Network Parameterisation and Activation Functions in Deep Learning

Trimmel, Martin

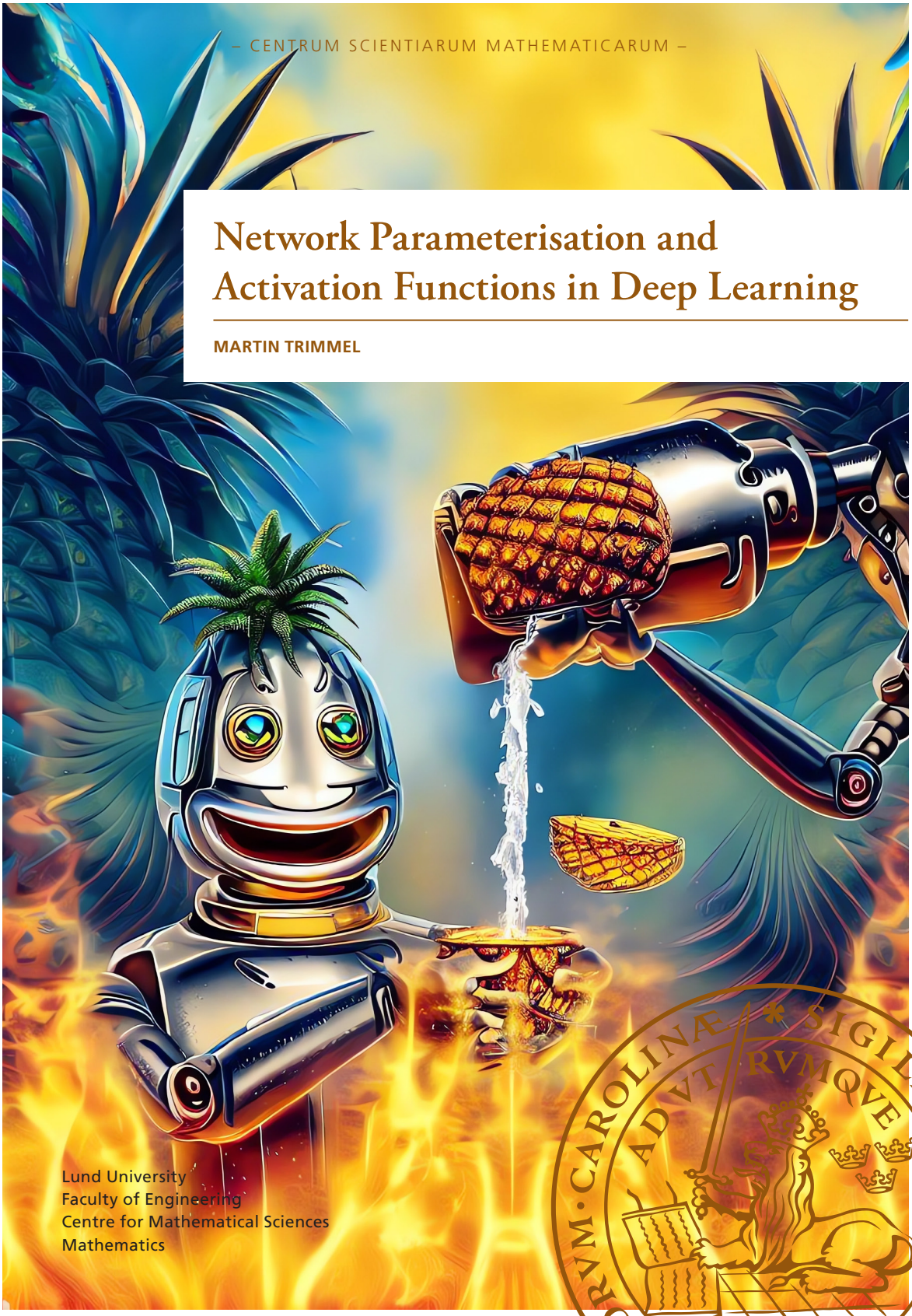
2023

Document Version:
Publisher's PDF, also known as Version of record

– CENTRUM SCIENTIARUM MATHEMATICARUM –

Network Parameterisation and Activation Functions in Deep Learning

MARTIN TRIMMEL



Lund University
Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Network Parameterisation and Activation Functions in Deep Learning

Network Parameterisation and Activation Functions in Deep Learning

by Martin Trimmel

LUND
UNIVERSITY

Thesis for the degree of Doctor of Philosophy

Thesis advisors
Prof Cristian Sminchisescu, Prof Anders Heyden, Dr Henning Petzka

Faculty opponent
Prof Guido Montúfar, University of California, Los Angeles (UCLA)

To be presented, with the permission of the Faculty of Engineering of Lund University, for public criticism in the lecture hall MH:Hörmander at the Centre for Mathematical Sciences on Tuesday, the 16th of May 2023 at 15:15.

Organization
LUND UNIVERSITY
Centre for Mathematical Sciences
Box 118
SE–221 00 LUND
Sweden

Document name
DOCTORAL THESIS
Date of disputation
2023-05-16
Sponsoring organization

Author(s)
Martin Trimmel

Title and subtitle
Network Parameterisation and Activation Functions in Deep Learning

Abstract
Deep learning, the study of multi-layered artificial neural networks, has received tremendous attention over the course of the last few years. Neural networks are now able to outperform humans in a growing variety of tasks and increasingly have an impact on our day-to-day lives. There is a wide range of potential directions to advance deep learning, two of which we investigate in this thesis:

(1) One of the key components of a network are its activation functions. The activations have a big impact on the overall mathematical form of the network. The *first paper* studies generalisation of neural networks with rectified linear activations units ("ReLUs"). Such networks partition the input space into so-called linear regions, which are the maximally connected subsets on which the network is affine. In contrast to previous work, which focused on obtaining estimates of the number of linear regions, we proposed a tropical algebra-based algorithm called TropEx to extract coefficients of the linear regions. Applied to fully-connected and convolutional neural networks, TropEx shows significant differences between the linear regions of these network types. The *second paper* proposes a parametric rational activation function called ERA, which is learnable during network training. Although ERA only adds about ten parameters per layer, the activation significantly increases network expressivity and makes small architectures have a performance close to large ones. ERA outperforms previous activations when used in small architectures. This is relevant because neural networks keep growing larger and larger and the computational resources they require result in greater costs and electricity usage (which in turn increases the CO2 footprint).

(2) For a given network architecture, each parameter configuration gives rise to a mathematical function. This functional realisation is far from unique and many different parameterisations can give rise to the same function. Changes to the parameterisation that do not change the function are called symmetries. The *third paper* theoretically studies and classifies all the symmetries of 2-layer networks using the ReLU activation. Finally, the *fourth paper* studies the effect of network parameterisation on network training. We provide a theoretical analysis of the effect that scaling layers have on the gradient updates. This provides a motivation for us to propose a Cooling method, which automatically scales the network parameters during training. Cooling reduces the reliance of the network on specific tricks, in particular the use of a learning rate schedule.

Key words
deep learning, linear region, network parameterisation, activation function, network calibration, conformal prediction, tropical algebra, rational function, temperature scaling, network symmetries

Classification system and/or index terms (if any)

Supplementary bibliographical information
Language
English

ISSN and key title
1404-0034
ISBN
978-91-8039-572-4 (print)
978-91-8039-573-1 (pdf)

Recipient's notes
Number of pages
220
Price
Security classification

Network Parameterisation and Activation Functions in Deep Learning

by Martin Trimmel

LUND
UNIVERSITY

Cover illustration front: The author generated this image with Microsoft Bing Image Creator, which is based on the DALL·E 2 generative model.

Funding information: This thesis work was supported in part by the European Research Council Consolidator grant SEED, CNCSUEFISCDI PN-III-PCCF-2016-0180, Swedish Foundation for Strategic Research (SSF) Smart Systems Program, as well as the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Faculty of Engineering, Centre for Mathematical Sciences

Doctoral Thesis in Mathematical Sciences 2023:3
ISBN: 978-91-8039-572-4 (print)
ISBN: 978-91-8039-573-1 (pdf)
ISSN: 1404-0034
LUTFMA: 1082-2023

Printed in Sweden by Media-Tryck, Lund University, Lund 2023

Dedicated in gratefulness
to my parents
Günter and Agnes Trimmel

In a new system of education there should be time for teachers to sit down with the students to listen to each other. Because both students and teachers have suffering in them. And if teachers and students understand each other's suffering, they will stop making each other suffer more. There will be good communication and the work of teaching and learning will become much easier.
Thich Nhat Hanh

Contents

I Background **v**

Preface **vii**

1 Abstract . . . vii
2 List of publications . . . viii
3 Acknowledgements . . . ix
4 Popular Summary . . . xi
5 Populärwissenschaftliche Zusammenfassung . . . xiii
6 Populärvetenskaplig sammanfattning . . . xv

1 Overview **1**

1 Notation in This Chapter . . . 2
2 Introduction . . . 4
3 Research Questions . . . 7
4 Formalism of Deep Learning . . . 8
4.1 Machine Learning Terminology . . . 8
4.2 Neural Networks . . . 9
4.3 Optimisation . . . 12
5 Deep Learning Techniques . . . 12
5.1 Learning Rate Schedule . . . 13
5.2 Regularisation . . . 13
5.3 Normalisation . . . 14
6 Activation Functions . . . 15
6.1 General Overview . . . 15
6.2 Rational Activation Functions . . . 17
7 Network Symmetries . . . 18
8 Piecewise-Linear Neural Networks . . . 20
8.1 Tropical Algebra . . . 20
8.2 Linear Regions . . . 22
9 Network Calibration . . . 24
9.1 Measuring (Mis-)Calibration . . . 24
9.2 Calibration Methods . . . 25
10 Contributions and Outlook . . . 26

II Scientific Publications **41**

Paper I: TropEx: An Algorithm for Extracting Linear Terms in Deep Neural Networks **43**

1 Introduction . . . 46
2 Background and Overview . . . 48
3 Method . . . 49
3.1 Matrix Representation of Tropical Rational Maps . . . 49
3.2 Extracting Linear Terms of a Classification Network . . . 49
4 Experiments . . . 51
5 Conclusion . . . 57
A Notation . . . 62
B Equivalence Between Tropical Matrices and Tropical Polynomials . . . 64
B.1 Basics of Tropical Algebra . . . 64
B.2 Tropical Matrix Operations . . . 65
B.3 Tropical Evaluation . . . 66
B.4 The Semiring of Tropical Matrices . . . 67
C Derivation of the Algorithm . . . 69
C.1 Introduction . . . 69
C.2 Proof of Lemma 3.2 . . . 70
C.3 Converting Activations Into Their Tropical Form . . . 71
C.4 Merging of Tropical Layers . . . 72
C.5 Selection of Terms . . . 76
C.6 Converting the Last Layer Into a Tropical Rational Function . . . 79
C.7 Proof of Theorem 3.3 . . . 80
C.8 Using Different Values of K . . . 80
C.9 An Observation on Cross Terms . . . 81
D Setup of Experiments . . . 82
E Additional Experiments . . . 84
E.1 Accuracy of Extracted Functions - Individual result for each architecture . . . 84
E.2 Visual Exploration of Generalization . . . 85
E.3 Leaky ReLU . . . 85
E.4 Evolution of test accuracy during extraction . . . 85
E.5 Similarity of activation patterns . . . 86
E.6 Comparing Network Coefficients after Network Retraining . . . 87
E.7 Visualization of Coefficients . . . 88
E.8 Exploring the neighborhood and estimating volume and number of linear regions in practice . . . 88
F Example: Applying TropEx to a Toy Network . . . 90
G Potential for Future Work . . . 98

Paper II: ERA: Enhanced Rational Activations **99**
1 Introduction . . . 102
2 Background . . . 104
3 Enhanced Rational Function . . . 105
4 Experiments . . . 106
4.1 Image Classification . . . 106
4.2 3D Human Pose and Shape Reconstruction . . . 110
5 Conclusions . . . 115
A Image Classification . . . 122
A.1 Additional Results on CIFAR10 . . . 123
A.2 Results on MNIST . . . 125
A.3 Results on ImageNet . . . 126
B 3D Human Pose and Shape Reconstruction . . . 135
B.1 Additional ablations . . . 135

Paper III: Notes on the Symmetries of 2-Layer ReLU-Networks **139**
1 Introduction . . . 142
2 Odd activation functions . . . 143
3 Two-layer ReLU regression networks . . . 144
4 Deep neural networks . . . 148
5 Conclusion . . . 150
A Proof of Lemma 3.2 . . . 152
B Proof of Lemma 3.3 . . . 152
C Proof of Theorem 3.4 . . . 154
D Example of a pair of degenerate (R1,R2,R3)-irreducible networks that implement the same network function . . . 156

Paper IV: How to Keep Cool While Training **159**
1 Introduction . . . 162
2 Background and Notation . . . 164
2.1 Notation . . . 164
2.2 Notions of Network Equivalence . . . 164
2.3 Network Calibration . . . 165
3 Method . . . 166
3.1 Analysis of Gradient Values . . . 166
3.2 Cooling While Training . . . 168
4 Experiments . . . 172
4.1 Motivation and General Setup . . . 172
4.2 Impact on Gradients . . . 172
4.3 Impact on Performance . . . 173
4.4 Impact on Calibration . . . 174
4.5 Which version of Cooling is the best? . . . 175

5 Conclusion 175
A Additional Proofs 182
B Training Setup 183
C Additional Results on CIFAR10 185
C.1 ResNet50 185
C.2 Batch Normalization and Dropout 185
C.3 Vector Scaling 185
C.4 Gradient Norms 185

Part I

Background

1 Abstract

Deep learning, the study of multi-layered artificial neural networks, has received tremendous attention over the course of the last few years. Neural networks are now able to outperform humans in a growing variety of tasks and increasingly have an impact on our day-to-day lives. There is a wide range of potential directions to advance deep learning, two of which we investigate in this thesis:

(1) One of the key components of a network are its activation functions. The activations have a big impact on the overall mathematical form of the network. The *first paper* studies generalisation of neural networks with rectified linear activations units ("ReLUs"). Such networks partition the input space into so-called linear regions, which are the maximally connected subsets on which the network is affine. In contrast to previous work, which focused on obtaining estimates of the number of linear regions, we proposed a tropical algebra-based algorithm called TropEx to extract coefficients of the linear regions. Applied to fully-connected and convolutional neural networks, TropEx shows significant differences between the linear regions of these network types. The *second paper* proposes a parametric rational activation function called ERA, which is learnable during network training. Although ERA only adds about ten parameters per layer, the activation significantly increases network expressivity and makes small architectures have a performance close to large ones. ERA outperforms previous activations when used in small architectures. This is relevant because neural networks keep growing larger and larger and the computational resources they require result in greater costs and electricity usage (which in turn increases the CO2 footprint).

