eta_{init}^{BP-fixed} = 0.001$. All learning rates decayed linearly towards $\eta_{init}/5$ over the training epochs.

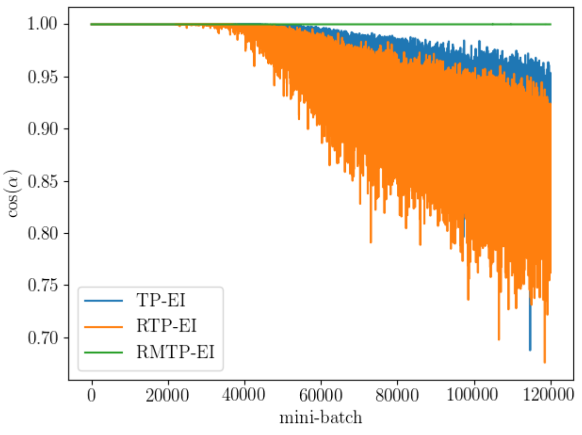
Figure 5.7 shows that the performance of RMTP-EI is only slightly better than BP with fixed hidden layer parameters and much worse compared to BP, TP-EI and RTP-EI. RMTP-EI differs from RTP-EI by using $D_{s_i}^{-1}$ instead of D_{s_i} for computing its parameter updates, in order to resemble more closely the Gauss-Newton optimization method. As the diagonal matrix D_{s_i} has entries smaller or equal to 1 when a leaky-ReLU non-linearity is used, the step sizes of RMTP-EI will always be greater or equal to those of RTP-EI. Consequently, the step size of RMTP-EI has to be taken very small in order to make the training method stable, as no adaptive step sizes are used in the implementation. This explains the very slow training progress of RMTP-EI. From this insight, we can conclude that the modified target propagation method and its variants will only have a chance of good performance when adaptive step sizes are used in order to stabilize the training progress. This is left for future research.

Finally, we investigate the reconstruction errors occurring in the training process of TP-EI, RTP-EI and RMTP-EI. Figure 5.8 shows the magnitude of the approximation error $\boldsymbol{e}_1^{approx}$ defined in equation (5.41) and the angle between the propagated learning signal $\hat{\boldsymbol{h}}_1 - \boldsymbol{h}_1 = \boldsymbol{e}_1^{TP} + \boldsymbol{e}_1^{approx}$ and the useful learning signal $\boldsymbol{e}_1^{TP}$ defined in equation (5.39). This angle indicates whether the direction of the propagated learning signal is still useful for training the network. Figure 5.8a shows that the magnitude of $\boldsymbol{e}_1^{approx}$ stays stable around 10e-4 for TP-EI and RTP-EI. This magnitude stayed the same for different values of the output step size $\hat{\eta}$ (not shown in the figure), indicating that $\boldsymbol{e}_1^{Taylor} \approx 0$ and $\boldsymbol{e}_1^{approx} = \boldsymbol{e}_1^{rec}$. The approximation error is thus almost entirely caused by the limited numerical accuracy of the inverse matrix computation. Figure 5.8b shows that after 40000 mini-batches, the propagated learning signals $\hat{\boldsymbol{h}}_1 - \boldsymbol{h}_1$ of TP-EI and RTP-EI are not perfectly aligned anymore with the useful learning signal $\boldsymbol{e}_1^{TP}$. As $\boldsymbol{e}_1^{approx}$ has the same magnitude throughout the whole training process, this dealigning of the learning signals is due to the decay in magnitude of the useful learning signal $\boldsymbol{e}_1^{TP}$ during training. This decay in magnitude is normal, as the optimum comes closer and the gradients consequently become smaller. As the useful learning signal $\boldsymbol{e}_1^{TP}$ is directly proportional to the output step size $\hat{\eta}$, a logical remedy for the dealigning of the learning signal is to increase $\hat{\eta}$. However, when $\hat{\eta}$ becomes too big, $\boldsymbol{e}_1^{Taylor}$ will grow, because the chance that $\boldsymbol{h}_1$ and $\hat{\boldsymbol{h}}_1$ find themselves on different parts of the piece-wise linear function grows bigger. We thus observe that two opposite forces influence the angle α between the propagated learning signal $\hat{\boldsymbol{h}}_1 - \boldsymbol{h}_1 = \boldsymbol{e}_1^{TP} + \boldsymbol{e}_1^{approx}$

and the useful learning signal $\boldsymbol{e}_1^{TP}$: (1) increasing $\hat{\eta}$ increases $\boldsymbol{e}_1^{TP}$, which lowers α and (2) increasing $\hat{\eta}$ also increases $\boldsymbol{e}_1^{Taylor}$ and consequently increases $\boldsymbol{e}_1^{approx}$, which raises α. $\boldsymbol{e}_1^{rec}$ is independent from $\hat{\eta}$. During experiments, $\hat{\eta} = 0.1$ turned out to be a good equilibrium value for these two forces. Note that in section 4.2.3 we opted to keep $\hat{\eta}$ as small as possible to reduce $\boldsymbol{e}_1^{Taylor}$, but now we see that it needs to be big enough to counter the numerical inaccuracies in TP-EI and RTP-EI. In figure 5.8 we see that for RMTP-EI, $\boldsymbol{e}_1^{approx}$ approx grows while the angle α stays equal to zero. This is due to the weight matrices of the network, which keep on growing during the training. Consequently, both $\hat{\boldsymbol{h}}_1$ and $\boldsymbol{h}_1$ grow during training, thereby increasing both $\boldsymbol{e}_1^{approx}$ and $\hat{\boldsymbol{h}}_1 - \boldsymbol{h}_1$. This indicates that the training process of RMTP-EI was not entirely stable. This instability issue could be solved by further decreasing the learning rate of the training. However, this would make the training progress so slow that even the benchmark of backpropagation with fixed hidden layer parameters is not reached during 60 epochs, so we refrained from further lowering the learning rate.



(a)



(b)

Figure 5.8: **Approximation errors occurring during the training process of the student-teacher regression problem trained by the target propagation method with exact inverses and its variants.** (a) The norm of the approximation error, as defined in equation (5.41). (b) The cosinus of the angle between the propagated learning signal $\hat{\boldsymbol{h}}_1 - \boldsymbol{h}_1 = \boldsymbol{e}_1^{TP} + \boldsymbol{e}_1^{approx}$ and the useful learning signal $\boldsymbol{e}_1^{TP}$ defined in equation (5.39). All results are averaged over 7 random generated network initializations and teacher datasets, and a moving average window of length 8 was used in the time dimension for clarity of the results. The following training hyperparameters were used: $\hat{\eta} = 0.1$, $\epsilon = 0.5$, $\eta_{init}^{TP-EI} = \eta_{init}^{RTP-EI} = 0.01$, $\eta_{init}^{RMTP-EI} = 0.0005$. All learning rates decayed linearly towards $\eta_{init}/5$ over the training epochs.

Conclusion. To conclude this student-teacher toy experiment, we reiterate that the performance of TP-EI and RTP-EI is comparable to the performance of error backpropagation. This indicates that, unfortunately, TP-EI and RTP-EI display no approximate second-order characteristics of fast initial convergence. We hypothesized that this is mainly due to the non-adaptive learning rate used in TP-EI and RTP-EI, while Gauss-Newton optimization uses implicitly an adaptive learning rate. Furthermore, we observed that by using the randomized target propagation training scheme, the training process is more stable, however, more experiments should be done to confirm this hypothesis. The modified target propagation training scheme was not successful in improving the normal target propagation training scheme, because the learning rate needs to be very small to stabilize this method, leading to too slow convergence. Finally, we observed that the output step size $\hat{\eta}$ needs to be chosen carefully in order to balance on the one hand the relative magnitude of the useful learning signal compared to the reconstruction errors, with on the other hand the Taylor approximation errors.

5.2 Experiments on target propagation with approximate inverses

In the previous section, we explored the behaviour of target propagation with exact inverses. Now we investigate the learning dynamics of target propagation with approximate inverses. This setting is biologically more realistic, as the weights (synapses) Q_i of the feedback mapping g_i need to be learned in the brain. First, the relevant theoretical predictions are verified, after which the training behaviour of target propagation and its variants are investigated on a student-teacher non-linear regression problem.

5.2.1 Experimental verification of theoretical predictions

In the following, we verify the theoretical predictions made by theorems 4.6 and 4.7 on learning the pseudo-inverse of the forward weights, and lemma 4.8 on the network architecture constraints for the factorization of the pseudo-inverse of the total Jacobian of the network.

Learning the pseudo-inverse of the feed-forward weights

For the experimental verifications of theorems 4.6 and 4.7, we used a toy network of two layers $\boldsymbol{h}_1$ and $\boldsymbol{h}_2$ of size n_1 and n_2, respectively. Layer $\boldsymbol{h}_2$ has feed-forward weights W and layer $\boldsymbol{h}_1$ has feedback weights Q. The feed-forward and feedback mapping of $\boldsymbol{h}_1$ and $\boldsymbol{h}_2$ are given by:

$$\boldsymbol{h}_2 = f(\boldsymbol{h}_1) = s(W\boldsymbol{h}_1) \tag{5.42}$$

$$g(\boldsymbol{h}_2) = Qs^{-1}(\boldsymbol{h}_2), \tag{5.43}$$

with s an element-wise Leaky-ReLU nonlinearity with a negative slope of 0.35. During the experiments, the forward weights W are kept fixed, while the backward weights Q are learned by doing gradient steps on the inverse loss L^{inv} of equation (4.91) or the regularized inverse loss $L^{inv,r}$ of equation (4.97), which are both restated here for the convenience of the reader:

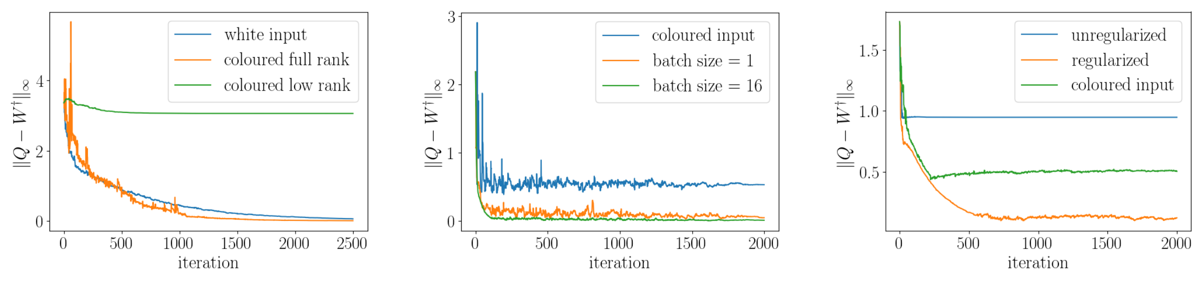
$$L^{inv}\Big(g\big(f(\boldsymbol{h}_1)\big),\boldsymbol{h}_1\Big) = \|g\big(f(\boldsymbol{h}_1)\big) - \boldsymbol{h}_1\|_2^2 \tag{5.44}$$

$$L^{inv,r}\Big(g\big(f(\boldsymbol{h}_1)\big),\boldsymbol{h}_1\Big) = \|g\big(f(\boldsymbol{h}_1)\big) - \boldsymbol{h}_1\|_2^2 + \lambda\|Q\|_F^2. \tag{5.45}$$

$g\big(f(\boldsymbol{h}_1)\big) - \boldsymbol{h}_1$ can be interpreted as the reconstruction error, making L^{inv} equal to the squared norm of the reconstruction error and $L^{inv,r}$ the regularized form of the squared norm of the reconstruction error. In the following, we consider three distinct cases: (1) layers of equal size ($n_1 = n_2$), (2) contracting layers ($n_1 > n_2$) and (3) expanding layers ($n_1 < n_2$).

Layers of equal sizes. For layers of equal sizes, theorem 4.6 predicts that Q converges to W^{-1} if and only if the covariance matrix Γ of $\boldsymbol{h}_1$ is of full rank. For the experiment, we take an equal layer size of $n_1 = n_2 = 5$. Both Q and W are square and start with robust invertible values by design ($\sigma_{min} \cdot \sigma_{max} > 0.4$ for both matrices, with σ_i the singular values). W is kept fixed throughout the experiment and Q is updated with gradient steps on L^{inv}. Figure 5.9a shows that when $\boldsymbol{h}_1$ is white noise ($\boldsymbol{h}_1$ has a covariance matrix $\Gamma = \sigma^2 I$) or full rank coloured noise ($\boldsymbol{h}_1$ has a full rank covariance matrix $\Gamma \neq \sigma^2 I$), Q converges to the pseudo-inverse of W. In this case, $W^\dagger = W^{-1}$, as W is square and of full rank by design. Figure 5.10a shows that in both cases, the reconstruction error $g\big(f(\boldsymbol{h}_1)\big) - \boldsymbol{h}_1$ converges to zero, which was expected, as Q learns to be the perfect inverse of W. When $\boldsymbol{h}_1$ does not have a covariance matrix of full rank, e.g. when it is a linear transformation of a lower dimensional vector, figure 5.9a indicates that the feedback weights Q do not converge to the inverse of the forward weights W, as multiple solutions for Q exist for which $L^{inv} = 0$, due to the singularity of Γ. Figure 5.10a confirms this explanation, as the reconstruction error converges to zero, indicating that $L^{inv} = 0$. Note that in the standard deep learning setting, $\boldsymbol{h}_i$ are nonlinear transformations of the lower layer $\boldsymbol{h}_{i-1}$, so $\boldsymbol{h}_i$ has in general a full-rank covariance matrix. This indicates that only the first two discussed cases appear in the standard deep learning setting, and Q will always converge to W^{-1} in expectation.

Contracting layers. For contracting layers and if W has linear independent rows, theorem 4.6 predicts that Q converges to $W^\dagger$ if and only if $\boldsymbol{h}_1$ has a covariance matrix $\Gamma = \sigma^2 I$, representing white noise. For the experiment, we take $n_1 = 5$ and $n_2 = 3$. Both Q and W are initialized with robust singular values by design ($\sigma_{min} \cdot \sigma_{max} > 0.4$ for both matrices, with σ_i the singular values), ensuring that W has linear independent rows. W is kept fixed throughout the experiment and Q is updated with gradient steps on L^{inv}. Figure 5.9b shows that if $\boldsymbol{h}_1$ is white noise (orange and green curve), Q indeed approaches $W^\dagger$. For a mini-batch size of 1, the estimate of the expected gradient of L^{inv} is too noisy, causing that Q does not completely converge towards $W^\dagger$. If a larger mini-batch size is taken, the estimate of the expected gradient of L^{inv} is more accurate and Q converges to $W^\dagger$. For a coloured activation $\boldsymbol{h}_1$, figure 5.10b shows that Q does not converge to $W^\dagger$, as expected from the theory. Note that for all three cases, the reconstruction



(a) (b) (c)

Figure 5.9: **The approximation error of the feedback weights Q on learning the pseudo-inverse of the forward weights W.** The infinity norm of the approximation error $Q - W^{\dagger}$ is given during the optimizing iterations of the inverse loss $L^{inv,r}$. g and f take the form as defined in equations (5.42) and (5.43). **(a)** The approximation error for equal sized layers of dimension 5. For the blue curve, white noise was used for $\boldsymbol{h}_1$. For the orange and green curve, coloured noise of the form $\boldsymbol{h}_1 = C\boldsymbol{h}_0$ was used, with $\boldsymbol{h}_0$ white noise of dimension 5 and 2, respectively, and C a fixed random Gaussian matrix of size 5×5 and 5×2, respectively. An initial step size of 0.01 was used that decayed linearly to 0.008 over 2500 iterations, no regularization was used ($\lambda = 0$) and a mini-batch size of 1 was taken. **(b)** The approximation error for contracting layers of size 5 and 3 respectively. For the blue curve, coloured noise of the form $\boldsymbol{h}_1 = C\boldsymbol{h}_0$ was used, with $\boldsymbol{h}_0$ white noise of dimension 5 and C a fixed random Gaussian matrix of size 5×5. For the orange and green curve, white noise was used for $\boldsymbol{h}_1$ with a mini-batch size of 1 and 16, respectively. An initial step size of 0.01 was used that decayed linearly to 0.0001 over 2000 iterations and no regularization was used ($\lambda = 0$). **(c)** The approximation error for expanding layers of size 3 and 5 respectively. For the blue and orange curve, white noise was used for $\boldsymbol{h}_1$, and the regularizing parameter of $L^{inv,r}$ was set to $\lambda = 0$ and $\lambda = 0.2$, respectively. For the green curve, coloured noise of the form $\boldsymbol{h}_1 = C\boldsymbol{h}_0$ was used, with $\boldsymbol{h}_0$ white noise of dimension 5 and C a fixed random Gaussian matrix of size 5×5, and the regularizing parameter was set to $\lambda = 0.2$. An initial step size of 0.01 was used that decayed linearly to 0.008 over 2000 iterations and a mini-batch size of 1 was taken.

error $g\big(f(\boldsymbol{h}_1)\big) - \boldsymbol{h}_1$ does not converge to zero, because there exist no perfect inverse mapping from the higher layer $\boldsymbol{h}_2$ towards the lower layer $\boldsymbol{h}_1$, as shown by figure 5.10b. Consequently, a network with contracting layers will always suffer from reconstruction errors that deteriorate the learning signal if a target propagation training method is used. For such networks, it is necessary to use difference target propagation or a similar method that cancels out the reconstruction errors.

Expanding layers. For expanding layers, theorem 4.6 predicts that Q does not converge to $W^{\dagger}$ by minimizing the inverse loss L^{inv}. However, if the regularized inverse loss $L^{inv,r}$ is minimized, theorem 4.7 predicts that the backward weights Q do converge to $W^{\dagger}$ in the limit of $\lambda \to 0$, if and only if $\boldsymbol{h}_1$ has a covariance matrix $\Gamma = \sigma^2 I$, thus representing white noise. For the experiment, we take $n_1 = 3$ and $n_2 = 5$. Both Q and W are initialized with robust singular values by design ($\sigma_{min} \cdot \sigma_{max} > 0.4$ for both matrices, with σ_i the singular values). W is kept fixed throughout the experiment and Q is updated with gradient steps on $L^{inv,r}$. Figure 5.9c shows that if $\lambda = 0$, representing the unregularized loss L^{inv}, the backward weights Q do not converge to $W^{\dagger}$, conform with the theory. The reconstruction error however does converge to zero as visualized in figure 5.10c, because Q has converged to a left inverse of W (which is not unique), making the inverse loss L^{inv} equal to zero. If white noise is taken as $\boldsymbol{h}_1$ and $\lambda > 0$, we see that Q approaches more closely $W^{\dagger}$. However, Q does not completely converge to $W^{\dagger}$, but instead converges towards $W^T(WW^T + \frac{\lambda}{\sigma^2}I)^{-1}$, as predicted by theorem 4.7. Due to the regularizing term in $L^{inv,r}$, the reconstruction error is not completely minimized to zero, as shown in figure 5.10c. If coloured noise is taken as $\boldsymbol{h}_1$, minimizing the regularized inverse loss $L^{inv,r}$ with $\lambda > 0$ does not make Q converge towards $W^{\dagger}$, as shown in figure 5.9c.

Original target propagation. To complete this set of experiments on learning the inverse g of the forward mapping f, we compare our results with the original target propagation method as proposed by the authors of [9, 5]. Our forward and backward mappings are given by equations (5.42) and (5.43).

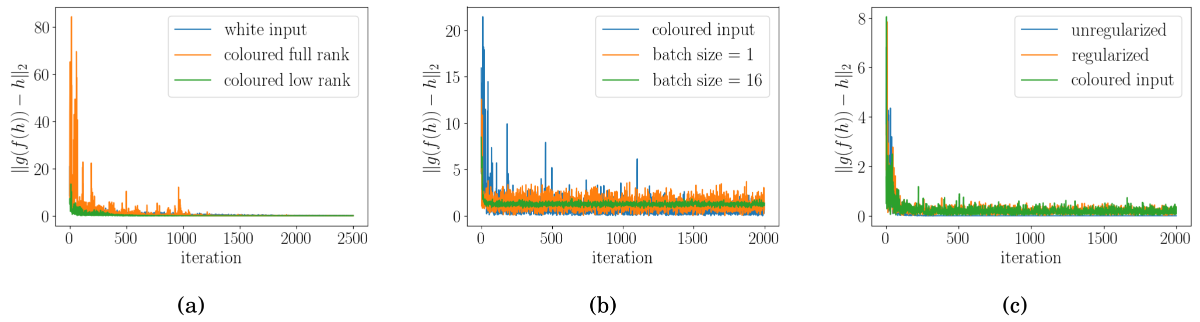


Figure 5.10: **The reconstruction error of the sequential feed-forward mapping f and feedback mapping g.** The 2-norm of the reconstruction error $g\big(f(\boldsymbol{h}_1)\big) - \boldsymbol{h}_1$ is given during the optimizing iterations of the inverse loss $L^{inv,r}$. g and f take the form as defined in equations (5.42) and (5.43). **(a)** The reconstruction error for equal sized layers of dimension 5. For the blue curve, white noise was used for $\boldsymbol{h}_1$. For the orange and green curve, coloured noise of the form $\boldsymbol{h}_1 = C\boldsymbol{h}_0$ was used, with $\boldsymbol{h}_0$ white noise of dimension 5 and 2, respectively, and C a fixed random Gaussian matrix of size 5×5 and 5×2, respectively. An initial step size of 0.01 was used that decayed linearly to 0.008 over 2500 iterations, no regularization was used ($\lambda = 0$) and a mini-batch size of 1 was taken. **(b)** The reconstruction error for contracting layers of size 5 and 3 respectively. For the blue curve, coloured noise of the form $\boldsymbol{h}_1 = C\boldsymbol{h}_0$ was used, with $\boldsymbol{h}_0$ white noise of dimension 5 and C a fixed random Gaussian matrix of size 5×5. For the orange and green curve, white noise was used for $\boldsymbol{h}_1$ with a mini-batch size of 1 and 16, respectively. An initial step size of 0.01 was used that decayed linearly to 0.0001 over 2000 iterations and no regularization was used ($\lambda = 0$). **(c)** The reconstruction error for expanding layers of size 3 and 5 respectively. For the blue and orange curve, white noise was used for $\boldsymbol{h}_1$, and the regularizing parameter of $L^{inv,r}$ was set to $\lambda = 0$ and $\lambda = 0.2$, respectively. For the green curve, coloured noise of the form $\boldsymbol{h}_1 = C\boldsymbol{h}_0$ was used, with $\boldsymbol{h}_0$ white noise of dimension 5 and C a fixed random Gaussian matrix of size 5×5, and the regularizing parameter was set to $\lambda = 0.2$. An initial step size of 0.01 was used that decayed linearly to 0.008 over 2000 iterations and a mini-batch size of 1 was taken.

However, in the original target propagation method they are inspired on auto-encoders and given by [9, 5]:

$$\boldsymbol{h}_2 = f(\boldsymbol{h}_1) = s(W\boldsymbol{h}_1) \tag{5.46}$$

$$g(\boldsymbol{h}_2) = s(Q\boldsymbol{h}_2), \tag{5.47}$$

with s the tangens hyperbolicus non-linearity. However, as mentioned in section 4.3.2, this represents an auto-encoder without a hidden layer, so the universal approximation theorem does not hold anymore and in general, it will not be possible to achieve a reconstruction error $g(f(\boldsymbol{h}_1)) - \boldsymbol{h}_1$ of zero. This theoretical hypothesis is confirmed by experimental results shown in figure 5.11. For all three previous discussed cases (equal, contracting and expanding layers), minimizing the inverse loss L^{inv} with f and g defined by equations (5.46) and (5.47) does not succeed in driving the reconstruction error towards zero. Networks trained with the original target propagation method will thus always suffer from reconstruction errors that deteriorate the learning signal, unless the difference target propagation scheme is used or a similar method that cancels out the reconstruction errors.

Architecture constraints for the factorization of the pseudo-inverse

In the following paragraphs, we investigate the constraints, stated by lemma 4.8, on the network architecture which ensure that the pseudo inverse of the Jacobians $J_i = \frac{\partial \boldsymbol{h}_L}{\partial \boldsymbol{h}_i}$ are factorizable. Lemma 4.8 states that the pseudo-inverses of all Jacobians J_i are factorizable as:

$$J_i^{\dagger} = J_i^{f} \triangleq \prod_{k=i+1}^{L} W_k^{\dagger} D_{s_k}^{\dagger} \tag{5.48}$$

if and only if $n_L = n_{L-1} = ... = n_2$ and $n_2 \leq n_1$, with n_i the dimension of the i-th layer, W_i is of full rank for $i = 3,...,L-1$, W_2 is of full row rank and D_{s_k} are square and of full rank. This lemma poses very strict constraints on the possible architectures of the network, so this experiment aims to investigate whether these constraints can be relaxed, while still ensuring that $J_i^{\dagger} = J_i^{f}$ approximately holds. During

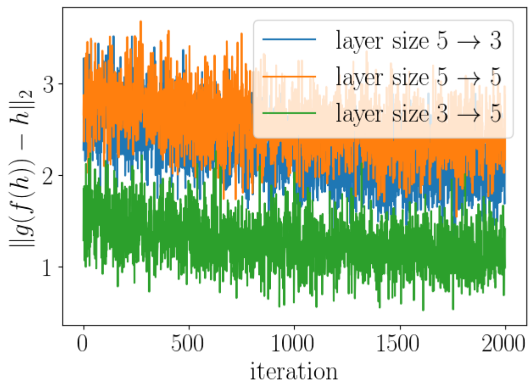


Figure 5.11: **The reconstruction error of the sequential feed-forward mapping f and feedback mapping g for the original target propagation method.** The 2-norm of the reconstruction error $g\big(f(\boldsymbol{h}_1)\big) - \boldsymbol{h}_1$ is given during the optimizing iterations of the inverse loss L^{inv}. g and f take the form as defined in equations (5.46) and (5.47). The blue curve shows the reconstruction error for contracting layers of size $n_1 = 5$ and $n_2 = 3$, the orange curve for equally sized layers of size $n_1 = n_2 = 5$ and the green curve for expanding layers of size $n_1 = 3$ and $n_2 = 5$. An initial step size of 0.002 was used that decayed linearly to 0.0001 over 2000 iterations, a mini-batch size of 8 was taken and no regularization was used ($\lambda = 0$).

the experiments, we ensured that both W_i and D_{s_i} had singular values significantly greater than zero, in order to prevent $J_i^\dagger$ from exploding and to fulfill the conditions on W_i and D_{s_i} stated in lemma 4.8. As J_i^f is used in the target propagation scheme for computing the layer activation updates $\Delta\boldsymbol{h}_i = J_i^f \boldsymbol{e}_L$, we investigated the distance between the update computed with the factorized pseudo-inverse $J_i^f \boldsymbol{e}_L$, as done in target propagation, and the update computed with the real pseudo-inverse $J_i^\dagger \boldsymbol{e}_L$, as done in Gauss-Newton optimization. For all experiments, we did a 1000 repetitions to obtain a reliable histogram of the distance $\|J_i^f \boldsymbol{e}_L - J_i^\dagger \boldsymbol{e}_L\|_2$, we took $\boldsymbol{e}_L$ as a random Gaussian vector and averaged each result over a mini-batch-size of 8. We investigated a network architecture of 2 hidden layers in three different cases: (1) all the hidden layers and the output layer have the same size ($n_1 = n_2 = n_3$) (2) the first hidden layer has a larger size ($n_1 > n_2 = n_3$) and (3) the output layer has a smaller size ($n_1 = n_2 > n_3$).

Equally sized network. This network setting serves as a control, as for equally sized layers J_i is square and its pseudo-inverse is equal to its inverse, which is always factorizable as in equation (5.48). Figure 5.12a shows that for layer dimensions of 5 the distance $\|J_i^f \boldsymbol{e}_L - J_i^\dagger \boldsymbol{e}_L\|_2$ is indeed approximately zero. The deviations from zero are due to the limited accuracy of the inverse matrix computations.

Larger first hidden layer. For this network setting, the size of the first hidden layer is $n_1 = 7$ and the sizes of the second hidden layer and output layer are $n_2 = n_3 = 5$. In this case, lemma 4.8 predicts that it is still allowed to factorize the pseudo-inverse as $J_i^\dagger = J_i^f$. From figure 5.12b, we see that the distance $\|J_i^f \boldsymbol{e}_L - J_i^\dagger \boldsymbol{e}_L\|_2$ is indeed approximately zero and comparable to the distances obtained in the equally sized network of figure 5.12a.

Smaller output layer. A network architecture with a smaller output layer is common in the field of deep learning for classification purposes, as the number of classes is typically lower than the number of features. Hence it would be beneficial if equation (5.48) still approximately holds in this situation, because then theorem (4.9) also approximately holds. In this network setting, the size of the hidden layers are taken equal to $n_1 = n_2 = 5$ and the size of the output layer is taken equal to $n_3 = 3$. Figure 5.12c shows that, unfortunately, the distance $\|J_i^f \boldsymbol{e}_L - J_i^\dagger \boldsymbol{e}_L\|_2$ is not approximately zero anymore and that for many random cases, the distance is actually significantly big. Similar results are obtained when other network architectures are tried out that are not conform with the architecture constraints from lemma 4.8, but are not shown here for brevity reasons. From these results, we conclude that the constraints on the network architecture of lemma 4.8 are necessary for $J_i^\dagger$ to be factorizable and that in general, the constraints cannot be relaxed while keeping $J_i^\dagger = J_i^f$ approximately true.

