UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

Investigating a Second-Order Optimization Strategy for Neural Networks

Bernhard Bermeitinger

Dissertation eingereicht an der Fakultät für Informatik und Mathematik der Universität Passau zur Erlangung des Grades eines Doktors der Naturwissenschaften

A dissertation submitted to the faculty of computer science and mathematics in partial fulfillment of the requirements for the decree of doctor of natural sciences

Advisor:	Prof. Dr. Siegfried Handschuh
Second Advisor:	Prof. Dr. Björn Schuller

Rigorosum: Passau, 10. April 2024

List of Publications

The following chronologically ordered list of papers are part of this cumulative dissertation. They are peer-reviewed and published in conference proceedings, or submitted to a peer-reviewed conference. Citations, references, equations, tables, and figures have been visually adapted to match a common format.

The papers [BHH19a; HBH22a; HBH22b; BHH23] are reproduced in Chapters 1 and 3 to 5 by courtesy of the publisher *SCITEPRESS*, respectively.

The paper [BHH19b] is reproduced in Chapter 2 with permission from the publisher *Springer Nature* (license number *5574061456455*).

[BHH19a] Bernhard Bermeitinger, Tomas Hrycej, and Siegfried Handschuh. "Representational Capacity of Deep Neural Networks: A Computing Study." In: *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 1: KDIR*. 11th International Conference on Knowledge Discovery and Information Retrieval. Vienna, Austria: SCITEPRESS, 2019, pp. 532–538. ISBN: 978-989-758-382-7. DOI: 10.5220/0008364305320538.

[BHH19b] Bernhard Bermeitinger, Tomas Hrycej, and Siegfried Handschuh. "Singular Value Decomposition and Neural Networks." In: *Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning*. 28th International Conference on Artificial Neural Networks. Ed. by Igor V. Tetko, Věra Kůrková, Pavel Karpov, and Fabian Theis. Vol. 11728. Lecture Notes in Computer Science. Munich, Germany: Springer, Cham, Sept. 9, 2019, pp. 153–164. DOI: 10.1007/978-3-030-30484-3_13.

[HBH22a] Tomas Hrycej, Bernhard Bermeitinger, and Siegfried Handschuh. "Number of Attention Heads vs. Number of Transformer-encoders in Computer Vision." In: *Proceedings of the 14th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*. 14th International Conference on Knowledge Discovery and Information Retrieval. Valletta, Malta: SCITEPRESS - Science and Technology Publications, 2022, pp. 315–321. ISBN: 978-989-758-614-9. DOI: 10.5220/0011578000003335.

[HBH22b] Tomas Hrycej, Bernhard Bermeitinger, and Siegfried Handschuh. "Training Neural Networks in Single vs. Double Precision." In: *Proceedings of the 14th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*. 14th International Conference on Knowledge Discovery and Information Retrieval. Valletta, Malta: SCITEPRESS, 2022, pp. 307–314. ISBN: 978-989-758-614-9. DOI: 10.5220/0011577900003335.

[BHH23] Bernhard Bermeitinger, Tomas Hrycej, and Siegfried Handschuh. "Make Deep Networks Shallow Again." In: *Proceedings of the 15th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*. 15th International Conference on Knowledge Discovery and Information Retrieval. Rome, Italy: SciTePress, 2023, pp. 339–346. ISBN: 978-989-758-671-2. DOI: 10.5220/0012203800003598.

Zusammenfassung

Die vorliegende kumulative Dissertation untersucht den Einsatz des *Konjugierten Gradienten* („*Conjugate Gradient* (CG)“) auf dessen Tauglichkeit für die Optimierung von künstlichen neuronalen Netzwerken („*Neural Network* (NN)“). Das zentrale Forschungsinteresse liegt in der Evaluation der Eignung von CG, insbesondere im Vergleich zu gängigen Optimierungsverfahren erster Ordnung, wie dem *Stochastische Gradientenabstieg* („*Stochastic Gradient Descent* (SGD)“) und dessen Varianten.

Der Optimierungsalgorithmus über den CG ist eine numerische Optimierungsmethode der zweiten Ordnung. Ein NN besteht grundsätzlich aus mehreren Schichten: Einer Eingabeschicht, einer beliebigen Anzahl versteckter Schichten und einer Ausgabeschicht. Die Aktivierungsfunktionen der versteckten Schichten sind jeweils nichtlinear. Je nach Anforderung der Problemstellung kann die Ausgabeschicht ebenfalls nichtlinear sein. Wenn die Topologie eines NN eine versteckte Schicht aufweist, wird es als *flach* bezeichnet. Ab zwei versteckten Schichten ist es ein *tiefes* NN. Je mehr sequentiell verkettete versteckte Schichten in der Topologie eines NN vorhanden sind, umso tiefer ist es.

Die Arbeit zielt darauf ab, CG im Kontext verschiedener NN-Architekturen, sowohl in der Tiefe als auch in der Breite, zu bewerten. Im Rahmen dieser Bewertung werden tiefe NNs mit flachen NNs verglichen, wobei die Anzahl der trainierbaren Parameter zwischen den beiden Typen gleich bleibt.

Üblicherweise werden Algorithmen erster Ordnung, wie SGD, *Root Mean Square Propagation* (RMSprop), oder Abwandlungen davon, wie *Adam*, eingesetzt, um NNs auf eine vorgegebene Trainingsmenge zu trainieren. Einer der Nachteile dieser numerischen Optimierungsmethoden erster Ordnung ist, dass sie keine Konvergenz der Fehlerfunktion garantieren können. Sie werden vor allem wegen ihrer Einfachheit in der Implementierung eingesetzt und, obwohl vergleichsweise viele Optimierungsschritte dafür notwendig sind, dennoch für die jeweilige Problemstellung akzeptable Ergebnisse vorweisen können. Auf den Einsatz von verschiedenen Optimierungsverfahren zweiter Ordnung wird in der Literatur weitestgehend verzichtet.