(2) For a given network architecture, each parameter configuration gives rise to a mathematical function. This functional realisation is far from unique and many different parameterisations can give rise to the same function. Changes to the parameterisation that do not change the function are called symmetries. The *third paper* theoretically studies and classifies all the symmetries of 2-layer networks using the ReLU activation. Finally, the *fourth paper* studies the effect of network parameterisation on network training. We provide a theoretical analysis of the effect that scaling layers have on the gradient updates. This provides a motivation for us to propose a Cooling method, which automatically scales the network parameters during training. Cooling reduces the reliance of the network on specific tricks, in particular the use of a learning rate schedule.

2 List of publications

This thesis is based on the following publications, referred to by their Roman numerals:

I **TropEx: An Algorithm for Extracting Linear Terms in Deep Neural Networks**

Martin Trimmel*, Henning Petzka*, Cristian Sminchisescu
International Conference on Learning Representations (ICLR), Virtual Conference, 2021

II **ERA: Enhanced Rational Activations**

Martin Trimmel*, Mihai Zanfir*, Richard Hartley, Cristian Sminchisescu
European Conference on Computer Vision (ECCV), Tel Aviv, Israel, 2022

III **Notes on the Symmetries of 2-Layer ReLU-Networks**

Henning Petzka, **Martin Trimmel**, Cristian Sminchisescu
Northern Lights Deep Learning Conference (NLDL), Tromsø, Norway, 2020

IV **How to Keep Cool While Training**

Martin Trimmel, Mihai Zanfir, Richard Hartley, Cristian Sminchisescu
Preprint, April 2023

3 Acknowledgements

Even though only my name can be found on the cover of this book, this thesis is actually the result of the joint endeavour of many people. Without their efforts, the papers presented here would not have been possible, so I would like to express my sincere thanks to them.

First of all, I would like to thank my supervisor Cristian Sminchisescu for all the support he has given me during the last couple of years. Cristian always encouraged me to pursue new paths in research and he skillfully enabled many of the collaborations that helped produce this thesis. Thanks to him, I was able to visit many different places to discuss deep learning with people from around the world. I would also like to thank my co-supervisor Anders Heyden for having been happy to provide guidance whenever I asked for it. I am also very grateful to Richard Hartley for providing deep insights into various aspects of deep learning and Mihai Zanfir for his great suggestions and light-hearted conversations about research. An extra big thank-you goes to my co-supervisor Henning Petzka, whose guidance has really exceeded the ordinary and from whom I could learn a lot in my first projects. Henning was a great partner on travels between the alps and the arctic circle and I even enjoyed his googling of jokes about Austrians on a dizzying bus ride on the curvy roads along the Norwegian fjords. Despite moving away from Lund, Henning gratefully continued to offer his help and even spent his free time to support me in my defence and thesis process.

I am very lucky to have had great PhD siblings in our *research group family*, in which I was the last one to start. First, I would like to express my gratitude to my older PhD brother Aleksis Pirinen for being an awesome friend who greatly cares about all the people around him and the planet he lives on. I've really appreciated Aleksis' intelligent advice, his desire and ability to help others, many walks in nature together and a funny train and boat journey to Zurich. I thank my older PhD brother David Nilsson for his calm presence and interesting conversations. My thanks go to my older PhD sister Maria Priisalu for her infectious cheerfulness and laughter, a great collaboration on the time series project, precious travel advice and enjoyable conversations on culture and history. Finally, I'd also like to thank my slightly older twin brother Erik Gärtner for having been a great first officemate and computer expert who initially helped me a lot with getting used to my computer and to CUDA. My thanks also go to the friendly postdoc uncle Ted Kronvall for interesting discussions on generative models and a thriving "Skvättiväg" plant.
Outside of my closest research group, I would like to especially thank my second officemate Axel Berg for many interesting conversations about Transformers and point clouds and for teaching me a lot of exciting things about Swedish culture and history.

I would like to send a big thank-you to Carl Olsson for a lot of support and his readiness to help during the last few months. A big thanks also goes to Carl-Gustav Werner and James Hakim for their steady and fast computer support throughout the last few years, which made life much easier. I would also like to thank the administrative personnel at the mathematics centre, in particular Lena Lööf, Eva-Lena Borgström, Sara Ingelstrand and Jessica Kareseit, for making many things run more smoothly.

I would like to thank three fellow PhD students from other universities: First, thanks go to Frieder Simon from the University of Oxford for many entertaining conversations about Transformers and the mathematics of deep learning (including thinking about what a neural network actually is). I would also like to thank Jan-Paul Lerch from Bielefeld University for interesting mathematical conversations and insights throughout the last few years and for having invited me to a city which almost surely does not exist. A particular thanks also goes to Erik Sandström from ETH Zurich for a great time when learning more about generative models and for an unforgettable weekend on Öland.

"To be a friend means to offer happiness." I would therefore like to send a big thank you to some close friends, who are not related to my PhD or mathematics education, but who offered me a lot of happiness during the last few years. In alphabetical order: Annie & Pontus Bladkrona, Olle Claesson, Mike Huang, Ana Maria Martinez Moreno & Gustaf Sörnmo, Arlind Reuter and Alfredo Zombrano. It has been a big joy for me to have each of them in my life. I would especially like to water the flowers of Olle for his incredible openness and ability to listen, which helped me to get to know myself better. A lot of extra water also goes to Arlind for being herself like a happy flower that supports all the people around her. A special thanks also goes to Csaba Kurunci for having been a considerate flatmate and a great kifli- and kladdkaka-baker for the last two years.

Finally, I would like to express my gratitude to my family: I thank my parents Agnes and Günter Trimmel for their tremendous and constant support during all of my educational path and for always being there for me. Last, but not least, I send a big bow to my sister Simone Trimmel for a lot of interesting conversations throughout many years and for being such a great listener and travel partner.

Martin Trimmel
Lund, April 2023

4 Popular Summary

The idea of artificial, man-made intelligence has fascinated people since ancient times: starting with Talos, a giant forged from bronze that guarded the island of Crete in ancient Greek mythology, this idea was also found in the form of golems in ancient Jewish mysticism and later in the European Middle Ages with Paracelsus, who spoke of creating artificial men.

Today, we have the extraordinary chance to live in a time in which it seems possible for the first time to build machines that are in some respects more intelligent than us humans. In the last few decades there have been tremendous advances in machine learning. Computers are now able to recognise faces better, transcribe languages better and play chess and Go better than we can. They are also already making valuable contributions to medicine, including the ability to detect certain types of cancer. As 2022 has shown, computer programmes could also soon replace artists and serve as our conversation partners.

The main reason for these amazing advances is artificial neural networks, whose research and development is called Deep Learning. Although these networks were originally inspired by the human brain, in practice there are very big differences in how computers and humans process information.

Our lack of mathematical understanding of neural networks is a major problem. Often these networks are seen as a black box of which we know that it works, but not why it works. The first article in this thesis attempts to contribute to a better understanding of a widely used class of neural networks. It is well known that such networks are collections of an astronomically high number of linear functions. In fact, even small networks already ”contain” more linear functions than atoms in the entire universe. Using a special kind of mathematics called tropical algebra, we derive an algorithm to study the interplay of these linear functions and compare different types of neural networks.

Another disadvantage of neural networks is the gigantic amounts of energy consumed to train them. Although we are already aware of the dramatic consequences of the looming environmental and climate catastrophes, the trend towards ever larger networks (and thus higher power consumption) has continued in recent years. One possible measure can be to make networks more efficient and ensure that they do not have to perform as many calculations to reach the desired result. The second article in this thesis attempts to make certain building blocks of networks, called activation functions, more efficient and introduces an improved rational activation function that has significant advantages over other activation functions for small, efficient networks.

Networks can also become overconfident in their predictions and give undue credence to false predictions. There are techniques that calibrate ready-trained networks and reduce overconfidence. We apply one of these techniques during training and find that it makes the training itself much easier. The reason is this: the success of the training usually depends on a specific configuration (the so-called hyperparameters). One of the most important hyperparameters is the learning rate, which determines how fast the network changes during training. We show that calibration techniques make the ”right” learning rate less important and the training more stable.

5 Populärwissenschaftliche Zusammenfassung

Die Vorstellung künstlicher, menschengemachter Intelligenz fasziniert Menschen schon seit der Antike: Angefangen mit Talos, einem aus Bronze geschmiedeten Giganten, der in der altgriechischen Mythologie die Insel Kreta bewachte, fand sich diese Idee in Form von Golems auch schon in der alten jüdischen Mystik und später im europäischen Mittelalter bei Paracelsus wieder, der von der Herstellung künstlicher Männer sprach. Wir haben heute das außerordentliche Glück, in einer Zeit zu leben, in der es möglich scheint, erstmals Maschinen zu bauen, die in mancherlei Hinsicht intelligenter als wir Menschen sind. Gerade in den letzten Jahrzehnten hat es gewaltige Fortschritte im maschinellen Lernen gegeben. Computer sind nunmehr imstande, Gesichter besser zu erkennen, Sprachen besser zu transkribieren und auch besser Schach und Go zu spielen als wir. Sie leisten auch schon wertvolle Beiträge in der Medizin und sind unter anderem imstande, gewisse Arten von Krebs zu erkennen. Wie das Jahr 2022 gezeigt hat, könnten Computerprogramme auch schon bald Künstler ersetzen und uns als Konversationspartner dienen.

Der Hauptgrund für diese erstaunlichen Fortschritte stellen künstliche neuronale Netzwerke dar, deren Erforschung und Entwicklung man Deep Learning (Tiefes Lernen) nennt. Obwohl diese Netzwerke ursprünglich vom menschlichen Gehirn inspiriert wurden, gibt es in der Praxis sehr große Unterschiede, wie Computer und Menschen Informationen verarbeiten.

Unser fehlendes mathematisches Verständnis von neuronalen Netzwerken ist ein wesentliches Problem. Oftmals werden diese Netzwerke als Black Box betrachtet, von der man weiß, dass sie funktioniert, aber nicht wieso sie funktioniert. Der erste Artikel in dieser Abhandlung versucht, zum besseren Verständnis einer weit verbreiteten Klasse neuronaler Netzwerke beizutragen. Es ist bekannt, dass solche Netzwerke Ansammlungen unheimlich vieler linearer Funktionen sind. (In der Tat “beinhalten” selbst kleine Netzwerke bereits mehr lineare Funktionen als Atome im gesamten Universum). Mithilfe einer speziellen Art von Mathematik namens tropischer Algebra leiten wir einen Algorithmus her, um das Zusammenspiel dieser linearen Funktionen zu untersuchen und neuronale Netzwerke miteinander zu vergleichen.

Ein weiterer Nachteil neuronaler Netzwerke sind die gigantischen Mengen an Energie, die verbraucht werden, um sie zu trainieren. Obwohl uns die dramatischen Folgen der dräuenden Umwelt- und Klimakatastrophen bereits heute bewusst sind, hat sich der Trend zu immer größer werdenden Netzwerken (und damit zu höherem Stromverbrauch) in den vergangenen Jahren fortgesetzt. Eine mögliche Maßnahme kann es sein, Netzwerke effizienter zu gestalten und dafür zu sorgen, dass sie weniger viele Berechnungen durchführen müssen, um zum gewünschten Ergebnis zu kommen. Der zweite Arti-

kel dieser Abhandlung versucht, bestimmte Bausteine von Netzwerken, sogenannte Aktivierungsfunktionen, effizienter zu gestalten, und stellt eine verbesserte rationale Aktivierungsfunktion vor, die bei kleinen, effizienten Netzwerken signifikante Vorteile gegenüber anderen Aktivierungsfunktionen hat.

Netzwerke können auch zu selbstsicher in ihren Vorhersagen werden und falschen Vorhersagen übertriebenen Glauben schenken. Es gibt Techniken, die fertig trainierte Netzwerke kalibrieren und übermäßiges Selbstvertrauen vermindern. Wir wenden eine dieser Techniken bereits während des Trainings an und finden heraus, dass dadurch das Training selbst wesentlich erleichtert wird. Der Grund dafür ist folgender: Der Erfolg des Trainings hängt normalerweise von einer speziellen Konfiguration (den sogenannten Hyperparametern) ab. Einer der wichtigsten Hyperparameter ist die Lernrate, die bestimmt, wie schnell sich das Netzwerk während des Trainings verändert. Wir zeigen, dass mithilfe von Kalibrierungstechniken die “richtige” Lernrate weniger wichtig und das Training stabiler wird.

6 Populärvetenskaplig sammanfattning

Idén om artificiell, konstgjord intelligens har fascinerat människor sedan antiken: från Talos, en jätte av brons som vaktade ön Kreta i den grekiska mytologin, fanns idén också i form av golems i den judiska mystiken och senare under medeltiden i Europa med Paracelsus, som talade om att skapa konstgjorda människor.

Idag lever vi i fascinerande en tid då det för första gången verkar möjligt att bygga maskiner som i vissa avseenden är intelligentare än vi människor. Bara under de senaste decennierna har det gjorts enorma framsteg inom maskininlärning. Datorer kan nu känna igen ansikten bättre, transkribera språk bättre och spela schack och Go bättre än vi. De kommer också redan idag med värdefulla bidrag till medicinen, bland annat genom att de kan upptäcka vissa typer av cancer. Under 2022 har vi till och med sett att datorprogrammen snart kan ersätta konstnärer och bli våra samtalspartners.