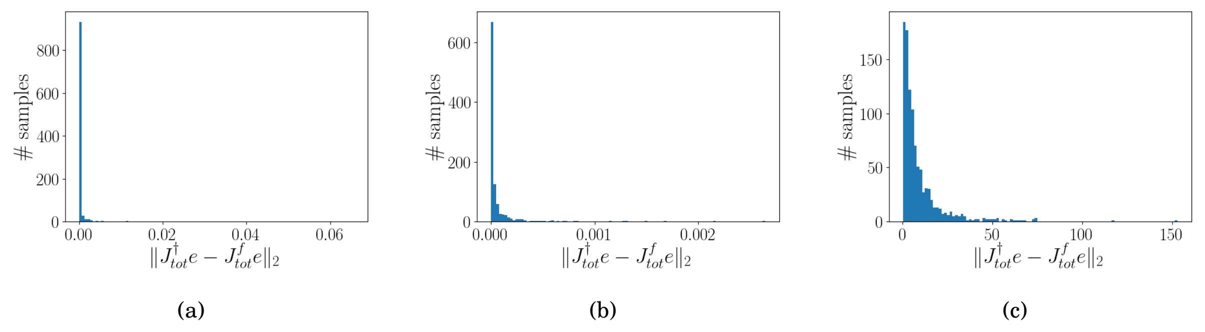


Figure 5.12: **Histograms of the distances $\|J_i^f \boldsymbol{e}_L - J_i^\dagger \boldsymbol{e}_L\|_2$ between updates computed with the factorized pseudo-inverse J^f and the real pseudo-inverse $J^\dagger$ for different network settings.** For each experiment, 1000 repetitions were done to create the histograms and each result is averaged over a mini-batch size of 8 for $\boldsymbol{e}_L$. Random Gaussian vectors were used for $\boldsymbol{e}_L$ during all experiments. W_i and D_{s_i} are taken as random Gaussian matrices with $\sigma_{min} \cdot \sigma_{max} > 0.4$ and D_{s_i} are diagonal. For all three experiments, a network architecture with two hidden layers and one output layer was used. (a) A network architecture of equally sized layers with $n_1 = n_2 = n_3 = 5$. (b) A network architecture of layer sizes $n_1 = 7$ and $n_2 = n_3 = 5$. (c) A network architecture of layer sizes $n_1 = n_2 = 5$, $n_3 = 3$.

Conclusion and further implications of the experimental verifications

During the experiments on learning the pseudo-inverse of the feed-forward weights, we showed that the theoretical predictions from theorem 4.6 and 4.7 match perfectly with the experimental results. Consequently, it is allowed to assume that a network can always reach a state during pre-training in which its backward weights are the pseudo-inverse of the forward weights, when the correct hyper-parameters and training methods are used and the forward weights are kept fixed. This has as an important implication that for experiments, it is allowed to start with an initial state in which the backward weights are the pseudo-inverse of the forward weights. In a more biological setting, we hypothesize that during the development of the brain, the neural networks could be trained such that the backward weights are approximately the pseudo-inverse of the forward weights. In this way, the learning signals can propagate correctly through the network when the brain starts learning from its environment and experiences.

In the experimental verification of the architecture constraints put forward in lemma 4.8, we showed that the architecture constraints cannot be relaxed in a general manner if the pseudo-inverses of the Jacobians $J_i = \frac{\partial \boldsymbol{h}_L}{\partial \boldsymbol{h}_i}$ need to be factorizable as $J_i^\dagger = \prod_{k=i+1}^{L} W_k^\dagger D_{s_k}^\dagger$. So in order for difference target propagation to approximate Gauss-Newton optimization, according to theorem 4.9, the following constraints are put on the network architecture:

$$n_1 > n_2 = \dots = n_i = \dots = n_L \tag{5.49}$$

with n_i the size of the i-th layer. Note that there are no constraints on the size of the input layer n_0. In the literature on difference target propagation [5, 11], the architecture constraints were satisfied for difference target propagation (DTP) and auxiliary-output simplified DTP (AO-SDTP), as both authors used equally sized hidden layers and used a trick for propagating the target from the low-dimensional output layer to the last hidden layer. In difference target propagation [5], error-backpropagation was used for propagating an error signal from the output layer towards the last hidden layer. Hence, a target for the high-dimensional last hidden layer was computed corresponding with equation (4.4) instead of computing it for the low-dimensional output layer, after which that target is propagated through equally sized layers. In AO-SDTP [11], they append the low-dimensional output layer with extra random features computed from the last hidden layer, making the appended output layer of equal size to the hidden layers. In simplified DTP (SDTP) [11], however, no tricks are used for propagating the low dimensional output target through the network. Consequently, theorem 4.9 does not hold for this training method if contracting layers are used, which explains its inferior performance compared to DTP and AO-SDTP. Finally, note that when only one hidden layer is used, the output layer can be of lower dimension, as then $L = 2$, so the constraints expressed in equation (5.49) still hold.

5.2.2 Non-linear regression on a student-teacher network

In this section, we investigate the training behaviour of target propagation with approximate inverses and its variants on a student-teacher non-linear regression problem. We investigate 6 different training methods: (1) target propagation with approximate inverses, as discussed in section 4.3, (2) difference target propagation with approximate inverses, as discussed in section 4.3.3, (3) original target propagation from the literature [9, 5], (4) original difference target propagation from the literature [5], (5) error backpropagation and finally (6) error backpropagation with fixed hidden layer parameters as a control. The first 4 methods are investigated both in a randomized as not randomized setting. We do not discuss modified target propagation with approximate inverses, as even with batch normalization, it suffered from the same instability as RMTP-EI, which was already discussed in the previous section. First, we explain the used methods for this series of experiments, after which we discuss the obtained results.

Methods

We structure the methods in (1) the used network setting, (2) the used loss function, (3-9) the six above mentioned training methods, (10) the used expressions for the approximation errors and learning signals and finally (11) the implementation in Python.

Network setting. In these series of experiments, we use three different network architectures: (1) all equally sized layers of dimension $n_0 = n_1 = n_2 = 6$ with one hidden layer, (2) contracting layer setting with one hidden layer and $n_0 = n_1 = 6$ and $n_2 = 4$ and (3) equally sized layers with 4 hidden layers and $n_0 = n_1 = n_2 = n_3 = n_4 = n_5 = 6$. All three network architectures satisfy the architecture constraints of lemma 4.8, hence, theorem 4.9 will hold for the discussed difference target propagation method. For both network architectures, a Leaky-ReLU non-linearity with a negative slope of 0.35 is used for the forward non-linearity s_1 of the hidden layer, while a linear output layer is used. Following the same reasoning as section 5.1.3, all forward weight matrices of the student networks are initialized with a Frobenius distance of 8.0 from the corresponding teacher weight matrices. All backward weight matrices $\boldsymbol{Q}_i$ are initialized to the (pseudo-)inverse of the forward weight matrices W_{i+1}, following the discussion in section 5.2.1 in which we showed that any well-trained network can reach this state during pre-training. For all training methods discussed below, a mini-batch size of 32 is used, with 60 mini-batches in one epoch.

Loss function. For our series of non-linear regression experiments, we chose an L_2 output loss, which is a widely used loss for regression problems. This L_2 loss is expressed as:

$$L(\boldsymbol{h}_L, \boldsymbol{t}_L) = \sum_{b=1}^{B} \|\boldsymbol{h}_L^{(b)} - \boldsymbol{t}_L^{(b)}\|_2^2, \tag{5.50}$$

with B the size of the mini-batch and $\boldsymbol{t}_L$ the output value of the used dataset. For the local layer losses L_i to train the forward weights, we also used an L_2 loss based on the layer activation and target. For training the backward weights, we used the regularized inverse loss defined in equation (4.97).

Target propagation with approximate inverses. In target propagation with approximate inverses (TP-AI), the backward mapping g_i learns to be the inverse of the forward mapping f_{i+1}. For g_i we take the new form proposed in section 4.3.1:

$$g_i(\hat{\boldsymbol{h}}_{i+1}) = \boldsymbol{Q}_i s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1}). \tag{5.51}$$

Note that this form of g_i differs from the original target propagation papers [9, 5]. The forward parameters W_i are updated conform with equation (4.12) with learning rate η_i. The backward parameters $\boldsymbol{Q}_i$ are updated by a gradient step on the inverse loss $L_i^{inv,r}$ defined in equation (4.97) with white noise as input to the loss function, following the results of theorem 4.7. Algorithm A.10 gives a detailed overview of the used target propagation method with approximate inverses. The randomized version of TP-AI, randomized target propagation with approximate inverses (RTP-AI) uses exactly the same method as TP-AI, with as only difference that during each training iteration, only one layer k gets randomly selected to update its forward parameters W_k. All the backward parameters $\boldsymbol{Q}_i$ are still updated during each training iteration. Algorithm A.11 gives a detailed overview of RTP-AI.

Difference target propagation with approximate inverses. The difference target propagation method with approximate inverses (DTP-AI) uses the same training method for its forward and backward weights as the above explained target propagation method with approximate inverses, only the targets are propagated differently through the network. Restating equation (4.101), the following backward mapping is used in DTP-AI:

$$\hat{\boldsymbol{h}}_i = \boldsymbol{h}_i + g_i(\hat{\boldsymbol{h}}_{i+1}) - g_i(\boldsymbol{h}_{i+1}) \tag{5.52}$$

$$= \boldsymbol{h}_i + Q_i s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1}) - Q_i s_{i+1}^{-1}(\boldsymbol{h}_{i+1}) \tag{5.53}$$

The forward and backward parameters are updated with the same equations as in TP-AI. Algorithm A.12 and A.13 give a detailed overview of the DTP-AI method and its randomized version RDTP-AI, respectively.

Original target propagation. In the original target propagation method (original-TP) [9, 5], a different form for the backward mapping g_i is used, inspired on auto-encoders:

$$\hat{\boldsymbol{h}}_i = g_i(\hat{\boldsymbol{h}}_{i+1}) = s_i(Q_i \hat{\boldsymbol{h}}_{i+1}). \tag{5.54}$$

The non-linearity of the backward mapping is thus the same as for the forward mapping. The forward weights are updated with the same equations as in TP-AI, while a slightly different method for the backward weights is used: a gradient step on the inverse loss function L_i^{inv} is done, with the current sample activation $\boldsymbol{h}_i$ as input to the loss function instead of white noise. Algorithm A.14 and A.15 give a detailed overview of the original-TP method and its randomized version original-RTP, respectively.

Original difference target propagation. The original difference target propagation method (original-DTP) [5] uses the same training method as original-TP to update its forward and backward weights, with as only difference the backward mapping of the targets. Similar to DTP-AI, original-DTP uses the following backward mapping:

$$\hat{\boldsymbol{h}}_i = \boldsymbol{h}_i + g_i(\hat{\boldsymbol{h}}_{i+1}) - g_i(\boldsymbol{h}_{i+1}) \tag{5.55}$$

$$= \boldsymbol{h}_i + s_i(Q_i \hat{\boldsymbol{h}}_{i+1}) - s_i(Q_i \boldsymbol{h}_{i+1}) \tag{5.56}$$

Algorithm A.16 and A.17 give a detailed overview of the original-DTP method and its randomized version original-RDTP, respectively. Note that, in order to stay close to DTP-AI, we do not use error backpropagation to propagate the learning signal to the second-to-last layer, as was done in the original difference target propagation method. Therefore, original-DTP refers to the simplified difference target propagation method [11].

Error backpropagation The same error backpropagation method is used as in section (5.1.3), with as only difference that now a mini-batch size of 32 is used for the experiments instead of a mini-batch size of 1. Algorithm A.7 gives a detailed description of the used backpropagation method.

Error backpropagation with fixed hidden layer parameters. Following the same reasoning as in section 5.1.3, we use the error backpropagation method with fixed hidden layer parameters (BP-fixed) as a bench-mark to see if the other methods succeed in propagating useful learning signals towards their hidden layers. In the BP-fixed method, only the forward weights of the output layer are trained.

Approximation errors and learning signals Similar to section 5.1.3, we define the following learning signals and approximation errors:

$$\boldsymbol{e}_i^{TP} \triangleq -\hat{\eta}\left[\prod_{k=i}^{L-1} J_{g_k}\right]\boldsymbol{e}_L \tag{5.57}$$

$$\boldsymbol{e}_i^{rec} \triangleq \big(g_i(\boldsymbol{h}_{i+1}) - \boldsymbol{h}_i\big) + \sum_{j=i+1}^{L-1}\left[\left(\prod_{k=i}^{j-1} J_{g_k}\right)\big(g_j(\boldsymbol{h}_{j+1}) - \boldsymbol{h}_j\big)\right]. \tag{5.58}$$

$$\boldsymbol{e}_i^{approx} \triangleq \hat{\boldsymbol{h}}_i - \boldsymbol{h}_i - \boldsymbol{e}_i^{TP} = \boldsymbol{e}_i^{rec} + \boldsymbol{e}_i^{Taylor}, \tag{5.59}$$

with $\boldsymbol{e}_i^{Taylor}$ the fifth term in the right-hand side of equation (5.38). Note that for the difference target propagation variants, $\boldsymbol{e}_i^{approx} = \boldsymbol{e}_i^{Taylor}$, as the reconstruction errors $\boldsymbol{e}_i^{rec}$ are canceled out by the altered scheme for propagating the targets through the network (see section 4.3.3). As now approximate inverses are used for g_i, the learning signal $\boldsymbol{e}_i^{TP}$ is not exactly the same anymore to the learning signal $\boldsymbol{e}_i^{GN}$ obtained by Gauss-Newton optimization, defined as:

$$\boldsymbol{e}_i^{GN} \triangleq -\hat{\eta}\left[\prod_{k=i+1}^{L} W_k^{\dagger} D_{s_k}^{-1}\right]\boldsymbol{e}_L \tag{5.60}$$

Therefore, it is interesting to see the evolution of the angle between the propagated learning signal $\hat{\boldsymbol{h}}_i - \boldsymbol{h}_i$ and $\boldsymbol{e}_i^{GN}$ throughout the training process for the different methods. Finally, we also track the angle between $\hat{\boldsymbol{h}}_i - \boldsymbol{h}_i$ and $\boldsymbol{e}_i^{BP}$, the learning signal obtained by error-backpropagation, to check whether $\hat{\boldsymbol{h}}_i - \boldsymbol{h}_i$ lies in a descent direction. $\boldsymbol{e}_i^{BP}$ is defined as:

$$\boldsymbol{e}_i^{BP} \triangleq -\hat{\eta}\left[\prod_{k=i+1}^{L} W_k^{T} D_{s_k}\right]\boldsymbol{e}_L \tag{5.61}$$

Implementation. The experiments are created within the Pyprop framework in Python, similar to section 5.1.3.

Results and discussion

In the following, we present and discuss the results of the experiments on the three different network architectures: (1) 4 hidden layers in an equally sized network, (2) 1 hidden layer in an equally sized network and (3) 1 hidden layer in a contracting network.

4 hidden layers in an equally sized network. We investigate this deeper architecture in order to examine more clearly the difference between the randomized target propagation scheme and the target propagation scheme with full updates (all layers at once). Following the reasoning of section 5.1.2, we expect that the randomized version will work better, as this version is more theoretically well-founded. As the simulations were very heavy, we only compare TP-AI to RTP-AI and use BP and fixed-BP as baselines for comparison. Figure 5.13 shows the obtained simulation results. It is clear that the randomized version RTP-AI has a more stable training process compared to TP-AI which does full layer updates. At the end of the training, RTP-AI outperforms TP-AI. Note that the forward weights of TP-AI are on average updated 5 times more than those of RTP-AI due to the randomization of RTP-AI. This emphasizes that the parameter updates of RTP-AI are more effective compared to those of TP-AI. Note that only a single training run was investigated. Hence, future research should verify if these results can be reproduced.

When the performance of both TP-AI and RTP-AI is compared to the performance of BP and fixed-BP, we see that BP clearly outperforms the target propagation variants, while TP-AI and RTP-AI still perform better than fixed-BP, indicating that the target propagation variants succeed in propagating useful learning signals towards their hidden layers. To improve the performance of the target propagation variants, future experiments could use RDTP-AI and DTP-AI, as these training methods do not suffer from reconstruction errors. It would also be interesting to investigate in future work whether useful learning signals are propagated towards all 4 hidden layers, or whether the same performance could be reached with less hidden layers. However, the main purpose of this experiment was to compare the randomized version of target propagation to the version with full layer updates, in which we succeeded.

1 hidden layer in an equally sized network. In this series of experiments, we investigate in detail the learning behaviour of all the variants of target propagation with approximate inverses. All results are averaged over 15 random runs. First, the performances of the randomized methods, in general, are compared to the performances of the not-randomized methods, after which each randomized target propagation variant is discussed in more detail.

The training performances of the randomized variants of target propagation and the normal variants of target propagation are shown in figure 5.14 and 5.15, respectively. For each training method, a separate small grid search was done to find the best learning rates for the forward weights and the backward weights and to find the ideal regularizer parameter λ for the inverse loss $L_i^{inv,r}$. By comparing the two

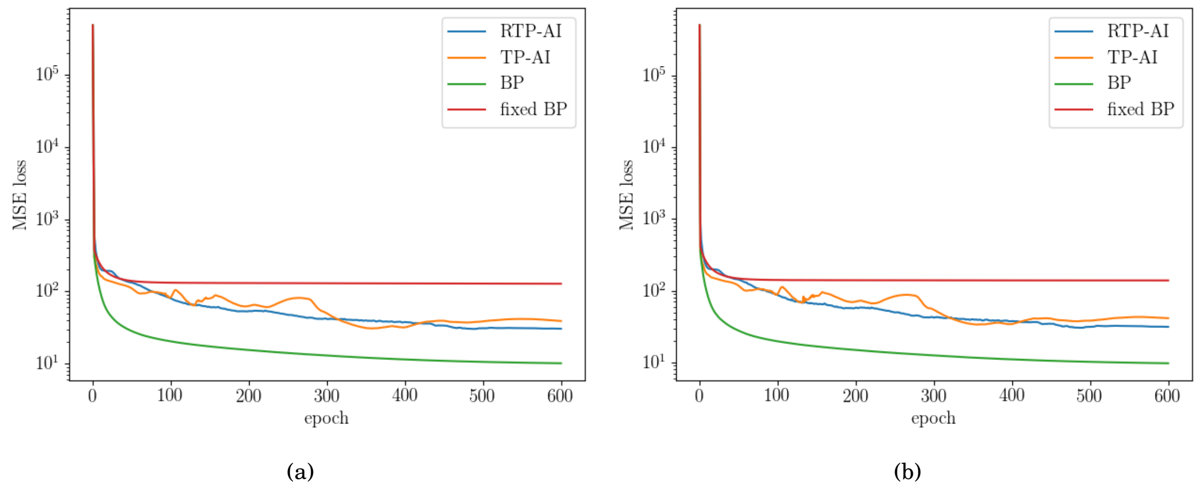


Figure 5.13: **Training progress of the student-teacher regression problem with 4 hidden layers, trained by the target propagation method with approximate inverses.** The output mean-squared-error (MSE) loss, also known as L_2 loss, is given in function of the training epoch. Results for RTP-AI, TP-AI, BP and fixed-BP are shown. (a) The training loss, evaluated on the training dataset. (b) The test loss, evaluated on the test dataset. For all experiments, a mini-batch size of 32 was used with 120 mini-batches in one epoch, and a network setting as explained in the methods of section 5.2.2. The output step size was taken equal to $\hat{\eta} = 0.1$ and the backward weight regularizer equal to $\lambda = 0.0$ for RTP-AI and TP-AI. A limited grid-search was done to find the best learning rates for each method: $\eta_{init}^{TP-AI} = \eta_{init}^{RTP-AI} = 1e-4$ and $\eta_{init}^{BP} = \eta_{init}^{BP-fixed} = 5e-6$. All learning rates decayed linearly towards $\eta_{init}/5$ over the training epochs.

figures, we see that the randomized variants of target propagation perform slightly worse than the not-randomized variants. The forward weights in randomized training versions only get updated half of the time on average, compared to the not-randomized training versions. This is most likely the cause of their lesser performance. This indicates that, while the randomized methods have an advantage in deeper structures due to their added stability, these benefits of added stability are overshadowed in more shallow architectures by the dip in performance due to lesser parameter updates. Apart from their slightly lesser performance, the training behaviour of the randomized variants of target propagation is very similar to the training behaviour of their corresponding not-randomized variant. Hence we only discuss the randomized training variants in the following, in order to minimize the length of the discussion.

The performance of the original target propagation method [5, 9] is comparable to the performance of back-propagation with fixed hidden layer parameters, as shown in figure 5.14. This implies that the original-RTP method does not succeed in sending useful learning signals towards its hidden layer and only the forward weights of the output layer are properly trained. In section 5.2.1 we showed that the original-TP method does not succeed in decreasing its reconstruction errors towards zero. Figure 5.16a shows that indeed, the approximation errors (largely determined by the reconstruction errors) are high compared to the other methods. Figure 5.16b indicates that the approximation errors deteriorate the learning signals in such a way that the learning signal received by the hidden layer is almost completely dealigned with the useful learning signal $\boldsymbol{e}^{TP}$. Furthermore, we see in figure 5.17b that the propagated learning signal $\hat{\boldsymbol{h}}_i - \boldsymbol{h}_i$ does not even point in a descent direction, as it has on average an angle more than 90 degrees away from the gradient direction $\boldsymbol{e}^{BP}$.

In section 4.3.3 we showed that difference target propagation and its variants effectively cancel out the reconstruction errors, such that a clean learning signal $\boldsymbol{e}^{TP}$ gets propagated though the network. In figure 5.14 we see that the performance of original-RDTP is much better than the performance of original-TP, due to the clean learning signal. Figure 5.16 confirms that the approximation errors $\boldsymbol{e}^{approx}$ and consequently the reconstruction errors $\boldsymbol{e}^{rec}$ are very small and that the propagated learning signal $\hat{\boldsymbol{h}}_i - \boldsymbol{h}_i$ is completely aligned with the useful learning signal $\boldsymbol{e}^{TP}$. Figure 5.17b shows that $\hat{\boldsymbol{h}}_i - \boldsymbol{h}_i$ lies in a descent direction. However, from figure 5.17a we see that $\hat{\boldsymbol{h}}_i - \boldsymbol{h}_i$ does not align completely with the Gauss-Newton learning signal $\boldsymbol{e}^{GN}$. In section 5.2.1 we showed that it is not possible for g_i to learn the perfect inverse of f_{i+1}

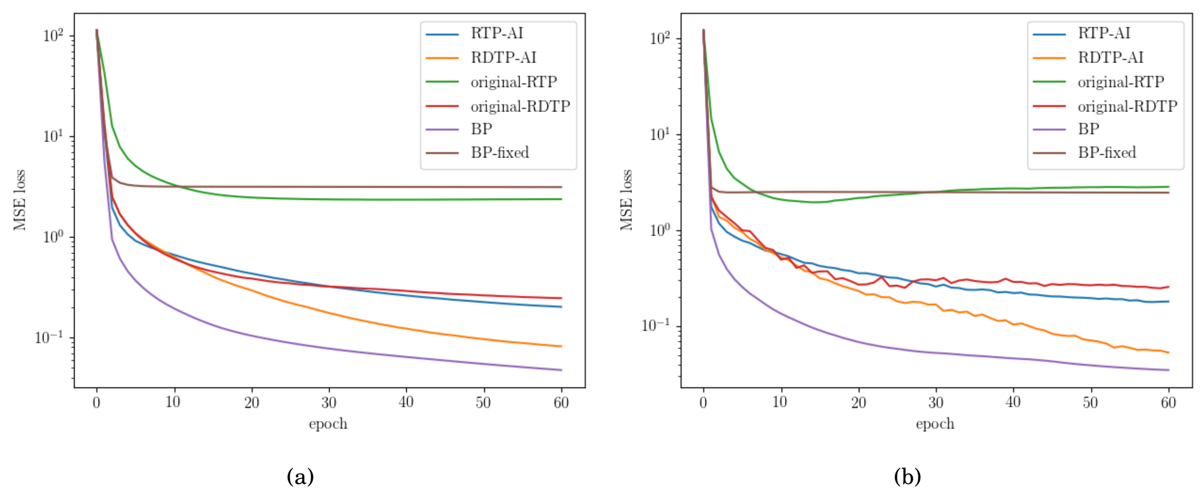
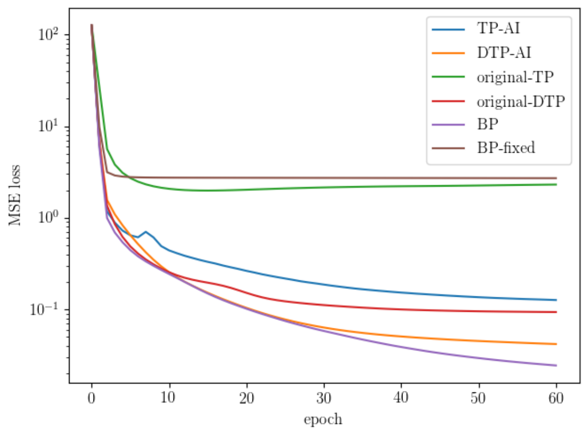


Figure 5.14: **Training progress of the student-teacher regression problem with 1 hidden layer and an equally sized network, trained by the randomized target propagation method with approximate inverses and its variants.** The output mean-squared-error (MSE) loss, also known as L_2 loss, is given in function of the training epoch. Results for RTP-AI, RDTP-AI, original-RTP, original-RDTP, BP and fixed-BP are shown. (a) The training loss, evaluated on the training dataset. (b) The test loss, evaluated on the test dataset. For all experiments, a mini-batch size of 32 was used with 60 mini-batches in one epoch, and a network setting with equally sized layers and one hidden layer, as explained in the methods of section 5.2.2. All results are averaged over 15 random generated network initializations and teacher datasets. The output step size was taken equal to $\hat{\eta} = 0.1$ and the backward weight regularizer equal to $\lambda = 0.0$ for RTP-AI, RDTP-AI, original-RTP and original-RDTP. A limited grid-search was done to find the best learning rates for each method: $\eta_{init}^{RTP-AI} = 0.09$, $\eta_{init}^{RDTP-AI} = \eta_{init}^{original-RDTP} = 0.1$ and $\eta_{init}^{original-RTP} = \eta_{init}^{BP} = \eta_{init}^{BP-fixed} = 0.01$. All learning rates decayed linearly towards $\eta_{init}/5$ over the training epochs.

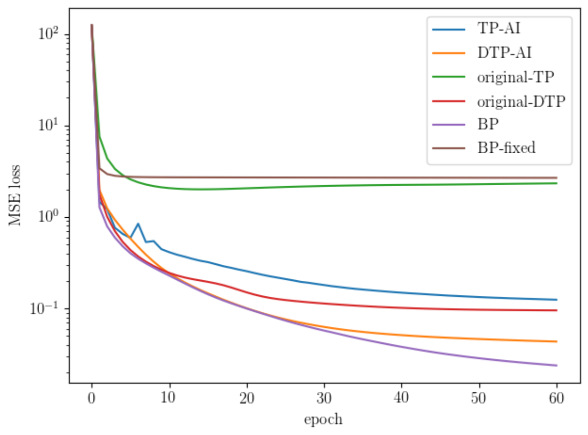
in original-TP and original-DTP, thereby explaining why $\hat{\boldsymbol{h}}_i - \boldsymbol{h}_i$ cannot align with $\boldsymbol{e}^{GN}$, as $\boldsymbol{e}^{GN}$ uses the Jacobian of the perfect inverse of f_{i+1}.

The RTP-AI method, which makes use of our newly proposed form for g_i, outperforms both original-RTP as original-RDTP, as shown in figure 5.14. This increase in performance has two reasons: (1) the new form of g_i allows the network to learn perfect inverses when layers of equal size are used, therefore the reconstruction errors $\boldsymbol{e}^{rec}$ are small and do not interfere much with the useful learning signal $\boldsymbol{e}^{TP}$ and (2) due to the fact that perfect inverses can be learned, the propagated learning signal $\hat{\boldsymbol{h}}_i - \boldsymbol{h}_i$ aligns much better with the Gauss-Newton learning signal $\boldsymbol{e}^{GN}$. Figure 5.16 shows the course of the approximation error $\boldsymbol{e}^{approx}$ during training. As the network is initialized in a setting with perfect inverses, $\boldsymbol{e}^{approx}$ starts very small. However, during early training, the gradients for the forward weights are very big, causing the forward weights to change rapidly. Consequently, the backward weights cannot learn the inverse fast enough, leading to a fast increase in $\boldsymbol{e}^{approx}$. When the movement of the forward weights slows down, the backward weights can catch up again and $\boldsymbol{e}^{approx}$ decreases again, making $\hat{\boldsymbol{h}}_i - \boldsymbol{h}_i$ to align again approximately with $\boldsymbol{e}^{TP}$. Note that this alignment is never complete, because the forward weights keep making small movements, preventing the backward weights to fully converge to the exact inverse. Figure 5.17a confirms that the propagated learning signal of RTP-AI does indeed approximately align with the Gauss-Newton learning signal, explaining its improved performance compared to original-RTP and original-RDTP. The not-randomized version TP-AI performs slightly worse than original-DTP. This is likely due to the faster movement of the forward weights, as during each iteration the forward weights of all layers are updated instead of only one of the layers. This makes it harder for the backward weights to catch up, which gives an advantage to difference target propagation and its variants, because they do not suffer from the increased reconstruction errors. Future work can introduce multiple training iterations for the backward weights after each training iteration for the forward weights in order to reduce this problem.