Aufgrund des Nischendaseins dieser Algorithmen, verschiebt sich der Fokus der aktuellen Forschung weg von der eigentlichen Optimierung der NNs auf deren Topologie und Regularisierung sowie ebenfalls auf die Größe und Struktur der Datensätze. Im Zuge der weiteren Annahme, dass SGD als Optimierer eingesetzt wird, werden die NNs immer tiefer, folglich die Anzahl der sequentiellen Nichtlinearitäten erhöht. Die lange Verkettung der Schichten verursacht zunehmend das Problem des *Vanishing Gradient*, was bedeutet, dass der Gradient der Fehlerfunktion bei der *Backpropagation* umso kleiner wird, je weiter sich der Gradient vom Ausgang der NNs entfernt, bis er schließlich so klein ist, dass ein sinnvoller Optimierungsschritt unmöglich wird. (Das Gegenteil, eine multiplikative Explosion des Gradienten, ist ebenso denkbar mit ähnlich negativen Auswirkungen auf die Optimierung.)

CG kann dieses Problem zwar nicht lösen, optimiert aber flachere NNs zu einem im Vergleich niedrigeren Fehlerwert, was bedeutet, dass die tiefen NNs für die Problemstellung überdimensioniert waren und die Repräsentationskapazität kleinerer NNs ausreichend wären.

Ein kritischer Punkt bei der Anwendung von CG ist die Maschinengenauigkeit. In Experimenten wird bestätigt, dass CG grundsätzlich in der Lage ist, kleine neuronale Netzwerke mit wenigen Tausend trainierbaren Parametern, aber auch mit mehreren Millionen, so zu optimieren, dass die Fehlerfunktion superlineare Konvergenz aufweist. Nachteilig ergibt sich, dass vor allem bei wachsender Nichtlinearität der zu lösenden künstlich erzeugten Datensätze, CG zu keiner Lösung kommt, wenn die übliche Maschinengenauigkeit von 32-Bit den Berechnungen zugrunde liegt. Der Trend geht zu noch geringerer Präzision; es wird vermehrt die 16-Bit-Genauigkeit für das Training verwendet. Als Grund wird die kürzere Rechenzeit und kleinerer Speicherbedarf angeführt. Für CG ist das keine Alternative, da im 32-Bit-Raum, die Richtungssuche („line-search“) schon bei leicht nichtlinearen Datensätzen keine Optimierungsrichtung finden kann und vorzeitig abbricht. Wird die Optimierung mit 64-Bit-Fließkommazahlen durchgeführt, profitiert CG bei moderat nichtlinearen Datensätzen superlinear. Bei stark nichtlinearen Problemen zeigt sich die superlineare Konvergenz zwar nicht, die erreichten Fehlerwerte durch CG sind allerdings immer niedriger als die von RMSprop [vgl. HBH22b, in Kapitel 4].

Ein weiterer zentraler Aspekt der Arbeit ist die Initialisierung der trainierbaren Parameter der NNs: Gewichtsmatrix und Biasvektor. Üblicherweise werden die trainierbaren Parameter aus einer zuvor definierten Wahrscheinlichkeitsverteilung gezogen. Für den experimentellen Test der Hypothese werden kontrolliert künstliche Datensätze erzeugt, deren Lösung bekannt ist und die als Trainingsdaten dienen. Initiale Experimente zeigen, dass die zufällige Wahl der Parameter unterschiedliches Verhalten von ansonsten identischen NNs und Trainingseinstellungen erzeugt. So ist insbesondere SGD nicht in der Lage, eine zufriedenstellende Lösung für manche der zufällig initialisierten NNs zu finden.

Für die nichtlineare Aktivierungsfunktion *Sigmoid* gilt, dass sie sich in einem begrenzten Intervall linear verhält. Ein NN mit einer solchen nichtlinearen Schicht ist demnach ähnlich einem linearen NN für kleine Werte der Aktivierungsfunktion, die in dieses Intervall fallen. Diese Eigenschaft kann ausgenutzt und die Initialparameter können statt zufällig, durch vorige Verarbeitung durch Singulärwertzerlegung („*Singular Value Decomposition* (SVD)“), gesetzt werden. Experimentell ziehen alle getesteten Optimierungsalgorithmen einen Vorteil daraus und zeigen um Größenordnungen niedrigere Fehlerwerte. Insbesondere gilt das für CG, der in den durchgeführten Rechenexperimenten die NNs zu den niedrigsten Fehlerwerten optimiert und zusätzlich dafür eine wesentlich kleinere Anzahl an Optimierungsschritten benötigt als die Vergleichsoptimierer [vgl. BHH19b, in Kapitel 2]. Die Vorverarbeitung der Daten für SVD-Initialisierung der trainierbaren Parameter ist zu bevorzugen.

Es stellt sich die Frage, ob tiefe NNs eine höhere Repräsentationskapazität haben als solche mit wenigen bzw. nur einer nichtlinearen Schicht. Hierfür werden von verschieden tiefen NNs künstliche Daten erzeugt, deren Lösung bekannt ist, also deren Fehlerwert für das jeweilige erzeugende Referenz-NN genau null ist. NNs mit verschiedenen Tiefen sollen diese Trainingsdaten abbilden können und werden darauf mit unterschiedlichen Optimierungsalgorithmen trainiert. Überraschenderweise sind die flachen NNs, im Hinblick auf den Fehlerwert, besser in der Lage, die Datensätze abzubilden, als die tiefen NNs mit der gleichen Anzahl an trainierbaren Parametern. Dabei ist es unerheblich, ob ein tiefes oder flaches NN die Trainingsdaten erzeugt. Bei Experimenten, in denen die sonstigen Testvariablen (Initialisierung, Anzahl der Parameter, Aktivierungsfunktion, Datensatz, etc.) identisch vorgegeben sind, zeigt sich, dass vor allem die Wahl des Optimierungsalgorithmus ausschlaggebend für den erreichten Fehlerwert ist. Die Optimierung durch CG konvergiert in allen Durchläufen gegen den jeweils kleinsten Fehlerwert [vgl. BHH19a, in Kapitel 1].