Huvudorsaken till dessa fantastiska framsteg är artificiella neuronnät, vars forskning och utveckling kallas Deep Learning (djupinlärning). Även om dessa nätverk ursprungligen inspirerades av den mänskliga hjärnan finns det i praktiken mycket stora skillnader mellan hur datorer och människor bearbetar information.

Vår bristande matematiska förståelse av neuronnät är ett stort problem. Ofta ses dessa nätverk som en svart låda - vi vet att de fungerar, men inte varför de fungerar. I den första artikeln i denna avhandling försöker vi bidra till en bättre förståelse av en ofta använd klass av neuronnät. Det är välkänt att sådana nätverk består av ett otroligt stort antal linjära funktioner. Faktum är att även små nätverk redan innehåller fler linjära funktioner än antalet atomer som finns i hela universum. Med hjälp av en speciell typ av matematik som kallas tropisk algebra har vi tagit fram en algoritm för att studera samspelet mellan dessa linjära funktioner och jämföra neuronnät.

En annan nackdel med neuronnät är de enorma mängder energi som går åt för att träna dem. Även om vi redan är medvetna om de dramatiska konsekvenserna av de annalkande miljö- och klimatkatastroferna har trenden mot allt större nätverk (och därmed högre energiförbrukning) fortsatt under de senaste åren. En möjlig åtgärd kan vara att göra nätverken effektivare och se till att de inte behöver göra så många beräkningar för att uppnå ett önskat resultat. I den andra artikeln försöker vi göra vissa byggstenar i nätverken (så kallade aktiveringsfunktioner) effektivare och introducerar en förbättrad rationell aktiveringsfunktion som har stora fördelar jämfört med andra aktiveringsfunktioner för små, effektiva nät.

Nätverken kan också bli alltför självsäkra i sina förutsägelser och tilldela felaktiga förutsägelser en otillbörlig trovärdighet. Det finns tekniker som kalibrerar färdigtränade nätverk så att de minskar sin övertro på sina förutsägelser. Vi har tillämpat en av dessa

tekniker under träningsfasen och funnit att det gör själva träningen mycket enklare. Orsaken är följande: Träningens framgång beror vanligtvis på en specifik konfiguration (de så kallade hyperparametrarna). En av de viktigaste hyperparametrarna är inlärningshastigheten, som bestämmer hur snabbt nätverket förändras under träningen. Vi visar att kalibreringsmetoder gör att den rätta inlärningshastigheten blir mindre viktig och att träningen blir mer stabil.

Chapter 1

Overview

1 Notation in This Chapter

General notation

The uppercase version of any real-valued variable is a random variable for which the lower case variable denotes a value. (E.g. if $x \in \mathbb{R}$, then X is a real-valued random variable.)

Lowercase bold variables denote vectors and corresponding non-bold variables with a subscript denote their entries. (E.g. $\mathbf{x} \in \mathbb{R}^d$ is a vector and x_j are its entries for $1 \leq j \leq d$.)

Table of notation

Symbol	Meaning
$\{i, \dots, j\}$	set of integers k such that $i \leq k \leq j$
$\mathcal{A}(\mathbf{x})$	activation pattern of a network at a data point $\mathbf{x}$
α	hyperparameter
β	hyperparameter
B_j	j^{th} bin corresponding to the interval $(\frac{j-1}{M}, \frac{j}{M}]$
$\mathbf{b}$	bias vector
$\mathbf{b}_i$	bias vector corresponding to the i^{th} layer of a neural network
$C(D, D')$	set of continuous functions $D \to D'$
c	ground truth label
$\widehat{c}$	predicted label
$\mathrm{CE}(p)$	calibration error depending on the probability p
$\mathrm{Ch}(v)$	set of children of a node v
D, D', D_i	domain of a function
δ_{ij}	Kronecker delta of i and j
d, d_i	dimensions in $\mathbb{Z}_{\geq 0}$
E	set of edges
$\mathcal{F}$	space of (neural network) functions
F	set of local functions
f	function
f_θ	function parameterised by a collection of parameters θ
$\mathcal{G}$	directed graph
η	learning rate
K	number of classes for classification
$\mathcal{L}_{\mathbf{j}}$	set of multi-indices corresponding to learnable axes
ℓ	linear layer of a neural network, equal to a function $\mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$
L	loss function $\mathcal{X} \times \Theta \to \mathbb{R}_{\geq 0}$
λ	hyperparameter

$\mathcal{N}_\theta$	neural network and its function
N_b	number of interval bins
N_ℓ	number of layers of a network
N_n	number of neurons of a network
N_v	number of vertices of a graph
p	probability value or likelihood, dropout hyperparameter
$\widehat{p}$	confidence estimate of a network, maximum of $\widehat{\mathbf{y}}$
$\mathbb{P}$	probability measure
$\mathrm{Pa}(v)$	set of parents of a node v
$\mathbb{R}$	real numbers
$\mathbb{R}_{\geq 0}$	non-negative real numbers
$\mathcal{S}_{\mathbf{j}}$	set of multi-indices corresponding to standardisation axes
s	parameterisation symmetry $s : \Theta \to \Theta$
σ	activation function
θ	parameter configuration in Θ
Θ	set of all possible parameter configurations
V	set of vertices
v	node/vertex in a graph
$\mathbf{W}$	weight matrix
$\mathbf{W}_i$	weight matrix corresponding to the i^{th} layer of a neural network
$\mathbf{W}^{\text{pos}}$	positive part of a matrix defined by entries $w_{ij}^{\text{pos}} = \max\{0, w_{ij}\}$
$\mathbf{W}^{\text{neg}}$	negative part of a matrix defined by entries $w_{ij}^{\text{neg}} = \max\{0, -w_{ij}\}$
$\mathcal{X}$	data set
$\mathcal{X}_{\text{test}}$	test data set
$\mathcal{X}_{\text{train}}$	training data set
$\mathcal{X}_{\text{val}}$	validation data set
$\mathbf{x}$	vector describing data sample or input to layer of a neural network
$\mathbf{x}_i$	output of the i^{th} layer of a neural network, $\mathbf{x}_i = \rho(\mathbf{W}_i \mathbf{x}_{i-1} + \mathbf{b}_i)$
x_j	vector components of $\mathbf{x}$ in $\mathbb{R}$
x	real number
$\mathbf{y}$	vector of ground-truth class probabilities in $\mathbb{R}^K$
$\widehat{\mathbf{y}}$	vector of predicted class probabilities in $\mathbb{R}^K$
$\mathbf{z}$	vector of predicted logits in $\mathbb{R}^K$

2 Introduction

During the last decade, artificial intelligence has successfully emerged out of science fiction books into our daily lives. Large scale machine learning models are now widely used in science, by companies and even by pupils and students who prefer generating essays within a few seconds over spending hours writing them. At the core of these social changes lies *deep learning*, the study of large-scale artificial neural networks. Such networks are used to represent mathematical functions $\mathcal{N}_\theta : \mathbb{R}^d \to \mathbb{R}^{d'}$ that can be applied to perform tasks like machine translation or image classification. The weights $\theta \in \Theta$ of the network are optimised on large datasets to make f_θ take the desired form. This thesis studies neural networks from two perspectives:

(1) *Activation functions*. Informally speaking, activation functions are the non-linear components of neural networks. Network architectures often alternate between scantly-parameterised activations and heavily-parameterised linear layers. The overall mathematical form of the network function depends greatly on the choice of activation function: (i) Piecewise linear activation functions induce a partition of the input space into an astromically large number of *linear regions* on which the network function is affine. For such activation functions, understanding the interplay of the linear regions can be conducive to understanding network generalisation. (ii) Since parametric activations are much more flexible than their non-parametric counterparts, they have the potential of increasing the expressiveness of a given network architecture. This is highly relevant because network keep ever increasing in size and have sometimes reached more than a trillion parameters. Training such networks requires vast amounts of computational resources, which in turn raises the associated financial and environmental costs.

(2) *Network parameterisation*. When we optimise a network, we search for a suitable network function $\mathcal{N}_\theta$. This search does not directly take place in the function space, but we are actually traversing a path in the parameter space Θ and each change in parameters implicitly also changes the network function. It is therefore natural to take a closer look at the relationship between functions and parameters. For a given neural architecture, the function $\mathcal{N}_\bullet : \Theta \to C(\mathbb{R}^d, \mathbb{R}^{d'})$ assigning to each parameter configuration its corresponding network function is not surjective, but for non-polynomial activation functions, its image is dense in $C(\mathbb{R}^d, \mathbb{R}^{d'})$. The function $\mathcal{N}_\bullet$ is not injective either, and changes to the network parameterisation which do not change the underlying network function, are called *symmetries* of the architecture. Even though differently parameterised networks can represent the same function, the choice of parameterisation as a general rule affects the gradient-based parameter updates and thus has a significant effect on the training of the network. Hence, apart from finding and classifying all symmetries for a given architecture, it is at least equally important to understand what effect the choice of parameterisation has on network training.

This overview is structured in the following way: In Section 3 we describe the research questions we studied in this thesis and how they relate to the presented papers. In Section 4 we formally introduce the terminology and describe the most fundamental concepts of the thesis. In Section 5 we give a brief overview of some of the most important deep learning techniques used in our papers. In Section 6 we present an overview over previous work on activation functions and place a special focus on parametric and rational activation functions because of their relation to Paper II. Sections 7-9 introduce some of the more advanced deep learning concepts, relating to network parameterisation, piecewise-linear neural networks and network calibration. Finally, we summarise the contributions of the papers and provide an outlook on potential future work in Section 10.

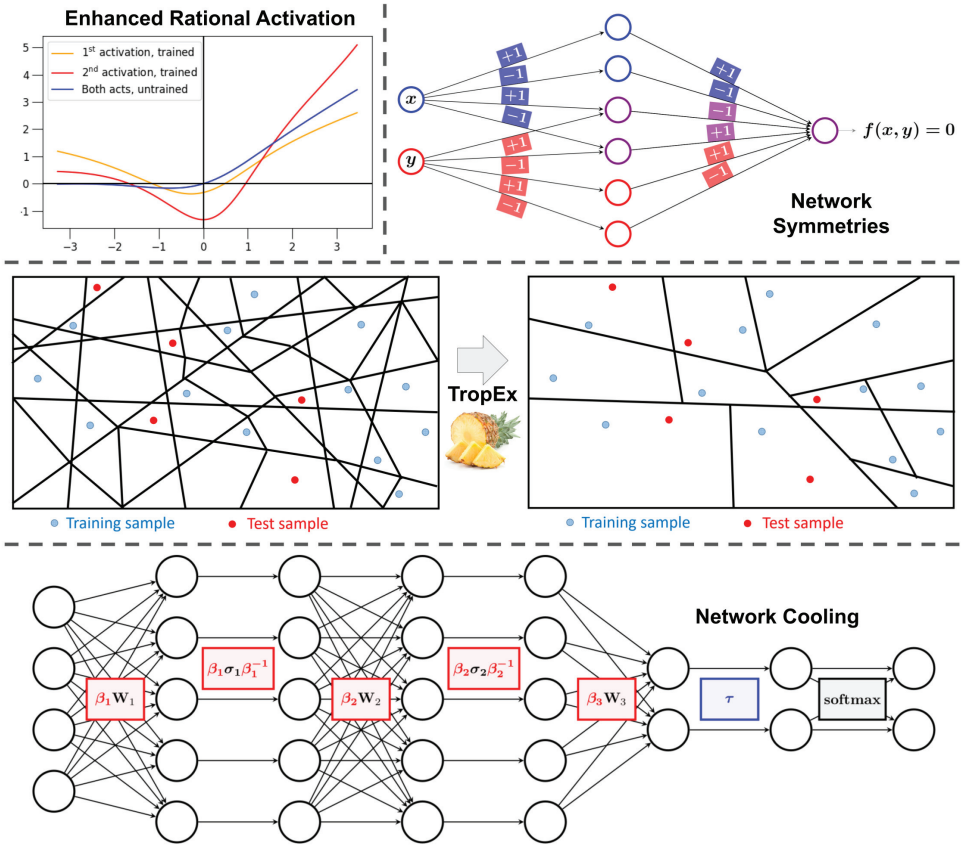


Figure 1.1: In this thesis, we have proposed methods for improving and better understanding activation functions and network parameterisation. **Top, left:** Plot of the first two Enhanced Rational Activations (ERAs) in a network before and after training. **Top, right:** We classified the symmetries of 2-layer ReLU neural networks. The plot shows a network whose parameters are non-zero, but which represents the zero function. **Middle:** We developed TropEx, an algorithm using tropical algebra to extract the coefficients of linear regions of piecewise linear (e.g. ReLU) networks. We used TropEx to study the effect that different network parameterisations (fully-connected vs convolutional) have on the linear regions. **Bottom:** We investigated the effect of network parameterisation on network training and proposed a Cooling method to adaptively reparameterise the network during training.

3 Research Questions

The overall research question of this thesis is how to better theoretically understand and how to improve network parameterisation and activation functions. We have grouped the questions according to the topic they are related to:

Network Parameterisation

[**PQ1**] Given a network architecture, what are its symmetries?

[**PQ2**] How does changing the parameterisation of the network function[1] affect the linear regions?

[**PQ3**] How does changing the parameterisation of the linear regions affect the network function[1]?

[**PQ4**] How does the parameterisation of a network affect its training?

Activation Functions

[**AQ1**] When are parametric activation functions preferable over non-parametric ones?

[**AQ2**] How can rational activation functions be made more stable and performant?

Please note that the last two parameterisation questions [**PQ2**] and [**PQ3**] are actually also questions concerning activation functions, since they are specific to continuous piecewise linear activation functions.

We investigate all of the questions above in this thesis. The activations function questions [**AQ1**] and [**AQ2**] are studied in Paper II, where we propose an Enhanced Rational Activation (ERA), which can be initialised randomly and which are particularly beneficial when used in smaller networks. We provide a partial answer to question [**PQ1**] in Paper III, where we classify all the symmetries of 2-layer ReLU networks. Paper I focuses on questions [**PQ2**] and [**PQ3**], where the proposed TropEx-algorithm is used to compare convolutional and fully-connected networks. Finally, we investigate question [**PQ4**] in Paper IV, where we perform an analysis of the gradients of rescaled networks and propose a Cooling method to make the network being better parameterised during training.