DTP-AI and RDTP-AI perform best of all variants of target propagation with approximate inverses, which can be seen in figures 5.14 and 5.15. This good performance can be explained by two reasons: (1) DTP-like



(a)



(b)

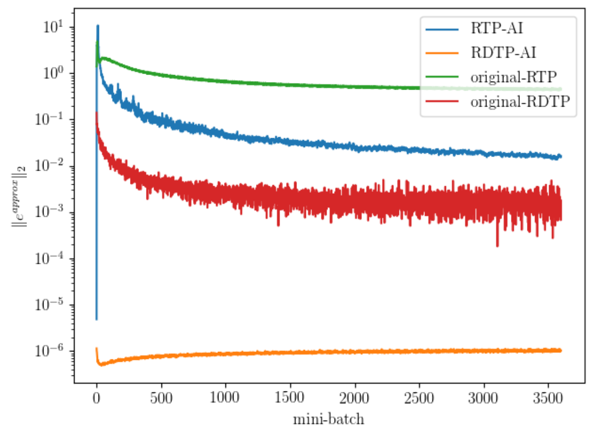
Figure 5.15: **Training progress of the student-teacher regression problem with 1 hidden layer and an equally sized network, trained by the target propagation method with approximate inverses and its variants.** The output mean-squared-error (MSE) loss, also known as L_2 loss, is given in function of the training epoch. Results for TP-AI, DTP-AI, original-TP, original-DTP, BP and fixed-BP are shown. (a) The training loss, evaluated on the training dataset. (b) The test loss, evaluated on the test dataset. For all experiments, a mini-batch size of 32 was used with 60 mini-batches in one epoch, and a network setting with equally sized layers and one hidden layer, as explained in the methods of section 5.2.2. All results are averaged over 15 random generated network initializations and teacher datasets. The output step size was taken equal to $\hat{\eta} = 0.1$ for RTP-AI, RDTP-AI, original-RTP and original-RDTP. A limited grid-search was done to find the best learning rates for each method: $\eta_{init}^{TP-AI} = 0.09$, $\eta_{init}^{DTP-AI} = \eta_{init}^{original-DTP} = 0.1$ and $\eta_{init}^{original-TP} = \eta_{init}^{BP} = \eta_{init}^{BP-fixed} = 0.01$. All learning rates decayed linearly towards $\eta_{init}/5$ over the training epochs.

methods do not suffer from reconstruction errors and (2) due to our new form of g_i, the propagated learning signal is much better aligned with the Gauss-Newton learning signal $\boldsymbol{e}^{GN}$ compared to original-DTP and original-RDTP. Figures 5.16 and 5.17 confirm these hypotheses.

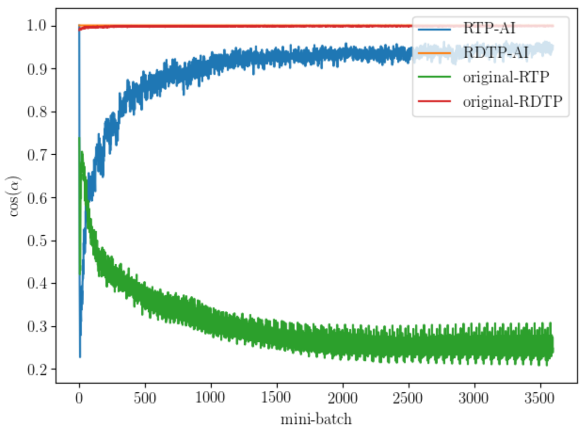
Finally, when we compare all variants of the target propagation method to the error-backpropagation method, we see that BP clearly outperforms all other methods. Chapter 4 showed that DTP-AI and RDTP-AI are a mixture of Gauss-Newton optimization and gradient descent. Hence, the observation that BP outperforms both DTP-AI and RDTP-AI indicates that our implementation of the mixture of Gauss-Newton optimization and gradient descent does not succeed in improving plain gradient descent. Following the discussion of section 5.1.3, we hypothesize that the performance of target propagation and its variants can be improved greatly by introducing adaptive learning rates, such that it can benefit more from its relation to Gauss-Newton optimization, which makes implicitly use of adaptive learning rates.

1 hidden layer in a contracting network. In this last series of experiments, we investigate the learning behaviour of target propagation and its variants in a contracting network setting. Note that the used architecture with one hidden layer still satisfies the architecture constraints of lemma 4.8, hence theorem 4.9 still applies for DTP-AI and RDTP-AI. All results are averaged over 15 random runs. In the following, we only discuss the randomized versions of the training methods, as their training behaviour is very similar to the not-randomized versions.

Section 5.2.1 showed that for contracting layers, there does not exist a perfect inverse of $f_2(\boldsymbol{h}_1)$, hence the reconstruction error $\boldsymbol{e}^{rec}$ cannot be reduced to zero. Consequently, we expect RTP-AI and original-RTP to suffer from this reconstruction error. Figure 5.18a confirms that there is a decrease in performance of for RTP-AI and original-RTP, and figure 5.18b shows that the propagated learning signal $\hat{\boldsymbol{h}}_1 - \boldsymbol{h}_1$ does not align with the useful learning signal $\boldsymbol{e}^{TP}$ due to the reconstruction error. From these results, we conclude that for contracting networks, it is necessary to use methods such as difference target propagation that cancel out the reconstruction errors, as it is not possible to reduce those reconstruction errors towards zero.

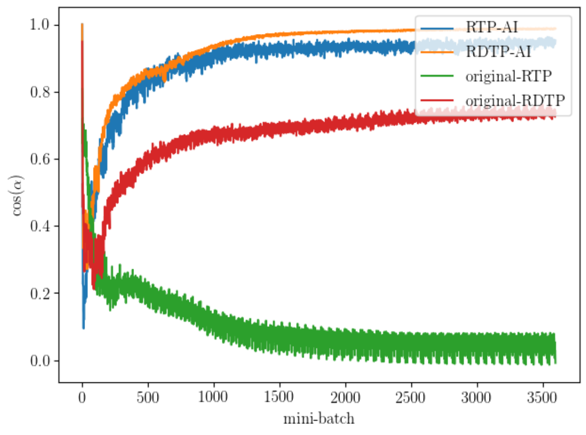


(a)

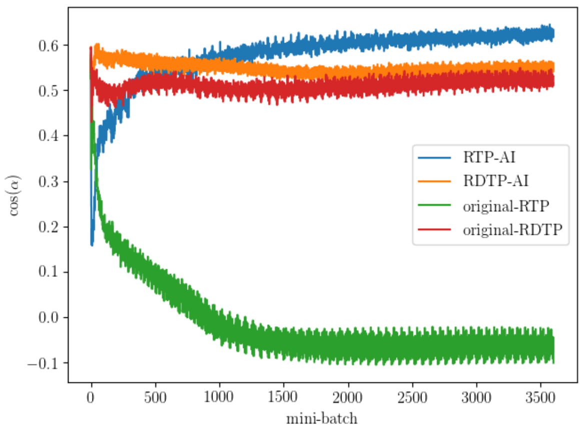


(b)

Figure 5.16: **Approximation errors occurring during the training process of the student-teacher regression problem trained by the randomized target propagation method with approximate inverses and its variants.** (a) The norm of the approximation error, as defined in equation (5.59). (b) The cosinus of the angle between the propagated learning signal $\hat{\boldsymbol{h}}_1 - \boldsymbol{h}_1 = \boldsymbol{e}_1^{TP} + \boldsymbol{e}_1^{approx}$ and the useful learning signal $\boldsymbol{e}_1^{TP}$ defined in equation (5.57). A network with equally sized layers and one hidden layer was used, as defined in the methods of section 5.2.2. All results are averaged over 15 random generated network initializations and teacher datasets. Mini-batches of size 32 were used. The same hyper parameters as in figure 5.14 were used.



(a)



(b)

Figure 5.17: **The angles between the propagated learning signal and the GN and BP learning signals during the training process of the student-teacher regression problem trained by the randomized target propagation method with approximate inverses and its variants.** (a) The cosinus of the angle between the propagated learning signal $\hat{\boldsymbol{h}}_1 - \boldsymbol{h}_1$ and the GN learning signal $\boldsymbol{e}_1^{GN}$ as defined in equation (5.60). (b) The cosinus of the angle between the propagated learning signal $\hat{\boldsymbol{h}}_1 - \boldsymbol{h}_1$ and the GN learning signal $\boldsymbol{e}_1^{GN}$ as defined in equation (5.60). A network with equally sized layers and one hidden layer was used, as defined in the methods of section 5.2.2. All results are averaged over 15 random generated network initializations and teacher datasets. Mini-batches of size 32 were used. The same hyper parameters as in figure 5.14 were used.

More surprisingly, we see in figure 5.18a that also the performance of RDTP-AI has significantly decreased, and that it now has a performance comparable with original-DTP. Figure 5.18b shows that the propagated learning signal is still perfectly aligned with $\boldsymbol{e}^{TP}$ and figure 5.19a shows that it is also perfectly aligned with the Gauss-Newton learning signal $\boldsymbol{e}^{GN}$. Hence, the decrease in performance is not due to approximation errors or other inaccuracies during the training process, but needs to be explained by the theory on target propagation. We do not have a conclusive answer for this decrease in performance, but we would like to propose a possible theory that is not yet confirmed by experimental evidence. As the output layer is of lower dimension than the hidden layer, the Jacobian $J_1 = \frac{\partial \boldsymbol{h}_2}{\boldsymbol{h}_1}$ has more columns than rows. Hence, the Gauss-Newton curvature matrix $G = J_1^T J_1$ is not of full rank and multiple solutions to the Gauss-Newton system $G\Delta \boldsymbol{h}_1 = J_1^T \boldsymbol{e}_L$ are possible. Target propagation solves this problem by taking the solution $\Delta \boldsymbol{h}_1$ with the lowest norm by using the pseudoinverse of J_1 ($\Delta \boldsymbol{h}_1 = J_1^\dagger \boldsymbol{e}_L$). Maybe there exist other and better choices for $\Delta \boldsymbol{h}_1$ that would increase the performance of RDTP-AI again compared to original-RDTP. However, in most settings for Gauss-Newton optimization, the expectation of the curvature matrix G over different samples is taken, which makes sure that G is almost always of full rank. So maybe the root of the problem is the singularity of G, indicating that this problem will not be solved by taking another solution for $\Delta \boldsymbol{h}_1$. If this is the case, the problem is inherent to the target propagation method and more drastic changes need to be made to the target propagation method to increase its performance. We encourage future research to investigate this issue in more detail.

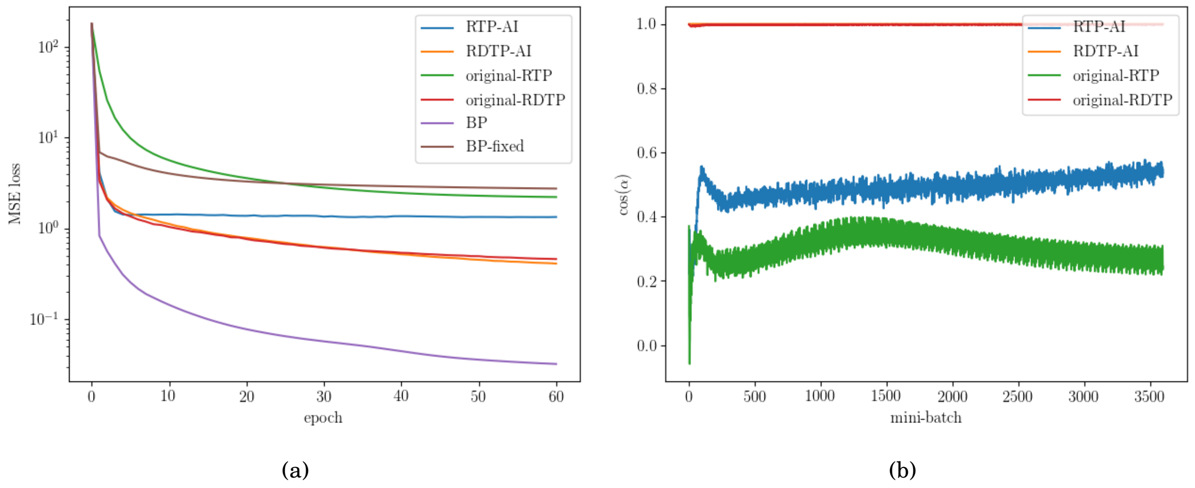
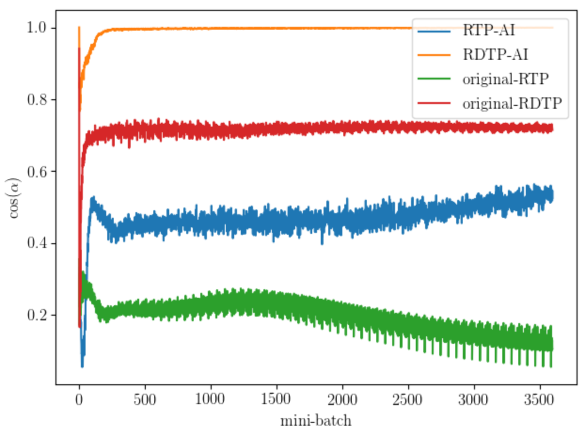
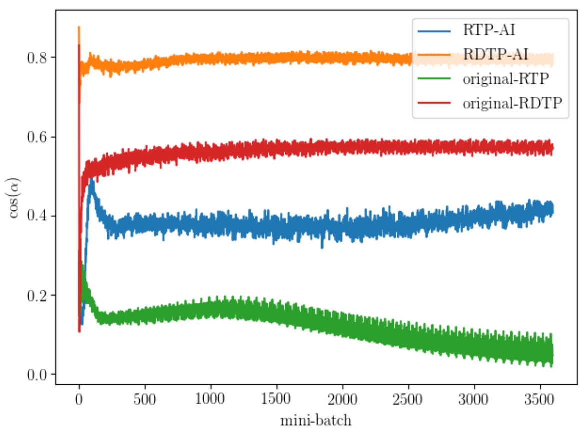


Figure 5.18: **The test loss and approximation errors during the training process of the student-teacher regression problem with contracting layers, trained by the randomized target propagation method with approximate inverses and its variants.** (a) The MSE test loss, evaluated on the training dataset. (b) The cosinus of the angle between the propagated learning signal $\hat{\boldsymbol{h}}_1 - \boldsymbol{h}_1 = \boldsymbol{e}_1^{TP} + \boldsymbol{e}_1^{approx}$ and the useful learning signal $\boldsymbol{e}_1^{TP}$ defined in equation (5.57). The results are averaged over 15 random generated network initializations and teacher datasets. For all experiments, a mini-batch size of 32 was used with 60 mini-batches in one epoch. A network setting with contracting layers as explained in the methods of section 5.1.3 was used. The output step size was taken equal to $\hat{\eta} = 0.1$ and the backward weight regularizer equal to $\lambda = 0.0$. A limited grid-search was done to find the best learning rates for each method: $\eta_{init}^{TP-AI} = 0.09$, $\eta_{init}^{DTP-AI} = \eta_{init}^{original-DTP} = 0.1$, $\eta_{init}^{original-TP} = 0.005$ and $\eta_{init}^{BP} = \eta_{init}^{BP-fixed} = 0.05$. All learning rates decayed linearly towards $\eta_{init}/5$ over the training epochs.

Conclusion To conclude this series of experiments on the student-teacher non-linear regression problem, we reiterate our most important findings. By investigating a deep architecture of 4 hidden layers, we indicated that the randomized version of target propagation provides extra stability to the training process, confirming our theoretical predictions. However, the benefits from the randomized training scheme only surface in deep architectures. In more shallow architectures, the training process is already stable for the not-randomized versions, and the randomized version slows down the training progress because only one layer gets updated each training iteration. During the experiments of an equally sized network with one hidden layer, we showed that our new variants of target propagation TP-AI, RTP-AI, DTP-AI and RDTP-AI outperformed their existing counterparts in the literature (original-TP and original-DTP), because our form of g_i allows the network to find the perfect inverses of f_{i+1}, which is not possible in original-TP and original-DTP. In a contracting network setting, we showed that there exists no perfect inverse of f_{i+1}, making it necessary to use methods that get rid of the reconstruction errors, such as DTP-AI and original-DTP and their randomized versions.



(a)



(b)

Figure 5.19: **The angles between the propagated learning signal and the GN and BP learning signals during the training process of the student-teacher regression problem with contracting layers, trained by the randomized target propagation method with approximate inverses and its variants.** (a) The cosinus of the angle between the propagated learning signal $\hat{\boldsymbol{h}}_1 - \boldsymbol{h}_1$ and the GN learning signal $\boldsymbol{e}_1^{GN}$ as defined in equation (5.60). (b) The cosinus of the angle between the propagated learning signal $\hat{\boldsymbol{h}}_1 - \boldsymbol{h}_1$ and the GN learning signal $\boldsymbol{e}_1^{GN}$ as defined in equation (5.60). A network with contracting layers was used, as defined in the methods of section 5.2.2. All results are averaged over 15 random generated network initializations and teacher datasets. Mini-batches of size 32 were used. The same hyper parameters as in figure 5.18 were used.

5.3 Conclusion

This chapter investigated experimentally the learning dynamics of target propagation and its variants and verified the theoretical assumptions and predictions from chapter 4.

The chapter started by investigating an easy interpretable toy example. In this, we showed that target propagation not only uses Gauss-Newton optimization to compute its local layer targets, but that it is also closely related to Gauss-Newton optimization for the layer parameters W_i if the other layer parameters $W_{j\neq i}$ stay fixed. However, due to the gradient step on the local loss, target propagation still has gradient-descent-like behaviour. In order to make the behaviour of target propagation more close to second-order optimization, we introduced *modified target propagation*. This new method uses batch-normalization to reduce the influence of the covariance of the layer activations $\boldsymbol{h}_i$ on the current parameter update W_i and counters the part of the curvature of the loss function that is due to the nonlinearity of the current layer. Unfortunately, the modified target propagation training scheme was not successful in improving the normal target propagation training scheme, because the learning rate needs to be very small to stabilize this method, leading to slow convergence.

Further, we investigated the theoretical assumptions made in chapter 4 on target propagation with exact inverses. This revealed that the block-diagonal approximation of the GN curvature matrix is not accurate in the setting of target propagation. Therefore, we introduced *randomized target propagation*, in which only one randomly chosen weight matrix W_i is updated during each training iteration. This randomized version of target propagation uses Gauss-Newton optimization with the correct curvature matrix to compute its local layer targets. By comparing the performance of normal target propagation and randomized target propagation on a student-teacher non-linear regression problem in section 5.1.3 and 5.2.2, it became clear that the randomized version of target propagation stabilizes the training process. The stabilizing effect of randomized target propagation is most clear in deeper architectures, where it outperformed normal target propagation. In architectures with only 1 hidden layer, the training process of normal target propagation is already stable, and the randomized version lowers the performance of target propagation due to its fewer parameter updates per training iteration.

During the experiments on the student-teacher networks for both target propagation with exact inverses and approximate inverses, we showed that the performance of target propagation and its variants is comparable to the performance of error backpropagation. This indicates that, unfortunately, target propagation displays no approximate second-order characteristics of fast initial convergence. We hypothesized that this is mainly due to the non-adaptive learning rate used in target propagation, while Gauss-Newton optimization in deep learning [85] uses implicitly an adaptive learning rate and explicitly a line-search during each training iteration. Therefore, we encourage future research to look into adaptive learning rates for target propagation, such that its approximate second-order characteristics can surface.

Lastly, we compared the performance of all the discussed variants of target propagation. We showed that the poor performance of the original target propagation method [9] can be explained by the large occurring reconstruction errors in this method. The experiments confirmed that difference target propagation and its variants effectively cancel out the reconstruction errors, thereby explaining their good performance. The results indicated that our new form of g_i improved the performance of both target propagation and difference target propagation significantly, because this new form can approximate the inverse of f_{i+1} to arbitrary precision if layers of equal size are used. When the network architecture has contracting layers, the experiments showed that the reconstruction errors never disappear, making it necessary to use difference target propagation or other methods that cancel out the reconstruction errors. Finally, we observed that the output step size $\hat{\eta}$ needs to be chosen carefully in order to balance on the one hand the relative magnitude of the useful learning signal compared to the reconstruction errors, with on the other hand the Taylor approximation errors.

Chapter 6

Biological implementations of target propagation

After a thorough theoretical and experimental analysis of target propagation and difference target propagation, we now circle back to the beginning of this thesis, where we started with a description of the biological properties of neurons. This chapter proposes two biologically plausible implementations of target propagation to solve the credit assignment problem, building further upon the models for deep learning with segregated dendrites as discussed in section 3.3. The first model is based on a mixture of feed-forward and feedback signals inside the somatic compartment of the neuron. The second model uses multiplexing to separate the feed-forward and feedback signals inside a single neuron and is briefly explained in the outlook section at the end of this chapter. Both models had to make significant modifications to the pure mathematical target propagation method, emphasizing that even the 'biologically plausible methods' for credit assignment from the literature, are not straight forward to implement in a biologically realistic setting. Both models are only discussed theoretically, due to the limited timespan of this thesis. Future work should experimentally validate the performance and learning dynamics of these biologically plausible networks.

6.1 Introduction

Pure target propagation [9] and difference target propagation [5] use a discrete, two-step updating scheme to calculate neural activations in the network. From a machine learning point of view, this is faster to simulate, but from the biological point of view, it is not realistic. This chapter introduces a form of target propagation learning, implemented in a continuous-time nonlinear dynamical network with abstract pyramidal neurons. The network is inspired by the work of João Sacramento [7], who also provided the idea to investigate this dynamical target propagation network. We will first discuss the biological setting and the network model, after which a theoretical analysis is done for the case with exact inverses and approximate inverses. The important contribution of this new network is that it can operate in continuous time without the need for separate phases and that it generates target-propagation-like dynamics with very simple model equations. We name our network model *dynamical target propagation with segregated dendrites*.

6.2 Biological setting

The dynamical target propagation with segregated dendrites model is based on the segregated compartment model of a pyramidal neuron. The human cerebral cortex consists of $70-85\%$ out of pyramidal neurons [17], motivating the choice for pyramidal neurons in this biological network model. We only summarize the important properties of pyramidal neurons, as they were already discussed in section 2.1.1. Figure 6.1b and 6.1c schematize the segregated compartment model of the pyramidal neuron. The feed-forward connections are integrated into the basal dendritic compartment B, which has its own separate voltage level. The feedback connections are integrated into the apical dendritic compartment A, which also has its own voltage level. In both dendritic compartments, the voltage level gives rise to dendritic spikes that propagate to the somatic compartment S of the neuron [104, 105]. In the somatic compartment, those dendritic spikes influence the voltage level of the cell body and based on that voltage level, an output spike train is sent along the neuron's axon. This model uses a rate-based network: the output spike train of neuron j in layer i is represented by a single scalar $h_i^{(j)}$ indicating the average firing rate. All the output firing rates of a layer i are combined in the vector $\boldsymbol{h}_i$.

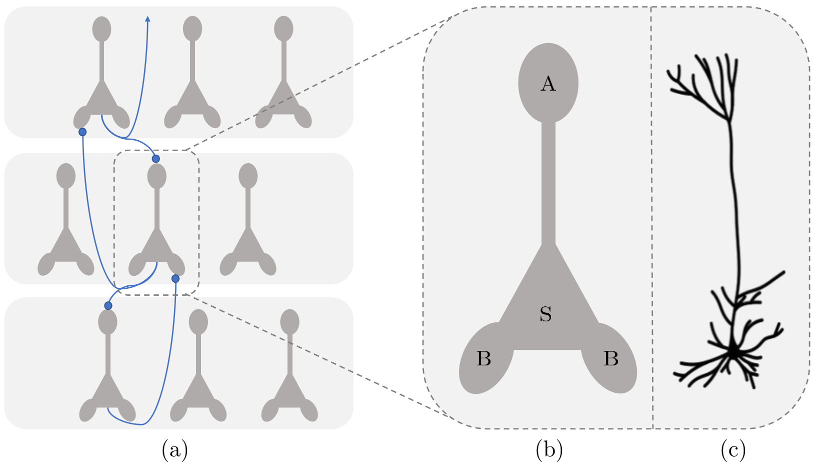


Figure 6.1: **Biological network implementation of pyramidal neurons with segregated dendritic compartments** (figure inspired on Sacramento et al. [7]). (a) A network consisting of multiple layers of pyramidal neurons with segregated dendritic compartments: (1) the feed-forward connections originate from the axons of pyramidal cells of layer i and end at the basal dendritic compartment of pyramidal cells in layer $i+1$, (2) the feedback connections originate from the axons of pyramidal cells of layer $i+1$ and end at the apical dendritic compartment of pyramidal cells in layer i and (3) the feed-forward and feedback signals are combined in the somatic compartment of the pyramidal neuron. The combining mechanism of the feed-forward and feedback signals are specified in the network models of section 6.1 and 6.9. For visual clarity, only a few neuron connections are visualized, however, fully connected layers are used in the biological network implementations. The blue lines represent the axons and the blue dots the synapses. (b) A close-up of a pyramidal neuron with the following segregated compartments: the apical dendritic compartment A, the basal dendritic compartment B and the somatic compartment S. The axon is not shown on the figure. (c) An anatomical drawing of a pyramidal neuron in the human brain, used for comparison (figure adapted from [8]).

6.3 Network model

The network dynamics presented here are specified in terms of rates. Working with rates instead of potentials results in simpler network dynamics and is closer to the original TP method. The dendritic spikes that occur in pyramidal neurons also give an intuitive biological explanation for the rate-based network dynamics. The following dynamical equations are proposed:

$$\tau \frac{d}{dt}\boldsymbol{h}_i = -\boldsymbol{h}_i + (1-\mu) f_i(\boldsymbol{h}_{i-1}) + \mu g_i(\boldsymbol{h}_{i+1}), \quad i = 1, \ldots, L. \tag{6.1}$$

$\boldsymbol{h}_i$ represent the output firing rates of the neurons, based on the somatic voltage level. $f_i(\boldsymbol{h}_{i-1})$ represent the dendritic firing rates of the basal compartments, based on the basal voltage levels which depend on the feed-forward inputs $\boldsymbol{h}_{i-1}$. $g_i(\boldsymbol{h}_{i+1})$ represent the dendritic firing rates of the apical compartments, based on its voltage levels which depend on the feedback inputs $\boldsymbol{h}_{i+1}$. In this model, the output firing rates are a linear mixture of the two dendritic firing rates. For brevity, we have omitted the notation of the time dependence of the firing rates $\boldsymbol{h}_i(t)$. τ is the neural integration time constant. The network has a layered structure with $L+1$ layers, including inputs $\boldsymbol{h}_0$. Figure 6.1 gives a visual representation of the network. The forward mapping is given by

$$f_i(\boldsymbol{h}_{i-1}) = s_i(W_i \boldsymbol{h}_{i-1}), \quad i = 1, \ldots, L \tag{6.2}$$

with s_i the point-wise forward nonlinearity representing the neural transfer function, and W_i the basal synaptic weights. The purpose of g_i is to approximate f_{i+1}^{-1}. Two different forms of g_i will be investigated in the following sections, so g_i will be specified there. For the output layer, we set g to the desired target activity $\boldsymbol{h}_L^{trgt}$:

$$g_L(\boldsymbol{h}_{L+1}) = \boldsymbol{h}_L^{trgt}, \tag{6.3}$$

where we use $L+1$ to denote a virtual teaching layer. An important aspect in (6.1) is that the update is a *convex combination*, not a sum, of forward predictions $f_i(\boldsymbol{h}_{i-1})$ and feedback activity $g_i(\boldsymbol{h}_{i+1})$, with mixing coefficient $\mu \in (0;0.5)$. In a more detailed biophysical model this could be achieved via shunting inhibition [106].

6.4 Network with exact inverses

As a starting point for the analysis of this network, we assume that f_i is a bijection (so layers of equal size are needed) and that g_i is the perfect inverse of f_{i+1}:

$$g_i(\boldsymbol{h}_{i+1}) = f_{i+1}^{-1}(\boldsymbol{h}_{i+1}) = W_i^{-1} s_i^{-1}(\boldsymbol{h}_{i+1}). \tag{6.4}$$

Note that it is not very likely that exact inverses can be computed in our brain, but investigating this ideal case will give inportant insights in this dynamical model.