Ein zentrales Experiment untersucht die Repräsentationsfähigkeit von modernen NNs unterschiedlicher Tiefe. Untersucht werden *Transformer-Encoder* auf verschiedenen Datensätzen

der Bildklassifikation. Eine Transformer-Encoder-Schicht besteht unter anderem aus der *Self-Attention-Funktionalität* und einem Multilayer-Perzeptron („*Multi-Layer Perceptron* (MLP)“). Üblicherweise ist die Self-Attention über mehrere *Heads* verteilt. Ein Head ist für sich individuell und operiert parallel zu den anderen Heads (ähnlich den Filtern in „*Convolutional Neural Networks* (CNNs)“). Die Anzahl der Heads gibt die Breite der Schicht vor. Die Tiefe entsteht durch die sequentielle Verkettung mehrerer Transformer-Encoder-Schichten. Untersucht wird das Verhältnis von Tiefe und Breite, also die Anzahl der Schichten mit der Anzahl der Heads pro Schicht.

Ein wichtiges Maß, um die Generalisierungsmöglichkeit von NNs a priori abzuschätzen, ist die Angabe der Über- bzw. Unterbestimmtheit. Ein Gleichungssystem ist dann unterbestimmt, wenn die Anzahl der Lösungen unendlich ist, es also mehr Parameter als Gleichungen gibt. Falls es mehr Gleichungen als Parameter gibt, ist das System überbestimmt. Die beiden Möglichkeiten äußern sich so, dass im unterbestimmten Fall das NN ein sehr gutes Ergebnis auf der Trainingsmenge erzeugt, aber ein beliebig schlechtes auf der Testmenge. Das System ist überangepasst an die Trainingdaten („overfitting“) und kann die Testdaten nur mit großem Fehler vorhersagen. Ein überbestimmtes System dagegen zeichnet im Normalfall ein konsistentes Bild auf der Trainingsmenge und der Testmenge, was das bevorzugte Ergebnis ist [vgl. HBH22a, in Kapitel 3].

Der Vorteil von breiten, flachen NNs zeigt sich nicht nur mit Experimenten auf aktuellen Transformer-Architekturen, sondern auch in konventionellen Bildklassifikationsmodellen wie CNNs. Die Linearisierung über die Taylorentwicklung der Konvolutionsschichten und der jeweiligen Identitätsfunktionen zum Überspringen der Nichtlinearitäten („Residual Connections“) ermöglicht es, tiefe, sequentielle Schichten zu parallelisieren. Eine Reihe von 6,912 Rechenexperimenten zeigt, dass die Tiefe der NN keine höhere Güte aufweist; die parallelisierten (flach, aber breit) NNs zeigen im Durchschnitt wenig Overfitting und überdurchschnittlich oft zwar einen höheren Fehlerwert für die Trainingsmenge, allerdings einen niedrigeren Fehlerwert bei der Testmenge als deren sequentiellen Pendants [vgl. BHH23, in Kapitel 5]. Die Optimierung wird von RMSprop übernommen; wobei die CG eine weitere Verbesserung versprechen könnte.

Zusammenfassend untersucht die vorliegende kumulative Dissertation die Anwendung des konjugierten Gradienten (CG) zur Optimierung künstlicher neuronaler Netzwerke (NNs) und vergleicht diese Methode mit verbreiteten Optimierungsverfahren erster Ordnung, insbesondere dem Stochastischem Gradientenabstieg (SGD).

Die in den Arbeiten präsentierten Forschungsergebnisse zeigen, dass CG in der Lage ist, sowohl kleinere als auch sehr große Netzwerke effektiv zu optimieren. Allerdings kann die Maschinengenauigkeit bei 32-Bit-Berechnungen zu Problemen führen, beste Ergebnisse werden erst in 64-Bit-Fließkommazahlen erreicht. Die Forschung betont auch die Bedeutung der Initialisierung der NN-Parameter und zeigt, dass eine Initialisierung mittels Singulärwertzerlegung zu deutlich geringeren Fehlerwerten führt. Überraschenderweise erzielen flachere NNs bessere Ergebnisse als tiefe NNs mit einer vergleichbaren Anzahl an trainierbaren Parametern, unabhängig vom jeweiligen NN, das die künstlichen Daten erzeugt. Es zeigt sich auch, dass flache, breite NNs, sowohl in Transformer-, als auch in CNN-Architekturen oft besser abschneiden als ihre tieferen Gegenstücke. Insgesamt empfehlen die Forschungsergebnisse eine Neubewertung der bisherigen Präferenz für extrem tiefe NNs und betonen das Potential von CG als Optimierungsmethode.

Summary

This cumulative dissertation examines the application of the *Conjugate Gradient* (CG) for its suitability for optimizing artificial *Neural Networks* (NNs). The central research interest lies particularly in the comparison of CG with common first-order methods like *Stochastic Gradient Descent* (SGD) and its variants.

The CG optimization algorithm is a second-order numerical optimization method. A NN generally consists of several consecutive *layers* an input layer, an arbitrary number of hidden layers, and an output layer. The activation functions of the hidden layer are nonlinear. Depending on the problem statement and requirements, the output layer's activation function might also be nonlinear. If the topology of a NN exhibits one hidden layer, it is called a *shallow* NN. From two hidden layers on, it is a *deep* NN. The deeper the NN is, the more sequential hidden layers are present in its topology.

This dissertation aims to evaluate CG in the context of various NN architectures, both in their width and in their depth. As part of this evaluation, deep NNs are compared with shallow NNs, while the number of trainable parameters between the two instance types is kept equal.

Typically, first-order algorithms, such as SGD, *Root Mean Square Propagation* (RMSprop), or derivatives thereof, such as *Adam*, are used to train NNs on a given training set. One of the disadvantages of these first-order numerical optimization methods is that they cannot guarantee convergence of the error function. Their main advantage is their ease of implementation. They can still show acceptable results for the respective problem, although a relatively large number of optimization steps are required. The use of various second-order optimization methods is largely neglected in the literature.

Due to the niche existence of these algorithms, the focus of current research is shifting away from the actual optimization of NNs to their topology, regularization, and the size and structure of datasets. Following the assumption that SGD is used as an optimizer, NNs are becoming deeper, thus increasing the number of sequential nonlinearities. The long chaining of hidden layers increasingly causes the problem of *vanishing gradient*, which means that the gradient of the error function in backpropagation increasingly becomes smaller. This effect multiplies the further away the gradient is from the output of the respective NN until it finally becomes so small that a meaningful optimization step turns impossible. (The opposite, a multiplicative explosion of the gradient, is also conceivable with similar adverse effects on optimization.)