[1] Refers to networks with piecewise linear activation functions.

4 Formalism of Deep Learning

The aim of this section is to formally define the machine and deep learning terminology and notation used in this thesis. The concepts introduced here relate to all the research questions from Section 3 and can be seen as setting the stage for the paper-specific overview starting in Section 6. We start by briefly defining terms from general machine learning and then present some essential terminology relating to deep learning, including a formal definition of an artifical neural network.

4.1 Machine Learning Terminology

Machine learning is the study of algorithms to make computers carry out tasks they are **not explicitly programmed for**. Instead, the computers "learn" from data how to perform the task. It would be very hard to write a program that tells a computer explicitly how to detect faces on photos. Hence, in machine learning we teach the computer how to do face detection using a lot of sample data: photos with and without faces.

There is a great **variety of tasks** that we want to solve with machine learning: Amongst others, we want to be able to predict values (*regression*), assign labels to data points (*classification*), group data points (*clustering*), extract key features (*dimensionality reduction*) and generate new data points (*generative modelling*). There is a **variety of models** to achieve some of these tasks. E.g. For regression, there are methods like ridge regression [38, 39], Bayesian regression and Support Vector Machines [8, 73]. For classification, there is least squares classification, Fisher's discriminant [25], logistic regression [15] and again Support Vector Machines.

The **way how we try to perform a task** is always the same:

1. We think of the task as a mathematical function f^*, called the *target function*.
2. We try to approximate the function f^* on a dataset $\mathcal{X}$ by our modelling function f_θ. The variable θ denotes the parameters of the function. We denote the set of all parameter configurations by Θ.
3. We use data to optimise θ on a *training dataset* $\mathcal{X}_{\text{train}}$ so that f_θ becomes similar to f^*. To this end, we often minimise a *loss function* $L : \mathcal{X}_{\text{train}} \times \Theta \to \mathbb{R}_{\geq 0}$, which measures the similarity between f_θ and f^*.

We note that $\mathcal{X}_{\text{train}} \subset \mathcal{X}$, i.e. the training dataset is a strict subset of the general data distribution we are interested in. Often, we also use a *validation dataset* $\mathcal{X}_{\text{val}}$ to compare

models trained with different hyperparameter configurations with each other. Finally, we use a *test set* $\mathcal{X}_{\text{test}}$ to determine the performance of the model on unseen data.

We assume in this thesis that all data points $\mathbf{x}$ are multi-dimensional arrays in $\mathbb{R}^{d_1 \times \cdots \times d_n}$ for some dimensions $d_1, \ldots, d_n$. We will call such data points *tensors*. In particular, RGB $h \times w$ pixel images are assumed to be elements of $\mathbb{R}^{h \times w \times 3}$, where each pixel takes integer values in the range $[0, 255]$. In practice, data points tend to be normalised to the range $[0, 1]$ or $[-1, 1]$ before entering the model. The labels $k \in \{1, \ldots, C\}$, representing one of C classes to which a data points can belong to, are commonly converted into one-hot vectors $\mathbf{y} \in \{0, 1\}^C$, where entry $y_i = 1$ if and only if $i = k$.

The optimal choice of model depends on the task and the data at hand. Generally speaking, when the *complexity* of the dataset $\mathcal{X}$ is small, a less expressive model with fewer parameters is required than for a more complex dataset. There are two risks: (1) the model is not expressive enough for the dataset (*underfitting*); (2) the model is too expressive for the dataset (*overfitting*). Both of these problems are often discernable via the loss function or other evaluation metrics: If f_θ performs poorly on the training data, the model may underfit the data and we may need to make it more expressive. On the other hand, if f_θ performs well on the training data, but poorly on the test data, the model is overfitting the data and it may be too expressive for the given task.

4.2 Neural Networks

More basic models mentioned in Section 4.1 provide suitable functions f_θ for accomplishing a great variety of tasks. However, they often are often not flexible enough to model a large class of different functions f^*. That's why we are interested in a more general model: Artifical Neural Networks. Since these models are graphs, we will need some graph-theoretic vocabulary before defining them:

Definition 4.1

Let $\mathcal{G} = (V, E)$ be a directed graph. If there is an edge $(v, v') \in E$, we say that v' is a *child* of v and v is a *parent* of v'. We denote the sets of children and parents of a node v by $\text{Ch}(v)$ and $\text{Pa}(v)$, respectively. If a node v has no parents, we call it a *root* and if it has no children, we call it a *leave*.

We note that there exists no generally accepted mathematical definition of what an artificial neural network constitutes [6, 24, 31]. For this reason, we present our own definition, which draws inspiriation from the formalism presented in [14]. Our definition covers all the networks used in this thesis and aims for utmost generality. Broadly speaking, we define a neural network to be a graph, whose edges correspond to parameterised, differentiable functions.

Definition 4.2

Let $\mathcal{G} = (V, E, F)$ be a tuple consisting of a finite connected graph (V, E) and a set F defined by

- the set $V = \{\mathbf{X}_i \mid i \in \{1, \dots, N_v\}\}$ is an ordered set of random variables $\mathbf{X}_i$, such that
  - all $\mathbf{X}_i \in D_i \subseteq \mathbb{R}^{d_i}$ for some $d_i \in \mathbb{N}$;
  - the first r nodes $\mathbf{X}_1, \dots, \mathbf{X}_r$ are the roots and are called *input nodes*;
  - the last l nodes $\mathbf{X}_{N_v-l+1}, \dots, \mathbf{X}_{N_v}$ are the leaves and are called *output nodes*;
  - the other $N_v - r - l$ nodes $\mathbf{X}_{r+1}, \dots, \mathbf{X}_{N_v-l}$ are called *hidden nodes*;
- the set E contains directed edges;
- the set

$$F = \left\{ f_i : \left(\prod_{j:\mathbf{X}_j \in \mathrm{Pa}(\mathbf{X}_i)} D_j \right) \times \Theta_i \to D_i \;\middle|\; i \in \{r+1, \dots, n\} \right\} \tag{1.1}$$

$$(\mathbf{X}_{j_1}, \dots, \mathbf{X}_{j_n}; \theta_i) \mapsto \mathbf{X}_i \tag{1.2}$$

  consists of parameterised *local functions* defining the relations between hidden or output nodes and their parents. The function f_i is parametrised by the set Θ_i (where it is possible that there exist some i with $\Theta_i = \varnothing$, i.e. that some f_i are non-parametric).

Again, we denote by $\Theta = \prod_i \Theta_i$ the set of (global) parameter configurations. For a given configuration $\theta \in \Theta$, we can evaluate $\mathcal{G}$ by entering data at the input nodes and by consecutively evaluating the local functions. (If there are any cycles, it may be necessary to initialise some of the nodes by some values.) This allows us to define a function, which we will denote by $\mathcal{G}_\theta : \prod_{i=1}^{r} D_i \to \prod_{i=n-l+1}^{N_v} D_i$.

If $(\mathbf{X}_i, \mathbf{X}_j) \in E$ implies that $i < j$, we say that $\mathcal{G}$ is a *computational graph*. (In particular, computational graphs are acyclic.) If all local functions $f_i \in F$ are differentiable almost everywhere (with respect to the Lebesgue measure), we call the tuple a *neural network architecture*, which we denote by $\mathcal{N}$. For a given parameter configuration, the function $\mathcal{N}_\theta$ is then called a *network function*. In line with most of the literature, we will use the term *neural network* ambiguously and sometimes employ it to denote the architecture $\mathcal{N}$ and sometimes an associated network function $\mathcal{N}_\theta$. We say that a *neuron* of a network is an entry of one of the vectors $\mathbf{X}_i \in V$. We denote the total number of neurons of a network by N_n.

If a neural network is acyclic, its is called a *feedforward neural network*. Otherwise, if it contains at least one loop, it is called a *recurrent neural network (RNN)*. We only consider feedforward neural networks in this thesis. Therefore, **from now on we always mean only feedforward networks when using the term "neural network".** We now introduce some further important terminology concerning the components of neural networks:

Definition 4.3

Let $\mathcal{N}$ be a neural network and for some N_ℓ, let $1 \leq i \leq N_\ell$ and let $\mathbf{W}_i \in \mathbb{R}^{d_i \times d_{i-1}}$, $\mathbf{b}_i \in \mathbb{R}^{d_i}$, $\sigma_i : D_i \to D_i$ be some matrices, vectors and non-linear functions, respectively. We define a network *layer* ℓ_i to be the function given by

$$\mathbf{x}_i = \sigma_i(\mathbf{W}_i \mathbf{x}_{i-1} + \mathbf{b}_i). \tag{1.3}$$

Let $\mathbf{x}_0$ and $\mathbf{x}_{N_\ell}$ be the input and the output of the network, respectively. If $N_\ell = 2$, we say that $\mathcal{N}$ is a *shallow* network. Otherwise, we say that $\mathcal{N}$ is a *deep* network and call it a *multi-layer perceptron (MLP)*. We call $\mathbf{W}_i$, $\mathbf{b}_i$ and σ_i a *weight matrix*, a *bias vector* and an *activation function*, respectively. We say that a layer is *dense* if the parameters in $\mathbf{W}_i$ can take any real value. We say that a network is *fully-connected* if all of its layers are dense.

Using only a stack of plain fully-connected MLPs would lead to computer memory problems even for small input images. For this reason, it is necessary to impose additional structure on the weight matrices. One of the most common ways to achieve this, is to use convolutions:

Definition 4.4

A layer ℓ is called *convolutional*, if the affine part of ℓ is computed by applying a cross-correlation[2]with a filter F to the data point:

$$\mathbf{x}' = F * \mathbf{x} + \mathbf{b}.$$

The bias is applied channel-wise here: To all pixels in channel i of $F * \mathbf{x}$, the same number b_i is added. A *convolutional neural network (CNN)* is a feedforward neural network that contains at least one convolutional layer.

In practice, convolutional neural networks are preferred over fully-connected MLPs. According to [27], CNNs enjoy 3 benefits: (1) they have sparse interactions, i.e. most of their connections in the weight matrix are set to 0; (2) they share their parameters,

[2]Even though cross-correlations are used, the networks are in practice called *convolutional* by an abuse of notation. This can be justified by the fact that applying a cross-correlation with filter F is the exactly the same as applying the convolution of with F flipped horizontally and vertically.

so they do not need to learn a separate set of parameters at each pixel; (3) they are translation equivariant, i.e. if the input to a convolution layer is translated, then the outputs gets translated in the same manner. However, we still lack a deep understanding of why convolutional networks work so well [80]. It is unclear why they achieve higher accuracies than their fully-connected counterparts. In Paper I in this thesis we investigate differences between fully-connected MLPs and CNNs from the view of linear regions.

4.3 Optimisation

When *training* a neural network $\mathcal{N}_\theta$, we optimise its parameters θ to make $\mathcal{N}_\theta$ minimise a loss L on the training dataset. Since $\mathcal{N}_\theta$ is differentiable with respect to θ, we can use the gradients in the optimisation procedure to obtain a parameter update of the form

$$\theta \leftarrow \theta - \eta \nabla_\theta L, \tag{1.4}$$

where the scalar hyperparameter η is called the *learning rate*. When we compute the gradients over all the data points in the dataset, this is called *gradient descent*. This standard procedure is, however, mostly infeasible in practice, since it requires computing the loss based on the entire dataset, which often does not even fit into computer memory. For this reason, the network function is optimised using *stochastic gradient descent (SGD)*, where the gradient updates are computed on only a *batch* of data points. Apart from vanilla SGD, some other common optimisation methods are AdaGrad [21], RMSProp [71] and Adam [44]. Some of these more advanced methods employ decaying averages of the gradients, which often works better than directly updating the parameters.

The gradients $\nabla_\theta L$ are most commonly computed by *backpropagation* [47], an algorithm based on the chain rule of differentiation. Informally speaking, backpropagation amounts to reversing all edges in E and traversing the graph backwards until all the parameters have been reached. For that purpose, we calculate all the derivatives of the local functions and compute the desired derivatives with respect to the parameters using the chain rule.

5 Deep Learning Techniques

In this section we give an overview of some commonly used techniques that help (and in some cases enable) the training of neural networks. All the material presented here is generally relevant for all the papers of this thesis. However, there are also some paper-specific differences: For Paper IV, the first two subsections on learning rate schedules and regularisation are important for because the paper proposes a technique

which reduces the reliance on deep learning "tricks" for successful training, in particular the use of a learning rate schedule. The last subsection on normalisation is relevant for Paper II because that paper makes use of various normalisation techniques to stabilise the training of rational activation functions.

5.1 Learning Rate Schedule

Instead of keeping the learning rate η constant during training, most often a *learning rate schedule* is being used. Two noticeable basic schedules are (1) a piecewise-linear one, where the linear rate drops by a certain factor after a given number of epochs and (2) an exponential one, where the learning rate continuously decays exponentially throughout training. A more advanced method is cosine scheduling [48], where the learning rate behaves like a cosine function which, depending on the implementation, may or may not restart after having reached zero for the first time. Another option is to start with a small learning rate and to gradually increase it until it has reached a maximal value [29, 35]. This latter option is called *learning rate warmup* and it is motivated by the desire to avoid overfitting to the first training samples, which may happen if the learning rate is too large.

5.2 Regularisation

There is a wide range of *regularisation* tricks and techniques to reduce the gap between training and test performance. When using *weight decay*, a term $\lambda|\theta|$ is added to the loss function which penalises weights with large magnitudes. (λ is a hyperparameter here.) We check how well our model performs on a validation dataset during training. We stop training when the loss (or another metric) on the validation dataset stops improving, which is called *early stopping*. Another option is to use *dropout* ([68]): During training, each node is set to 0 with probability $p \in (0, 1)$. Nodes not set to zero are multiplied by $1/p$. After training, the layer is used normally. (p is a hyperparameter here.)