Fixed point equations. We assume that the network dynamics admit a stable fixed point $\{\boldsymbol{h}_i^*\}_{i=1}^L$, under a fixed input pattern $\boldsymbol{h}_0^*$:

$$\boldsymbol{h}_i^* = (1-\mu) f_i(\boldsymbol{h}_{i-1}^*) + \mu g_i(\boldsymbol{h}_{i+1}^*), \quad i = 1, \dots, L. \tag{6.5}$$

In section 6.7 we investigate whether the assumption on the stability of the fixed points is permissible. If we now assume that the feedback is small comparing to the feed-forward activation ($\mu \to 0$) we can make a first order Taylor approximation for g_i around $f_{i+1}(\boldsymbol{h}_i^*)$:

$$\begin{aligned} g_i(\boldsymbol{h}_{i+1}^*) &= g_i\big((1-\mu) f_{i+1}(\boldsymbol{h}_i^*) + \mu g_{i+1}(\boldsymbol{h}_{i+2}^*)\big) & (6.6)\\ &\approx g_i\big(f_{i+1}(\boldsymbol{h}_i^*)\big) + \mu J_{g_i}\big(g_{i+1}(\boldsymbol{h}_{i+2}^*) - f_{i+1}(\boldsymbol{h}_i^*)\big) & (6.7)\\ &= \boldsymbol{h}_i^* + \mu J_{f_{i+1}}^{-1}\big(g_{i+1}(\boldsymbol{h}_{i+2}^*) - f_{i+1}(\boldsymbol{h}_i^*)\big), & (6.8) \end{aligned}$$

with J_{g_i} the Jacobian of g_i evaluated at $f_{i+1}(\boldsymbol{h}_i^*)$ and $J_{f_{i+1}}$ the Jacobian of f_{i+1} evaluated at $\boldsymbol{h}_i^*$. The last step is based on the inverse function theorem and the fact that g_i is the perfect inverse of f_{i+1}. By filling in the approximation of $g_i(\boldsymbol{h}_{i+1}^*)$ into (6.5) and rearranging the terms, we get:

$$\boldsymbol{h}_i^* \approx f_i(\boldsymbol{h}_{i-1}^*) + \frac{\mu^2}{1-\mu} J_{f_{i+1}}^{-1}\Big(g_{i+1}(\boldsymbol{h}_{i+2}^*) - f_{i+1}(\boldsymbol{h}_i^*)\Big) \tag{6.9}$$

This equation has some interesting properties. First, if we do not provide a target to the output layer (by providing $\boldsymbol{h}_L^{trgt} = \boldsymbol{h}_L$ as a target for the last layer), by recursion one can show that the activity at every layer is equal to the forward prediction alone. This is possible due to the convex combination of (6.5). Second, if we do provide a target, we see in equation (6.9) that the difference of this target with the forward activation is backpropagated through the network. It seems that errors can propagate backwards through the network in an implicit fashion, without the need for dedicated circuitry as in [7]. These errors can be used as learning signals for the network. Third, a mixture state of the form (6.5) is probably useful for updating f_i without the need for distinct phases. We will first investigate how the target error gets propagated backwards through the network, after which we will propose possible weight updates to exploit this backpropagated target error.

Backpropagation of the target error. In a similar way to the original target propagation, let us define the output target $\boldsymbol{h}_L^{trgt}$ as

$$\begin{aligned} \boldsymbol{h}_L^{trgt} &= \boldsymbol{h}_L^* - \hat{\eta} \boldsymbol{e}_L & (6.10)\\ \boldsymbol{e}_L &\triangleq \frac{\partial L}{\partial \boldsymbol{h}_L^*}, & (6.11) \end{aligned}$$

with $L(\boldsymbol{h}_L^*, \boldsymbol{h}_L^{true})$ the loss for having output $\boldsymbol{h}_L^*$ instead of the wished true output $\boldsymbol{h}_L^{true}$ and $\hat{\eta}$ the output target step size. Filling this target into the fixed point equation (6.5) of the last layer, using (6.3) and rearranging the terms gives us the following fixed point equation for the last layer:

$$\boldsymbol{h}_L^* = f_L(\boldsymbol{h}_{L-1}^*) - \frac{\mu}{1-\mu}\hat{\eta}\boldsymbol{e}_L. \tag{6.12}$$

From equation (6.5) we observe that

$$g_{i+1}(\boldsymbol{h}_{i+2}^*) - f_{i+1}(\boldsymbol{h}_i^*) = \frac{1}{\mu}\big(\boldsymbol{h}_{i+1}^* - f_{i+1}(\boldsymbol{h}_i^*)\big) \tag{6.13}$$

Combining this with equation (6.9) gives us the following approximations for $\boldsymbol{h}_{L-1}^*$ and all other $\boldsymbol{h}_i^*$:

$$\boldsymbol{h}_{L-1}^* \approx f_{L-1}(\boldsymbol{h}_{L-2}^*) - \hat{\eta}\frac{\mu^2}{(1-\mu)^2}J_{f_L}^{-1}\boldsymbol{e}_L \tag{6.14}$$

$$\boldsymbol{h}_i^* \approx f_i(\boldsymbol{h}_{i-1}^*) - \hat{\eta}\frac{\mu^{L+1-i}}{(1-\mu)^{L+1-i}}\Big[\prod_{k=i}^{L-1} J_{f_{k+1}}^{-1}\Big]\boldsymbol{e}_L \tag{6.15}$$

$$= f_i(\boldsymbol{h}_{i-1}^*) - \hat{\eta}\frac{\mu^{L+1-i}}{(1-\mu)^{L+1-i}}\Big[\prod_{k=i}^{L-1} W_{k+1}^{-1}D_{s_{k+1}}^{-1}\Big]\boldsymbol{e}_L, \tag{6.16}$$

with D_{s_k} a diagonal matrix containing the partial derivatives of $s_k(W_k\boldsymbol{h}_{k-1}^*)$ with respect to $W_k\boldsymbol{h}_{k-1}^*$. The obtained expression (6.16) is almost identical to the one of pure target propagation with exact inverses (4.11), as $f_i(\boldsymbol{h}_{i-1}^*)$ is similar to the feed-forward activation $\boldsymbol{h}_i$ in target propagation. However, now there is no need for separate phases, as we work with only one signal and continuous time dynamics. In (6.16), a layer dependent scaling factor $\frac{\mu^{L+1-i}}{(1-\mu)^{L+1-i}}$ appeared. This scaling factor decays exponentially for layers towards the bottom, leading to a decay of the propagated target error. This could be compensated by a layer specific learning rate in the weight updates, as is done in [7]. However, in a noisy environment this can result in noise signals dominating the learning signal.

Weight updates. Similar to the forward weights update in target propagation (4.12), we can define purely local dynamics for the feed-forward weights.

$$\frac{dW_i}{dt} = -\eta_i D_{s_i}(f_i(\boldsymbol{h}_{i-1}^*) - \boldsymbol{h}_i^*)\boldsymbol{h}_{i-1}^{*T} \tag{6.17}$$

$$\approx -\eta_i\frac{\mu^{L+1-i}}{(1-\mu)^{L+1-i}}D_{s_i}\Big(\hat{\eta}\Big[\prod_{k=i}^{L-1} W_{k+1}^{-1}D_{s_{k+1}}^{-1}\Big]\boldsymbol{e}_L\Big)\boldsymbol{h}_{i-1}^{*T} \tag{6.18}$$

$$= -\tilde{\eta}_i D_{s_i}\Big(\hat{\eta}\Big[\prod_{k=i}^{L-1} W_{k+1}^{-1}D_{s_{k+1}}^{-1}\Big]\boldsymbol{e}_L\Big)\boldsymbol{h}_{i-1}^{*T}, \tag{6.19}$$

with $\tilde{\eta}_i = \eta_i\frac{\mu^{L+1-i}}{(1-\mu)^{L+1-i}}$, a layer specific learning rate. These weight dynamics can be seen as the continuous version of the discrete weight update rules discussed in chapter 4. When this continuous weight update (6.19) with the discrete weight update of target propagation with exact inverses (4.12), it can be seen that both are equal, except from the layer specific learning rate. However, now there is no need for separate phases or separate channels to propagate the target signals, it is done implicitly. Note that the backwards weights from g_i are equal to W_{i+1}^{-1} in this ideal case. They thus don't need to be learned, as they are directly computed from W_{i+1}^{-1} (e.g. with the Sherman-Morrison formula).

6.5 Network with approximate inverses

We now loosen the condition of exact inverses to approximate inverses and repeat the analysis of the previous section. The analysis makes abstraction of the specific parametric form of g_i.

Fixed point equations. By filling in the approximation of $g_i(\boldsymbol{h}_i^*)$ (6.7) into the fixed point equations (6.5) and rearranging the terms (subtract $\mu\boldsymbol{h}_i^*$ from both sides of the equation and divide by $(1-\mu)$), we get the following equations for $\boldsymbol{h}_i^*$:

$$\boldsymbol{h}_i^* \approx f_i(\boldsymbol{h}_{i-1}^*) + \frac{\mu}{1-\mu}\Big(g_i\big(f_{i+1}(\boldsymbol{h}_i^*)\big) - \boldsymbol{h}_i^*\Big) + \frac{\mu^2}{1-\mu}J_{g_i}\Big(g_{i+1}(\boldsymbol{h}_{i+2}^*) - f_{i+1}(\boldsymbol{h}_i^*)\Big) \tag{6.20}$$

When compared to the fixed point equations obtained with exact inverses, we see that now a reconstruction error term has appeared in the equations, similar to the reconstruction error term in target propagation with approximate inverses. When g_i is the exact inverse of f_{i+1} the reconstruction error term disappears. Now we will investigate how this reconstruction error gets propagated backwards through the network.

Backpropagation of the reconstruction error. Similar to the analysis with perfect inverses, we now obtain the following approximations for the fixed point equations in the case of approximate inverses:

$$\boldsymbol{h}^*_{L-1} \approx f_{L-1}(\boldsymbol{h}^*_{L-2}) + \frac{\mu}{1-\mu}\Big(g_{L-1}\big(f_L(\boldsymbol{h}^*_{L-1})\big) - \boldsymbol{h}^*_{L-1}\Big) - \hat{\eta}\frac{\mu^2}{(1-\mu)^2} J_{g_{L-1}} \boldsymbol{e}_L \tag{6.21}$$

$$\begin{aligned} \boldsymbol{h}^*_i &\approx f_i(\boldsymbol{h}^*_{i-1}) + \frac{\mu}{1-\mu}\Big(g_i\big(f_{i+1}(\boldsymbol{h}^*_i)\big) - \boldsymbol{h}^*_i\Big) + \sum_{j=i+1}^{L-1}\Big[\frac{\mu^{j+1-i}}{(1-\mu)^{j+1-i}}\Big(\prod_{k=i+1}^{j} J_{g_k}\Big)\Big(g_j(\boldsymbol{h}^*_{j+1}) - \boldsymbol{h}^*_j\Big)\Big].. \\ &- \hat{\eta}\frac{\mu^{L+1-i}}{(1-\mu)^{L+1-i}}\Big[\prod_{k=i}^{L-1} J_{g_k}\Big]\boldsymbol{e}_L, \quad i = 1,...,L-2 \end{aligned} \tag{6.22}$$

We now see that the reconstruction errors accumulate when propagating backward through the network. Hence, the useful learning signal $\left[\prod_{k=i}^{L-1} J_{g_k}\right]\boldsymbol{e}_L$ gets more and more deteriorated when going deeper in the network. The obtained equations are again very similar to the ones obtained with target propagation with approximate inverses, except from the layer specific scaling factors $\frac{\mu^{j+1-i}}{(1-\mu)^{j+1-i}}$.

Forward weight updates. For the forward weight updates, we can take the same update rule as with perfect inverses (6.17). However, the reconstruction errors dominate the target error (scale factor of $\frac{\mu}{1-\mu}$ instead of $\frac{\mu^{L+1-i}}{(1-\mu)^{L+1-i}}$), thus it can be expected that this update rule will perform poorly if the approximation of the inverses is not accurate enough.

Backward weight updates. Now that the backward mapping functions g_i are not anymore the perfect inverses of f_{i+1}, we need to define a parameterization of g_i. Let us stay close to our analysis of target propagation in chapter 4 and take:

$$g_i(\boldsymbol{h}_{i+1}) \triangleq Q_i t_i(\boldsymbol{h}_{i+1}), \tag{6.23}$$

with t_i the backward nonlinearity (neural transfer function in the apical dendrites) and Q_i the synaptic weights of the apical dendrites. Ideally, we want to minimize for each layer the regularized inverse approximation loss as defined in chapter 4:

$$L_i^{inv,r} = \Big|\Big|\boldsymbol{h}^*_i - g_i\big(f_{i+1}(\boldsymbol{h}^*_i)\big)\Big|\Big|_2^2 + \lambda\|Q_i\|_F^2, \tag{6.24}$$

with $\|Q_i\|_F$ the Frobenius norm of Q_i. By computing the gradient, this leads to the following weight update:

$$\Delta Q_i = -\eta_i^{inv}\Big(g_i\big(f_{i+1}(\boldsymbol{h}^*_i)\big) - \boldsymbol{h}^*_i\Big)t_i\big(f_{i+1}(\boldsymbol{h}^*_i)\big)^T - \eta_i^{inv}\lambda Q_i, \tag{6.25}$$

The weight decay term could be biologically implemented by a mechanism similar to homeostatic plasticity. As seen in theorem 4.6, under the condition that $t_i = s_{i+1}^{-1}$, this weight update will drive Q_i towards W_{i+1}^{-1} if W_{i+1} is invertible or will drive Q_i towards $W_{i+1}^{\dagger}$ if $\boldsymbol{h}_i$ is white noise and W_{i+1} is not invertible. As the mixing factor μ is small, we can approximate this weight update by the following weight dynamics:

$$\frac{dQ_i}{dt} = -\eta_i^{inv}\Big(g_i\big(\boldsymbol{h}^*_{i+1}\big) - \boldsymbol{h}^*_i\Big)t_i(\boldsymbol{h}^*_{i+1})^T - \eta_i^{inv}\lambda Q_i. \tag{6.26}$$

Note however that due to this approximation, Q_i will not approximate exactly W_{i+1}^{-1} or $W_{i+1}^{\dagger}$ anymore. A possible solution for this problem in networks with equal layer sizes is to introduce a *sleep phase*, where no teaching signal is applied on the network. Then $\boldsymbol{h}^*_L = f_L(h^*_{L-1})$ (see equation (6.3)) and if the Q_i's are trained until optimality starting from the top layers to the bottom layers, $f_i(\boldsymbol{h}^*_{i-1}) = \boldsymbol{h}^*_i$ due to the convex mixture used in the network dynamics. This results in equation (6.26) being an exact approximation of equation (6.25). For networks with unequal layer sizes, this solution is not possible, as no exact inverses of f_i exist, introducing reconstruction error terms in the network. Even if no teaching signal is applied on the network, $\boldsymbol{h}^*_i$ will not be equal to $f_i(\boldsymbol{h}^*_{i-1})$ in equilibrium.

6.6 Neural dynamics for difference target propagation

In section 6.5, we showed that reconstruction errors are propagated through the network when g_i is not the perfect inverse of f_{i+1}. Importantly, these reconstruction errors are scaled $\frac{\mu}{1-\mu}$, whereas the learning signal (the target error) has a scaling factor of $\frac{\mu^{L+1-i}}{(1-\mu)^{L+1-i}}$. This means that when g_i is not the perfect inverse of f_{i+1}, these reconstruction errors will completely dominate the learning signal, resulting in a network without useful learning signals. In section 4.3.3 we showed that difference target propagation [5] gets rid of the reconstruction errors that deteriorate the learning signal, by adjusting the propagated target to:

$$\hat{\boldsymbol{h}}_i = \boldsymbol{h}_i + g_i(\hat{\boldsymbol{h}}_{i+1}) - g_i(\boldsymbol{h}_{i+1}). \tag{6.27}$$

When a L_2 loss function is used for the local layer loss L_i, it can be simplified to:

$$L_i = \|\boldsymbol{h}_i - \hat{\boldsymbol{h}}_i\|_2^2 = \|g_i(\boldsymbol{h}_{i+1}) - g_i(\hat{\boldsymbol{h}}_{i+1})\|_2^2 \tag{6.28}$$

Now we see that a difference between the backpropagated target and the backpropagated forward activation is used for the local layer loss, and consequently also for the learning signal to update the forward weights W_i. Now we propose a modification to the dynamical network described in section 6.3 in order to incorporate a similar mechanism as difference target propagation to get rid of the reconstruction errors that deteriorate the learning signal. The new network consists of two phases: (1) a free phase, where no teaching signal is applied on the network, such that $\boldsymbol{h}_L^{trgt} = \boldsymbol{h}_L^*$ and (2) a learning phase, where a teaching signal is applied on the last layer of the network, according to equation (6.10). The learning signal for the forward weights W_i can then be obtained by taking the difference between the two phases. We first analyse the two phases, after which we discuss possible learning rules for both the forward weights W_i and the backward weights Q_i.

Free phase. In the free phase, the target output $\boldsymbol{h}_L^{trgt}$ is taken equal to the output equilibrium activation $\boldsymbol{h}_L^{f*}$ (imagine it as short-circuiting the output activation to the target output activation input). The input $\boldsymbol{h}_0$ remains fixed. The superscript f is used to indicate the free phase. Following equation (6.12), this results in

$$\boldsymbol{h}_L^{f*} = f_L(\boldsymbol{h}_{L-1}^{f*}). \tag{6.29}$$

Now equation (6.20) and (6.13) can be used to obtain first order Taylor approximations of $\boldsymbol{h}_i^{f*}$ during the free phase:

$$\boldsymbol{h}_{L-1}^{f*} = f_{L-1}(\boldsymbol{h}_{L-2}^{f*}) + \frac{\mu}{1-\mu}\Big(g_{L-1}\big(f_L(\boldsymbol{h}_{L-1}^{f*})\big) - \boldsymbol{h}_{L-1}^{f*}\Big) \tag{6.30}$$

$$\boldsymbol{h}_i^{f*} \approx f_i(\boldsymbol{h}_{i-1}^{f*}) + \frac{\mu}{1-\mu}\Big(g_i\big(f_{i+1}(\boldsymbol{h}_i^{f*})\big) - \boldsymbol{h}_i^{f*}\Big) + \sum_{j=i+1}^{L-1}\Bigg[\frac{\mu^{j+1-i}}{(1-\mu)^{j+1-i}}\Big(\prod_{k=i+1}^{j} J_{g_k}\Big)\Big(g_j(\boldsymbol{h}_{j+1}^{f*}) - \boldsymbol{h}_j^{f*}\Big)\Bigg] \tag{6.31}$$

The second and third term of the above equation represent the reconstruction error of the current layer and the backpropagated reconstruction errors of the layers on top, respectively.

Learning phase. During the learning phase, the target output $\boldsymbol{h}_L^{trgt}$ is set according to equation (6.3). The input $\boldsymbol{h}_0$ remains fixed. The same derivation as the one of equation (6.22) can be used to obtain the first order Taylor approximations of $\boldsymbol{h}_i^{l*}$ during the learning phase:

$$\boldsymbol{h}_{L-1}^{l*} \approx f_{L-1}(\boldsymbol{h}_{L-2}^{l*}) + \frac{\mu}{1-\mu}\Big(g_{L-1}\big(f_L(\boldsymbol{h}_{L-1}^{l*})\big) - \boldsymbol{h}_{L-1}^{l*}\Big) - \hat{\eta}\frac{\mu^2}{(1-\mu)^2}J_{g_{L-1}}\boldsymbol{e}_L \tag{6.32}$$

$$\begin{aligned}\boldsymbol{h}_i^{l*} \approx{}& f_i(\boldsymbol{h}_{i-1}^{l*}) + \frac{\mu}{1-\mu}\Big(g_i\big(f_{i+1}(\boldsymbol{h}_i^{l*})\big) - \boldsymbol{h}_i^{l*}\Big) + \sum_{j=i+1}^{L-1}\Bigg[\frac{\mu^{j+1-i}}{(1-\mu)^{j+1-i}}\Big(\prod_{k=i+1}^{j} J_{g_k}\Big)\Big(g_j(\boldsymbol{h}_{j+1}^{l*}) - \boldsymbol{h}_j^{l*}\Big)\Bigg].. \\ &- \hat{\eta}\frac{\mu^{L+1-i}}{(1-\mu)^{L+1-i}}\Big[\prod_{k=i}^{L-1} J_{g_k}\Big]\boldsymbol{e}_L, \quad i = 1,\dots,L-2\end{aligned} \tag{6.33}$$

The second and third term of the above equation represent the reconstruction error of the current layer and the backpropagated reconstruction errors of the layers on top, respectively. The last term represents the target error, which is the useful learning signal.

Forward weights update. Similar as in difference target propagation, we opt for a local layer loss of the following form:

$$L_i = \|\boldsymbol{h}_i^{f*} - \boldsymbol{h}_i^{l*}\|_2^2 \tag{6.34}$$

When comparing equation (6.31) and (6.33), we see that the second and third term of the equations, the reconstruction errors, are almost equal. They have exactly the same form, only $\boldsymbol{h}_i^{f*}$ differs from $\boldsymbol{h}_i^{l*}$. As the same fixed input $\boldsymbol{h}_0$ is used for both phases, the mixing constant μ is small and the output target $\boldsymbol{h}_L^{trgt}$ in the learning phase is only slightly moved towards lower loss with step size $\hat{\eta}$, it can be expected that the equilibrium points $\boldsymbol{h}_i^{f*}$ will only slightly differ from $\boldsymbol{h}_i^{l*}$. Therefore, in the local loss function L_i of equation (6.34), the reconstruction error terms are almost completely cancelled out against each other. Therefore, the useful backpropagated output error term $\boldsymbol{e}_L$ is less deteriorated by the reconstruction errors, and an almost clean learning signal remains. Based on L_i the following dynamics for W_i are proposed:

$$\frac{dW_i}{dt} = -\eta_i D_{s_i}^l (\boldsymbol{h}_{i-1}^{f*} - \boldsymbol{h}_i^{l*}) \boldsymbol{h}_{i-1}^{l*T} \tag{6.35}$$

$$\approx -\eta_i D_{s_i}^l \left(\frac{\mu^{L+1-i}}{(1-\mu)^{L+1-i}} \hat{\eta} \left[\prod_{k=i}^{L-1} J_{g_k} \right] \boldsymbol{e}_L + \boldsymbol{e}_i^{phase} \right) \boldsymbol{h}_{i-1}^{l*T}, \tag{6.36}$$

with $D_{s_i}^l$ the diagonal matrix with on its entries the derivatives of s_i, evaluated at $W_i \boldsymbol{h}_{i-1}^{l*}$, and $\boldsymbol{e}_i^{phase}$ the difference between sum of the second and third term of (6.31) and the sum of the ones of (6.33), which is expected to be small compared to the learning signal $\boldsymbol{e}_L$. Note that these weight dynamics cannot be directly interpreted as gradient descent dynamics on the local loss function L_i, because both $\boldsymbol{h}_i^{f*}$ and $\boldsymbol{h}_i^{l*}$ depend on W_i. The specific form of the weight dynamics were chosen to stay close to the originial difference target propagation method.

Backward weights update. As the reconstruction errors are almost completely cancelled out due to the difference-target-propagation-like network setting, it is no major problem if the backward mappings g_i only approximate the inverse forward mappings f_{i+1} up to limited precision. Therefore, the same weight dynamics as proposed in section (6.5) can be used:

$$\frac{dQ_i}{dt} = -\eta_i^{inv} \Big(g_i(\boldsymbol{h}_{i+1}^{f*}) - \boldsymbol{h}_i^{f*} \Big) t_i(\boldsymbol{h}_{i+1}^{f*})^T - \eta_i \lambda Q_i. \tag{6.37}$$

The backward weights Q_i will only approximate W_{i+1}^{-1} or $W_{i+1}^{\dagger}$ up to a limited precision, due to the approximations made in this weight dynamics (see section 6.5). However, this is not a major issue due to the small influence of the remainder of the reconstruction errors $\boldsymbol{e}_i^{phase}$ on the learning signal. As $\boldsymbol{h}_{i+1}^*$ is more close to $f_{i+1}(\boldsymbol{h}_i^*)$ during the free phase, it is best to update the backward weights during that phase.

To conclude, this two-phase modification of the dynamical target propagation network reduced the influence of the useless reconstruction errors on the useful learning signal, at the cost of introducing separate phases into the network dynamics. During the free phase, the plasticity of the backward weights Q_i should be turned on and the plasticity of the forward weights W_i turned off, whereas during the learning phase, the plasticity of the forward weights W_i should be turned on and those of the backward weights Q_i turned off. Section 6.8 will briefly discuss how this 2-phase network could operate in a biological setting. Future work should experimentally check whether this two-phase scheme can indeed reduce the reconstruction errors sufficiently to have a clean enough learning signal that can be used to train the network on challenging tasks.

6.7 Stability analysis of the fixed points

In this section, a local stability analysis is done for the fixed point $\{\boldsymbol{h}_i^*\}_{i=1}^L$ defined in equation (6.5), under the condition that g_i is the perfect inverse of f_{i+1}. During the local stability analysis, it is assumed that the input $\boldsymbol{h}_0$ remains constant. Define $\bar{\boldsymbol{h}}$ as the concatenated vector of $\{\boldsymbol{h}_i\}_{i=1}^L$, $\bar{f}(\bar{\boldsymbol{h}})$ as the concatenated vector function $\{f_i(\boldsymbol{h}_{i-1})\}_{i=1}^L$ and $\bar{g}(\bar{\boldsymbol{h}})$ as the concatenated vector function $\{g_i(\boldsymbol{h}_{i+1})\}_{i=1}^L$, with $g_L(\boldsymbol{h}_{L+1}) = \boldsymbol{h}_L^{trgt}$ and $\boldsymbol{h}_L^{trgt}$ fixed. Now the full set of equations of the network dynamics can be written as:

$$\tau \frac{d}{dt} \bar{\boldsymbol{h}} = -\bar{\boldsymbol{h}} + (1-\mu)\bar{f}(\bar{\boldsymbol{h}}) + \mu \bar{g}(\bar{\boldsymbol{h}}). \tag{6.38}$$

In order to analyse the local stability of this dynamical system, a first order Taylor approximation is made:

$$\tau \frac{d}{dt}\bar{\boldsymbol{h}} \approx \bar{J}\bar{\boldsymbol{h}} \tag{6.39}$$

$$\bar{J} = -I + (1-\mu)\frac{\partial \bar{f}(\bar{\boldsymbol{h}})}{\partial \bar{\boldsymbol{h}}}\Big|_{\bar{\boldsymbol{h}}=\bar{\boldsymbol{h}}^*} + \mu \frac{\partial \bar{g}(\bar{\boldsymbol{h}})}{\partial \bar{\boldsymbol{h}}}\Big|_{\bar{\boldsymbol{h}}=\bar{\boldsymbol{h}}^*}, \tag{6.40}$$

with $\bar{\boldsymbol{h}}^*$ the solution of $\bar{\boldsymbol{h}}^* = (1-\mu)\bar{f}(\bar{\boldsymbol{h}}^*) + \mu\bar{g}(\bar{\boldsymbol{h}}^*)$. The Jacobian matrix $\bar{J}$ can be written as a block tridiagonal matrix due to the local dynamics of the network layers (only interaction with itself, the layer on top and the layer below). Now define the local Jacobian block J_{f_i} as:

$$J_{f_i} = \frac{\partial f_i(\boldsymbol{h}_{i-1})}{\partial \boldsymbol{h}_{i-1}}\Big|_{\boldsymbol{h}_{i-1}=\boldsymbol{h}^*_{i-1}} = D_{s_i}W_i, \tag{6.41}$$

with W_i the forward weights of layer i and D_{s_i} the diagonal matrix with as entries the derivatives of the element-wise function s_i, evaluated at $\boldsymbol{h}_{i-1} = \boldsymbol{h}^*_{i-1}$. As this stability analysis is done for a network with perfect inverses, the following equality holds due to the inverse function theorem:

$$\frac{\partial g_i(\boldsymbol{h}_{i+1})}{\partial \boldsymbol{h}_{i+1}}\Big|_{\boldsymbol{h}_{i+1}=\boldsymbol{h}^*_{i+1}} = J^{-1}_{f_{i+1}}. \tag{6.42}$$

$\bar{J}$ is thus a block tridiagonal matrix with the square blocks $-I$ on its main diagonal, the square blocks $\{(1-\mu)J_{f_i}\}_{i=2}^L$ on its lower diagonal and the square blocks $\{\mu J^{-1}_{f_i}\}_{i=2}^L$ on its upper diagonal. For the fixed point $\{\boldsymbol{h}^*_i\}_{i=1}^L$ to be stable, $\bar{J}$ has to be negative definite (only strictly negative eigenvalues). It was checked experimentally that a matrix of this form is always negative definite, thereby indicating that the fixed point $\{\boldsymbol{h}^*_i\}_{i=1}^L$ is a stable fixed point. Hence, the assumption of a stable fixed point made in section 6.4 is valid locally for networks with exact inverses. Future research could investigate whether this fixed point is also globally stable, or in a certain region around the fixed point, similar to the work of Suykens et al. [107, 108]. Furthermore, it should be investigated whether the fixed point is still locally stable when approximate inverses are used. For the experiment, 100000 random cases were investigated, with J_i random matrices with a Gaussian distribution and randomly selected variance and with μ taken randomly in the interval (0;0.5).