Although, CG cannot solve this particular issue, it optimizes shallower NNs to a lower error value in comparison to SGD. Effectively, the deep NNs were over-parameterized for the problem, and the representational capacity of smaller NNs would have been sufficient in this instance.

Machine precision is a crucial point in applying CG. Experiments confirm that CG is generally able to optimize small NNs with a few thousand trainable parameters, but also larger ones with several million parameters, such that the error function exhibits superlinear convergence. When using the default machine precision in common frameworks of 32 bits, CG sometimes fails to find the next optimization direction. The trend towards lower machine precisions, like 16 bits or even lower, is common and generally recommended. The shorter computation

time and smaller memory footprint are cited as the reasons. This is not a valid route for CG and problems designed to be increasingly nonlinear, as the *line-search* method cannot find an optimization direction in 32-bit space and aborts prematurely. For strongly nonlinear problems, the convergence of the error function is not superlinear anymore, however, the error values achieved by CG are consistently lower than those of RMSprop [see HBH22b, in Chapter 4].

Another central aspect of the dissertation is the initialization of the NNs' trainable parameters: weight matrix and bias vector. Typically, the trainable parameters are drawn from a predefined probability distribution. For the experimental test of the hypothesis, controlled artificial datasets are generated, whose solutions are known and which serve as training data. Preceding experiments show that the random choice of parameters causes different behavior in otherwise identical NNs and training settings. For example, SGD is not able to find a satisfactory solution for some of the randomly initialized NNs.

The nonlinear activation function *sigmoid* is linear in a limited interval. A NN with this activation function behaves therefore similar to a linear NN for small values of the activation function that fall into this interval. This property can be exploited, and the initial parameters can be set instead of randomly, by preprocessing the datasets with *Singular Value Decomposition* (SVD). Experimentally, all tested optimization algorithms benefit from this setting and show orders of magnitude lower error values. CG is particularly efficient, producing the lowest error values in the experiments and benefiting from a substantially lower number of required optimization steps [see BHH19b, in Chapter 2]. Data preprocessing by SVD for initialization is to be preferred.

The question arises whether deep NNs have a higher representational capacity than those with few or only one nonlinear layer. For this purpose, artificial datasets are generated by NNs with various depths, whose solution is known, i.e., whose error value for the respective generating NN instance is exactly zero. NNs with different depths are trained and evaluated on these datasets with different optimization algorithms. Surprisingly, the shallow NNs are better able to map the datasets in terms of the error value than the deeper NNs. For consistency, the number of trainable parameters is kept equal. Surprisingly, the shallow NNs are better able to map the datasets in terms of the error value than the deeper NNs, keeping the number of trainable parameters equal. It is irrelevant whether a shallow or deep NN generates the training data. In experiments in which the other test variables (initialization, number of parameters, activation function, dataset, etc.) are identically configured, it is shown that the choice of the optimization algorithm is decisive for the achieved error value. The optimization by CG converges in all runs to the smallest error value [see BHH19a, in Chapter 1].

A central experiment investigates the representation capacity of modern NNs of different depths. The focus is on *Transformer-Encoders* for their performance on various well-known image classification datasets. A Transformer-encoder layer consists mainly of the *self-attention* functionality and a *Multi-Layer Perceptron* (MLP). The self-attention is typically distributed over multiple *heads*. Each head is individual and operates in parallel to the other heads—similar to the filters in *Convolutional Neural Networks* (CNNs). The number of heads defines the width of the layer. The depth is generated by the sequential concatenation of multiple Transformer-encoder layers. The ratio of depth to width, i.e., the number of layers versus the number of heads per layer, is investigated.

A key measure to estimate the generalization ability of NNs is the indication of *overdetermination* or *underdetermination*. An equation system is underdetermined if the number of solutions is infinite, i.e., if there are more parameters than equations. If there are more equations than parameters, the system is overdetermined. The two possibilities manifest themselves in such a way that in the underdetermined case, the NN produces a very good result on the training

set, but an arbitrarily high error value for the test set. The system is *overfitting* to the training data and fails to predict the test data. An overdetermined system, on the other hand, usually behaves similarly on the training and test set, which is the preferred situation [see HBH22a, in Chapter 3].

The advantage of wide, shallow NNs is not only shown experimentally on modern Transformer architectures but also in conventional image classification models such as CNNs. Linearization with the Taylor expansion of the convolutional layers and the respective identity function to skip the nonlinearities (*residual connections*) enables deep, sequential layers to be parallelized. A series of 6,912 computational experiments show that the deeper NNs do not generally exhibit better performance. The parallelized (shallow but wide) NNs tend not to overfit on average. Above average, in comparison to their deep counterparts, they show a higher error value on the training set but a lower error value on the test set [see BHH23, in Chapter 5]. The optimization is done with RMSprop, however, CG could promise further improvement.

In summary, this cumulative dissertation investigates the application of the conjugate gradient method CG for the optimization of artificial neural networks (NNs) and compares this method with common first-order optimization methods, especially the stochastic gradient descent (SGD).

The presented research results show that CG can effectively optimize both small and very large networks. However, the default machine precision of 32 bits can lead to problems. The best results are only achieved in 64-bits computations. The research also emphasizes the importance of the initialization of the NNs' trainable parameters and shows that an initialization using singular value decomposition (SVD) leads to drastically lower error values. Surprisingly, shallow but wide NNs, both in Transformer and CNN architectures, often perform better than their deeper counterparts. Overall, the research results recommend a re-evaluation of the previous preference for extremely deep NNs and emphasize the potential of CG as an optimization method.