One can also use techniques, which are more data-based: If we apply *label smoothing*, i.e. we replace the one-hot encoded labels by $\mathbf{y} \in \{\alpha, 1-\alpha\}^C$ for some small constant α. This reduces overfitting and makes models predict probability values which are more closely aligned with the probabilities of actually making correct predictions [55]. In order to generate new training points from the given dataset, *data augmentation* is commonly used. For images, this includes rescaling the image or changing its colour intensities, applying rotations, translations or noise.

5.3 Normalisation

We asked in [**AQ2**] how rational activation functions can be made more stable. In our investigation in Paper II we noticed that one of the biggest problems for rational activations is the so-called *covariate shift*, i.e. the change in input distribution to the activation function. This is a general problem for deep networks, since updating the parameters of earlier layers affects the input of later layers, and hence the layer not only needs to learn "new things", but also to adapt to the changes caused by the parameters updates of the earlier layers. For this reason, various *normalisation techniques* have been developed to standardise the input to the layers and to reduce covariate shift. These normalisation methods always follow the same pattern, which we will now describe, following [76]. Let $\mathbf{x} \in \mathbb{R}^{d_1 \times \cdots \times d_n}$ be a tensor representing a batch of data, where the data points are stacked along the first axis and the channels are along the n^{th} axis. Let $\mathbf{j} = (j_1, \ldots, j_n)$ be a multiindex, where each entry is in the range $0 \leq j_l \leq d_l$. We denote by $\mathcal{S}_\mathbf{j}$ a set of multiindices, depending on the multiindex $\mathbf{j}$. This set depends on the normalisation method and will be specified further below. We compute the mean μ and the standard deviation σ of the data by

$$\mu_\mathbf{j} = \frac{1}{|\mathcal{S}_\mathbf{j}|} \sum_{\mathbf{k} \in \mathcal{S}_\mathbf{j}} x_\mathbf{k}, \quad \sigma_\mathbf{j} = \sqrt{\frac{1}{|\mathcal{S}_\mathbf{j}|} \sum_{\mathbf{k} \in \mathcal{S}_\mathbf{j}} (x_\mathbf{k} - \mu_\mathbf{j})^2 + \epsilon}. \tag{1.5}$$

(The small constant ϵ is added for numerical stability.) The data is normalised via the formula $x'_\mathbf{j} = (x_\mathbf{j} - \mu_\mathbf{j})/\sigma_\mathbf{j}$. Finally, the data gets rescaled to $x''_\mathbf{j} = \gamma_\mathbf{j} \mathbf{x}'_\mathbf{j} + \beta_\mathbf{j}$, for some learnable parameter tensors γ and β and sets $\mathcal{L}_\mathbf{j}$ of multiindices.

There are different ways how to choose the indices:
Batch normalisation ([40]): We normalise each channel separately and learn a parameter for each channel. This results in:
$\mathcal{S}_\mathbf{j} = \{\mathbf{k} \mid k_n = j_n\}, \mathcal{L}_\mathbf{j} = \{\mathbf{k} \mid k_l = j_l; 1 \leq l \leq n-1\}$
Layer normalisation ([5]): We normalise each data point separately and learn a parameter for each entry of a data point. This results in:
$\mathcal{S}_\mathbf{j} = \{\mathbf{k} \mid k_1 = j_1\}, \mathcal{L}_\mathbf{j} = \{\mathbf{k} \mid k_1 = j_1\}$
Instance normalisation ([72]): We normalise each data point and each channel separately and learn a parameter for each channel. This results in:
$\mathcal{S}_\mathbf{j} = \{\mathbf{k} \mid k_1 = j_1, k_n = j_n\}, \mathcal{L}_\mathbf{j} = \{\mathbf{k} \mid k_l = j_l; 1 \leq l \leq n-1\}$
Group normalisation ([76]): We normalise each data point and each of G groups of channels separately and learn a parameter for each channel group. This results in:
$\mathcal{S}_\mathbf{j} = \{\mathbf{k} \mid k_1 = j_1, \left\lfloor \frac{k_n}{d_n/G} \right\rfloor = \left\lfloor \frac{j_n}{d_n/G} \right\rfloor\},$
$\mathcal{L}_\mathbf{j} = \{\mathbf{k} \mid k_l = j_l; 1 \leq l \leq n-1; \left\lfloor \frac{k_n}{d_n/G} \right\rfloor = \left\lfloor \frac{j_n}{d_n/G} \right\rfloor\}$

We note that group normalisation becomes layer normalisation for $G = 1$ and instance normalisation for $G = d_n$. In Paper II we find that for convolutional and fully-connected neural networks, instance and layer normalization, respectively, benefit the training of rational activation functions. Batch normalization shows a diminished effect, which may be due to the fact that for each data point, the inputs distributions to the activations functions get distorted by statistics from other data points within the same batch.

6 Activation Functions

Since Paper II and research questions [**AQ2**] and [**AQ1**] are about understanding and developing new parametric activation functions, the material covered in this section is mainly related to that paper and these research questions. However, parts of the general introduction at the beginning can also be deemed relevant for Paper I and Paper III, since these papers study only networks with a specific class of activation functions, namely continuous piecewise linear ones.

6.1 General Overview

The overall mathematical function of a neural network greatly depends on the activation functions of the nodes. In theory, every node can have its own activation function and every function can be used as activation function, but in practice, all nodes of a given layer tend to have the same activation function. According to Theorem 3.1 in [63] (*universal approximation theorem*), every MLP using a non-polynomial continuous activation function σ at all of its nodes can approximate any continuous function. Hence, if we would like to be able to approximate any continuous function, then we are in theory free to use any continuous function as activation function, as long it is not a polynomial.

Traditionally, the *sigmoid* function $x \mapsto \frac{\exp(x)}{1+\exp(x)}$ and the *tangens hyperbolicus* (tanh) function $x \mapsto \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$ were widely used activation functions. However, their derivatives go to 0 for small and large inputs, which is problematic in deep networks, where the use of the chain rule to compute the gradients requires the multiplication of many factors. If these factors are smaller than 1, the resulting derivatives are close to 0 (*vanishing gradients*).

Consequently, the *rectified linear unit (ReLU)* [26, 33, 56] $x \mapsto \max\{x, 0\}$ has become popular since the early 2010s. Since the derivative of the ReLU is constant at 1 for inputs $x > 0$, there is a reduced risk of gradient vanishing. Improved versions of

the ReLU often have a non-zero derivative also for inputs $x < 0$. Notable examples of such activations are the *leaky ReLU* ([49], defined by $x \mapsto \max\{x, \alpha x\}$ for a hyperparameter α), the *exponential linear unit (ELU)* ([13], defined by $x \mapsto x$ for $x > 0$ and $x \mapsto \alpha(\exp(x) - 1)$ for $x < 0$ and for a hyperparameter α) and the *scaled exponential linear unit (SELU)* ([45], defined by $x \mapsto \beta x$ for $x > 0$ and $x \mapsto \beta\alpha(\exp(x) - 1)$ for $x < 0$ for $\alpha \approx 1.67326$ and $\beta \approx 1.0507$). Despite its simple mathematical form, the ReLU and its variants have been widely used, in particular in CNNs.

Since the advent of transformers, there has been a beginning trend towards slightly more mathematically sophisticated activation functions like *GELU* ([36]; $x \cdot \Phi(x)$, where Φ is the cumulative distribution function of the standard normal distribution), *Swish* (also called *SiLU*, [23, 36, 66]; $x \mapsto x \cdot \text{sigmoid}(x)$) and *Mish* ([51]; $x \cdot \tanh(\text{softplus}(x))$).

Almost all of the most widely used activation functions (and all of those mentioned in this section so far) are non-parametric.[3] A notable exception is the *parametric ReLU (PReLU)* [34], defined by $x \mapsto \max\{x, \alpha x\}$ for a learnable parameter α). There are also a small number of other parametric equation functions: [2] proposed *adaptive piecewise linear (APL)* units, which are given by $x \mapsto \max\{x, 0\} + \sum_{j=1}^{S} a_j \max\{0, -x + b_j\}$. [70] proposed *simple piecewise linear and adaptive with symmetric hinges (SPLASH)* units, which are defined by $x \mapsto \sum_{j=1}^{S} \left(a_j^+ \max\{x - b_j, 0\} + a_j^- \max\{-x - b_j, 0\}\right)$. [22] proposed to learn the coefficients of a Fourier sum of sines and cosines and to use the result as an activation function. More recently, [1] proposed an evolutionary population-based search method for finding parametric activation functions. They did not propose a generally suited activation function, but found activations which were optimal for specific neural architectures.

There have also been attempts to train a neural network to learn an activation function: [74] propose *hyperactivations*, which are activation functions represented by so-called activation networks whose parameters are predicted by a *hypernetwork* [32]. However, the activation network and the hypernetwork in turn require a specific choice of activation function ([74] chose a ReLU).

Using the coefficients of polynomials as learnable parameters could turn polynomials into parametric activations. However, apart from not enjoying the universal approximation property mentioned above, polynomial networks have a further disadvantage: the derivatives of polynomials of degree greater than 1 are unbounded, which can lead to optimization problems (*exploding gradients*). In spite of these hindrances, [28] managed to train polynomial networks by using Batch Normalization before the activation.

[3] We note that the version of Swish proposed by [66] included a learnable parameter β which is multiplied with x inside the sigmoid. However, the parametric version does not seem to be commonly used because both TensorFlow's and PyTorch's implementations are non-parametric.

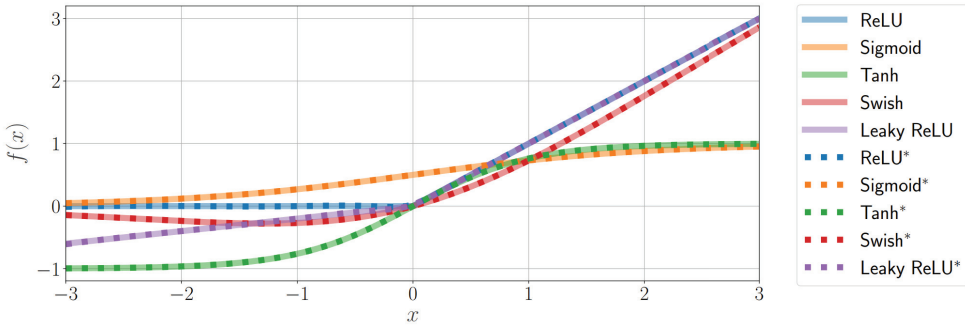


Figure 1.2: Graphs of common activation functions (ReLU, sigomid, tanh, Swish, leaky ReLU) overlaid with the graphs of corresponding rational function approximations (marked with *). Rational functions of a degree 5/4 are able to almost perfectly approximate all standard activations on the real interval [-3, 3]. (Image taken from [52].)

6.2 Rational Activation Functions

Instead of using polynomials directly, [12, 52] found that a more promising approach is to use their quotients, i.e. rational functions. We start with a formal definition:

Definition 6.1

We say that $f : \mathbb{R} \to \mathbb{R}$ is a *rational function* if there are numbers $a_0, \dots a_m, b_0, \dots, b_n \in \mathbb{R}$ such that

$$f(x) = \frac{P(x)}{Q(x)} = \frac{a_0 + a_1 x + \cdots + a_m x^m}{b_0 + b_1 x + \cdots + b_n x^n} \tag{1.6}$$

and $b_j \neq 0$ for at least one j. The numbers a_i, b_j are called the *coefficients* of f. If m and n are the largest indices such that $a_m \neq 0, b_n \neq 0$, we say that $\deg(f) := m/n$ is the *degree* of f.

Rational functions have a decisive advantage over polynomials: If the degree m of the numerator is one larger than the degree n of the denominator, then they behave like linear functions at $\pm\infty$. Hence, their derivatives are bounded and the problem of exploding gradients is solved. Moreover, they enjoy the universal approximation property like all other non-polynomial continuous functions. Another advantage of rational funtions is that even for low degrees (like 3/2 or 5/4) they are able to approximate all standard activation functions. Figure 1.2 shows plots of 5 different activation functions overlaid with their rational approximations on the interval [-3, 3]. It is evident from the plot that these standard activations are virtually identical to their rational approximations. In fact, rational functions are normally not initialised randomly but by approximating a standard activation. The reason for this is that random initialisation can create ill-behaved functions that can be take very large and small values (and which consequently lead to very small or large gradients). [52], [9] and [19] initialise the rational functions

as Leaky ReLUs, ReLUs and identity functions, respectively.

However, general rational functions carry a substantial risk: They are undefined at poles, i.e. points x when the denominator $Q(x)$ is zero. Moreover, the magnitude of the function output (and consequently, also the gradients) blows up close to the poles. In order to avoid this problem [52] proposed to make rational functions "safe" by taking absolute values in the denominator, which resulted in a function where the denominator is always strictly positive. Strictly speaking, such a function is no longer rational and only piecewise smooth. In certain cases, it may, however, be possible to directly use "unsafe" rational functions because [9] did so without observing instabilities. It should be noted, that the latter paper's rational functions only had degree 3/2 as opposed to the degree 5/4 rational functions used in [52]. Rational functions have further been employed by [19] for reinforcement learning. Apart from their use as activation functions, rational functions have also been directly used to represent neural networks by [64]. They constructed a continued fraction, where the denominators inside the fraction were predicted by linear models. Such as representation has the advantage that it is interpretable via continuants and an easily computable power series expansion.

7 Network Symmetries

In this section we formally define what we mean by symmetries in a network and cover existing work in this area. Since the material in this section concerns [**PQ1**], it is particularly relevant for Paper III. We also study a special type of symmetries in Paper IV, so the contents here also relates to that paper.