6.8 Discussion and theoretical predictions

After a thorough theoretical analysis of the network proposed at the beginning of section 6.1, we can now discuss the implications of the main assumptions and propose theoretical predictions on the performance of this biological implementation of target propagation. Due to the limited size and timespan of this thesis, the theoretical predictions are not experimentally validated, but this can be done in future work. During this discussion, we will make a distinction between the single-phase network dynamics of section 6.3 and the two-phase network dynamics of section 6.6. We structure this discussion in the theoretical predictions we can make for its performance and the important assumptions we have made for our biological network implementations.

Influence of the reconstruction errors on the performance. In section 6.5, we showed that reconstruction errors are propagated through the single-phase network when g_i is not the perfect inverse of f_{i+1}. Importantly, these reconstruction errors are scaled with $\frac{\mu}{1-\mu}$, whereas the learning signal (the target error) has a scaling factor of $\frac{\mu^{L+1-i}}{(1-\mu)^{L+1-i}}$. This means that when g_i is not the perfect inverse of f_{i+1}, these reconstruction errors will completely dominate the learning signal, resulting in a network without useful learning signals. The single-phase network should thus always be designed in a way that ensures the existence of the inverses of the forward mappings f_i, implying that layers of equal size need to be used and that $t_i = s^{-1}_{i+1}$. If no perfect inverses of f_i exist, the single-phase network has no chance of success. In section (6.6), we proposed a two-phase network to get rid of the reconstruction errors that propagate through the network. We predict that this network can operate properly even when no perfect inverses of f_i exist, based on recent results in difference target propagation [5, 11] and biological networks that incorporate difference target propagation-like methods [8, 7]. This claim should however be experimentally verified in future work.

Distinct phases. The two-phase network model of section 6.6 uses two distinct phases: a free phase and a learning phase. This imposes two biological feasibility issues on the network: (1) two distinct global phases of neural processing are needed that need to be globally coordinated and (2) there exists a temporal gap between the two phases, indicating that the weight update of equation (6.35) needs to be computed over time. These issues could, theoretically, be solved by biologically plausible mechanisms [8]. The two distinct phases could be globally coordinated by neuromodulatory systems [109] or alternatively by oscillations that indicate which phase is on [110]. Taking the difference between the neuron activations at the free phase and learning phase (which is tens of milliseconds apart) is possible if there exists a cellular signal that is dependent on the firing rate of the neuron, has a slow time constant and reacts differently depending on which global phase is active. Whether such biological mechanisms exist and whether they are used in the way described by our two-phase network model is highly speculative, we only want to indicate in this discussion that they would be possible in theory. Future experimental research should determine whether these mechanisms do in fact exist and are used for the same purposes as we describe.

Comparison with related work. Our biological network implementation is mostly related to the work of Sacramento et al. [7] and Guerguiev et al. [8], which both were discussed at the end of chapter 3. Compared to the work of Sacramento et al., the advantage of our network is that it does not need a specialized cortical microcircuit to operate. The authors of [7] need a separate interneuron for each pyramidal neuron, which is inconsistent with the anatomical distribution of neurons in the cortex (the cortex consists out of $70-85\%$ pyramidal neurons [17]). The disadvantage of our work compared to the work of Sacramento et al. is that it needs two distinct phases to be able to operate in a noisy environment, whereas the cortical microcircuits of Sacramento et al. continuously operate in a single phase. Compared to the work of Guerguiev et al. [8], our model has the following advantages: (1) the feedback weights of our network are plastic, which is also observed in biology and (2) our model can naturally be implemented for deep architectures, whereas in the model of Guerguiev et al., each hidden layer needs a direct feedback connection from the output layer and it is reported that adding more than two hidden layers does not help the performance of the network. The disadvantage of our network model is that the learning signal exponentially decays for deeper network architectures. This exponentially decaying of the learning signal is inherently to network models that use a mixture of feed-forward and feedback signals and thus also occurs in the network of Sacramento et al. Similar to our work, the authors of [7] solve this problem by layer-specific learning rates.

Conclusion. To conclude, we emphasize the conceptual importance of this network model. It showed that a target-propagation-like method can be incorporated in a biologically realistic network model with very simple and intuitive dynamics, encouraging further research in both target propagation and biological network models with similar dynamics. The possible biological implementations of this network model are speculative and only partially in line with experimental results from neuroscience, but nonetheless, it provides the field with a new possible way forward to find a solution for the credit assignment problem in our brain and can lead to new insights in both neuroscience and deep learning. We end this chapter with a small outlook section on how a new hypothesis from theoretical neuroscience, neuronal multiplexing, promises to solve the issue of the exponentially decaying of the learning signal in our network model.

6.9 Outlook

The exponential decaying of the learning signal in the network discussed in this chapter is inherent to network models that use a mixture of feed-forward and feedback signals. Hence, a logical solution for the exponential decay is to develop a network model that does not mix feed-forward and feedback signals, but instead keeps them in separate channels. However, as a neuron only has a single output channel, the axon, on which it has to send both its feed-forward signal to the next layer and feedback signal to the lower layer, this poses a problem. In electrical engineering, such problems are solved by multiplexing: two or more signals are encoded on a single channel and a decoder separates and decodes the encoded signals at the end of the channel. A new hypothesis from theoretical neuroscience states that, similar to electrical engineering, pyramidal neurons can multiplex two separate signals through their axons.

6.9.1 Biological setting

This new proposed network model makes use of the segregated compartment model of pyramidal neurons as visualized in figure 6.1. As discussed in section 2.1.2, a recent hypothesis from theoretical neuroscience states that pyramidal neurons can multiplex through bursting spikes [29]. For a detailed description of the multiplexing mechanism, the reader is referred to section 2.1.2. The important results from this hypothesis are that a single neuron can now output two signals on its axon through bursting spikes: (1) an event rate, representing the feed-forward signal and (2) a bursting probability, representing the feedback signal. The event rate (feed-forward signal) is decoded at the synapses on the basal dendrites through a low-pass filter mechanism called short-term-depression and the bursting probability is decoded at the synapses of the apical dendrites through a high-pass filter mechanism called short-term-facilitation (see section 2.1.4). The basal (feed-forward) inputs determine the output event rate of the neuron, whereas the apical (feedback) inputs determine the output bursting rate of the neuron via plateau potentials. The feed-forward connections roughly all arrive at the basal dendrites [20, 19] and the feedback connections roughly all arrive at the apical dendrites [22, 23]. This gives rise to two completely separated and independent signal paths: (1) a feed-forward path encoded by the event rate and (2) a feedback path encoded by the bursting probabilities. Both signal paths also have their own non-linear activation function s_i and t_i, respectively, as they represent different neuronal mechanisms. The network wiring is again the same as in the network model of 6.3 and is visualized in figure 6.1.

6.9.2 Network model

Due to the decoupling of the feed-forward and feedback signals, the network dynamics are straight forward and no mixture model is needed. Let us define $\boldsymbol{a}_i^b$ as the voltage level of the basal dendritic compartment of neurons in layer i, $\boldsymbol{a}_i^a$ as the voltage level of the apical dendritic compartment, $\boldsymbol{h}_i$ as the output event rate of the neurons in layer i and $\hat{\boldsymbol{h}}_i$ as the output bursting probability of the neurons in layer i. Our new proposed network model has the following decoupled dynamics:

$$\frac{d\boldsymbol{a}_i^b}{dt} = -\boldsymbol{a}_i^b + W_i \boldsymbol{h}_{i-1} \tag{6.43}$$

$$\frac{d\boldsymbol{a}_i^a}{dt} = -\boldsymbol{a}_i^a + Q_i \hat{\boldsymbol{h}}_{i+1} \tag{6.44}$$

$$\boldsymbol{h}_i = s_i(\boldsymbol{a}_i^b) \tag{6.45}$$

$$\hat{\boldsymbol{h}}_i = t_i(\boldsymbol{a}_i^a), \tag{6.46}$$

with W_i the forward weights (basal synapses) and Q_i the backward weights (apical synapses), s_i the neuron activation function transforming the basal voltage level to an output event rate and t_i the neuron activation function transforming the apical voltage level to an output bursting potential.

Equilibrium points. The network has the following equilibrium points:

$$\boldsymbol{a}_i^{b*} = W_i \boldsymbol{h}_{i-1}^* \tag{6.47}$$

$$\boldsymbol{a}_i^{a*} = Q_i \hat{\boldsymbol{h}}_{i+1}^* \tag{6.48}$$

$$\boldsymbol{h}_i^* = s_i(\boldsymbol{a}_i^{b*}) = s_i(W_i \boldsymbol{h}_{i-1}^*) = f_i(\boldsymbol{h}_{i-1}^*) \tag{6.49}$$

$$\hat{\boldsymbol{h}}_i^* = t_i(\boldsymbol{a}_i^{a*}) = t_i(Q_i \hat{\boldsymbol{h}}_{i+1}^*) = g_i(\hat{\boldsymbol{h}}_{i+1}^*), \tag{6.50}$$

In equilibrium state, the network is thus entirely characterized by its event rates $\boldsymbol{h}_i^*$, burst probabilities $\hat{\boldsymbol{h}}_i^*$, forward mapping f_i and backward mapping g_i. This network can be interpreted as a target propagation network: the event rates $\boldsymbol{h}_i$ represent the feed-forward signal, whereas the burst probabilities $\hat{\boldsymbol{h}}_i$ represent the target signal. The neuron then tries to match the event rate to the target bursting probability by adapting its basal and apical synapses.

Learning signal. We propose a difference target propagation-like method for solving the credit assignment problem in our network implementation, similar to section 6.6. The output bursting rate $\hat{\boldsymbol{h}}_L$ of layer L is defined by:

$$g_L(\hat{\boldsymbol{h}}_{L+1}^*) = \boldsymbol{h}_L^{trgt}, \tag{6.51}$$

where $\hat{\boldsymbol{h}}^*_{L+1}$ can be seen as a virtual teaching layer, and $\boldsymbol{h}^{trgt}_L$ as the wished event rate output of the network. We introduce now two phases: a free phase and a learning phase. From now on, bursting probabilities during the free phase are denoted by a superscript $\hat{\boldsymbol{h}}^{*f}_i$ and during the learning phase by $\hat{\boldsymbol{h}}^{*l}_i$. The event rates $\boldsymbol{h}^*_i$ are the same during both phases, as the dynamics are decoupled and only the output target burst probability is fixed to a different value between the phases. During the free phase, $\boldsymbol{h}^{trgt}_L = \boldsymbol{h}^*_L$. In other words, the bursting probability $\hat{\boldsymbol{h}}^{f*}_L$ of the output layer L is short circuited to its event rate $\boldsymbol{h}^*_L$. If we assume that the backward mappings g_i are approximate inverses of f_{i+1}, $\hat{\boldsymbol{h}}^{f*}_i$ will lay close to $\boldsymbol{h}^*_i$ in this free phase, and $\hat{\boldsymbol{h}}^{f*}_i$ can be approximated by a first order Taylor expansion around $\boldsymbol{h}^*_i$ (derivation similar to section 4.3.1):

$$\hat{\boldsymbol{h}}^{f*}_{L-1} = g_{L-1}(\hat{\boldsymbol{h}}^{f*}_L) = g_{L-1}(\boldsymbol{h}^*_L) = \boldsymbol{h}^*_{L-1} + \left(g_{L-1}(\boldsymbol{h}^*_L) - \boldsymbol{h}^*_{L-1}\right) \tag{6.52}$$

$$\hat{\boldsymbol{h}}^{f*}_i \approx \boldsymbol{h}^*_i + \left(g_i(\boldsymbol{h}^*_{i+1}) - \boldsymbol{h}^*_i\right) + \sum_{j=i+1}^{L-1}\left[\left(\prod_{k=i}^{j-1} J_{g_k}\right)\left(g_j(\boldsymbol{h}^*_{j+1}) - \boldsymbol{h}^*_j\right)\right] \quad i = 1,\dots,L-2 \tag{6.53}$$

The approximate invertible state of the network can be a result of early development in the brain, after which suitable weight updates for W_i and $\boldsymbol{Q}_i$ keep the approximate invertible state intact. During the learning phase, $\boldsymbol{h}^{trgt}_L$ is tweaked towards lower loss:

$$\boldsymbol{h}^{trgt}_L = \boldsymbol{h}^*_L - \hat{\eta}\boldsymbol{e}_L \tag{6.54}$$

$$\boldsymbol{e}_L \triangleq \frac{\partial L}{\partial \boldsymbol{h}^*_L}, \tag{6.55}$$

Following the same derivation as equation (4.84), we can make the following first order Taylor approximations of $\hat{\boldsymbol{h}}^{l*}_i$ around $\boldsymbol{h}_i$:

$$\hat{\boldsymbol{h}}^{l*}_{L-1} \approx \boldsymbol{h}^*_{L-1} + \left(g_{L-1}(\boldsymbol{h}^*_L) - \boldsymbol{h}^*_{L-1}\right) + \hat{\eta} J_{g_{L-1}}\boldsymbol{e}_L \tag{6.56}$$

$$\hat{\boldsymbol{h}}^{l*}_i \approx \boldsymbol{h}^*_i + \left(g_i(\boldsymbol{h}^*_{i+1}) - \boldsymbol{h}^*_i\right) + \sum_{j=i+1}^{L-1}\left[\left(\prod_{k=i}^{j-1} J_{g_k}\right)\left(g_j(\boldsymbol{h}^*_{j+1}) - \boldsymbol{h}^*_j\right)\right] - \hat{\eta}\left[\prod_{k=i}^{L-1} J_{g_k}\right]\boldsymbol{e}_L. \tag{6.57}$$

The second and third term represent reconstruction errors of g_i and the fourth term represents the useful learning signal (the same as in the original target propagation method). The second and third term of equation (6.55) are exactly the same reconstruction error terms. Hence, we propose the following weight dynamics for the forward weights W_i in order to cancel out the reconstruction error terms:

$$\frac{dW_i}{dt} = -\eta_i D_{s_i}(\hat{\boldsymbol{h}}^{f*}_{i-1} - \hat{\boldsymbol{h}}^{l*}_i)\boldsymbol{h}^{*T}_{i-1} \tag{6.58}$$

$$\approx -\eta_i D_{s_i}\left(\hat{\eta}\left[\prod_{k=i}^{L-1} J_{g_k}\right]\boldsymbol{e}_L\right)\boldsymbol{h}^{l*T}_{i-1}, \tag{6.59}$$

We see that the reconstruction errors have exactly cancelled out in the first order Taylor expansion, leaving the W_i dynamics with a clean learning signal. Furthermore, there is no decaying scaling term present like in the mixture model of 6.3, making this model more suitable for deep architectures. The backward weights $\boldsymbol{Q}_i$ could be kept fixed, similar to the work of Guerguiev et al. [8], but more interestingly would be to find weight dynamics for $\boldsymbol{Q}_i$ such that g_i approximates the (pseudo-)inverse of f_{i+1}. In this way, the Gauss-Newton properties of difference target propagation (see theorem 4.9) could be exploited. We leave it to future research to find such update rules for $\boldsymbol{Q}_i$.

Chapter 7

Conclusion

This thesis focussed on the field of biologically plausible deep learning which seeks to bridge the gap between deep learning and neuroscience. We identified two current shortcomings in this field: (1) many of the biologically plausible learning methods do not yet have a solid mathematical foundation and (2) many of those methods are not closely linked to the properties of biological neurons, making it hard for neuroscientists to deduce workable hypotheses from them. In this thesis, we decided to focus on target propagation [9, 5], a promising biologically plausible learning method. We aimed to bridge the two above mentioned gaps by (1) creating a well-founded mathematical theory around the learning dynamics of target propagation, (2) improving the target propagation method by the gained mathematical insights and (3) developing a biological network model based on pyramidal neurons that exhibits target-propagation-like learning dynamics.

Through developing a mathematical framework around target propagation, based on Taylor approximations of the network signals, we discovered which optimization methods target propagation uses to minimize its output loss function. Under well-specified conditions, target propagation with exact inverses uses Gauss-Newton optimization to compute its local layer targets, after which it performs a gradient descent step on the local layer losses to update the forward weights of the network. As a second main theoretical result of this thesis, we discovered that reconstruction errors interfere with the propagated learning signals when approximate inverses are used to propagate the targets through the network. These reconstruction errors explain the poor performance of pure target propagation that was reported in the literature [5]. We showed that difference target propagation cancels out these reconstruction errors by adding a correction term to the propagated targets, which explains the good performance of difference target propagation in the literature [5, 11]. As the third main theoretical result of this thesis, we proved under well-specified conditions that difference target propagation uses Gauss-Newton optimization to compute its local layer targets, even when only approximate inverses are used to propagate the targets.

Based on the gained mathematical insight on target propagation, we proposed various improvements for the method. First, we introduced a new parametrization for the backward mapping function of the targets, which ensures that it can learn the inverse of the forward mapping to arbitrary precision when layers of equal size are used. Experimental results showed that this new parametrization significantly improved the performance of target propagation. Secondly, we proposed a modified version of target propagation that changes the gradient step of target propagation to a form more closely related to Gauss-Newton optimization. We hoped that this modified version would approximate more closely the second-order optimization characteristics of Gauss-Newton. Unfortunately, this method is not robustly stable, thus could not improve the existing target propagation method. Thirdly, we showed that target propagation uses a block-diagonal approximation of the Gauss-Newton curvature matrix, which is frequently done in the field of approximate second-order optimization for deep learning [85]. However, this block-diagonal approximation is not accurate in the setting of target propagation. Therefore, we proposed a randomized version of target propagation, which does not make use of this block-diagonal approximation. Experimental results showed that this randomized version of target propagation stabilizes the training process for deep network architectures. Lastly, experimental results showed that the target propagation method and its variants have slightly worse performance compared to error-backpropagation, which is a first-order optimization method. Hence, the mix of first- and second-order optimization in target propagation does not succeed in improving standard first-order optimization methods for deep learning. We hypothesized that this is mainly due to the non-adaptive learning rate used in target propagation, while Gauss-Newton optimization in deep learning [85] uses implicitly an adaptive learning rate and explicitly a line-search during each training iteration. Therefore, we encourage future research to look into adaptive learning rates for target propagation, such that its approximate second-order characteristics can surface.

In the final part of this thesis, we developed two biological network models that exhibit target-propagation-like learning dynamics, based on the segregated dendrites of pyramidal neurons. The first model is a simple mixture model of the apical dendritic spikes and the basal dendritic spikes that occur in the pyramidal neuron. This model has as significant contribution that it operates in continuous time without the need for separate phases, and it shows that target-propagation-like learning behaviour can be reached with very simple and biologically realistic model equations. A more detailed theoretical study of the model revealed that it suffers from reconstruction errors, similar to the target propagation method. Therefore, we introduced two separate phases in the model, resulting in difference-target-propagation-like behaviour. This reduced the influence of the reconstruction errors on the propagated learning signals. Lastly, we showed that the propagated learning signals in this model decay exponentially with depth, which is an inherent property of mixture-based models. Consequently, we introduced briefly a second biological model which is not based on mixtures but uses multiplexing to separate the feed-forward and feedback signals in the network. This model does not suffer from exponentially decaying learning signals. The learning performance of these two models is only worked out in theory. Hence, we encourage future research to simulate these networks to experimentally test their performance. If they perform well, future research should link the models in more detail towards real biological processes inside neural networks, such that neuroscientists can deduce workable hypotheses from them on the inner working of our brain.

To conclude, this thesis has made two main contributions to the field of biologically plausible deep learning and the scientific community as a whole. First, we created an extensive mathematical foundation for the target propagation method, which has led to improved variants of target propagation and which will help future research to further improve the target propagation method and other alternatives to error backpropagation. Hopefully, this work will encourage other researchers and accelerate the mathematical understanding of biologically plausible learning methods. Secondly, we developed two biologically more realistic models of target propagation. In future research, these two models can be linked in more detail to biological mechanisms inside neurons, such that they can give new hypotheses for neuroscience on how credit assignment is performed in biological networks.

Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, p. 533, 1986.

[4] Y. Bengio, D.-H. Lee, J. Bornschein, T. Mesnard, and Z. Lin, "Towards biologically plausible deep learning," *arXiv preprint arXiv:1502.04156*, 2015.

[5] D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio, "Difference target propagation," in *Joint european conference on machine learning and knowledge discovery in databases*. Springer, 2015, pp. 498–515.

[6] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature communications*, vol. 7, p. 13276, 2016.

[7] J. Sacramento, R. P. Costa, Y. Bengio, and W. Senn, "Dendritic error backpropagation in deep cortical microcircuits," *arXiv preprint arXiv:1801.00062*, 2017.

[8] J. Guerguiev, T. P. Lillicrap, and B. A. Richards, "Towards deep learning with segregated dendrites," *ELife*, vol. 6, p. e22901, 2017.

[9] Y. Bengio, "How auto-encoders could provide credit assignment in deep networks via target propagation," *arXiv preprint arXiv:1407.7906*, 2014.

[10] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[11] S. Bartunov, A. Santoro, B. A. Richards, G. E. Hinton, and T. Lillicrap, "Assessing the scalability of biologically-motivated deep learning algorithms and architectures," *arXiv preprint arXiv:1807.04587*, 2018.

[12] M. F. Bear, B. W. Connors, and M. A. Paradiso, *Neuroscience*. Lippincott Williams & Wilkins, 2007, vol. 2.

[13] P. Dayan, L. F. Abbott, and L. Abbott, "Theoretical neuroscience: computational and mathematical modeling of neural systems," 2001.

[14] S. T. Brady and L. Tai, "Cell biology of the nervous system," in *Basic Neurochemistry (Eighth Edition)*. Elsevier, 2012, pp. 3–25.

[15] S. Herculano-Houzel, "The human brain in numbers: a linearly scaled-up primate brain," *Frontiers in human neuroscience*, vol. 3, p. 31, 2009.

[16] A. M. Zador, "A critique of pure learning: What artificial neural networks can learn from animal brains," *BioRxiv*, p. 582643, 2019.

[17] J. DeFelipe and I. Fariñas, "The pyramidal neuron of the cerebral cortex: morphological and chemical characteristics of the synaptic inputs," *Progress in neurobiology*, vol. 39, no. 6, pp. 563–607, 1992.

[18] S. Manita, T. Suzuki, C. Homma, T. Matsumoto, M. Odagawa, K. Yamada, K. Ota, C. Matsubara, A. Inutsuka, M. Sato *et al.*, "A top-down cortical circuit for accurate sensory perception," *Neuron*, vol. 86, no. 5, pp. 1304–1316, 2015.

[19] J. M. Budd, “Extrastriate feedback to primary visual cortex in primates: a quantitative analysis of connectivity,” *Proceedings of the Royal Society of London B: Biological Sciences*, vol. 265, no. 1400, pp. 1037–1044, 1998.

[20] M. Spratling, “Cortical region interactions and the functional role of apical dendrites,” *Behavioral and cognitive neuroscience reviews*, vol. 1, no. 3, pp. 219–228, 2002.

[21] M. E. Larkum, J. J. Zhu, and B. Sakmann, “A new cellular mechanism for coupling inputs arriving at different cortical layers,” *Nature*, vol. 398, no. 6725, p. 338, 1999.

[22] M. E. Larkum, J. Waters, B. Sakmann, and F. Helmchen, “Dendritic spikes in apical dendrites of neocortical layer 2/3 pyramidal neurons,” *Journal of Neuroscience*, vol. 27, no. 34, pp. 8999–9008, 2007.

[23] M. E. Larkum, T. Nevian, M. Sandler, A. Polsky, and J. Schiller, “Synaptic integration in tuft dendrites of layer 5 pyramidal neurons: a new unifying principle,” *Science*, vol. 325, no. 5941, pp. 756–760, 2009.

[24] R. S. Kasevich and D. LaBerge, “Theory of electric resonance in the neocortical apical dendrite,” *PLoS One*, vol. 6, no. 8, p. e23412, 2011.

[25] W. Nernst, “Die elektromotorische wirksamkeit der jonen,” *Zeitschrift für physikalische Chemie*, vol. 4, no. 1, pp. 129–181, 1889.

[26] A. L. Hodgkin, A. F. Huxley, and B. Katz, “Measurement of current-voltage relations in the membrane of the giant axon of loligo,” *The Journal of physiology*, vol. 116, no. 4, pp. 424–448, 1952.

[27] Wikipedia. (2018) Action potential. [Online]. Available: https://en.wikipedia.org/wiki/Action_potential

[28] M. Steriade, D. A. McCormick, and T. J. Sejnowski, “Thalamocortical oscillations in the sleeping and aroused brain,” *Science*, vol. 262, no. 5134, pp. 679–685, 1993.

[29] R. Naud and H. Sprekeler, “Burst ensemble multiplexing: A neural code connecting dendritic spikes with microcircuits,” *bioRxiv*, p. 143636, 2017.

[30] P. Gao, K. T. Sultan, X.-J. Zhang, and S.-H. Shi, “Lineage-dependent circuit assembly in the neocortex,” *Development*, vol. 140, no. 13, pp. 2645–2655, 2013.

[31] B. W. Connors and M. A. Long, “Electrical synapses in the mammalian brain,” *Annu. Rev. Neurosci.*, vol. 27, pp. 393–418, 2004.

[32] T. C. Sudhof, “The synaptic vesicle cycle,” *Annual review of neuroscience*, vol. 27, p. 509, 2004.

[33] D. M. Rosenbaum, S. G. Rasmussen, and B. K. Kobilka, “The structure and function of g-protein-coupled receptors,” *Nature*, vol. 459, no. 7245, p. 356, 2009.

[34] A. Citri and R. C. Malenka, “Synaptic plasticity: multiple forms, functions, and mechanisms,” *Neuropsychopharmacology*, vol. 33, no. 1, p. 18, 2008.

[35] R. S. Zucker and W. G. Regehr, “Short-term synaptic plasticity,” *Annual review of physiology*, vol. 64, no. 1, pp. 355–405, 2002.

[36] D. O. Hebb *et al.*, “The organization of behavior,” 1949.

[37] T. V. Bliss, A. Gardner-Medwin, and T. Lømo, “Synaptic plasticity in the hippocampal formation,” *Macromolecules and behaviour*, vol. 193, p. 203, 1973.

[38] T. V. Bliss and T. Lømo, “Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path,” *The Journal of physiology*, vol. 232, no. 2, pp. 331–356, 1973.

[39] G. G. Turrigiano and S. B. Nelson, “Homeostatic plasticity in the developing nervous system,” *Nature Reviews Neuroscience*, vol. 5, no. 2, p. 97, 2004.

[40] W. C. Abraham and M. F. Bear, “Metaplasticity: the plasticity of synaptic plasticity,” *Trends in neurosciences*, vol. 19, no. 4, pp. 126–130, 1996.

[41] J. Z. Tsien, P. T. Huerta, and S. Tonegawa, “The essential role of hippocampal ca1 nmda receptor–dependent synaptic plasticity in spatial memory,” *Cell*, vol. 87, no. 7, pp. 1327–1338, 1996.

[42] R. C. Malenka and M. F. Bear, “Ltp and ltd: an embarrassment of riches,” *Neuron*, vol. 44, no. 1, pp. 5–21, 2004.

[43] V. Chevaleyre, K. A. Takahashi, and P. E. Castillo, “Endocannabinoid-mediated synaptic plasticity in the cns,” *Annu. Rev. Neurosci.*, vol. 29, pp. 37–76, 2006.

[44] J. A. Cummings, R. M. Mulkey, R. A. Nicoll, and R. C. Malenka, “Ca2+ signaling requirements for long-term depression in the hippocampus,” *Neuron*, vol. 16, no. 4, pp. 825–833, 1996.

[45] W. Gerstner, M. Lehmann, V. Liakoni, D. Corneil, and J. Brea, “Eligibility traces and plasticity on behavioral time scales: Experimental support of neohebbian three-factor learning rules,” *Frontiers in Neural Circuits*, 2018.

[46] W. Gerstner, R. Kempter, J. L. van Hemmen, and H. Wagner, “A neuronal learning rule for sub-millisecond temporal coding,” *Nature*, vol. 383, no. 6595, p. 76, 1996.

[47] Y. Dan and M.-m. Poo, “Spike timing-dependent plasticity of neural circuits,” *Neuron*, vol. 44, no. 1, pp. 23–30, 2004.

[48] Y. Dan and M.-M. Poo, “Spike timing-dependent plasticity: from synapse to perception,” *Physiological reviews*, vol. 86, no. 3, pp. 1033–1048, 2006.

[49] G. G. Turrigiano, K. R. Leslie, N. S. Desai, L. C. Rutherford, and S. B. Nelson, “Activity-dependent scaling of quantal amplitude in neocortical neurons,” *Nature*, vol. 391, no. 6670, p. 892, 1998.

[50] B. D. Philpot, K. K. Cho, and M. F. Bear, “Obligatory role of nr2a for metaplasticity in visual cortex,” *Neuron*, vol. 53, no. 4, pp. 495–502, 2007.

[51] F. Rosenblatt, “The perceptron–a perciving and recognizing automation,” *Report 85-460-1 Cornell Aeronautical Laboratory, Ithaca, Tech. Rep.*, 1957.

[52] ——, “The perceptron: a probabilistic model for information storage and organization in the brain.” *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[53] C. Börgers, *An introduction to modeling neuronal dynamics*. Springer, 2017, vol. 66.