Contents

List of Acronyms

CG Conjugate Gradient

CNN Convolutional Neural Network

CV Computer Vision

MLP Multi-Layer Perceptron

MSE Mean Square Error

NLP Natural Language Processing

NN Neural Network

RMSprop Root Mean Square Propagation

ReLU Rectified Linear Unit

SGD Stochastic Gradient Descent

SVD Singular Value Decomposition

Chapter 1

Representational Capacity of Deep Neural Networks: A Computing Study

Abstract There is some theoretical evidence that deep neural networks with multiple hidden layers have a potential for more efficient representation of multidimensional mappings than shallow networks with a single hidden layer. The question is whether it is possible to exploit this theoretical advantage for finding such representations with help of numerical training methods. Tests using prototypical problems with a known mean square minimum did not confirm this hypothesis. Minima found with the help of deep networks have always been worse than those found using shallow networks. This does not directly contradict the theoretical findings—it is possible that the superior representational capacity of deep networks is genuine while finding the mean square minimum of such deep networks is a substantially harder problem than with shallow ones.

1.1 Introduction

At present, there is a strong revival of interest in layered neural networks. Although the basic structures and algorithms are similar to those developed in the eighties of the past century, there are some shifts in the focus. The most important of them is the emphasis on large networks with more than one hidden layer. The current interest wave is motivated by numerous reports about positive computing experience with such multi-layer networks for very large mapping tasks. Typical applications are corpus-based semantics and *Computer Vision* (CV) (e.g. [Ber+16]). In particular the former is characterized by a strong under-determination of model parameters—there are substantially more parameters than labeled training examples. There are some review works on deep learning, summarizing both the success stories and the theoretical justifications for the success. This paper takes the extensive work by Lecun et al. [LBH15] as a frequent reference.

The term *deep networks* will be used in the sense of deep learning, denoting networks with more than one hidden layer. Networks with a single hidden layer will be referred to as *shallow networks*.

Besides some experimental findings of representational efficiency of deep neural networks, there are also several attempts for theoretical justifications. They state essentially the following: deep networks exhibit larger representation capacity than shallow networks. That is, they are capable of approximating a broader class of functions with the same number of parameters. [Mon+14]

To compare the representation capacity of deep and shallow networks, several interesting results have been published. Bengio et al. [Ben+03] have investigated a class of algebraic functions that can be represented by a special structure of deep networks (alternating summation and product

layers). They showed that for a certain type of deep networks the number of hidden units necessary for a shallow representation would grow exponentially while in a deep network it grows only polynomial. Montufar et al. [Mon+14] have used a different approach for evaluating the representational capacity. They investigated how many different linear hyperplane regions of input space can be mapped to an output unit. They derived statements for the maximum number of such hyperplanes in deep networks, showing that this number grows exponentially with the number of hidden layers. The activation units used have been *Rectified Linear Unit* (ReLU) and softmax units. It is common to these findings that they do not make statements about arbitrary functions. The result of [Ben+03] is valid for algebraic functions representable by the given deep network architecture, but not for arbitrary algebraic terms. Montufar et al. [Mon+14] have derived the maximum number of representable hyperplanes, but it is not guaranteed, that this maximum can be attained for an arbitrary function to be represented. In other words, there are function classes that can be efficiently represented by a deep network, while other functions cannot. This is not unexpected: knowing that some N_1 dimensional function is to be identified from N_2 training examples (which is equivalent to satisfying $N = N_1 \times N_2$ equations), it cannot be generally represented by less than N parameters although cases representable by fewer parameters exist. Another familiar analogy is that of algebraic terms. Some of them can be made compact by the distributive law, others cannot.

This finding can be summarized in the following way. There exist mappings that can be represented by deep neural networks more economically than by shallow ones. These mappings are characterized by multiple usages of intermediary (or hidden) concepts. This may be typical for cognitive mappings, so that deep networks may be adequate for cognitive tasks.

Various studies are claiming the superiority of deep networks based on positive computing experience. Most of them concern particular architectures such as networks using convolutional layers (e.g., [Goo+14]). This network type has a strong justification for image recognition since the convolutional layers are closely related to spacial operators known to be important for image processing. It is then logical to expect that a network with an appropriate number of such layers is superior to a shallow network that offers only the possibility of using a single convolutional layer followed directly by the final processing to the output.

A few works address the issue of the representational capacity of fully connected deep networks. Erhan et al. [Erh+09] is aware of problems with convergence properties of optimization algorithms for deep networks, but claim their superiority to shallow networks on test sets. Their particular focus is to show the usefulness of unsupervised pre-training of some layers, rather than the comparison of the representational capacity so that the choice of test problems makes it not fully appropriate for clarifying the representational capacity of deep and shallow networks.

- Their study compares networks with different total numbers of network parameters (weights and biases) so that the comparison is favorable for deeper networks having more parameters.
- First-order optimization methods with fixed learning rates are used so that the danger of influencing the results by a poor convergence of the optimization method is serious.
- Some problems are under-determined, that is, having fewer constraints than parameters. In this case, the minimum of the training set error may be expected to be zero not only for a single optimum solution but also for large subsets of the parameter space.

For an objective review of deep and shallow networks, it is important to:

- use sufficiently determined problems (i.e., having more constraints than parameters),

- use optimization methods that can be expected to reliably converge at least to local minima,
- compare shallow and deep networks with identical or nearly identical numbers of free network parameters

A comparison following these principles is the goal of this study.

1.2 Attainable Representational Capacity

Even if some class of functions has a superior representational capacity, it is not guaranteed that this capacity can be fully exploited—the additional necessary component is an algorithm that is capable of attaining the functional fit.

In terms of shallow and deep networks, it would be necessary to have algorithms that exploit the assumed superior capacity of deep networks in fitting them to a set of training examples in an efficient way. This efficiency would have to be sufficient not to lose the representational advantage. In practical terms, the usefulness of a network type consists of both a representational capacity and the algorithm efficiency. So, a high capacity potential and a poorly converging algorithm may result in low exploitable capacity.

In the concrete case of mean square minimization, the numerical efficiency of the optimizing algorithm for a given problem is the key parameter. For strictly convex minimization tasks, the usual measure of the potential efficiency is the condition number of the Hessian matrix (i.e., the matrix of the second derivatives of the error function with regard to the network parameters W, with W_i being the parameters of the i-th hidden layer). This condition number is defined as the ratio of the largest and the smallest eigenvalue of the Hessian. Unfortunately, this objective measure cannot be applied for our comparison, for two reasons:

- The error function is not convex at points far away from the minimum.
- Even at the points where the error function is convex, some eigenvalues are very close to zero, corresponding to search directions which have no or very small effect on the error function.