For a given neural network architecture $\mathcal{N}$, we denote by $\mathcal{F}$ the set of all possible network functions $\mathcal{N}_\theta$ for $\theta \in \Theta$. This obviously gives rise to a function $\mathcal{N}_\bullet : \Theta \to \mathcal{F}$, assigning to each parameter configuration θ the corresponding network function $\mathcal{N}_\theta$. The function $\mathcal{N}_\bullet$ is called a *realisation map* [41] and $\mathcal{N}_\theta$ is called a *realisation* of the architecture $\mathcal{N}$ [60]. The realisation map $\mathcal{N}_\bullet$ is not injective because for any architecture, permuting a node and all of its in- and outgoing weights within a layer keeps the underlying network function unchanged. For any function $f \in \mathcal{F}$, we call an element of the pre-image $\mathcal{N}_\bullet^{-1}(\{f\})$ a *parameterisation* of f. We say that a function $s : \Theta \to \Theta$ is a *symmetry* of $\mathcal{N}$ if $\mathcal{N}_{s(\theta)} = \mathcal{N}_\theta$ for all $\theta \in \Theta$, i.e. if it keeps the network function unchanged. We note that since researchers have studied network symmetries for a very long time, symmetries can be found under various names in the literature: They are called *equioutput transformations*, *function-preserving parameter transformations* and *reparameterisation invariances* by [11], [62] and [42, 61], respectively.

There are two main reasons why understanding the symmetries of a wide range of

architectures is important:

1. *Practical reason*: [57] noticed that the optimisation of a neural network depends not only on the network function, but also on its parameterisation. The value of the loss function itself is completely determined by the network function, but computing the gradients via the chain rule generally leads to different results for different parameterisations. Since [57] found this undesirable, they proposed an adapted version of SGD called Path SGD, which is invariant to reparameterisation. In general, however, it is still unclear which parameterisation is the best one for network training. This is the main subject of investigation of Paper IV.

2. *Theoretical reason*: Properties describing how well a network function generalises must be invariant to any symmetries of the architecture. For example, there exists the hypothesis [37, 43] that parameters generalise well if they correspond to a flat local minimum as opposed to a steep one (where the steepness is measured by the Hessian). However, [20] showed that various common measures of the flatness of the minimum depend on the parameterisation and hence, that sharp minima can also generalise well. For this reason, any measure of generalisation must depend on the network function, not its parameterisation.

We now discuss previous work on network symmetries. There are two wide classes of network symmetries that are applicable for many network architectures:

π Permutation of two neurons, including all incoming and outgoing weights, within a network layer.

ρ_λ Multiplication of all the ingoing weights of a neuron by λ and all of its outgoing weights (after applying the activation function) by $1/\lambda$.

Clearly, the permutation π is a symmetry of any network architecture. Moreover, odd[4] activation functions like the tanh function have permutations ρ_{-1} for which the signs of the weights are flipped. On the other hand, positive homogeneous[5] functions of degree 1 have symmetries ρ_λ for $\lambda > 0$. This includes the widely used ReLU activation and many of its derivations like the leaky ReLU. It is therefore natural to ask if there are any other symmetries apart from the just mentioned ones. Since the tanh activation used to be some of the most widely employed ones, there was initially a lot of focus on them. [69] showed that for 2-layer tanh networks, all analytic[6] symmetries are

[4] A function $f : \mathbb{R} \to \mathbb{R}$ is odd if $f(-x) = -f(x)$.

[5] A function $f : \mathbb{R} \to \mathbb{R}$ is homogeneous of degree a if $f(bx) = b^a f(x)$. It is positive homogeneous, if this only holds for all $b > 0$.

[6] A function $f : \mathbb{R} \to \mathbb{R}$ is analytic if for any $x \in \mathbb{R}$, f can be written as a power series on an open neighbourhood of x.

concatenations of permutations π and sign flips ρ_{-1}. [11] consequently extended this result to networks of arbitrary depth. Other notable work in this field includes [3], who produced the same result for infinitely differentiable activation functions σ with $\sigma(0) = \sigma'(0) = \sigma''(0) = 0$. Finally, [46] proved a corresponding result for 2-layer networks with sigmoid-activations.
So far, there has only been little previous work involving the ReLU activation. A notable exception is [62] who showed that for ReLU networks of decreasing width, symmetries which are concatentations of π and ρ_λ are indeed the only ones.

8 Piecewise-Linear Neural Networks

In this section, we introduce the necessary background for Paper I, which studies the effect of network parameterisation on linear regions. Since TropEx, the algorithm that the paper proposes, relies heavily on tropical algebra, we start with a brief introduction of some core tropical terminology and refer to [50] for a more in-depth introduction. We use the remainder of the section to give an overview of previous work specific to linear regions.

8.1 Tropical Algebra

8.1.1 Terminology

In the *tropical* analogue of elementary algebra, ordinary addition and multiplication are replaced by *tropical addition* and *tropical multiplication*, which in turn are just fancy names for maximum and addition. This yields a precise framework for studying mathematical functions where such operations appear frequently. We start by defining the central object of tropical algebra:

Definition 8.1

The tropical semiring *tropical semiring* is given by the tuple $\mathbb{T} = (\mathbb{R} \cup \{-\infty\}, \oplus, \odot)$, where $\oplus$ is called tropical addition and is defined by $x \oplus y = \max\{x, y\}$ and $\odot$ is called tropical multiplication and is defined by $x \odot y = x + y$.

Instead of this *max-plus algebra* it is also possible to make an analogous definition of a *min-plus algebra*, where $\oplus$ is defined by $x \oplus y = \min\{x, y\}$ and $-\infty$ is replaced by ∞. We will, however, stick to the definition above since it is more convenient when dealing with neural networks. The tuple $\mathbb{T}$ is a semiring (in fact even a semifield) because there is no inverse to tropical addition, but all other axioms of a ring are fulfilled.

The multiplicative inverse of $\mathbb{T}$, denoted by $\oslash$, is the same as ordinary subtraction. Similarly to addition and multiplication, it is also possible to define a *tropical power*, which is just repeated tropical multiplication, i.e. repeated ordinary addition which is ordinary multiplication. Hence we define $x^{\odot a} = a \cdot x$. We will slightly abuse the notation and we will omit the $\odot$-sign when referring to the tropical power in this subsection.

The same way as (ordinary) algebraic geometry studies polynomials, tropical algebraic geometry studies corresponding objects called tropical polynomials. These are defined in a completely analogous manner to their ordinary counterparts:

Definition 8.2

A *tropical monomial* in d variables is a an expression of the form $c \odot x_1^{a_1} \odot x_2^{a_2} \odot \cdots \odot x_d^{a_d}$, where $c \in \mathbb{T}$ and $a_1, \ldots, a_n \in \mathbb{Z}_{\geq 0}$. For improved readability, it can often be advisible to write the tropical monomial as $cx_1^{a_1}x_2^{a_2}\ldots x_d^{a_d}$ or even as $c\mathbf{x}^{\alpha}$, where $\alpha = (a_1, \ldots, a_d) \in \mathbb{Z}_{\geq 0}^d$ and $\mathbf{x} = (x_1, \ldots, x_d)$.
A *tropical polynomial* in d variables is a finite tropical sum $p(\mathbf{x}) = c_1\mathbf{x}^{\alpha_1} \oplus \cdots \oplus c_r\mathbf{x}^{\alpha_r}$ of tropical monomials. Without loss of generality, it is possible to assume that each term appears only once in the sum, i.e. that $\alpha_j \neq \alpha_k$ for $j \neq k$.
A *tropical rational function* is the tropical quotient $\nu(\mathbf{x}) = p(\mathbf{x}) \oslash q(\mathbf{x}) = p(\mathbf{x}) - q(\mathbf{x})$ of two tropical polynomials p and q.
A *tropical rational map* is a function $\nu : \mathbb{R}^d \to \mathbb{R}^C$, given by $\mathbf{x} \mapsto (\nu_1(\mathbf{x}), \ldots, \nu_s(\mathbf{x})$, where each ν_j is a tropical rational function.

We note that in ordinary notation, a tropical monomial is just the same as an affine function, a tropical polynomial the same as the maximum of multiple affine functions and a tropical rational function the difference of maxima of affine functions. It is also possible to define *generalised* versions of the above objects by allowing the tropical powers to be elements of $\mathbb{R}_{\geq 0}$ instead of only $\mathbb{Z}_{\geq 0}$.

8.1.2 Relation to Deep Learning

Given that ReLU-networks only consist of additions, multiplications and taking maxima, it should not come as a surprise that such networks can be studied using the tools of tropical algebra. The intimate relation between piecewise-linear networks and tropical mathematics was first discovered independently by [81] and [10]. We now present the most important theoretical result, which says that under some mild assumptions, piecewise-linear neural networks are exactly the same as tropical rational functions:

Theorem 8.3 (***Theorem 5.4 (i), [81]***)

Let $\nu : \mathbb{R}^d \to \mathbb{R}$. Then ν is a tropical rational function if and only if ν is a feedforward neural network satisfying the following assumptions:

1. *the weight matrices are integer-valued;*
2. *the bias vectors are real-valued;*
3. *for each layer, the activation functions take the form $\sigma_l(\mathbf{x}) = \max\{\mathbf{x}, \mathbf{a}_l\}$ for some threshold vector $\mathbf{a}_l \in \mathbb{R}^{d_l}$.*

We note that assumption (1) is indeed mild because real numbers can be arbitrarily closely approximated by rational numbers and rational weight matrices can be seen as equivalent to integer ones (cf. [81], p. 5).

8.2 Linear Regions

Note: *In order to enhance readability and avoid becoming unnecessarily verbose, we always mean a network with ReLU activation functions, whenever we use the word* network *in this subsection.*

We start with a formal definition of linear regions:

Definition 8.4

A *linear region* of a neural network $\mathcal{N}_\theta : \mathbb{R}^d \to \mathbb{R}^{d'}$ is a maximal connected subset of $\mathbb{R}^d$ on which $\mathcal{N}_\theta$ is linear.

There is also a related concept called the activation pattern:

Definition 8.5

We say that the neuron $x \in \mathcal{N}_\theta$ coming out of the ReLU activation function is *activated* if $x > 0$ and *inactive* otherwise (i.e. $x = 0$). If we impose some ordering on the network's neurons, we can consider the *activation pattern* $\mathcal{A}_{\mathcal{N}_\theta}(\mathbf{x}) \in \{0,1\}^{N_n}$ of a network at a data point $\mathbf{x}$ to be a vector where the i^{th} entry is equal to 1 if neuron i is activated and 0 otherwise.

We note that the number of linear regions is bounded from above by the number of activation patterns because different linear regions imply different activation patterns, but not vice versa. (Consider the 2-layer ReLU network with two hidden neurons implementing the identity function on $\mathbb{R}$ via weight vectors $(1,-1)$ and $(1,-1)^T$, which has two activation patterns but is linear on all of $\mathbb{R}$.) If the input space is d-dimensional,

then the boundaries between linear regions are $(d-1)$-dimensional hyperplanes that create a partition of the input space. The number of linear regions has for a long time been considered as a measure of the expressivity of the network. For this reason, a great deal of work has gone into obtaining estimates of the number of linear regions [4, 53, 54, 59, 65, 67, 78, 81]. However, it is intractable to perform an exact computation of the number of linear regions. Therefore, all effort has gone into deriving upper and lower bounds to the number of linear regions. To the best of my knowledge, [67] obtained the best upper and lower bounds: For a neural network of N_ℓ layers with d_l many neurons in layer l and input dimension d_0, the number of activation patterns is bounded from above by

$$\sum_{\mathbf{j}\in J}\prod_{l=1}^{N_\ell}\binom{d_l}{j_l} \tag{1.7}$$

where

$$J = \{\mathbf{j} \in \mathbb{Z}^{N_\ell} \mid 0 \leq j_l \leq \min\{d_0, d_1 - j_1, \ldots, d_{l-1} - j_{l-1}, d_l\}, \forall l = 1, \ldots, N_\ell\}.$$

Under the assumption that $d_l \geq 3d_0$ for all $l \leq N_\ell - 1$, a lower bound for the maximal (over all possible weight configurations) number of linear regions is given by

$$\left(\prod_{l=1}^{N_\ell-2}\left(\left\lfloor\frac{d_l}{d_0}\right\rfloor + 1\right)^{d_0}\right)\sum_{j=0}^{d_0}\binom{d_{N_\ell-1}}{j}. \tag{1.8}$$

[65] showed these bounds are asymptotically tight of order

$$\mathcal{O}(\min\{d_l \mid l = 1, \ldots, L\}^{d_0 N_\ell}).$$

The upper and lower bounds grow linearly in width and exponentially in depth, which captures the common understanding that deep and narrow networks are more expressive than shallow and wide networks. However, there is a problem with these numbers: They are astronomically high and exceed the number of atoms in the known universe ($\sim 10^{83}$) even for very small networks. On the other hand, even relatively large training datasets like ImageNet are much smaller and contain only some 10^6 datapoints. For this reason it is unclear whether the number of linear regions is actually a suitable predictor of the suitability of a network and, more generally, how information passes from the few linear regions containing training samples to the vast majority of regions without training data.

9 Network Calibration

The Cooling technique introduced in Paper IV is built on *temperature scaling*, a technique from network calibration. For this reason, we provide an overview of relevant material on calibration in this section. At first, we formally define calibration and how to measure it. In the second subsection, we describe the temperature scaling method and its generalisations, matrix and vector scaling.