[54] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[55] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[56] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein gan,” *arXiv preprint arXiv:1701.07875*, 2017.

[57] Y. LeCun, “Une procédure d’apprentissage pour réseau à seuil asymétrique,” *proceedings of Cognitiva 85*, pp. 599–604, 1985.

[58] L. Bottou and O. Bousquet, “The tradeoffs of large scale learning,” in *Advances in neural information processing systems*, 2008, pp. 161–168.

[59] D. Huber, D. Gutnisky, S. Peron, D. O’connor, J. Wiegert, L. Tian, T. Oertner, L. Looger, and K. Svoboda, “Multiple dynamic representations in the motor cortex during sensorimotor learning,” *Nature*, vol. 484, no. 7395, p. 473, 2012.

[60] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, “Occam’s razor,” *Information processing letters*, vol. 24, no. 6, pp. 377–380, 1987.

[61] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[62] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 1096–1103.

[63] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris *et al.*, "Simulation of networks of spiking neurons: a review of tools and strategies," *Journal of computational neuroscience*, vol. 23, no. 3, pp. 349–398, 2007.

[64] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *International journal of neural systems*, vol. 19, no. 04, pp. 295–308, 2009.

[65] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input," *Biological cybernetics*, vol. 95, no. 1, pp. 1–19, 2006.

[66] M. B. Goodman, D. H. Hall, L. Avery, and S. R. Lockery, "Active currents regulate sensitivity and dynamic range in c. elegans neurons," *Neuron*, vol. 20, no. 4, pp. 763–772, 1998.

[67] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[68] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random feedback weights support learning in deep neural networks," *arXiv preprint arXiv:1411.0247*, 2014.

[69] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[70] W. Xiao, H. Chen, Q. Liao, and T. Poggio, "Biologically-plausible learning algorithms can scale to large datasets," Center for Brains, Minds and Machines (CBMM), Tech. Rep., 2018.

[71] A. Nøkland, "Direct feedback alignment provides learning in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 1037–1045.

[72] A. Samadi, T. P. Lillicrap, and D. B. Tweed, "Deep learning with dynamic spiking neurons and fixed feedback weights," *Neural computation*, vol. 29, no. 3, pp. 578–602, 2017.

[73] H. Luo, J. Fu, and J. Glass, "Bidirectional backpropagation: Towards biologically plausible error signal transmission in neural networks," *arXiv preprint arXiv:1702.07097*, 2017.

[74] K. C. Bittner, C. Grienberger, S. P. Vaidya, A. D. Milstein, J. J. Macklin, J. Suh, S. Tonegawa, and J. C. Magee, "Conjunctive input processing drives feature selectivity in hippocampal ca1 neurons," *Nature neuroscience*, vol. 18, no. 8, p. 1133, 2015.

[75] K. C. Bittner, A. D. Milstein, C. Grienberger, S. Romani, and J. C. Magee, "Behavioral time scale synaptic plasticity underlies ca1 place fields," *Science*, vol. 357, no. 6355, pp. 1033–1036, 2017.

[76] J. Sherman, "Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix," *Annals of mathematical statistics*, vol. 20, no. 4, p. 621, 1949.

[77] C. D. Meyer, Jr, "Generalized inversion of modified matrices," *SIAM Journal on Applied Mathematics*, vol. 24, no. 3, pp. 315–323, 1973.

[78] C. F. Gauss, *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*. Perthes et Besser, 1809, vol. 7.

[79] R. W. Wedderburn, "Quasi-likelihood functions, generalized linear models, and the gauss—newton method," *Biometrika*, vol. 61, no. 3, pp. 439–447, 1974.

[80] W. contributors, "Gauss-Newton algorithm," 2019, online; accessed 5-June-2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Gauss%E2%80%93Newton_algorithm&oldid=893757020

[81] J. J. Moré, "The Levenberg-Marquardt algorithm: implementation and theory," in *Numerical analysis*. Springer, 1978, pp. 105–116.

[82] E. H. Moore, "On the reciprocal of the general algebraic matrix," *Bull. Am. Math. Soc.*, vol. 26, pp. 394–395, 1920.

[83] R. Penrose, "A generalized inverse for matrices," in *Mathematical proceedings of the Cambridge philosophical society*, vol. 51, no. 3. Cambridge University Press, 1955, pp. 406–413.

[84] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," in *International conference on machine learning*, 2015, pp. 2408–2417.

[85] A. Botev, H. Ritter, and D. Barber, "Practical Gauss-Newton optimisation for deep learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 557–565.

[86] S. Chen, C. Cowan, S. Billings, and P. Grant, "Parallel recursive prediction error algorithm for training layered neural networks," *International Journal of control*, vol. 51, no. 6, pp. 1215–1228, 1990.

[87] N. N. Schraudolph, "Fast curvature matrix-vector products for second-order gradient descent," *Neural computation*, vol. 14, no. 7, pp. 1723–1738, 2002.

[88] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[89] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o (1/k2)," in *Sov. Math. Dokl*, vol. 27, no. 2.

[90] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, 2013, pp. 1139–1147.

[91] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[92] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[93] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quarterly of applied mathematics*, vol. 2, no. 2, pp. 164–168, 1944.

[94] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[95] H. O. Hartley, "The modified gauss-newton method for the fitting of non-linear regression functions by least squares," *Technometrics*, vol. 3, no. 2, pp. 269–280, 1961.

[96] M. Rudelson and R. Vershynin, "The littlewood–offord problem and invertibility of random matrices," *Advances in Mathematics*, vol. 218, no. 2, pp. 600–633, 2008.

[97] A. Pinkus, "Approximation theory of the mlp model in neural networks," *Acta numerica*, vol. 8, pp. 143–195, 1999.

[98] M. Akrout, C. Wilson, P. C. Humphreys, T. Lillicrap, and D. Tweed, "Using weight mirrors to improve feedback alignment," *arXiv preprint arXiv:1904.05391*, 2019.

[99] A. Ben-Israel and D. Cohen, "On iterative computation of generalized inverses and associated projections," *SIAM Journal on Numerical Analysis*, vol. 3, no. 3, pp. 410–419, 1966.

[100] A. Y. Ng, "Feature selection, l 1 vs. l 2 regularization, and rotational invariance," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 78.

[101] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[102] A. D. Wyner, "Random packings and coverings of the unit n-sphere," *The Bell System Technical Journal*, vol. 46, no. 9, pp. 2111–2118, 1967.

[103] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[104] N. L. Golding and N. Spruston, “Dendritic sodium spikes are variable triggers of axonal action potentials in hippocampal ca1 pyramidal neurons,” *Neuron*, vol. 21, no. 5, pp. 1189–1200, 1998.

[105] N. L. Golding, N. P. Staff, and N. Spruston, “Dendritic spikes as a mechanism for cooperative long-term potentiation,” *Nature*, vol. 418, no. 6895, p. 326, 2002.

[106] R. Urbanczik and W. Senn, “Learning by the dendritic prediction of somatic spiking,” *Neuron*, vol. 81, no. 3, pp. 521–528, 2014.

[107] J. A. Suykens, B. L. De Moor, and J. Vandewalle, “NLq theory: a neural control framework with global asymptotic stability criteria,” *Neural Networks*, vol. 10, no. 4, pp. 615–637, 1997.

[108] J. A. Suykens, J. Vandewalle, and B. L. De Moor, “NLq theory: checking and imposing stability of recurrent neural networks for nonlinear modeling,” *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2682–2691, 1997.

[109] H.-J. Pi, B. Hangya, D. Kvitsiani, J. I. Sanders, Z. J. Huang, and A. Kepecs, “Cortical interneurons that specialize in disinhibitory control,” *Nature*, vol. 503, no. 7477, p. 521, 2013.

[110] J. Veit, R. Hakim, M. P. Jadi, T. J. Sejnowski, and H. Adesnik, “Cortical gamma band synchronization through somatostatin interneurons,” *Nature neuroscience*, vol. 20, no. 7, p. 951, 2017.

[111] A. M. Saxe, J. L. McClelland, and S. Ganguli, “Exact solutions to the nonlinear dynamics of learning in deep linear neural networks,” *arXiv preprint arXiv:1312.6120*, 2013.

[112] “Wikipedia: Moore-penrose inverse,” https://en.wikipedia.org/wiki/Moore\T1\textendashPenrose_inverse, 2019, accessed: 2019-03-02.

Appendices

Appendix A

Algorithms

Algorithm A.1: The update of one set of layer parameters with modified target propagation

```
**Result:** $\Delta W_k$: the modified target propagation update of the weights of the $k$-th layer, while all other weights remain fixed;
**Input:** $\{\boldsymbol{h}_0^{(b)}\}_{b=1}^B$: a mini-batch of input samples;
**for** *i in range(1,L)* **do**
    Batch normalize layer $i-1$: $\{\tilde{\boldsymbol{h}}_{i-1}^{(b)}\}_{b=1}^B = \{\frac{1}{\boldsymbol{\sigma}_{i-1}^{(b)}} \odot \left(\boldsymbol{h}_{i-1}^{(b)} - \boldsymbol{\mu}_{i-1}^{(b)}\right)\}_{b=1}^B$ ;
    Propagate mini-batch forward: $\{\boldsymbol{h}_i^{(b)}\}_{b=1}^B = \{f_i(\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B$ ;
Compute output target: $\{\hat{\boldsymbol{h}}_L^{(b)}\}_{b=1}^B = \{\boldsymbol{h}_L^{(b)} - \hat{\eta}\frac{\partial L}{\boldsymbol{h}_L^{(b)}}\}_{b=1}^B$;
**for** *i in range(L-1,k)* **do**
    Propagate target backwards: $\{\hat{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B = \{g_i(\hat{\boldsymbol{h}}_{i+1}^{(b)})\}_{b=1}^B$ ;
    Counter batch normalization: $\{\hat{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B = \{\boldsymbol{\sigma}_i^{(b)} \odot \hat{\boldsymbol{h}}_i^{(b)} + \boldsymbol{\mu}_i^{(b)}\}_{b=1}^B$;
$\Delta W_k = -\frac{1}{B}\sum_{b=1}^B \eta_k D_{s_k}^{-1,(b)}(\boldsymbol{h}_k^{(b)} - \hat{\boldsymbol{h}}_k^{(b)})\boldsymbol{h}_{k-1}^{T,(b)}$;
```

Algorithm A.2: Randomized target propagation training iteration

```
**Result:** The parameters $W_k$ of one randomly chosen layer are updated by the target propagation method;
**Input:** $\{\boldsymbol{h}_0^{(b)}\}_{b=1}^B$: a mini-batch of input samples;
**for** *i in range(1,L)* **do**
    Propagate mini-batch forward: $\{\boldsymbol{h}_i^{(b)}\}_{b=1}^B = \{f_i(\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B$ ;
Compute output targets: $\{\hat{\boldsymbol{h}}_L^{(b)}\}_{b=1}^B = \{\boldsymbol{h}_L^{(b)} - \hat{\eta}\frac{\partial L}{\boldsymbol{h}_L^{(b)}}\}_{b=1}^B$;
Chose one random layer $k$ from $\{k\}_{k=1}^L$ ;
**for** *i in range(L-1,k)* **do**
    Propagate target backwards: $\{\hat{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B = \{g_i(\hat{\boldsymbol{h}}_{i+1}^{(b)})\}_{b=1}^B$ ;
$\Delta W_k = -\frac{1}{B}\sum_{b=1}^B \eta_k D_{s_k}^{(b)}(\boldsymbol{h}_k^{(b)} - \hat{\boldsymbol{h}}_k^{(b)})\boldsymbol{h}_{k-1}^{T,(b)}$;
$W_k \leftarrow W_k + \Delta W_k$;
```

```
Algorithm A.3: Randomized modified target propagation training iteration
  Result: The parameters $W_k$ of one randomly chosen layer are updated by the modified target
          propagation method;
  Input: $\{\boldsymbol{h}_0^{(b)}\}_{b=1}^B$: a batch of input samples;
  for i in range(1,L) do
      Batch normalize layer $i-1$: $\{\tilde{\boldsymbol{h}}_{i-1}^{(b)}\}_{b=1}^B = \{\frac{1}{\boldsymbol{\sigma}_{i-1}^{(b)}} \odot (\boldsymbol{h}_{i-1}^{(b)} - \boldsymbol{\mu}_{i-1}^{(b)})\}_{b=1}^B$ ;
      Propagate mini-batch forward: $\{\boldsymbol{h}_i^{(b)}\}_{b=1}^B = \{f_i(\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B$ ;
  Compute output targets: $\{\hat{\boldsymbol{h}}_L^{(b)}\}_{b=1}^B = \{\boldsymbol{h}_L^{(b)} - \hat{\eta}\frac{\partial L}{\boldsymbol{h}_L^{(b)}}\}_{b=1}^B$;
  Chose one random layer $k$ from $\{k\}_{k=1}^L$ ;
  for i in range(L-1,k) do
      Propagate target backwards: $\{\hat{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B = \{g_i(\hat{\boldsymbol{h}}_{i+1}^{(b)})\}_{b=1}^B$ ;
      Counter batch normalization: $\{\hat{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B = \{\boldsymbol{\sigma}_i^{(b)} \odot \hat{\boldsymbol{h}}_i^{(b)} + \boldsymbol{\mu}_i^{(b)}\}_{b=1}^B$;
  $\Delta W_k = -\frac{1}{B}\sum_{b=1}^B \eta_k D_{s_k}^{-1,(b)}(\boldsymbol{h}_k^{(b)} - \hat{\boldsymbol{h}}_k^{(b)})\boldsymbol{h}_{k-1}^{T,(b)}$;
  $W_k \leftarrow W_k + \Delta W_k$;
```

```
Algorithm A.4: Training iteration of target propagation with exact inverses
  Result: The network parameters $W_i$, $i = 1,...,L$, and $W_i^{-1}$, $i = 2,...,L$ are robustly updated
          according to the target propagation training scheme with exact inverses.
  Input: $\boldsymbol{h}_0$: one input sample;
  for i in range(1,L) do
      Propagate sample forward: $\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(W_i\boldsymbol{h}_{i-1})$ ;
  Compute output target: $\hat{\boldsymbol{h}}_L = \boldsymbol{h}_L - \hat{\eta}\frac{\partial L}{\boldsymbol{h}_L}$;
  for i in range(L-1,1) do
      Propagate target backwards: $\hat{\boldsymbol{h}}_i = g_i(\hat{\boldsymbol{h}}_{i+1}) = W_{i+1}^{-1}s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1})$ ;
  for i in range(L,2) do
      Update parameters with the robust Sherman-Morrison update;
      $D_{s_i} = \frac{\partial s_i(W_i\boldsymbol{h}_{i-1})}{\partial W_i\boldsymbol{h}_{i-1}}$;
      $\boldsymbol{u}_i = -\eta_i D_{s_i}(\boldsymbol{h}_i - \hat{\boldsymbol{h}}_i)$ ;
      $\boldsymbol{v}_i = \boldsymbol{h}_{i-1}$;
      $d = \boldsymbol{v}_i^T W_i^{-1}\boldsymbol{u}_i$;
      if $|1+d| \geq \epsilon$ then
          $W_i \leftarrow W_i + \boldsymbol{u}_i\boldsymbol{v}_i^T$;
          $W_i^{-1} \leftarrow W_i^{-1} - \frac{W_i^{-1}\boldsymbol{u}_i\boldsymbol{v}_i^T W_i^{-1}}{1+d}$;
      else
          $\beta = \frac{1}{\epsilon - d}$;
          $W_i \leftarrow W_i + \beta\boldsymbol{u}_i\boldsymbol{v}_i^T$;
          $W_i^{-1} \leftarrow W_i^{-1} - \frac{W_i^{-1}\boldsymbol{u}_i\boldsymbol{v}_i^T W_i^{-1}}{\epsilon}$;
  Update the forward parameters of the first hidden layer;
  $W_1 \leftarrow W_1 - \eta_1 D_{s_1}(\boldsymbol{h}_1 - \hat{\boldsymbol{h}}_1)\boldsymbol{h}_0^T$;
```

Algorithm A.5: Training iteration of randomized target propagation with exact inverses

```
**Result:** One pair of randomly chosen layer parameters $W_k$, and $W_k^{-1}$, are robustly updated
        according to the randomized target propagation training scheme with exact inverses.
**Input:** $\boldsymbol{h}_0$: one input sample;
**for** *i in range(1,L)* **do**
    Propagate sample forward: $\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(W_i \boldsymbol{h}_{i-1})$ ;
Compute output target: $\hat{\boldsymbol{h}}_L = \boldsymbol{h}_L - \hat{\eta} \frac{\partial L}{\boldsymbol{h}_L}$;
Chose one random layer $k$ from $\{k\}_{k=1}^{L}$ ;
**for** *i in range(L-1,k)* **do**
    Propagate target backwards: $\hat{\boldsymbol{h}}_i = g_i(\hat{\boldsymbol{h}}_{i+1}) = W_{i+1}^{-1} s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1})$ ;
**if** *k>1* **then**
    Update parameter pair with the robust Sherman-Morrison update;
    $D_{s_k} = \frac{\partial s_k(W_k \boldsymbol{h}_{k-1})}{\partial W_k \boldsymbol{h}_{k-1}}$;
    $\boldsymbol{u}_i = -\eta_k D_{s_k}(\boldsymbol{h}_k - \hat{\boldsymbol{h}}_k)$ ;
    $\boldsymbol{v}_i = \boldsymbol{h}_{k-1}$;
    $d = \boldsymbol{v}_i^T W_k^{-1} \boldsymbol{u}_i$;
    **if** $|1+d| \geq \epsilon$ **then**
        $W_k \leftarrow W_k + \boldsymbol{u}_i \boldsymbol{v}_i^T$;
        $W_k^{-1} \leftarrow W_k^{-1} - \frac{W_k^{-1} \boldsymbol{u}_i \boldsymbol{v}_i^T W_k^{-1}}{1+d}$;
    **else**
        $\beta = \frac{1}{\epsilon - d}$;
        $W_k \leftarrow W_k + \beta \boldsymbol{u}_i \boldsymbol{v}_i^T$;
        $W_k^{-1} \leftarrow W_k^{-1} - \frac{W_k^{-1} \boldsymbol{u}_i \boldsymbol{v}_i^T W_k^{-1}}{\epsilon}$;
**else**
    Update the forward parameters of the first hidden layer;
    $W_1 \leftarrow W_1 - \eta_1 D_{s_1}(\boldsymbol{h}_1 - \hat{\boldsymbol{h}}_1) \boldsymbol{h}_0^T$;
```

Algorithm A.6: Training iteration of randomized modified target propagation with exact inverses

```
Result: One pair of randomly chosen layer parameters $W_k$, and $W_k^{-1}$, are robustly updated
        according to the randomized modified target propagation training scheme with exact
        inverses.
Input: $\boldsymbol{h}_0$: one input sample;
for i in range(1,L) do
    Propagate sample forward: $\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(W_i \boldsymbol{h}_{i-1})$ ;
Compute output target: $\hat{\boldsymbol{h}}_L = \boldsymbol{h}_L - \hat{\eta}\frac{\partial L}{\boldsymbol{h}_L}$;
Chose one random layer $k$ from $\{k\}_{k=1}^{L}$ for i in range(L-1,k) do
    Propagate target backwards: $\hat{\boldsymbol{h}}_i = g_i(\hat{\boldsymbol{h}}_{i+1}) = W_{i+1}^{-1} s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1})$ ;
if k>1 then
    Update parameter pair with the robust Sherman-Morrison update;
    $D_{s_k} = \frac{\partial s_k(W_k \boldsymbol{h}_{k-1})}{\partial W_k \boldsymbol{h}_{k-1}}$;
    $\boldsymbol{u}_i = -\eta_k D_{s_k}^{-1}(\boldsymbol{h}_k - \hat{\boldsymbol{h}}_k)$ ;
    $\boldsymbol{v}_i = \boldsymbol{h}_{k-1}$;
    $d = \boldsymbol{v}_i^T W_k^{-1} \boldsymbol{u}_i$;
    if $|1+d| \geq \epsilon$ then
        $W_k \leftarrow W_k + \boldsymbol{u}_i \boldsymbol{v}_i^T$;
        $W_k^{-1} \leftarrow W_k^{-1} - \frac{W_k^{-1}\boldsymbol{u}_i \boldsymbol{v}_i^T W_k^{-1}}{1+d}$;
    else
        $\beta = \frac{1}{\epsilon - d}$;
        $W_k \leftarrow W_k + \beta \boldsymbol{u}_i \boldsymbol{v}_i^T$;
        $W_k^{-1} \leftarrow W_k^{-1} - \frac{W_k^{-1}\boldsymbol{u}_i \boldsymbol{v}_i^T W_k^{-1}}{\epsilon}$;
else
    Update the forward parameters of the first hidden layer;
    $W_1 \leftarrow W_1 - \eta_1 D_{s_1}^{-1}(\boldsymbol{h}_1 - \hat{\boldsymbol{h}}_1)\boldsymbol{h}_0^T$;
```

Algorithm A.7: Error backpropagation with stochastic gradient descent.

```
Result: The network parameters $W_i$, $i = 1,...,L$, are updated according to the error
        backpropagation method in combination with the stochastic gradient descent method with
        mini-batches of one randomly chosen layer are updated by the target propagation method;
Input: $\{\boldsymbol{h}_0^{(b)}\}_{b=1}^{B}$: a mini-batch of input samples;
for i in range(1,L) do
    Propagate mini-batch forward: $\{\boldsymbol{h}_i^{(b)}\}_{b=1}^{B} = \{f_i(\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^{B} = \left\{s_i(\boldsymbol{a}_{i-1}^{(b)})\}_{b=1}^{B} = \{s_i(W_i \boldsymbol{h}_{i-1}^{(b)})\right\}_{b=1}^{B}$ ;
Compute output error: $\{\boldsymbol{\delta}_L^{(b)}\}_{b=1}^{B} = \left\{\frac{\partial L}{\boldsymbol{a}_L^{(b)}}\right\}_{b=1}^{B}$ ;
for i in range(L-1,1) do
    Propagate error signal backwards: $\{\boldsymbol{\delta}_i^{(b)}\}_{b=1}^{B} = \left\{D_{s_i} W_{i+1}^T \boldsymbol{a}_{i+1}^{(b)})\right\}_{b=1}^{B}$ ;
    Update parameters: $\Delta W_i = -\frac{1}{B}\sum_{b=1}^{B} \eta_i \boldsymbol{\delta}_i^{(b)} \boldsymbol{h}_{i-1}^{T,(b)}$;
    $W_i \leftarrow W_i + \Delta W_i$;
```

```
Algorithm A.8: Update backward weights $Q_i$
  Result: The backward weights $Q_i$, $i = 1,...,L-1$, are updated by performing a gradient step on the
          regularized inverse loss $L_i^{inv,r}$, defined in equation (4.97)
  Input: backward learning rates $\eta_i^b$ and regularizing parameters $\lambda_i$ ;
  for i in range(1,L-1) do
      Sample input mini-batch $\{\boldsymbol{h}_i^{(b)}\}_{b=1}^B$ from a white Gaussian distribution ;
      Propagate mini-batch forward: $\{\boldsymbol{h}_{i+1}^{(b)}\}_{b=1}^B = \{f_{i+1}(\boldsymbol{h}_i^{(b)})\}_{b=1}^B$;
      Propagate mini-batch backward: $\{\tilde{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B \{g_i(\boldsymbol{h}_{i+1}^{(b)})\}_{b=1}^B$ ;
      Update backward parameters: $Q_i \leftarrow (1-\eta_i^b\lambda_i)Q_i - \eta_i^b \frac{1}{B}\sum_{b=1}^B \big(\tilde{\boldsymbol{h}}_i^{(b)} - \boldsymbol{h}_i^{(b)}\big)s_{i+1}^{-1}(\boldsymbol{h}_{i+1}^{(b)})^T$
```

```
Algorithm A.9: Update backward weights $Q_i$ in the original TP methods
  Result: The backward weights $Q_i$, $i = 1,...,L-1$, are updated by performing a gradient step on the
          inverse loss $L_i^{inv}$, defined in equation (4.91)
  Input: the layer activations $\{\boldsymbol{h}_i^{(b)}\}_{b=1}^B$, $i = 1,...,L$ and the backward learning rates $\eta_i^b$;
  for i in range(1,L-1) do
      Propagate upper layer mini-batch backward: $\{\tilde{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B \{g_i(\boldsymbol{h}_{i+1}^{(b)})\}_{b=1}^B$ ;
      Update backward parameters;
      $\{D_{s_i}^{(b)}\}_{b=1}^B = \left\{ \frac{\partial s_i(Q_i\boldsymbol{h}_{i+1}^{(b)}}{\partial Q_i \boldsymbol{h}_{i+1}^{(b)}} \right\}_{b=1}^B$  $Q_i \leftarrow (1-\eta_i^b\lambda_i)Q_i - \eta_i^b \frac{1}{B}\sum_{b=1}^B D_{s_i}^{(b)}\big(\tilde{\boldsymbol{h}}_i^{(b)} - \boldsymbol{h}_i^{(b)}\big)(\boldsymbol{h}_{i+1}^{(b)})^T$
```

```
Algorithm A.10: Training iteration of the target propagation method with approximate inverses
  Result: All forward weights $W_i$, $i = 1,...,L$ and all backward weights $Q_i$, $i = 1,...,L-1$ are updated
          by the target propagation method with approximate inverses;
  Input: $\{\boldsymbol{h}_0^{(b)}\}_{b=1}^B$: a mini-batch of input samples;
  for i in range(1,L) do
      Propagate mini-batch forward: $\{\boldsymbol{h}_i^{(b)}\}_{b=1}^B = \{f_i(\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B = \{s_i(W_i\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B$ ;
  Compute output targets: $\{\hat{\boldsymbol{h}}_L^{(b)}\}_{b=1}^B = \left\{ \boldsymbol{h}_L^{(b)} - \hat{\eta}\frac{\partial L}{\boldsymbol{h}_L^{(b)}} \right\}_{b=1}^B$ ;
  for i in range(L-1,1) do
      Propagate target backwards: $\{\hat{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B = \{g_i(\hat{\boldsymbol{h}}_{i+1}^{(b)})\}_{b=1}^B = \{Q_i s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1}^{(b)})\}_{b=1}^B$ ;
      Update forward parameters;
      $\{D_{s_i}^{(b)}\}_{b=1}^B = \left\{ \frac{\partial s_i(W_i\boldsymbol{h}_{i-1}^{(b)})}{\partial W_i \boldsymbol{h}_{i-1}^{(b)}} \right\}_{b=1}^B$ ;
      $\Delta W_i = -\frac{1}{B}\sum_{b=1}^B \eta_i D_{s_i}^{(b)}(\boldsymbol{h}_i^{(b)} - \hat{\boldsymbol{h}}_i^{(b)})\boldsymbol{h}_{i-1}^{T,(b)}$;
      $W_i \leftarrow W_i + \Delta W_i$;
  Update backward parameters conform with algorithm A.8;
```

Algorithm A.11: Training iteration of the randomized target propagation method with approximate inverses

```
Result: The parameters $W_k$ of one randomly chosen layer and all backward weights $Q_i$,
        $i = 1,...,L-1$ are updated by the randomized target propagation method with approximate
        inverses;
Input: $\{\boldsymbol{h}_0^{(b)}\}_{b=1}^B$: a mini-batch of input samples;
for i in range(1,L) do
    Propagate mini-batch forward: $\{\boldsymbol{h}_i^{(b)}\}_{b=1}^B = \{f_i(\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B = \{s_i(W_i\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B$ ;
Compute output targets: $\{\hat{\boldsymbol{h}}_L^{(b)}\}_{b=1}^B = \left\{\boldsymbol{h}_L^{(b)} - \hat{\eta}\frac{\partial L}{\boldsymbol{h}_L^{(b)}}\right\}_{b=1}^B$ ;
Chose one random layer $k$ from $\{k\}_{k=1}^L$ ;
for i in range(L-1,k) do
    Propagate target backwards: $\{\hat{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B = \{g_i(\hat{\boldsymbol{h}}_{i+1}^{(b)})\}_{b=1}^B = \{Q_i s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1}^{(b)})\}_{b=1}^B$ ;
Update forward parameters;
$\{D_{s_k}^{(b)}\}_{b=1}^B = \left\{\frac{\partial s_k(W_k\boldsymbol{h}_{k-1}^{(b)})}{\partial W_k\boldsymbol{h}_{k-1}^{(b)}}\right\}_{b=1}^B$ ;
$\Delta W_k = -\frac{1}{B}\sum_{b=1}^B \eta_k D_{s_k}^{(b)}(\boldsymbol{h}_k^{(b)} - \hat{\boldsymbol{h}}_k^{(b)})\boldsymbol{h}_{k-1}^{T,(b)}$;
$W_k \leftarrow W_k + \Delta W_k$;
Update backward parameters conform with algorithm A.8;
```