  The condition number is then near to infinity. These directions result from redundancies inherent to the neural networks. This is the case, for example, due to the rotational invariance of the hidden layers at the points of nearly linear activation. Then, the mapping $W_{i+1}W_i$ is identical with $W_{i+1}H^{-1}HW_i$, for any non-singular square matrix H of the dimension corresponding to the width of the i-th hidden layer. So, the neural network constitutes the same mapping with the matrices W_i and W_{i+1} as with HW_i and $W_{i+1}H^{-1}$.

For the lack of theoretical alternatives, the only possibility is to assess the efficiency of particular problems experimentally. To make reliable conclusions from the experiments, it is important to use reliable optimization methods whose results are as little as possible subject to random disturbances of the solution path. This is why a widespread numerical optimization procedure with well-defined convergence properties, the *Conjugate Gradient* (CG) method, has been used in addition to the optimization methods usual in the neural network community.

The problems were generated so that they have a known MSE minimum equal to zero (see Section 1.3), attainable by a particular network architecture (shallow or deep). Since it is hardly possible to generate over-determined problems that have a zero minimum for both a shallow and a deep network, cross-validation has been used. A pair of zero minimum problems have

been generated, one (P_s) with a zero minimum for shallow network N_s, and another (P_d) with a zero minimum for deep network N_d. Both network architectures have the same dimensions of input and output vectors, and hidden layer sizes such that the overall numbers of network parameters are close to each other (the completely identical parameter numbers are difficult to reach).

Both problems, P_s and P_d, are fitted by shallow network N_s and deep network N_d. Comparing the minima reached allows conjectures about the attainable representational capacity of shallow and deep networks.

The scope of this study is only fully connected networks. There seems to be no doubt that specific deep architectures such as those using convolution networks (mimicking spatial operators in image processing) are optimal for specific problems. So, a comparison of deep and shallow networks with such special architectures would make sense only in such specific application settings.

1.3 Test Problems with a Known Minimum

To assess the performance of training algorithms, it is desirable to use test problems with a known optimum. A construction method is presented in this section. Many statements about the performance of deep learning are based on computing experience with practical problems from various application domains. In the very most cases, the problem is fitting the deep network to real data. This implies that the real minimum is not known—the size of real problems makes it impossible to figure out. This may distort the performance evaluation. Reaching "acceptable" or even "the best known" results from the application problem view lets the unanswered question how far we are from the real optimum. It cannot be excluded that all methods find solutions far away from the optimum and the statements about the algorithms are to a large degree arbitrary. To clarify this aspect, this investigation will make use of problems with a known minimum. Such problems can be generated in the following way:

A network with a set of arbitrary (e.g., random) weights w_0 is generated. Furthermore, a set of random input vectors U for a mapping to be fitted is produced. These inputs, applied to the network $f(u, w)$, result in a set of output vectors Y.

$$y_i = f(u_i, w_0), \quad i = 1, \ldots, n \tag{1.1}$$

The pairs

$$(u_i, y_i), \quad i = 1, \ldots, n \tag{1.2}$$

constitute the training set to be fitted. The least-squares objective function

$$E = \sum_{i=1}^{n} \big(y_i - f(u_i, w)\big)^2 \tag{1.3}$$

has an obvious minimum of zero. This minimum may not be unique in terms of the parameter vector w. So, the success of the fitting is measured only via a minimum value of E reached. The random weights are generated from a uniform distribution:

$$w_i \in \left(\frac{-w_f}{\sqrt{n+1}}, \frac{w_f}{\sqrt{n+1}}\right), \qquad i = 1, \ldots, n \tag{1.4}$$

Table 1.1: Overview of test problems.

Problem	Input	Output	Data size	# hidden layers	# nodes (per hidden layer)	# parameters	# constraints
A_1	100	50	80	1	20	3,070	4,000
A_3	100	50	80	3	16	3,010	4,000
A_5	100	50	80	5	14	3,004	4,000
B_1	300	150	240	1	60	27,210	36,000
B_3	300	150	240	3	49	27,149	36,000
B_5	300	150	240	5	43	27,111	36,000
C_1	1,000	500	800	1	200	300,700	400,000
C_3	1,000	500	800	3	164	300,784	400,000
C_5	1,000	500	800	5	144	300,164	400,000

with n being the number of unit inputs from the preceding layer. There is a predefined factor w_f for controlling the degree of saturation within the network. The division by the square root of $n + 1$ has the goal of reaching an identical standard deviation of the weighted sum (including the bias) going as input to the nonlinear unit.

1.4 Optimization Methods

Neural networks were optimized by several methods implemented in the popular framework *Keras* [Cho+15] with the TensorFlow backend[1] [Aba+15]: *Stochastic Gradient Descent* (SGD) and *Root Mean Square Propagation* (RMSprop) were selected because of their widespread use, as well as Adadelta [Zei12].

These methods are first-order and there is a widespread opinion in the neural network community that second-order methods are not superior to the first-order ones. However, there are strong theoretical and empirical arguments in favor of the second-order methods from numerical mathematics (see, e.g. [BHH19b]). To make sure that the results in favor of shallow or deep networks are not biased by deficiencies of the optimization methods used, second-order methods should not be neglected. So, the CG method (see, e.g. [Pre+92]), as implemented in *SciPy* [OPJ+01], has also been included. This implementation uses the line search method based on the step length conditions of Wolfe [Wol69]. It exploits the derivative information and has excellent convergence properties for smooth functions.

Since the Keras-to-SciPy-to-Keras interface requires a custom-built bridge for information flow between the frameworks, fast GPU-enhanced execution is not possible and the run-time can't be evaluated comparably.

The performance of the optimization methods has been compared by the number of gradient calls (*epochs* in Keras). All these methods have been used with Keras' and SciPy's default settings.

[1] The actual version was `2.0.0-beta0`.