9.1 Measuring (Mis-)Calibration

We would like classification neural networks to be able to estimate the likelihood of making correct predictions. That is, we would like the predicted output probabilities $\widehat{\mathbf{y}}$ to correspond to true probabilities. For this to hold, a network making 100 predictions, each with 90% confidence, should make 90% correct classifications. More precisely, let us treat both input $\mathbf{X}_0 \in \mathcal{X}$, the corresponding output $\widehat{\mathbf{Y}} = \mathcal{N}(\mathbf{X}_0)$ and the predicted label $\widehat{K}$ as random variables. We let $\widehat{P} = \max_j \widehat{Y}_j$ be the confidence of the network. Then we say that a network is *perfectly calibrated* if

$$\mathbb{P}(\widehat{K} = k \mid \widehat{P} = p) = p \tag{1.9}$$

for all probabilities $p \in [0, 1]$. We will now introduce metrics from [30, 58] to measure how well a network is calibrated. We start by ordering the elements $\mathbf{x}$ of the dataset $\mathcal{X}_{\text{val}}$ by their corresponding network confidences $\widehat{p}$. Variables with a subscript i correspond to datapoint i after ordering. We define B_j to be the set of datapoint indices for which the confidence falls into the interval $(\frac{j-1}{N_b}, \frac{j}{N_b}]$. If we let

$$\operatorname{acc}(B_j) = \frac{1}{|B_j|} \sum_{i \in B_j} \delta_{\widehat{k}_i k_i} \tag{1.10}$$

$$\operatorname{conf}(B_j) = \frac{1}{|B_j|} \sum_{i \in B_j} \widehat{p}_i \tag{1.11}$$

be the average accuracy and the average confidence, respectively, of bin B_j, then equations (1.10) and (1.11) provide estimates for the left and the right hand side of equation (1.9), respectively. (δ denotes the Kronecker delta here.) Hence, if a model is perfectly calibrated, then $\operatorname{acc}(B_j) = \operatorname{conf}(B_j)$ for all j. Given that equation (1.9) specifies perfect calibration, it makes sense to define a quantity called the *calibration error* by

$$\operatorname{CE}(\widehat{P}) = \left| \mathbb{P}(\widehat{K} = k \mid \widehat{P}) - \widehat{P} \right|. \tag{1.12}$$

However, the calibration error is itself a random variable, depending on $\widehat{P}$. To obtain a scalar estimate of the calibration error, we can take the average (or, for safety-critical

applications, the maximum) over the data distribution. Using the $\mathrm{acc}-$ and $\mathrm{conf}-$ estimators from above, we thus define the *expected calibration error (ECE)* on a dataset $\mathcal{X}$ as

$$\mathrm{ECE} = \mathbb{E}_{\widehat{P}}(\mathrm{CE}(\widehat{P})) \approx \sum_{j=1}^{N_b} \frac{|B_j|}{|\mathcal{X}|} \left|\mathrm{acc}(B_j) - \mathrm{conf}(B_j)\right|; \tag{1.13}$$

and the *maximum calibration error (MCE)* as

$$\mathrm{MCE} = \max_{\widehat{P}}(\mathrm{CE}(\widehat{P})) \approx \max_{j} \frac{|B_j|}{|\mathcal{X}|} \left|\mathrm{acc}(B_j) - \mathrm{conf}(B_j)\right|. \tag{1.14}$$

[30] observed that for ResNets, the network capacity is indirectly proportional to the ECE, i.e. making the network deeper or wider or adding batch normalization increases the calibration error, while decreasing the network error. On the other hand, models regularized with strong weight decay displayed smaller calibration errors. The authors also noted that the increase of network capacity allowed the network to overfit to the negative log likelihood loss without overfitting to the test error: Larger models produce smaller test errors, but larger negative log likelihoods on the test data. In the classical understanding of learning theory, large models without regularization will overfit for the data. However, for ResNets, it seems that this overfitting is only happening with respect to the log likelihood and not with respect to the test error (cf. [80]).

9.2 Calibration Methods

There are a range of calibration methods for binary models, including histogram binning, isotonic regression, Bayesin binning into quantiles, and Platt scaling. (For more information on these methods, please confer [30].) More interestingly, apart from extensions of binning methods, [30] suggest *matrix scaling* for multiclass models. Here the output logits $\mathbf{z}$ are replaced by affinely transformed logits given by $\mathbf{W}\mathbf{z}+\mathbf{b}$ for a matrix $\mathbf{W}$ and a vector $\mathbf{b}$. The matrix and the vector consist of parameters that optimise the negative log likelihood on the validation dataset $\mathcal{X}_{\text{val}}$. Based on matrix scaling, [30] also propose two simpler variants: *vector scaling*, where the matrix $\mathbf{W}$ is diagonal and *temperature scaling*, where $\mathbf{W}$ is replaced by a single scalar $\tau > 0$ and there is no bias vector $\mathbf{b}$. Interestingly, it turns out that despite being the simplest variant, temperature scaling gives the best results in terms of ECE and MCE. Matrix and vector scaling have the disadvantage that they could possibly overfit to small validation datasets because they require more parameters. For temperature scaling, when $\tau = 1$, there is no scaling happening. As $\tau \to 0$, the modified output probabilities go to $\frac{1}{C}$ (where C is the number of labels). On the other hand, for $\tau \to \infty$, the maximum probability approaching 1 and the non-maximum probabilities go to 0. It should also be noted that since temperature scaling multiplies all logits by the same number, the maximum is unchanged and consequently, the accuracy of the network is not affected.

10 Contributions and Outlook

This section summarises the contributions of the authors to each of the papers and the contributions of the papers to the each of the research questions in Section 3. Moreover, we present an outlook on potential future work on each of the topics we studied.

The co-authors are abbreviated as follows: Cristian Sminchisescu (CS), Henning Petzka (HP), Martin Trimmel (MT), Mihai Zanfir (MZ), Richard Hartley (RH).

Paper I: TropEx: An Algorithm for Extracting Linear Terms in Deep Neural Networks

Martin Trimmel*, Henning Petzka*, Cristian Sminchisescu, *International Conference on Learning Representations (ICLR)*, Virtual Conference, 2021

Author contributions The project was conceived by HP. The method was developed by MT with continuous feedback from HP. MT implemented the method. Most of the experiments were run by MT, some also by HP. HP and MT wrote the manuscript with feedback from CS. Overall, HP and MT contributed equally to the project.

Paper contributions This paper studies the interplay between the parameterisation of the network and its linear regions and attempts to answer [**PQ2**] and [**PQ3**]. The main theoretical contribution is the derivation of an algorithm called TropEx, which uses **trop**ical algebra to **ex**tract the coefficients of linear regions of neural networks. We used TropEx to perform a range of experiments: First, we found that as a general rule, all training and test datapoints lie on different linear regions and hence that network generalisation cannot be explained by test samples inducing the same activation pattern as the training samples. We also noticed that the Euclidean norms and angles between the coefficients of different regions are not close to 0, so generalisation cannot be explained by similar activation patterns either. TropEx produces a tropical rational function with a drastically reduced number of linear regions (one for each sample instead of the original number which exceeded the number of atoms in the universe). We applied this tropical function on the MNIST and CIFAR10 test sets and noticed a stark difference between convolutional (CNNs) and fully connected neural networks (FCNs). The accuracy suffers a lot for CNNs and is much more stable for FCNs. This holds even for two networks that have the same number of neurons after each parameter layer. We also attempted to compute an estimate of the number of linear regions for these two networks and the CNN seems to have more of them, which contradicted our intuition, which

was based on measuing network expressivity in terms of the maximal number of linear regions. Further experiments indicated that the linear terms of the CNNs are also more diverse than those of the FCNs and that there is a benefit to shift the focus away from purely counting linear regions to attempt to achieve a deeper understanding of their interplay.

Outlook There are multiple avenues for future work: First, it would be interesting to see how robust to adversarial attacks the rational function extracted by TropEx is. In previous work, [16] proposed a loss function which led to enlarged linear regions and consequently to adversarial robustness. Since TropEx produces maximally enlarged linear regions, it is a natural candidate for further investigations in this direction. Another option would be to use TropEx for network pruning. Since tropical rational maps are the same as ReLU network functions by [81], using TropEx produced a new neural function. From the perspective of linear regions, this is a minimal function which is unchanged on all training data. However, there are two possible problems for actual pruning applications: First, the tropical extraction caused a considerable drop in test accuracy and secondly, the extracted function actually needs more space to be stored on a computer. These issues would have to be overcome for any practical applications.

Paper II: ERA: Enhanced Rational Activations

Martin Trimmel*, Mihai Zanfir*, Richard Hartley, Cristian Sminchisescu, *European Conference on Computer Vision (ECCV)*, Tel Aviv, Israel, 2022

Author contributions The project was conceived by MZ. The method was developed by MT with feedback from MZ. MT implemented the method. The experiments were run by MZ and by MT. Most of the manuscript was written by MT and one section was also written by MZ. RH and CS gave feedback on the manuscript. Overall, MZ and MT contributed equally to the project.

Paper contributions This paper aims to both improve rational functions in terms of performance and stability (answering [**AQ2**]) and to study possible use cases where they outperform conventional, non-parametric activation functions (answering [**AQ1**]). We found that there are multiple ways how rational functions can improved: (1) As opposed to previous approaches [9, 18, 52] that applied an absolute value in the denominator or that were unsafe with regards to poles, the proposed ERA function has a factorised denominator which enables the use of partial fractions. (2) Whereas previous papers

mainly relied on ReLU or Leaky ReLU initialisation, we found that intialising ERAs as Swish functions yields better results. (3) The training of rational functions can be greatly stabilised by normalising their input with standard normalisation techniques like layer normalisation or instance normalisation. (4) It is possible to even randomly initialise rational activation functions if gradient clipping is used to prevent large gradient causing divergence in pathological cases. Randomly initialised activation can almost get as good performance as Swish-initialised ones. Previous work [52] applied rational activation functions mainly in larger networks, where they made hardly any difference when being compared to other activation functions. In contrast, we found that ERAs are particularly benefitial in small networks, where they can increase the expressive power of the network without adding a significant amount of parameters. MLP-Mixers and Transformers trained for human pose and shape reconstruction achieve a per joint error almost as low as the state-of-the-art while having only 3.9M parameters instead of the 25M used in the original THUNDR [79] network.

Outlook We believe that future work on rational activations should focus on improving them further and finding more use cases where they trump non-parametric activation functions. [7] attempted to improve rational activation functions by using an orthogonal basis for the polynomial. Even though they perform better than [52], it is unclear if they are perferable over ERAs. We note that their approach and the ERA approach is mutually exclusive because to the best of our knowledge, it is not possible to have a factorised representation for polynomials using the orthogonal bases proposed in [7]. Another issue is that it may take longer to evaluate polynomials with an orthogonal basis because the values of multiple basis polynomials need to be computed. There are indications that rational activation functions could be benefitial for adversarial robustness: [17, 77] noted that smooth activation functions lead to more adversarially robust networks than non-smooth ones. It was also noted by [70] that a large second derivative of the activation function can help against adversarial attacks. Moreover, [82] found that symmetric activation functions tend to be more robust than non-symmetric ones. By putting restrictions on the parameter space, it can be easy to construct a rational activation function which is both smooth and symmetric and which has a large second derivative. Research in this direction could lead to networks that are less sensitive to adversarial examples.

Paper III: Notes on the Symmetries of 2-Layer ReLU-Networks

Henning Petzka, **Martin Trimmel**, Cristian Sminchisescu, *Northern Lights Deep Learning Conference (NLDL)*, Tromsø, Norway, 2020

Author contributions The project was conceived by HP. The theory was developed and most of the manuscript was written by HP, and both with contributions also from MT. CS gave feedback on the manuscript.

Paper contributions The paper provides a partial answer to question [**PQ1**] classifies the symmetries of 2-layer ReLU networks. Both weight-dependent symmetries and general symmetries, which are applicable to all networks, regardless of their weights, are considered. The paper shows that the general symmetries of 2-layer ReLU networks are exactly the permutations π and the scalar multiplications ρ_λ defined in Section II. For 2-layer networks, the paper also gives a precise description of the pathological parameter regions with non-trivial symmetries. The total region with weight-dependent symmetries turns out to be rather small, having Lebesgue measure 0. In deeper networks, the situation is more complicated and the paper shows that for such networks, there are regions of the parameter space of Lebesgue measure greater than 0 that suffer from weight-dependent symmetries.

Outlook There is a range of obvious ways to continue exploring network symmetries: The evident next step from our work would be to find and classify all the symmetries for ReLU networks of arbitrary depth. Another option would be to study the symmetries of new activation functions like Swish and GELU or to study the symmetries of of more complicated models like Transformers.

Paper IV: How to Keep Cool While Training

Martin Trimmel, Mihai Zanfir, Richard Hartley, Cristian Sminchisescu, *Preprint, April 2023*

Author contributions The project was conceived by RH. Most of the method was developed by RH, with some contributions also from MT. MT implemented the method. Most of the experiments were run by MT, with some also run by MZ. Most of the manuscript was written by MT, with contributions also from RH. RH and CS gave feedback on the manuscript.

Paper contributions This final paper studies [**PQ4**], i.e. how the parameterisation of a network affects its training. To start with, the paper presents a theoretical analysis of how scaling reparameterisation and a final scaling layer affect the backpropagated gradients. Scaling by a constant $\tau < 1$ at the end of the network produces more evenly distributed

("smoothed") gradients and scaling the linear layers by scalars $\beta_1 > \beta_2 > \cdots > \beta_N$ has the effect of increasing the magnitudes of the gradients. Motivated by this analysis, the paper proposes a Cooling method for adaptively reparameterising classification neural networks during training. Cooling is based on the commonly used temperature scaling method for network calibration and like the latter, it optimises a temperature parameter τ on a small validation dataset. This parameter is then used in a final scaling layer or redistributed to earlier layers to produce a sequence of scalars $\beta_1, \ldots, \beta_N$. The parameters τ and β_i fulfill the assumptions of the gradient analysis above and hence have the same effect as predict above. When plotting the magnitudes of the gradients, it is evident that Cooling can have a similar effect to common learning rate schedules on the gradients and hence, that it can be seen as a "data-dependent" learning rate schdule. Experiments on image classification and semantic segmentation datasets show that Cooling can outperform many of the most commonly used learning rate schedules, like piecewise constant schedules, exponential decay and cosine schedules. The effect of Cooling depends on how much of the temperature is being distributed back to reparameterise the earlier layers of the network. Keeping the temperature as a final scaling layer produces worse-calibrated, but better performant networks. On the other hand, reparamterising the earlier layers and not having a final scaling layers performs better than learning rate schedules and also improves the calibration of the network.

Outlook Cooling raises a number of interesting questions. Since simply reparameterising a network can have such a strong effect, what is the ideal parameterisation for network training? It is well-known that for successful network training, both the direction and the norms of the gradients are important. The paper begs the question: What should the gradient norms ideally be? There could also be more work on how should the β_i-parameters mentioned above should look like. The simply chose $\beta_i = \tau^{i/L}$, but a more sophisticated redistribution of the temperature may bring further benefits. Interestingly, [75] observed in their work that letting the loss depend only on the direction of the logits, not on their magnitudes, produced better calibrated networks. Hence, it would be exciting to see whether combing their approach with ours would give even greater benefits.