Algorithm A.12: Training iteration of the difference target propagation method with approximate inverses

```
Result: All forward weights $W_i$, $i = 1,...,L$ and all backward weights $Q_i$, $i = 1,...,L-1$ are updated
        by the difference target propagation method with approximate inverses;
Input: $\{\boldsymbol{h}_0^{(b)}\}_{b=1}^B$: a mini-batch of input samples;
for i in range(1,L) do
    Propagate mini-batch forward: $\{\boldsymbol{h}_i^{(b)}\}_{b=1}^B = \{f_i(\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B = \{s_i(W_i\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B$ ;
Compute output targets: $\{\hat{\boldsymbol{h}}_L^{(b)}\}_{b=1}^B = \left\{\boldsymbol{h}_L^{(b)} - \hat{\eta}\frac{\partial L}{\boldsymbol{h}_L^{(b)}}\right\}_{b=1}^B$ ;
for i in range(L-1,1) do
    Propagate target backwards;
    $\{\hat{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B = \{\boldsymbol{h}_i^{(b)} + g_i(\hat{\boldsymbol{h}}_{i+1}^{(b)}) - g_i(\boldsymbol{h}_{i+1}^{(b)})\}_{b=1}^B = \{\boldsymbol{h}_i^{(b)} + Q_i s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1}^{(b)}) - Q_i s_{i+1}^{-1}(\boldsymbol{h}_{i+1}^{(b)})\}_{b=1}^B$ ;
    Update forward parameters;
    $\{D_{s_i}^{(b)}\}_{b=1}^B = \left\{\frac{\partial s_i(W_i\boldsymbol{h}_{i-1}^{(b)})}{\partial W_i\boldsymbol{h}_{i-1}^{(b)}}\right\}_{b=1}^B$ ;
    $\Delta W_i = -\frac{1}{B}\sum_{b=1}^B \eta_i D_{s_i}^{(b)}(\boldsymbol{h}_i^{(b)} - \hat{\boldsymbol{h}}_i^{(b)})\boldsymbol{h}_{i-1}^{T,(b)}$;
    $W_i \leftarrow W_i + \Delta W_i$;
Update backward parameters conform with algorithm A.8;
```

Algorithm A.13: Training iteration of the randomized difference target propagation method with approximate inverses

```
Result: The parameters $W_k$ of one randomly chosen layer and all backward weights $Q_i$,
        $i = 1,...,L-1$ are updated by the randomized difference target propagation method with
        approximate inverses;
Input: $\{\boldsymbol{h}_0^{(b)}\}_{b=1}^B$: a mini-batch of input samples;
for i in range(1,L) do
    Propagate mini-batch forward: $\{\boldsymbol{h}_i^{(b)}\}_{b=1}^B = \{f_i(\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B = \{s_i(W_i\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B$ ;
Compute output targets: $\{\hat{\boldsymbol{h}}_L^{(b)}\}_{b=1}^B = \left\{\boldsymbol{h}_L^{(b)} - \hat{\eta}\frac{\partial L}{\boldsymbol{h}_L^{(b)}}\right\}_{b=1}^B$ ;
Chose one random layer $k$ from $\{k\}_{k=1}^L$ ;
for i in range(L-1,k) do
    Propagate target backwards;
    $\{\hat{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B = \{\boldsymbol{h}_i^{(b)} + g_i(\hat{\boldsymbol{h}}_{i+1}^{(b)}) - g_i(\boldsymbol{h}_{i+1}^{(b)})\}_{b=1}^B = \{\boldsymbol{h}_i^{(b)} + \boldsymbol{Q}_i s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1}^{(b)}) - \boldsymbol{Q}_i s_{i+1}^{-1}(\boldsymbol{h}_{i+1}^{(b)})\}_{b=1}^B$ ;
Update forward parameters;
$\{D_{s_k}^{(b)}\}_{b=1}^B = \left\{\frac{\partial s_k(W_k\boldsymbol{h}_{k-1}^{(b)})}{\partial W_k\boldsymbol{h}_{k-1}^{(b)}}\right\}_{b=1}^B$ ;
$\Delta W_k = -\frac{1}{B}\sum_{b=1}^B \eta_k D_{s_k}^{(b)}(\boldsymbol{h}_k^{(b)} - \hat{\boldsymbol{h}}_k^{(b)})\boldsymbol{h}_{k-1}^{T,(b)}$;
$W_k \leftarrow W_k + \Delta W_k$;
Update backward parameters conform with algorithm A.8;
```

Algorithm A.14: Training iteration of the original target propagation method

```
Result: All forward weights $W_i$, $i = 1,...,L$ and all backward weights $Q_i$, $i = 1,...,L-1$ are updated
        by the target propagation method with approximate inverses;
Input: $\{\boldsymbol{h}_0^{(b)}\}_{b=1}^B$: a mini-batch of input samples;
for i in range(1,L) do
    Propagate mini-batch forward: $\{\boldsymbol{h}_i^{(b)}\}_{b=1}^B = \{f_i(\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B = \{s_i(W_i\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B$ ;
Compute output targets: $\{\hat{\boldsymbol{h}}_L^{(b)}\}_{b=1}^B = \left\{\boldsymbol{h}_L^{(b)} - \hat{\eta}\frac{\partial L}{\boldsymbol{h}_L^{(b)}}\right\}_{b=1}^B$ ;
for i in range(L-1,1) do
    Propagate target backwards: $\{\hat{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B = \{g_i(\hat{\boldsymbol{h}}_{i+1}^{(b)})\}_{b=1}^B = \{\boldsymbol{Q}_i s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1}^{(b)})\}_{b=1}^B$ ;
    Update forward parameters;
    $\{D_{s_i}^{(b)}\}_{b=1}^B = \left\{\frac{\partial s_i(W_i\boldsymbol{h}_{i-1}^{(b)})}{\partial W_i\boldsymbol{h}_{i-1}^{(b)}}\right\}_{b=1}^B$ ;
    $\Delta W_i = -\frac{1}{B}\sum_{b=1}^B \eta_i D_{s_i}^{(b)}(\boldsymbol{h}_i^{(b)} - \hat{\boldsymbol{h}}_i^{(b)})\boldsymbol{h}_{i-1}^{T,(b)}$;
    $W_i \leftarrow W_i + \Delta W_i$;
Update backward parameters conform with algorithm A.9;
```

Algorithm A.15: Training iteration of the randomized original target propagation method

```
Result: The parameters $W_k$ of one randomly chosen layer are updated by the randomized target
        propagation method with approximate inverses and all backward weights $Q_i$,
        $i = 1,...,L-1$ are updated;
Input: $\{\boldsymbol{h}_0^{(b)}\}_{b=1}^B$: a mini-batch of input samples;
for i in range(1,L) do
    Propagate mini-batch forward: $\{\boldsymbol{h}_i^{(b)}\}_{b=1}^B = \{f_i(\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B = \{s_i(W_i\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B$ ;
Compute output targets: $\{\hat{\boldsymbol{h}}_L^{(b)}\}_{b=1}^B = \left\{\boldsymbol{h}_L^{(b)} - \hat{\eta}\frac{\partial L}{\boldsymbol{h}_L^{(b)}}\right\}_{b=1}^B$ ;
Chose one random layer $k$ from $\{k\}_{k=1}^L$ ;
for i in range(L-1,k) do
    Propagate target backwards: $\{\hat{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B = \{g_i(\hat{\boldsymbol{h}}_{i+1}^{(b)})\}_{b=1}^B = \{s_i(Q_i\hat{\boldsymbol{h}}_{i+1}^{(b)})\}_{b=1}^B$ ;
Update forward parameters;
$\{D_{s_k}^{(b)}\}_{b=1}^B = \left\{\frac{\partial s_k(W_k\boldsymbol{h}_{k-1}^{(b)})}{\partial W_k\boldsymbol{h}_{k-1}^{(b)}}\right\}_{b=1}^B$ ;
$\Delta W_k = -\frac{1}{B}\sum_{b=1}^B \eta_k D_{s_k}^{(b)}(\boldsymbol{h}_k^{(b)} - \hat{\boldsymbol{h}}_k^{(b)})\boldsymbol{h}_{k-1}^{T,(b)}$;
$W_k \leftarrow W_k + \Delta W_k$;
Update backward parameters conform with algorithm A.9;
```

Algorithm A.16: Training iteration of the original difference target propagation method

```
Result: All forward weights $W_i$, $i = 1,...,L$ and all backward weights $Q_i$, $i = 1,...,L-1$ are updated
        by the original difference target propagation method;
Input: $\{\boldsymbol{h}_0^{(b)}\}_{b=1}^B$: a mini-batch of input samples;
for i in range(1,L) do
    Propagate mini-batch forward: $\{\boldsymbol{h}_i^{(b)}\}_{b=1}^B = \{f_i(\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B = \{s_i(W_i\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B$ ;
Compute output targets: $\{\hat{\boldsymbol{h}}_L^{(b)}\}_{b=1}^B = \left\{\boldsymbol{h}_L^{(b)} - \hat{\eta}\frac{\partial L}{\boldsymbol{h}_L^{(b)}}\right\}_{b=1}^B$ ;
for i in range(L-1,1) do
    Propagate target backwards;
    $\{\hat{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B = \{\boldsymbol{h}_i^{(b)} + g_i(\hat{\boldsymbol{h}}_{i+1}^{(b)}) - g_i(\boldsymbol{h}_{i+1}^{(b)})\}_{b=1}^B = \{\boldsymbol{h}_i^{(b)} + s_i(Q_i\hat{\boldsymbol{h}}_{i+1}^{(b)}) - s_i(Q_i\boldsymbol{h}_{i+1}^{(b)})\}_{b=1}^B$ ;
    Update forward parameters;
    $\{D_{s_i}^{(b)}\}_{b=1}^B = \left\{\frac{\partial s_i(W_i\boldsymbol{h}_{i-1}^{(b)})}{\partial W_i\boldsymbol{h}_{i-1}^{(b)}}\right\}_{b=1}^B$ ;
    $\Delta W_i = -\frac{1}{B}\sum_{b=1}^B \eta_i D_{s_i}^{(b)}(\boldsymbol{h}_i^{(b)} - \hat{\boldsymbol{h}}_i^{(b)})\boldsymbol{h}_{i-1}^{T,(b)}$;
    $W_i \leftarrow W_i + \Delta W_i$;
Update backward parameters conform with algorithm A.9;
```

Algorithm A.17: Training iteration of randomized difference target propagation with approximate inverses

```
Result: The parameters $W_k$ of one randomly chosen layer and all backward weights $Q_i$,
        $i = 1,...,L-1$ are updated by the randomized original difference target propagation
        method ;
Input: $\{\boldsymbol{h}_0^{(b)}\}_{b=1}^B$: a mini-batch of input samples;
for i in range(1,L) do
    Propagate mini-batch forward: $\{\boldsymbol{h}_i^{(b)}\}_{b=1}^B = \{f_i(\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B = \{s_i(W_i\boldsymbol{h}_{i-1}^{(b)})\}_{b=1}^B$ ;
Compute output targets: $\{\hat{\boldsymbol{h}}_L^{(b)}\}_{b=1}^B = \left\{\boldsymbol{h}_L^{(b)} - \hat{\eta}\frac{\partial L}{\boldsymbol{h}_L^{(b)}}\right\}_{b=1}^B$ ;
Chose one random layer $k$ from $\{k\}_{k=1}^L$ ;
for i in range(L-1,k) do
    Propagate target backwards;
    $\{\hat{\boldsymbol{h}}_i^{(b)}\}_{b=1}^B = \{\boldsymbol{h}_i^{(b)} + g_i(\hat{\boldsymbol{h}}_{i+1}^{(b)}) - g_i(\boldsymbol{h}_{i+1}^{(b)})\}_{b=1}^B = \{\boldsymbol{h}_i^{(b)} + s_i(Q_i\hat{\boldsymbol{h}}_{i+1}^{(b)}) - s_i(Q_i\boldsymbol{h}_{i+1}^{(b)})\}_{b=1}^B$ ;
Update forward parameters;
$\{D_{s_k}^{(b)}\}_{b=1}^B = \left\{\frac{\partial s_k(W_k\boldsymbol{h}_{k-1}^{(b)})}{\partial W_k\boldsymbol{h}_{k-1}^{(b)}}\right\}_{b=1}^B$ ;
$\Delta W_k = -\frac{1}{B}\sum_{b=1}^B \eta_k D_{s_k}^{(b)}(\boldsymbol{h}_k^{(b)} - \hat{\boldsymbol{h}}_k^{(b)})\boldsymbol{h}_{k-1}^{T,(b)}$;
$W_k \leftarrow W_k + \Delta W_k$;
Update backward parameters conform with algorithm A.9;
```

Appendix B

Proofs

B.1 Theorem 4.1

Theorem. *Consider a feed-forward neural network with forward mapping function* $\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(W_i \boldsymbol{h}_{i-1})$, $i = 1,...,L$ *where* s_i *can be any differentiable, monotonically increasing and invertible element-wise function. Assume that the backward mapping functions* g_i*, used for propagating the target activations, are the exact inverses of* f_{i+1}*. Let* ΔW_i^{tp} *and* ΔW_i^{bp} *be the target propagation update and the back-propagation update in the i-th layer, respectively. If* $\hat{\eta}$ *in equation* (4.4) *is taken in limit to zero* ($\hat{\eta} \to 0+$)*, then the angle* α *between* ΔW_i^{tp} *and* ΔW_i^{bp} *is bounded by*

$$0 < cos(\alpha) \leq 1, \tag{B.1}$$

indicating a descent direction of ΔW_i^{tp}*, as* ΔW_i^{bp} *points in the opposite direction of the gradient.*

Proof. In order to prove that $0 < cos(\alpha) \leq 1$, it is necessary and sufficient that the inner product between the vectorized updates $vec(\Delta W_i^{tp})$ and $vec(\Delta W_i^{bp})$ is strictly positive for sufficiently small $\hat{\eta}$. Given a training example $(\boldsymbol{x}, \boldsymbol{y})$, the back-propagation update is given by

$$\Delta W_i^{bp} = -\eta_i D_{s_i} \left(\left[\prod_{k=i}^{L-1} J_{k+1}^T \right] \boldsymbol{e}_L \right) \boldsymbol{h}_{i-1}^T, \tag{B.2}$$

with $J_k = \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{h}_{k-1}} = D_{s_k} W_k$, as defined in equation 4.14. By means of a first order Taylor expansion, the target propagation update can be written as

$$\Delta W_i^{tp} = -\eta_i D_{s_i} \left(\hat{\eta} \left[\prod_{k=i}^{L-1} J_{k+1}^{-1} \right] \boldsymbol{e}_L + \mathcal{O}(\hat{\eta}^2) \right) \boldsymbol{h}_{i-1}^T, \tag{B.3}$$

similar to equation (4.12). For ease of notation, let us define $J = \prod_{k=L-1}^{i} J_{k+1}$ and $\boldsymbol{h} = \boldsymbol{h}_{i-1}$. Now the inner product of $vec(\Delta W_i^{tp})$ and $vec(\Delta W_i^{bp})$ can be written as follows:

$$\begin{aligned} < vec(\Delta W_i^{tp}), vec(\Delta W_i^{bp}) > &= \eta_i^2 \operatorname{Tr}\left(\left(D_{s_i} J^T \boldsymbol{e}_L \boldsymbol{h}^T \right)^T \left(\hat{\eta} D_{s_i} J^{-1} \boldsymbol{e}_L \boldsymbol{h}^T \right) + D_{s_i} \mathcal{O}(\hat{\eta}^2) \boldsymbol{h}^T \right) && \text{(B.4)} \\ &= \eta_i^2 \left(\hat{\eta} \operatorname{Tr}\left(\boldsymbol{h} \boldsymbol{e}_L^T J D_{s_i}^2 J^{-1} \boldsymbol{e}_L \boldsymbol{h}^T \right) + \operatorname{Tr}\left(\boldsymbol{h} \boldsymbol{e}_L^T J D_{s_i}^2 \mathcal{O}(\hat{\eta}^2) \boldsymbol{h}^T \right) \right). && \text{(B.5)} \end{aligned}$$

The first term of equation (B.5) is scaled by $\mathcal{O}(\hat{\eta})$, whereas the second term is scaled by $\mathcal{O}(\hat{\eta}^2)$. With $\hat{\eta}$ taken in limit to zero, the first term will thus dominate the expression and determine the sign of the inner product. For the proof, it is therefore sufficient to show that the first term is always strictly positive for $\boldsymbol{h} \neq \boldsymbol{0}$ and $\boldsymbol{e}_L \neq \boldsymbol{0}$. As s_i is an invertible element-wise function, $D_{s_i}^2$ is a diagonal matrix with strictly positive entries and thus with strictly positive eigenvalues. $J D_{s_i}^2 J^{-1}$ can be seen as a similarity transform of $D_{s_i}^2$, which means that $J D_{s_i}^2 J^{-1}$ has the same eigenvalues as $D_{s_i}^2$ and is positive definite. Hence, the first term of (B.5) can be written as

$$\hat{\eta} \operatorname{Tr}\left(\boldsymbol{h} \boldsymbol{e}_L^T J D_{s_i}^2 J^{-1} \boldsymbol{e}_L \boldsymbol{h}^T \right) = \hat{\eta} c \|\boldsymbol{h}\|_2^2, \quad c > 0. \tag{B.6}$$

This proves that the inner product is strictly positive for $\boldsymbol{h} \neq \boldsymbol{0}$ and $\boldsymbol{e}_L \neq \boldsymbol{0}$ and $\hat{\eta}$ taken in limit to zero. □

B.2 Lemma 4.4

Lemma. *Consider a feed-forward neural network with forward mapping $f_i(\boldsymbol{h}_{i-1}) = s_i(W_i\boldsymbol{h}_{i-1})$ and backward mapping $g_i(\boldsymbol{h}_{i+1}) = Q_i s_{i+1}^{-1}(\boldsymbol{h}_{i+1})$. In this network setting, the expected gradient $\mathbb{E}\left[\nabla_{Q_i} L_i^{inv}\right]$ with L_i^{inv} as specified in equation* (4.91) *is equal to*

$$\mathbb{E}\left[\nabla_{Q_i} L_i^{inv}\right] = 2\left(Q_i W_{i+1} - I\right)\Gamma_i W_{i+1}^T, \tag{B.7}$$

with Γ_i the covariance matrix of $\boldsymbol{h}_i$.

Proof.

$$\begin{aligned}
L_i^{inv}\Big(g_i\big(f_{i+1}(\boldsymbol{h}_i)\big), \boldsymbol{h}_i\Big) &= \|g_i\big(f_{i+1}(\boldsymbol{h}_i)\big) - \boldsymbol{h}_i\|_2^2 && \text{(B.8)}\\
&= \|Q_i\Big(s_{i+1}^{-1}\Big(s_{i+1}\big(W_{i+1}\boldsymbol{h}_i\big)\Big)\Big) - \boldsymbol{h}_i\|_2^2 && \text{(B.9)}\\
&= \|Q_i W_{i+1}\boldsymbol{h}_i - \boldsymbol{h}_i\|_2^2 && \text{(B.10)}\\
&= \mathrm{Tr}\Big(\big(Q_i W_{i+1}\boldsymbol{h}_i - \boldsymbol{h}_i\big)\big(Q_i W_{i+1}\boldsymbol{h}_i - \boldsymbol{h}_i\big)^T\Big) && \text{(B.11)}\\
&= \mathrm{Tr}\Big(\boldsymbol{h}_i\boldsymbol{h}_i^T\Big) + \mathrm{Tr}\Big(Q_i W_{i+1}\boldsymbol{h}_i\boldsymbol{h}_i^T W_{i+1}^T Q_i^T\Big) - 2\,\mathrm{Tr}\Big(Q_i W_{i+1}\boldsymbol{h}_i\boldsymbol{h}_i^T\Big), && \text{(B.12)}
\end{aligned}$$

with Tr the matrix trace. Taking the gradient of the above equation with respect to Q_i results in

$$\nabla_{Q_i} L_i^{inv} = 2\left(Q_i W_{i+1} - I\right)\boldsymbol{h}_i\boldsymbol{h}_i^T W_{i+1}^T \tag{B.13}$$

Now taking the expected value of the above equation, we get our final result:

$$\begin{aligned}
\mathbb{E}\Big[\nabla_{Q_i} L_i^{inv}\Big] &= 2\left(Q_i W_{i+1} - I\right)\mathbb{E}\Big[\boldsymbol{h}_i\boldsymbol{h}_i^T\Big] W_{i+1}^T && \text{(B.14)}\\
&= 2\left(Q_i W_{i+1} - I\right)\Gamma_i W_{i+1}^T && \text{(B.15)}
\end{aligned}$$

□

B.3 Lemma 4.5

Lemma. *Consider a feed-forward neural network with forward mapping $f_i(\boldsymbol{h}_{i-1}) = s_i(W_i\boldsymbol{h}_{i-1})$ and backward mapping $g_i(\boldsymbol{h}_{i+1}) = Q_i s_{i+1}^{-1}(\boldsymbol{h}_{i+1})$. If $\boldsymbol{h}_i$ is uncorrelated and has equal variance σ^2, the expectation of the inverse loss function L_i^{inv}, defined in equation* (4.91) *can be written as the following decoupled energy function $E(Q_i)$ depending on w_j and q_k, the columns of W_{i+1} and the rows of Q_i respectively:*

$$E(Q_i) = \sigma^2 \sum_j \left[\left(q_j^T w_j - 1\right)^2 + \sum_{k\neq j}\left(q_k^T w_j\right)^2\right]. \tag{B.16}$$

Proof. This proof follows a similar approach to the work of Saxe et al. [111], but tailored towards the target propagation method. If $\boldsymbol{h}_i$ is uncorrelated and has equal variance, its covariance matrix Γ_i can be written as a multiple of the identity matrix:

$$\Gamma_i = \sigma^2 I. \tag{B.17}$$

Following lemma 4.4, the expected gradient of L_i^{inv} is now defined by:

$$\mathbb{E}\left[\nabla_{Q_i} L_i^{inv}\right] = 2\sigma^2\left(Q_i W_{i+1} - I\right)W_{i+1}^T \tag{B.18}$$

The gradient with respect to a single row q_j of Q_i can be expressed as:

$$\mathbb{E}\left[\nabla_{q_j} L_i^{inv}\right] = 2\sigma^2\big((q_j^T w_j - 1)w_j + \sum_{k\neq j}(q_j^T w_k)w_k\big) \tag{B.19}$$

This gradient can be seen as the gradient of the following decoupled energy function of Q_i, thereby proving the lemma:

$$E(Q_i) = \sigma^2 \sum_j \left[\left(q_j^T w_j - 1\right)^2 + \sum_{k\neq j}\left(q_k^T w_j\right)^2\right]. \tag{B.20}$$

□

B.4 Theorem 4.6

Theorem. *Consider a feed-forward neural network with forward mapping $f_i(\boldsymbol{h}_{i-1}) = s_i(W_i \boldsymbol{h}_{i-1})$ and backward mapping $g_i(\boldsymbol{h}_{i+1}) = Q_i s_{i+1}^{-1}(\boldsymbol{h}_{i+1})$. If W_{i+1} is square and of full rank, the minimization of the inverse loss L_i^{inv}, defined in equation* (4.91) *leads in expectation towards the unique solution*

$$Q_i^* = W_{i+1}^{-1}, \tag{B.21}$$

if and only if Γ_i, the covariance matrix of $\boldsymbol{h}_i$, is of full rank.
If W_{i+1} is not square, the minimization of the inverse loss L_i^{inv}, defined in equation (4.91) *leads in expectation towards the unique solution*

$$Q_i^* = W_{i+1}^{\dagger}, \tag{B.22}$$

with $W_{i+1}^{\dagger}$ the Moore-Penrose pseudo-inverse of W_{i+1}[82, 83], if and only if W_{i+1} has linearly independent rows and if h_i is uncorrelated and has equal variances different from zero.

Proof. With the help of lemma 4.4, the optimality conditions for Q_i^* in order to find a minimum of L_i^{inv}, can be written as:

$$0 := \mathbb{E}\left[\nabla_{Q_i^*} L_i^{inv}\right] = 2\big(Q_i W_{i+1} - I\big)\Gamma_i W_{i+1}^T \tag{B.23}$$

$$\Leftrightarrow Q_i^* W_{i+1} \Gamma_i W_{i+1}^T = \Gamma_i W_{i+1}^T, \tag{B.24}$$

First the case of W_{i+1} square and of full rank is considered. If Γ_i is of full rank, equation (B.24) has a unique solution

$$Q_i^* = \Gamma_i W_{i+1}^T \big(W_{i+1} \Gamma_i W_{i+1}^T\big)^{-1} = W_{i+1}^{-1} \tag{B.25}$$

If Γ_i is not of full rank, $W_{i+1}\Gamma_i W_{i+1}^T$ is singular and consequently has a null space of dimension greater than zero, indicating that there exist multiple solutions of Q_i^* (hence no unique solution exist). Hereby the sufficiency and necessity of the condition on the rank of Γ_i is proven for the case of W_{i+1} square and of full rank.

Now the second case of W_{i+1} non-square is considered. If h_i is uncorrelated and has equal variances different from zero, its correlation matrix can be written as $\Gamma_i = \sigma^2 I$. Therefore, equation (B.24) can be rewritten as:

$$Q_i^* W_{i+1} W_{i+1}^T = W_{i+1}^T, \tag{B.26}$$

If W_{i+1} has linearly independent rows, $W_{i+1} W_{i+1}^T$ is invertible and there thus exist one unique solution for Q_i^* equal to:

$$Q_i^* = W_{i+1}^T \big(W_{i+1} W_{i+1}^T\big)^{-1} = W_{i+1}^{\dagger}, \tag{B.27}$$

with $W_{i+1}^{\dagger}$ the Moore-Penrose pseudo-inverse of W_{i+1} [82, 83, 112]. This proves the sufficiency of the conditions in the theorem. If W_{i+1} has linearly dependent rows, $W_{i+1} W_{i+1}^T$ is singular and there thus exist multiple solutions for Q_i^* of equation B.26, proving the necessity of the condition regarding the linear independence of the rows of W_{i+1}. If $\boldsymbol{h}_i$ is not uncorrelated and/or not of equal variance, Γ_i cannot be written as the multiple of the identity matrix. If Γ_i or $W_{i+1}\Gamma_i W_{i+1}^T$ is singular, the solution for Q_i^* of equation (B.24) is not unique and if $W_{i+1}\Gamma_i W_{i+1}^T$ is of full rank, the unique solution is $Q_i^* = \Gamma_i W_{i+1}^T \big(W_{i+1}\Gamma_i W_{i+1}^T\big)^{-1}$, which is only equal to $W_{i+1}^{\dagger}$ if Γ_i can be written as a multiple of the identity matrix or if W_{i+1} is of full rank. This concludes the necessity of the condition regarding that $\boldsymbol{h}_i$ is uncorrelated and has equal variance.

□

B.5 Theorem 4.7

Theorem. *Consider a feed-forward neural network with forward mapping $f_i(\boldsymbol{h}_{i-1}) = s_i(W_i \boldsymbol{h}_{i-1})$ and backward mapping $g_i(\boldsymbol{h}_{i+1}) = Q_i s_{i+1}^{-1}(\boldsymbol{h}_{i+1})$. If and only if h_i is uncorrelated and has equal variances σ^2, the* *minimization of the inverse loss $L_i^{inv,r}$, defined in equation* (4.97) *leads in expectation towards the unique solution*

$$Q_i^* = W_{i+1}^T \big(W_{i+1} W_{i+1}^T + \frac{\lambda}{\sigma^2} I\big)^{-1}. \tag{B.28}$$

When the weight-decay parameter λ is driven in limit to zero, this results in the unique solution

$$\lim_{\lambda \to 0} Q_i^* = W_{i+1}^\dagger \tag{B.29}$$

with $W_{i+1}^\dagger$ the Moore-Penrose pseudo-inverse of W_{i+1} [82, 83].