1.5 Computing Results

A series of computing experiments have been carried out to assess the relationship between attainable representational capacities of shallow and deep networks. The comparison is by using identical mapping problems (defined by input/output pairs) and observing the errors of both network architectures. To provide a reasonable meaning to the mean square figures attained and to make the results comparable, all problems have been deliberately defined to have a minimum at zero, according to the scheme of Section 1.3. To justify the use of the hidden layer as a feature extractor, its width should be smaller than the minimum of the input and output sizes. The dimensions have been chosen so that the full regression is under-determined (as typical for the application class mentioned above), but the relatively narrow hidden layer makes it slightly over-determined. So the effect of overfitting, harmful for generalization, is excluded.

1.5.1 Data Generation

Three problem sizes denoted as A, B, and C have been used. These classes are characterized by their input and output dimensions as well as by the size of the training set. For every class, a shallow network with a single hidden layer and two deep networks with three and five hidden layers have been generated. The problem of size class $X \in \{A, B, C\}$ with i hidden layers is denoted by X_i. The concrete network sizes, parameter numbers, and numbers of constraints are given in Table 1.1. The numbers of constraints are imposed by the reference outputs to be fitted. It is the product of the output dimension and the training set size. Comparing the number of constraints with the number of parameters defines the extent of over-determination or under-determination of the problem (e.g., a problem with more constraints than parameters is over-determined).

Hidden layer units are symmetric sigmoid functions rescaled to have a unity derivative at $x = 0$, defined by

$$s(x) = \frac{1}{1 + e^{-x}} \tag{1.5}$$

and

$$f(x) = 2s(2x) - 1 = \frac{2}{1 + e^{-2x}} - 1 \tag{1.6}$$

For every individual network architecture, fifteen different random parametrizations with corresponding training sets have been generated, all with a known mean square error minimum of zero.

1.5.2 Optimization Results

The results for the optimization methods for problem size class B are given in Table 1.3. The focus of this table is on showing the performance on shallow and deep networks for all optimization methods. Besides to the optimum of the error function F_{opt}, the error value at the initial random parameter set is shown to illustrate the extent of the error improvement. The number of iterations is fixed to 2,000 for Keras-methods while it is determined by the stopping rule for the CG. However, the maximum number of iterations for the CG is also set to 2,000.

The first three blocks of the table show the optimization results for shallow and deep networks individually. Each network is optimized to fit the training set generated for this network

Table 1.2: Results for all given problems and their ratio between shallow and deep networks.

Shallow setting	Deep setting	Data deep – NN shallow ($\times 10^{-3}$)	Data shallow – NN deep ($\times 10^{-3}$)	Ratio Deep/Shallow
A_1	A_3	0.038	5.368	140.5
	A_5	0.035	11.353	320.5
B_1	B_3	0.075	4.415	59.2
	B_5	0.070	9.629	138.1
C_1	C_3	0.182	4.663	25.7
	C_5	0.148	9.777	66.1

architecture. The error optimum is known to be zero under of the principles of Table 1.2. These optima set the baseline for those reached by the cross-checks in the following rows.

The following four blocks represent the cross-check itself. The column *Network* denotes the network architecture used for the applied optimization. The column *Source* points to the architecture for which the training set has been generated, with a known error optimum of zero. For the training runs presented in these rows, the optimum is not known. It is only known that it would be zero for the architecture given in the column *Source*, but not necessarily also for the architecture of the column *Network*, used for the fitting.

For example, for the first of these sixteen rows, the network architecture trained is B_1 (i.e., a shallow net with a single hidden layer). It is optimized to fit the training set for which it is known that a zero error can be reached by the architecture B_3 (i.e., a deep net with three hidden layers).

The column *Ratio to CG* displays how many times the error function value attained by the Keras methods was higher than that reached by the CG.

The column *Deep/Shallow* shows the ratio of the following error function values for the deep network and the shallow network.

Additionally, Table 1.2 shows mean square minima for all size classes using the Keras optimization method RMSprop. This table elucidates the development of the performance (*Mean Square Error* (MSE)) with shallow and deep networks for varying network sizes. Each row shows the performance of a pair of a shallow and a deep network with a comparable number of parameters. The average performance of a shallow network for a problem for which a zero error minimum is known to be attainable by a deep network is given in the column *Data deep – NN shallow*. The average performance of a deep network for a problem for which a zero error minimum is known to be attainable by a shallow network is given in the column *Data shallow – NN deep*. The ratio of both average performances is shown in the column *Ratio Deep/Shallow*.

The following can be observed:

- The MSE attained by shallow networks for problems having a zero MSE for some deep network are essentially lower than in the opposite situation.
- The difference tends to slightly decrease with the problem size.
- The by far weakest method was SGD, while the best was the CG. Adadelta and RMSprop were performing between them both, with RMSprop sometimes approaching the CG performance.

Table 1.3: Detailed results for problems of size B.

Network	Source	Algorithm	# iterations	F_{init} ($\times 10^{-3}$)	F_{opt} ($\times 10^{-3}$)	Ratio to CG	Ratio Deep/Shallow
B_1	B_1	Adadelta	2,000	332.2	6.748	578.43	
		RMSprop	2,000	332.2	0.098	8.40	
		SGD	2,000	332.2	90.402	7,748.77	
		CG	821	332.2	0.012		
B_3	B_3	Adadelta	2,000	143.3	6.446	173.67	
		RMSprop	2,000	143.3	0.243	6.54	
		SGD	2,000	143.3	50.485	1,360.22	
		CG	2,200	143.3	0.037		
B_5	B_5	Adadelta	2,000	83.8	4.915	41.04	
		RMSprop	2,000	83.8	0.277	2.31	
		SGD	2,000	83.8	33.233	277.51	
		CG	1,490	83.8	0.120		
B_1	B_3	Adadelta	2,000	235.8	2.577	70.41	
		RMSprop	2,000	235.8	0.075	2.04	
		SGD	2,000	235.8	45.396	1,240.02	
		CG	420	235.8	0.037		
B_1	B_5	Adadelta	2,000	206.7	1.594	52.44	
		RMSprop	2,000	206.7	0.070	2.29	
		SGD	2,000	206.7	32.404	1,065.83	
		CG	333	206.7	0.030		
B_3	B_1	Adadelta	2,000	237.8	28.331	6.75	11.0
		RMSprop	2,000	237.8	4.415	1.05	59.2
		SGD	2,000	237.8	118.896	28.34	2.6
		CG	1,072	237.8	4.195		114.6
B_5	B_1	Adadelta	2,000	208.6	44.980	5.32	28.2
		RMSprop	2,000	208.6	9.629	1.14	138.1
		SGD	2,000	208.6	136.118	16.11	4.2
		CG	2,125	208.6	8.451		278.0

- The difference between the performance with a shallow network on one hand and deep network, on the other hand, grows with the performance of the optimization method: the difference is relatively small for the worst-performing SGD and very large for the best-performing CG.