Index

activated, 22
activation function, 11
Activation functions, 4
activation pattern, 22
adaptive piecewise linear (APL), 16

backpropagation, 12
batch, 12
bias vector, 11

calibration error, 24
child, 9
classification, 8
clustering, 8
coefficients, 17
complexity, 9
computational graph, 10
convolutional, 11
convolutional neural network (CNN), 11
covariate shift, 14

data augmentation, 13
deep, 11
deep learning, 4
degree, 17
dense, 11
dimensionality reduction, 8
dropout, 13

early stopping, 13
expected calibration error (ECE), 25
exploding gradients, 16
exponential linear unit (ELU), 16

feedforward neural network, 11
fully-connected, 11

GELU, 16
generative modelling, 8
gradient descent, 12

hidden nodes, 10
hyperactivations, 16
hypernetwork, 16

inactive, 22
input nodes, 10

label smoothing, 13
layer, 11
leaky ReLU, 16
learning rate, 12
learning rate schedule, 13
learning rate warmup, 13
leave, 9
linear region, 22
linear regions, 4
local functions, 10
loss function, 8

matrix scaling, 25
maximum calibration error (MCE), 25
Mish, 16

multi-layer perceptron (MLP), 11

network function, 10
Network parameterisation, 4
neural network, 10
neural network architecture, 10
neuron, 10
normalisation techniques, 14

output nodes, 10
overfitting, 9

parameterisation, 18
parametric ReLU (PReLU), 16
parent, 9
perfectly calibrated, 24

rational function, 17
realisation, 18
realisation map, 18
rectified linear unit (ReLU), 15
recurrent neural network (RNN), 11
regression, 8
regularisation, 13
root, 9

scaled exponential linear unit (SELU), 16
shallow, 11
sigmoid, 15
SiLU, 16
simple piecewise linear and adaptive with symmetric hinges (SPLASH), 16
stochastic gradient descent (SGD), 12
Swish, 16
symmetries, 4
symmetry, 18

tangens hyperbolicus, 15
target function, 8
temperature scaling, 25
tensors, 9
test set, 9
training, 12
training dataset, 8
tropical, 20
tropical addition, 20
tropical monomial, 21
tropical multiplication, 20
tropical polynomial, 21
tropical power, 21
tropical rational function, 21
tropical rational map, 21
tropical semiring, 20

underfitting, 9
universal approximation theorem, 15

validation dataset, 8
vanishing gradients, 15
vector scaling, 25

weight decay, 13
weight matrix, 11

References

[1] Discovering parametric activation functions. *Neural Networks*, 148:48–65, 2022. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2022.01.001. URL https://www.sciencedirect.com/science/article/pii/S0893608022000016.

[2] F. Agostinelli, M. D. Hoffman, P. Sadowski, and P. Baldi. Learning activation functions to improve deep neural networks. *Workshop Track Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[3] F. Albertini and E. D. Sontag. Uniqueness of weights for neural networks. In *Artificial Neural Networks with Applications in Speech and Vision*. Chapman & Hall, 1993.

[4] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee. Understanding deep neural networks with rectified linear units. In *Proceedings of the 4th International Conference on Learning Representations*, 2016.

[5] J. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *ArXiv*, abs/1607.06450, 2016.

[6] J. Berner, P. Grohs, G. Kutyniok, and P. Petersen. The modern mathematics of deep learning. *CoRR*, abs/2105.04026, 2021. URL https://arxiv.org/abs/2105.04026.

[7] K. Biswas, S. Banerjee, and A. K. Pandey. Orthogonal-padé activation functions: Trainable activation functions for smooth and faster convergence in deep networks. *CoRR*, abs/2106.09693, 2021. URL https://arxiv.org/abs/2106.09693.

[8] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 144–152, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 089791497X. doi: 10.1145/130385.130401. URL https://doi.org/10.1145/130385.130401.

[9] N. Boulle, Y. Nakatsukasa, and A. Townsend. Rational neural networks, 2020. URL https://proceedings.neurips.cc/paper/2020/file/a3f390d88e4c41f2747bfa2f1b5f87db-Paper.pdf.

[10] V. Charisopoulos and P. Maragos. A tropical approach to neural networks with piecewise linear activations. *arXiv preprint arXiv:1805.08749*, 2018.

[11] A. M. Chen, H.-m. Lu, and R. Hecht-Nielsen. On the geometry of feedforward neural network error surfaces. In *Neural computation*. MIT Press, 1993.

[12] Z. Chen, F. Chen, R. Lai, X. Zhang, and C. Lu. Rational neural networks for approximating jump discontinuities of graph convolution operator. *CoRR*, abs/1808.10073, 2018. URL http://arxiv.org/abs/1808.10073.

[13] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/pdf/1511.07289.pdf.

[14] M. Collins. Computational graphs, and backpropagation. URL http://www.cs.columbia.edu/~mcollins/ff2.pdf.

[15] D. R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, 20(2):215–242, 1958. ISSN 00359246. URL http://www.jstor.org/stable/2983890.

[16] F. Croce, M. Andriushchenko, and M. Hein. Provable robustness of relu networks via maximization of linear regions. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2018.

[17] S. Dai, S. Mahloujifar, and P. Mittal. Parameterizing activation functions for adversarial robustness. In *2022 IEEE Security and Privacy Workshops (SPW)*, pages 80–87, 2022. doi: 10.1109/SPW54247.2022.9833884.

[18] Q. Delfosse, P. Schramowski, A. Molina, N. Beck, T.-Y. Hsu, Y. Kashef, S. Rüling-Cachay, and J. Zimmermann. Rational activation functions, 2020.

[19] Q. Delfosse, P. Schramowski, A. Molina, and K. Kersting. Recurrent rational networks. *CoRR*, abs/2102.09407, 2021. URL https://arxiv.org/abs/2102.09407.

[20] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio. Sharp minima can generalize for deep nets. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1019–1028. JMLR. org, 2017.

[21] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *Journal of Machine Learning Research (JMLR)*, 2011.

[22] C. Eisenach, Z. Wang, and H. Liu. Nonparametrically learning activation functions in deep neural nets. *Workshop Track Proceedings of the 5th International Conference on Learning Representations*, 2017.

[23] S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *CoRR*, abs/1702.03118, 2017. URL http://arxiv.org/abs/1702.03118.

[24] E. Fiesler. Neural network classification and formalization. *Computer Standards & Interfaces*, 16(3):231–239, 1994. ISSN 0920-5489. doi: https://doi.org/10.1016/0920-5489(94)90014-0. URL https://www.sciencedirect.com/science/article/pii/0920548994900140.

[25] E. M. Fisher. Linear discriminant analysis. *Statistics & Discrete Methods of Data Sciences*, 392:1–5, 1936.

[26] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[27] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[28] M. Goyal, R. Goyal, and B. Lall. Improved polynomial neural networks with normalised activations. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020. doi: 10.1109/IJCNN48605.2020.9207535.

[29] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. URL http://arxiv.org/abs/1706.02677.

[30] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/guo17a.html.

[31] E. Guresen and G. Kayakutlu. Definition of artificial neural networks with comparison to other networks. *Procedia Computer Science*, 3:426–433, 2011. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2010.12.071. URL https://www.sciencedirect.com/science/article/pii/S1877050910004461. World Conference on Information Technology.

[32] D. Ha, A. M. Dai, and Q. V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkpACe1lx.

[33] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature volume 405*, 2000.

[34] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[35] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2016.

[36] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[37] S. Hochreiter and J. Schmidhuber. Flat Minima. *Neural Computation*, 9(1):1–42, 01 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.1.1. URL https://doi.org/10.1162/neco.1997.9.1.1.

[38] A. E. Hoerl and R. W. Kennard. Ridge regression: Applications to nonorthogonal problems. *Technometrics*, 12(1):69–82, 1970. doi: 10.1080/00401706.1970.10488635. URL https://www.tandfonline.com/doi/abs/10.1080/00401706.1970.10488635.

[39] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970. doi: 10.1080/00401706.1970.10488634. URL https://www.tandfonline.com/doi/abs/10.1080/00401706.1970.10488634.

[40] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL http://arxiv.org/abs/1502.03167.

[41] A. Jacot, F. Gabriel, and C. Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/5a4be1fa34e62bb8a6ec6b91d2462f5a-Paper.pdf.

[42] C. Jang, S. Lee, F. C. Park, and Y.-K. Noh. A reparametrization-invariant sharpness measure based on information geometry. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=AVh_HTC76u.

[43] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=H1oyRlYgg.

[44] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[45] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/5d44ee6f2c3f71b73125876103c8f6c4-Paper.pdf.

[46] V. Kurková and P. C. Kainen. Functionally equivalent feedforward neural networks. *Neural Computation*, 6(3):543–558, 1994.

[47] S. Linnainmaa. Algoritmin kumulatiivinen pyöristysvirhe yksittäisten pyöristysvirheiden, 1970.

[48] I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

[49] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models, 2013.

[50] D. Maclagan and B. Sturmfels. *Introduction to Tropical Geometry*. Graduate Studies in Mathematics, vol. 161, AMS, 2015.

[51] D. Misra. Mish: A self regularized non-monotonic neural activation function. *arXiv preprint arXiv:1908.08681*, 2019.

[52] A. Molina, P. Schramowski, and K. Kersting. Padé activation units: End-to-end learning of flexible activation functions in deep networks, 2019.

[53] G. Montúfar. Notes on the number of linear regions of deep neural networks. *Presented at Mathematics of Deep Learning, Sampling Theory and Applications*, 2017.

[54] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems*, 2014.

[55] R. Müller, S. Kornblith, and G. E. Hinton. When does label smoothing help? In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/f1748d6b0fd9d439f71450117eba2725-Paper.pdf.

[56] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines, 2010.

[57] B. Neyshabur, R. R. Salakhutdinov, and N. Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/eaa32c96f620053cf442ad32258076b9-Paper.pdf.

[58] M. Pakdaman Naeini, G. Cooper, and M. Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), Feb. 2015. doi: 10.1609/aaai.v29i1.9602. URL https://ojs.aaai.org/index.php/AAAI/article/view/9602.

[59] R. Pascanu, G. Montufar, and Y. Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv:1312.6098*, 2013.

[60] P. Petersen and F. Voigtlaender. Optimal approximation of piecewise smooth functions using deep relu neural networks. *Neural Networks*, 108:296–330, 2018. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2018.08.019. URL https://www.sciencedirect.com/science/article/pii/S0893608018302454.

[61] H. Petzka, L. Adilova, M. Kamp, and C. Sminchisescu. A reparameterization-invariant flatness measure for deep neural networks. *CoRR*, abs/1912.00058, 2019. URL http://arxiv.org/abs/1912.00058.

[62] M. Phuong and C. H. Lampert. Functional vs. parametric equivalence of re{lu} networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Bylx-TNKvH.

[63] A. Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999. doi: 10.1017/S0962492900002919.

[64] I. Puri, A. Dhurandhar, T. Pedapati, K. Shanmugam, D. Wei, and K. R. Varshney. Cofrnets: Interpretable neural architecture inspired by continued fractions, 2021. URL https://openreview.net/forum?id=kGXlIEQgvC.

[65] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein. On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[66] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions, 2018. URL https://openreview.net/forum?id=SkBYYyZRZ.

[67] T. Serra, C. Tjandraatmadja, and S. Ramalingam. Bounding and counting linear regions of deep neural networks. *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[68] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[69] H. J. Sussmann. Uniqueness of the weights for minimal feedforward nets with a given input-output map. In *Neural network*. Elsevier, 1992.

[70] M. Tavakoli, F. Agostinelli, and P. Baldi. Splash: Learnable activation functions for improving accuracy and adversarial robustness. *Neural Networks*, 140: 1–12, 2021. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2021.02.023. URL https://www.sciencedirect.com/science/article/pii/S0893608021000733.

[71] T. Tieleman and G. Hinton. Lecture 6.5 - rmsprop, coursera: Neural networks for machine learning, 2012.

[72] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016. URL http://arxiv.org/abs/1607.08022.

[73] V. N. Vapnik and A. Y. Lerner. Recognition of patterns with help of generalized portraits. *Avtomat. i Telemekh.*, 24:6:774–780, 1963.

[74] C. J. Vercellino and W. Y. Wang. Hyperactivations for activation function exploration. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems, Workshop on Meta-learning*. Curran Associates, Inc., 2017. URL http://metalearning.ml/2017/papers/metalearn17_vercellino.pdf.

[75] H. Wei, R. Xie, H. Cheng, L. Feng, B. An, and Y. Li. Mitigating neural network overconfidence with logit normalization. 2022.

[76] Y. Wu and K. He. Group normalization. *CoRR*, abs/1803.08494, 2018. URL http://arxiv.org/abs/1803.08494.

[77] C. Xie, M. Tan, B. Gong, A. Yuille, and Q. V. Le. Smooth adversarial training. *arXiv preprint arXiv:2006.14536*, 2020.

[78] H. Xiong, L. Huang, M. Yu, L. Liu, F. Zhu, and L. Shao. On the number of linear regions of convolutional neural networks. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[79] M. Zanfir, A. Zanfir, E. G. Bazavan, W. T. Freeman, R. Sukthankar, and C. Sminchisescu. Thundr: Transformer-based 3d human reconstruction with markers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12971–12980, October 2021.

[80] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Sy8gdB9xx.

[81] L. Zhang, G. Naitzat, and L.-H. Lim. Tropical geometry of deep neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[82] Q. Zhao and L. D. Griffin. Suppressing the unusual: towards robust cnns using symmetric activation functions. *CoRR*, abs/1603.05145, 2016. URL http://arxiv.org/abs/1603.05145.

Printed by Media-Tryck, Lund 2023 NORDIC SWAN ECOLABEL 3041 0903

Doctoral Theses in Mathematical Sciences 2023:3
ISBN 978-91-8039-572-4
ISSN 1404-0034