Proof. As the gradient operator is linear, the gradient of the extra regularizer term $\lambda \|Q_i\|_F^2$ can be added with the expected gradient of L_i^{inv}. Together with lemma 4.4, this results in:

$$\mathbb{E}\left[\nabla_{Q_i} L_i^{inv,r}\right] = 2\big(Q_i W_{i+1} - I\big)\Gamma_i W_{i+1}^T + 2\lambda Q_i. \tag{B.30}$$

Requiring that $E\left[\nabla_{Q_i} L_i^{inv}\right] = 0$ gives the following optimality condition for Q_i:

$$Q_i^* \big(W_{i+1}\Gamma_i W_{i+1}^T + \lambda I\big) = \Gamma_i W_{i+1}^T. \tag{B.31}$$

As Γ_i is a covariance matrix, it is always positive semi-definite and symmetric. $W_{i+1}\Gamma_i W_{i+1}^T$ can thus be written as $\tilde{W}_{i+1}\tilde{W}_{i+1}^T$ with $\tilde{W}_{i+1} = W_{i+1}\Gamma_i^{\frac{1}{2}}$, proving that it is also positive semi-definite. Adding a positive multiple of the identity matrix to a positive semi-definite matrix makes it positive definite (trivial proof via the spectral decomposition of the positive semi-definite matrix). Therefore, $\big(W_{i+1}\Gamma_i W_{i+1}^T + \lambda I\big)$ is invertible and Q_i^* has a unique solution:

$$Q_i^* = \Gamma_i W_{i+1}^T \big(W_{i+1}\Gamma_i W_{i+1}^T + \lambda I\big)^{-1}. \tag{B.32}$$

If and only if Γ_i can be written as a multiple of the identity matrix $\sigma^2 I$, this expression is equal to:

$$Q_i^* = W_{i+1}^T \big(W_{i+1} W_{i+1}^T + \frac{\lambda}{\sigma^2} I\big)^{-1}. \tag{B.33}$$

The condition on Γ_i implies that h_i needs to be uncorrelated and of equal variances, thereby proving equation (B.33) and its necessary and sufficient condition. The last part of the theorem can be proven via the singular value decomposition (SVD) of $W_{i+1} = U\Sigma V^T$. Based on this SVD, equation (B.33) can be rewritten as:

$$\begin{aligned}
Q_i^* &= V\Sigma^T U^T \big(U\Sigma\Sigma^T U^T + \frac{\lambda}{\sigma^2} UU^T\big)^{-1} && \text{(B.34)}\\
&= V\Sigma^T \big(\Sigma\Sigma^T + \frac{\lambda}{\sigma^2} I\big)^{-1} U^T && \text{(B.35)}\\
&= V\tilde{\Sigma}^T U^T, && \text{(B.36)}
\end{aligned}$$

with $\tilde{\Sigma}$ of the same rectangular diagonal structure as Σ and with diagonal elements $\tilde{\sigma}_k$:

$$\tilde{\sigma}_k = \frac{\sigma_k}{\sigma_k^2 + \frac{\lambda}{\sigma^2}}, \tag{B.37}$$

with σ_k the singular values of W_{i+1}. For the limit of λ to zero, $\lim_{\lambda \to 0} \tilde{\sigma}_k = \sigma_k^{-1}$ if $\sigma_k \neq 0$ and $\lim_{\lambda \to 0} \tilde{\sigma}_k = 0$ if $\sigma_k = 0$. This implies that

$$\begin{aligned}
\lim_{\lambda \to 0} Q_i^* &= V\Sigma^\dagger U^T && \text{(B.38)}\\
&= W_{i+1}^\dagger, && \text{(B.39)}
\end{aligned}$$

thereby concluding the proof. □

B.6 Lemma 4.8

Lemma. *Consider a feed-forward network with L layers and with as forward mapping function* $\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(W_i\boldsymbol{h}_{i-1})$*,* $i = 1,...,L$ *where* s_i *can be any differentiable, monotonically increasing and invertible element-wise function. The Moore-Penrose pseudo-inverse of all Jacobians* $J_i = \frac{\partial \boldsymbol{h}_L}{\partial \boldsymbol{h}_i} = \prod_{k=L}^{i+1} D_{s_k} W_k$*,* $i = 1,...,L-1$*, can be factorized as* $J_i^\dagger = \prod_{k=i+1}^{L} W_k^\dagger D_{s_k}^\dagger$ *if and only if* $n_L = n_{L-1} = ... = n_2$ *and* $n_2 \leq n_1$*, with* n_i *the dimension of the i-th layer,* W_i *is of full rank for* $i = 3,...,L-1$ *and* W_2 *is of full row rank.*

Proof. $(AB)^\dagger = B^\dagger A^\dagger$ if one of the following conditions hold [112]:

- A has orthonormal columns
- B has orthonormal rows
- $B = A^*$ (B is the conjugate transpose of A)
- A has all columns linearly independent and B has all rows linearly independent

In general, the matrices in J_i do not have the first three properties, thus the fourth condition should hold for every factorization made in J_i.
In order to prove this lemma, we will follow a recursive approach. The key condition is that the pseudo-inverse of all Jacobians J_i, $i = 1,...,L-1$ need to be factorizable. The conditions imposed on the layer dimensions by factorizing $J_i^\dagger$ should thus also hold while factorizing all other $J_{j\neq i}^\dagger$. We start with J_{L-1}:

$$J_{L-1}^\dagger = \left(D_{s_L} W_L\right)^\dagger \tag{B.40}$$

D_{s_i} are always square, diagonal and of full rank, as s_i is a differentiable and invertible element-wise function, hence the fourth condition automatically holds for it. Throughout the rest of this proof, we will assume that every matrix with #rows ≥ #cols has full column rank and every matrix with #rows ≤ #cols has full row rank. So for the fourth condition to hold for W_L, it must be true that:

$$n_L \leq n_{L-1}. \tag{B.41}$$

Let's now consider J_{L-2}.

$$J_{L-2}^\dagger = \left(D_{s_L} W_L D_{s_{L-1}} W_{L-1}\right)^\dagger \tag{B.42}$$

As we now have 4 matrices, we will have to make 3 successive splits of the matrix product in 2 parts, for which condition four has to hold always for both parts of the split. Regardless of which split scheme is used, eventually there has to be a split at the right side of W_L and another one at the left side of W_{L-1}, leading to the conditions:

$$n_L \geq n_{L-1} \tag{B.43}$$

$$n_{L-1} \leq n_{L-2} \tag{B.44}$$

From equation (B.41) and (B.43) it follows that $n_L = n_{L-1}$. Thus $D_{s_L} W_L D_{s_{L-1}}$ can be written as a square matrix D. Together with equation (B.44), the argument can now be repeated for $J_i^\dagger$, $i = L-3,...,1$, leading to the conditions that $n_L = n_{L-1} = ... = n_2$ and $n_2 \leq n_1$. □

B.7 Theorem 4.9

Theorem. *Consider a feed-forward neural network with as forward mapping function* $\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(W_i\boldsymbol{h}_{i-1})$*,* $i = 1,...,L$ *where* s_i *can be any differentiable, monotonically increasing and invertible element-wise function. Take the backward mapping functions , used for propagating the target activations, equal to* $g_i(\hat{\boldsymbol{h}}_{i+1}) = Q_i s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1})$ *and assume that after each forward weight update, they are trained until optimality with white noise* $\boldsymbol{h}_i$ *and loss function* $L_i^{inv,r}$ *as defined in equation (4.97) with* $\lambda \to 0$ *in the limit. Furthermore assume a mini-batch size of 1, a sufficiently small output step size* $\hat{\eta}$ *and an* L_2 *output loss function. Finally, assume that* $n_L = n_{L-1} = ... = n_2$ *and* $n_2 \leq n_1$*, with* n_i *the dimension of the i-th layer, that* W_i *is of full rank for* $i = 3,...,L-1$ *and* W_2 *is of full row rank. Under these conditions, difference target propagation approximately uses Gauss-Newton optimization with a block-diagonal approximation of the Gauss-Newton Hessian to compute the local layer targets* $\hat{\boldsymbol{h}}_i$*.*

Proof. Under the conditions assumed in this theorem, the Gauss-Newton optimization step for the layer activations, with a block-diagonal approximation of the Gauss-Newton Hessian matrix with blocks equal to the layer sizes, is given by (a result of lemma 4.2):

$$\Delta \boldsymbol{h}_i = -\boldsymbol{J}_i^{\dagger} \boldsymbol{e}, \quad i = 1, ..., L-1, \tag{B.45}$$

with J_i defined as:

$$J_i = \prod_{k=L}^{i+1} \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{a}_k} \frac{\partial \boldsymbol{a}_k}{\partial \boldsymbol{h}_{k-1}} \tag{B.46}$$

$$J_i = \prod_{k=L}^{i+1} D_{s_k} W_k, \tag{B.47}$$

If the network has a structure as specified by lemma 4.8 ($n_L = n_{L-1} = ... = n_2$ and $n_2 \leq n_1$, with n_i the dimension of the i-th layer, W_i is of full rank for $i = 3, ..., L-1$ and W_2 is of full row rank), the pseudo-inverse can be factorized over the decomposed $\boldsymbol{J}_i$, leading to the following update for $\Delta \boldsymbol{h}_i$:

$$\Delta \boldsymbol{h}_i = -\Big(\prod_{k=i+1}^{L} W_k^{\dagger} D_{s_k}^{\dagger} \Big) \boldsymbol{e}. \tag{B.48}$$

Note that D_k are diagonal square matrices and that s_i are invertible non-linearities, thus $D_k^{\dagger} = D_k^{-1}$. Now lets define the layer target $\hat{\boldsymbol{h}}_i^{GN}$ as the updated layer activation:

$$\hat{\boldsymbol{h}}_i^{GN} = \boldsymbol{h}_i + \Delta \boldsymbol{h}_i. \tag{B.49}$$

Due to the block-diagonal approximation, it is common practice in the field to use an optimal step size $\hat{\eta}$ for the parameter update, leading to:

$$\hat{\boldsymbol{h}}_i^{GN} = \boldsymbol{h}_i + \hat{\eta} \Delta \boldsymbol{h}_i. \tag{B.50}$$

$$= \boldsymbol{h}_i - \hat{\eta} \Big(\prod_{k=i+1}^{L} W_k^{\dagger} D_{s_k}^{-1} \Big) \boldsymbol{e} \tag{B.51}$$

As shown in equation (4.102), the propagated targets $\hat{\boldsymbol{h}}_i$ by difference target propagation can be approximated with a first order Taylor expansion around $\boldsymbol{h}_i$ as follows:

$$\hat{\boldsymbol{h}}_i^{DTP} = \boldsymbol{h}_i - \hat{\eta} \Big[\prod_{k=i}^{L-1} J_{g_k} \Big] \boldsymbol{e}_L + \mathcal{O}(\hat{\eta}^2), \tag{B.52}$$

with $J_{g_k} = \frac{\partial g_k(\boldsymbol{h}_{k+1})}{\partial \boldsymbol{h}_{k+1}} = Q_i D_{s_{k+1}}^{-1}$. As shown by theorem 4.7, $Q_i = W_{i+1}^{\dagger}$ if Q_i is trained with white noise until optimality with loss function $L_i^{inv,r}$ and $\lambda \to 0$ in the limit. Therefore, under the assumptions of this theorem, the propagated targets $\hat{\boldsymbol{h}}_i$ by difference target propagation can be expressed as:

$$\hat{\boldsymbol{h}}_i^{DTP} = \boldsymbol{h}_i - \hat{\eta} \Big[\prod_{k=i+1}^{L} W_i^{\dagger} D_{s_k}^{-1} \Big] \boldsymbol{e}_L + \mathcal{O}(\hat{\eta}^2), \tag{B.53}$$

We see that $\hat{\boldsymbol{h}}_i^{GN}$ and $\hat{\boldsymbol{h}}_i^{DTP}$ are approximately equal with an error of $\mathcal{O}(\hat{\eta}^2)$ in equations (B.51) and (B.53) respectively, thereby proving the theorem. $\square$

Appendix C

PyProp: a neural network toolbox for alternatives to backpropagation

C.1 Introduction

PyProp is a Python based toolbox for creating and training multilayer perceptron models with alternatives to backpropagation. It is based on the PyTorch framework and defines its own types of layers and networks (so it does not use the layers and networks of torch.nn) that are easily customizable to training methods other than backpropagation. Currently, backpropagation and target propagation with both exact inverses as approximate inverses are implemented, but it can easily be extended to other methods, as long as there is a forward propagation and a backward propagation of training signals. This document serves as a brief introduction to PyProp, a more detailed documentation is still needed. It starts with explaining the structure of the PyProp package and ends with describing the typical training flow of models defined in PyProp. The toolbox can be found at `https://github.com/AlexanderMeulemans/PyProp`.

C.2 Structure of the toolbox

The toolbox can be structured in three main parts:

1. layers
2. networks
3. optimizers

A layer defines a single layer of a multi-layer perceptron model, a network consists out of multiple layer objects and defines a multi-layer perceptron model and an optimizer performs a training procedure on a network with a specified dataset.

UML diagrams. In the Github repository of PyProp, you can find a folder containing UML-diagrams of the different layers, networks and optimizers. `network_models.png` gives an overview of the inheritance of all the layers and networks. The other diagrams give a more detailed view of each module.

C.3 PyProp in action

In this section, the practical aspects of how to use PyProp for your own research are discussed.

C.3.1 Training a network in PyProp

Creating the network The first step is of course creating your network that you want to train. For this, you need first to create the layers of your network.

- If you want to train your network with backpropagation, you should use the child objects of the class `Layer`. The first layer of your network should be of the type `InputLayer` and the last layer should be a child of the class `OutputLayer`. The hidden layers can be all other children of `Layer`.

- If you want to create an invertible network and train it with target propagation with exact inverses, you should use the children of the class `InvertibleLayer` to create layers. Similarly, the input and output layer must be of types `InvertibleInputLayer` and `InvertibleOutputLayer` respectively.
- You can also create new types of layers to train with other methods such as difference target propagation. These can be created from other layer types with a limited amount of adjustments, see at the end of section C.3 for more information.

After all the needed layers are created, you can put them in a list in the right order and create a network with it. Use the `Network` class for this if your layers are children of `Layer` and use the `InvertibleNetwork` class for this if you have invertible layers.

Optimizer After you have created the network, you can train it by using an optimizer. Currently, stochastic gradient descent (SGD) and SGD with momentum are both implemented, however, the SGD with momentum currently only works with backpropagation layers. The optimizers also have methods to automatically train it on MNIST or a toy example dataset (see in the experiment folder for some examples on how to use it). Under the hood of the optimizer, the following actions are taken during each training step.

1. **Forward propagation:** The input mini-batch is propagated forward through all the layers. This is done by the method `propagateForward()`.
2. **Backward propagation:** At the output layer, the loss of the model output relative to the target values is computed, and teaching signals are propagated backwards through the network. With backpropagation, these teaching signals are the error signals, with target propagation variants, the teaching signals are the local layer targets. The backwards propagation of teaching signals is done by the method `propagateBackward()`
3. **Compute the gradients:** After the teaching signals are propagated, the weight gradients can be computed. Note that the term gradient refers to the gradient of the output loss with respect to the weights for backpropagation, and to the gradient of the local loss functions with respect to the local weights for the target propagation variants. The gradients are computed by the method `computeGradients`. For both the backpropagation layers and the invertible target propagation layers, there are only forward parameters that need to be updated, thus only gradients of the forward parameters are computed. In the future, also the backward parameters need to be trained for pure target propagation and difference target propagation.
4. **Update the parameters:** Now that the parameter gradients are computed, the parameters can be updated according to the specified learning rate. This is done by the method `updateForwardParameters` and the coordinating method `updateParameters`.

Visualize results The PyProp package uses TensorboardX to automatically visualize the training of the multilayer networks. At the beginning of the experiment, you should make a `SummaryWriter` object and specify the log directory to save the results to. This writer should be passed to all layers as an argument. Currently, the following results are visualized:

- After each mini-batch:
  - norm of the weights and biases of each layer
  - norm of the activations of each layer
  - norm of the gradients of the wieghts and biases of each layer
  - mini-batch training loss
  - the error of the inverse computation of the forward weights with the Sherman-Morrison trick
  - the norm of the reconstruction errors
- After each epoch:
  - the epoch test loss
  - the epoch training loss
  - histogram of the current weights and biases
  - histogram of the layer activations for the last mini-batch
  - histogram of the gradients for the last mini-batch

C.3.2 Create new layer types

If you want to investigate a new training method for neural networks, you should create a new layer type and a new corresponding network type (if needed).

New layer. For target propagation variants, you should make a new type of layer that inherits from `BidirectionalLayer` and that overwrites the following methods:

- `propagateBackward()`
- `computeForwardGradients()`
- `updateBackwardParameters()`
- `computeBacwardOutput()`

If the backward weights need to be trained, the method `updateBackwardParameters()` should use a new method `computeBackwardGradients()` to compute the direction in which the weights need to be changed (in order to be in line with the forward weights update scheme). Besides creating the new coordinating layer, you should also create children of it with specific nonlinearities and an input and output layer type.

New network. Normally, you can just use the `BidirectionalNetwork` with your new layer types. Only if you want your network to have extra functionalities, you should create a network type that inherits from `BidirectionalNetwork`.

Fiche masterproef

Student: Alexander Meulemans

Titel: Towards a mathematical understanding of biologically plausible learning methods for deep neural networks

Nederlandse titel: De ontwikkeling van een wiskundig theoretische basis van biologische plausibele leer methodes voor diepe neurale netwerken

UDC: 51-7

Korte inhoud:

Due to the recent successes of deep artificial neural networks (ANNs) and the resemblance of ANNs to the mammalian brain, the question has arisen whether the learning processes in the mammalian brain are driven by similar deep hierarchical models. This question is investigated in the newly emerged field of biologically plausible deep learning. Answering this question can contribute both to the field of machine learning and neuroscience. On the one hand, investigating biologically plausible deep learning can provide machine learning with new insights on hard challenges such as continuous learning, one-shot learning and general intelligence, which the human brain seems to achieve effortlessly. On the other hand, solving the credit assignment problem –how the strength of synapses in the brain must be changed to improve global behaviour– will help neuroscientists to develop new insights on how our brain (dys)functions in health and disease. Although a lot of progress has already been made, many of the biologically plausible learning methods currently lack a solid mathematical foundation and a well-defined link to the biological properties of neurons. This thesis focusses on target propagation (TP), a promising biologically plausible learning method for ANNs. Its main contributions to the field of TP are (1) a new extensive mathematical framework for TP, (2) various improvements to the TP method based on gained mathematical insights and (3) two biological network models of TP that are closely linked to the properties of pyramidal neurons. The new mathematical theory of TP has three main results. First, we prove that under well-specified conditions, TP with exact inverses uses Gauss-Newton optimization to compute its local layer targets, after which it performs a gradient descent step on the local layer losses to update the forward weights of the network. Second, we show that in TP with approximate inverses, reconstruction errors interfere with the propagated learning signals. We prove that difference target propagation (DTP), a variant of TP, cancels out these reconstruction errors by adding a correction term to its targets, which explains the better performance of DTP compared to TP. Third, we show that under well-specified conditions, DTP uses Gauss-Newton to compute its local layer targets, even when approximate inverses are used. Based on the gained mathematical insight on target propagation, we propose two improvements for the TP method. First, we introduce a new parametrization for the backward mapping function of the targets, which ensures that it can learn the inverse of the forward mapping to arbitrary precision when layers of equal dimension are used. Experimental results show that this new form led to significantly better performance of TP. Second, we propose a randomized version of TP which has better theoretical properties. Experimental results indicate that the randomized TP is more stable for deeper architectures. Finally, we develop two biological network models that exhibit TP-like learning dynamics and are closely linked to the properties of pyramidal neurons. The first model is a mixture model of the apical dendritic spikes and the basal dendritic spikes that occur in the pyramidal neuron. Both a single-phase and a two-phase version of this model are introduced, that exhibit TP-like and DTP-like learning dynamics, respectively. The second model makes use of multiplexing in pyramidal neurons to separate the feed-forward and feedback signals. The separation of signal paths provides this network with cleaner learning signals, compared to the previous mixture models. Both models are worked out in theory, while future research should verify their experimental performance. This thesis has as significant contributions to the field of biologically plausible deep learning that (1) it creates an extensive mathematical foundation for the TP method that led to improved variants of TP and (2) it developed two biological network models of TP that are closely linked to the biological properties of pyramidal neurons. Future research can use the mathematical framework to further improve TP and other biologically plausible learning methods and can use the biological network models in its quest for solving the credit assignment problem in the mammalian brain.

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: wiskundige ingenieurstechnieken

Promotoren: Prof. Benjamin Grewe
Prof. Johan Suykens

Assessoren: Prof. Daan Huybrechs
Dr. Nick Vannieuwenhoven

Begeleider: Prof. Benjamin Grewe

KU LEUVEN

ETH zürich

University of Zurich^UZH

Developing biologically plausible learning methods for deep neural networks

Introduction

In recent years, deep learning has reached incredible performance on many real-world tasks such as image and speech recognition. However, ANN's are still lacking behind in fields such as continuous learning, one-shot learning and general intelligence, which the human brain seems to achieve effortlessly. Therefore, it is useful to draw inspiration from biological networks, as has been done multiple times in the history of deep learning (e.g. convolutional networks). On the other hand, biologically plausible deep learning can give neuroscience testable hypotheses for synaptic plasticity mechanisms.

Goals of the thesis

1. Develop a deep learning training method that does not cope with the three main biological implausibilities of backpropagation:
   1. Weight transport
   2. Coupled plasticity of feedback weights
   3. Distinct clocked phases
2. Create testable hypothesis for the dependence of biological synaptic plasticity on the inner states of neurons in the human brain

Approach used in the thesis

1. Create a mathematical framework around the biologically inspired 'Target Propagation' (TP) method [1, 2]
2. Adjust the TP framework where needed to increase performance
3. Make a biologically realistic implementation of TP

Method

Target propagation (TP)

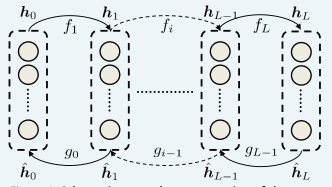


Figure 1: ***Schematic network representation of the target propagation learning scheme****. The input h_0 is propagated forward through the successive network layer mappings $f_i(\boldsymbol{h}_i)$. Afterwards, an output target $\hat{\boldsymbol{h}}_L$ is propagated backwards through the inverse network layer mappings $g_i(\hat{\boldsymbol{h}}_{i+1})$ to provide local layer targets.*

$$\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(\boldsymbol{a}_i) = s_i(W_i\boldsymbol{h}_{i-1})$$

$$\hat{\boldsymbol{h}}_L = \boldsymbol{h}_L - \hat{\eta}\frac{\partial L(\boldsymbol{h}_L, \boldsymbol{y})}{\partial \boldsymbol{h}_L}$$

$$\hat{\boldsymbol{h}}_i = g_i(\hat{\boldsymbol{h}}_{i+1}), \text{ with } g_i \approx f_{i+1}^{-1}$$

$$\Delta W_i = -\eta_i \frac{\partial L_i(\hat{\boldsymbol{h}}_i, \boldsymbol{h}_i)}{\partial W_i}$$

Error backpropagation (BP)

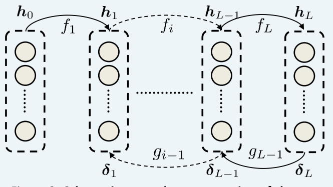


Figure 2: ***Schematic network representation of the error backpropagation learning scheme****. After the forward propagation, an error term $\boldsymbol{\delta}_L$ is propagated backwards through the network layer mappings $g_i(\boldsymbol{\delta}_{i+1})$ to provide local error signals. Note that the feedback weights need to be identical to the feedforward weights.*

$$\boldsymbol{h}_i = f_i(\boldsymbol{h}_{i-1}) = s_i(\boldsymbol{a}_i) = s_i(W_i\boldsymbol{h}_{i-1})$$

$$\boldsymbol{\delta}_L = \frac{\partial L(\boldsymbol{h}_L, \boldsymbol{y})}{\partial \boldsymbol{a}_L}$$

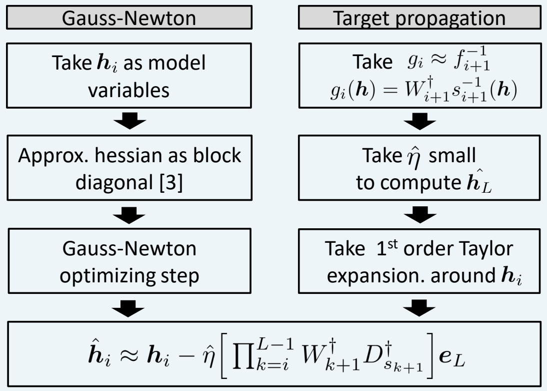
$$\boldsymbol{\delta}_i = g_i(\boldsymbol{\delta}_{i+1}) = s_i'(\boldsymbol{a}_i)^T W_{i+1}^T \boldsymbol{\delta}_{i+1}$$

$$\Delta W_i = -\eta \frac{\partial L}{\partial W_i} = -\eta \boldsymbol{\delta}_i \boldsymbol{h}_{i-1}^T$$

Results: theoretical

TP as an approximation of Gauss-Newton optimization

TP uses an approximation of Gauss-Newton optimization to compute its local layer targets $\hat{\boldsymbol{h}}_i$ based on output error $\boldsymbol{e}_L$:

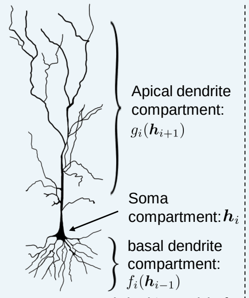


Gauss-Newton
Take $\boldsymbol{h}_i$ as model variables
Approx. hessian as block diagonal [3]
Gauss-Newton optimizing step
Target propagation
Take $g_i \approx f_{i+1}^{-1}$ $g_i(\boldsymbol{h}) = W_{i+1}^{\dagger} s_{i+1}^{-1}(\boldsymbol{h})$
Take $\hat{\eta}$ small to compute $\hat{\boldsymbol{h}}_L$
Take 1st order Taylor expansion. around $\boldsymbol{h}_i$
$\hat{\boldsymbol{h}}_i \approx \boldsymbol{h}_i - \hat{\eta}\Big[\prod_{k=i}^{L-1} W_{k+1}^{\dagger} D_{s_{k+1}}^{\dagger}\Big]\boldsymbol{e}_L$

Robust and efficient implementation of TP with exact inverses

- For rank 1 updates of W_i, its (pseudo-)inverse can be computed in $\mathcal{O}(n^2)$ with the Sherman-Morrison formula [4]
  $$(W_i + \boldsymbol{u}\boldsymbol{v}^T)^{-1} = W_i^{-1} - \frac{W_i^{-1}\boldsymbol{u}\boldsymbol{v}^T W_i^{-1}}{1+\boldsymbol{v}^T W_i^{-1}\boldsymbol{u}}$$
- By thresholding the denominator in the above update and correspondingly adjusting the stepsize of ΔW_i when needed, W_i can be prevented from becoming close to singular

Biological implementation of TP with segregated dendrites



Apical dendrite compartment: $g_i(\boldsymbol{h}_{i+1})$
Soma compartment: $\boldsymbol{h}_i$
basal dendrite compartment: $f_i(\boldsymbol{h}_{i-1})$

Figure 3: ***Segregated dendrite model of a pyramidal neuron****. The pyramidal neuron can be structured in three electrically distant compartments (figure adapted from [5]).*

Target propagation can be approximated by a set of biologically plausible differential equations :

$$\tau \frac{d}{dt}\boldsymbol{h}_i = -\boldsymbol{h}_i + (1-\lambda) f_i(\boldsymbol{h}_{i-1}) + \lambda g_i(\boldsymbol{h}_{i+1})$$

1st order Taylor expansion around the fixed points reveals TP dynamics:

$$\boldsymbol{h}_i^* = (1-\lambda) f_i(\boldsymbol{h}_{i-1}^*) + \lambda g_i(\boldsymbol{h}_{i+1}^*)$$

$$\boldsymbol{h}_i^* \approx f_i(\boldsymbol{h}_{i-1}^*) - \hat{\eta}\frac{\lambda^{L+1-i}}{(1-\lambda)^{L+1-i}}\Big[\prod_{k=i}^{L-1} W_{k+1}^{\dagger} D_{s_{k+1}}^{\dagger}\Big]\boldsymbol{e}_L$$

Results: experimental

Student-teacher network: nonlinear regression toy example

TP has comparable performance to BP on nonlinear regression toy example

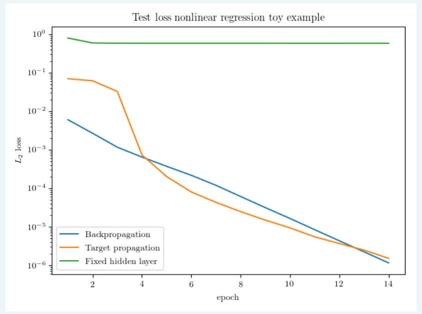


Figure 4: ***Test loss of nonlinear regression toy example****. A student network was trained to approximate a teacher network of the same layer dimensions (one hidden layer of 3 neurons and one output layer of 3 neurons).* ***Blue****: student network is trained with the error backpropagation method.* ***Orange****: student network is trained with the target propagation method with exact layer inverses ($g_i(\hat{\boldsymbol{h}}_{i+1}) = W_{i+1}^{-1} s_{i+1}^{-1}(\hat{\boldsymbol{h}}_{i+1})$).* ***Green:*** *student network with a fixed hidden layer, the output layer is trained with the error backpropagation method.*

Conclusion and outlook

Machine learning

- Approximate second order optimization for deep learning with $\mathcal{O}(n^2)$ computational cost
- Theoretical framework to analyse occurring phenomena in TP
- TP approach opens possibility for layer-specific loss functions

Neuroscience

- Theoretical hypothesis on how deep learning mechanisms can be implemented in our brain
- Testable hypothesis of how synaptic plasticity depends on the voltage levels in the neuron compartments

References

[1] Bengio *How Auto-Encoders Could Provide Credit Assignment in Deep Networks via Target Propagation*, 2014
[2] Lee et al. *Difference Target Propagation*, 2015
[3] Botev et al. *Practical Gauss-Newton Optimisation for Deep Learning*, 2017
[4] Meyer *Generalized inverse of modified matrices*, 1973
[5] Pinterest *https://www.pinterest.com/pin/495818240205349516/, accessed on 19/04/2019*

Master of Mathematical Engineering

Master's thesis
Alexander Meulemans

Supervisors
Benjamin F. Grewe
Johan Suykens

institute of neuroinformatics