1.6 Discussion

The computing experiments seem to essentially show the superiority of shallow networks in attaining low mean square minima for given mapping problems. This is not necessarily a contradiction to the theoretical results expecting the contrary. It is still possible that the representational capacity of deep networks is superior, while it is difficult to exploit this capacity by fitting the mapping with the help of numerical algorithms.

Shallow networks have been superior for all test problems and all optimizing algorithms. However, it is interesting to observe that the gap, although always large, was relatively smaller for weakly performing optimization methods (SGD and Adadelta) as well as for large networks.

A possible hypothesis explaining the both might be that the gap is low if the optimizing method fails to search for the minimum efficiently, approaching the performance of some kind of random search. This can result either from the weakness of the method itself or from the difficulty of the problem. Even sophisticated methods such as the conjugate gradient have growing difficulties with growing problem size. These difficulties may have to do with the machine precision necessary for stopping rules (testing for *zero gradient*) or with the number of iterations available.

So, the CG provides a theoretical guarantee for finding a minimum for an exactly quadratic problem of dimension q in q steps. This is a huge number of iterations for our test problems (and other real-world ones). In addition to this, our problems are far from being exactly quadratic (they may even be non-convex), which further increases the computing requirements. This makes clear that the adequacy of every optimization method decreases with the problem size. This still does not explain why deep networks should be more favorable if the optimization method is not adequate to the problem—at best, it may be argued that the search is then close to the random search, which might be indifferent to the functional parametrization optimized.

References

[Aba+15] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/.

[Ben+03] Yoshua Bengio et al. "Out-of-Sample Extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering." In: *Proceedings of the 16th International Conference on Neural Information Processing Systems*. Advances in Neural Information Processing Systems. NIPS'03. Cambridge, MA, USA: MIT Press, 2003, pp. 177–184. URL: https://dl.acm.org/doi/abs/10.5555/2981345.2981368.

[Ber+16] Bernhard Bermeitinger, André Freitas, Simon Donig, and Siegfried Handschuh. "Object Classification in Images of Neoclassical Furniture Using Deep Learning." In: *Computational History and Data-Driven Humanities*. International Workshop on Computational History and Data-Driven Humanities. IFIP Advances in Information and Communication Technology. Springer, Cham, May 25, 2016, pp. 109–112. ISBN: 978-3-319-46223-3. DOI: 10.1007/978-3-319-46224-0_10.

[BHH19b] Bernhard Bermeitinger, Tomas Hrycej, and Siegfried Handschuh. "Singular Value Decomposition and Neural Networks." In: *Artificial Neural Networks and Machine Learning – ICANN 2019: Deep Learning*. 28th International Conference on Artificial Neural Networks. Ed. by Igor V. Tetko, Věra Kůrková, Pavel Karpov, and Fabian Theis. Vol. 11728. Lecture Notes in Computer Science. Munich, Germany: Springer, Cham, Sept. 9, 2019, pp. 153–164. DOI: 10.1007/978-3-030-30484-3_13.

[Cho+15] François Chollet et al. *Keras*. 2015. URL: https://keras.io.

[Erh+09] Dumitru Erhan et al. "The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-Training." In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. Artificial Intelligence and Statistics. PMLR, Apr. 15, 2009, pp. 153–160. URL: https://proceedings.mlr.press/v5/erhan09a.html.

[Goo+14] Ian J. Goodfellow et al. *Multi-Digit Number Recognition from Street View Imagery Using Deep Convolutional Neural Networks*. Version 4. Apr. 14, 2014. DOI: 10.48550/arXiv.1312.6082. preprint.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning." In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687. DOI: 10.1038/nature14539.

[Mon+14] Guido F Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. "On the Number of Linear Regions of Deep Neural Networks." In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 2924–2932. URL: http://papers.nips.cc/paper/5422-on-the-number-of-linear-regions-of-deep-neural-networks.pdf.

[OPJ+01] Travis E. Oliphant, Pearu Peterson, Eric Jones, et al. *Scipy: Open Source Scientific Tools for Python*. 2001.

[Pre+92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*. USA: Cambridge University Press, 1992. ISBN: 978-0-521-43108-8.

[Wol69] Philip Wolfe. "Convergence Conditions for Ascent Methods." In: *SIAM Review* 11.2 (Apr. 1969), pp. 226–235. DOI: 10.1137/1011036.

[Zei12] Matthew D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*. Dec. 22, 2012. DOI: 10.48550/arXiv.1212.5701. preprint.

Chapter 2

Singular Value Decomposition and Neural Networks

Abstract *Singular Value Decomposition* (SVD) constitutes a bridge between the linear algebra concepts and multi-layer neural networks—it is their linear analogy. Besides of this insight, it can be used as a good initial guess for the network parameters, leading to substantially better optimization results.

2.1 Motivation

The utility of multi-layer neural networks is frequently being explained by their capability of extracting meaningful features in their hidden layers. This view is particularly appropriate for large size applications such as corpus-based semantics analyses where the number of training examples is too low for making the problem fully determined in terms of a direct mapping from the input to the output space.

This capability of feature extraction is mostly implicitly attributed to using nonlinear units in contrast to a linear mapping. The prototype of such linear mapping is linear regression, using multiplication of an input pattern by a regression matrix to get an estimate of the output pattern, omitting the possibility of using a sequence of two (or more) matrices corresponding to the use of a hidden layer of linear units. This possibility is usually considered to be obsolete with the argument that a product of two matrices is also a matrix and the result is thus equivalent to using a single matrix.

This argument, although superficially correct, hides the possibility of using a matrix of deliberately chosen low rank, which leads to the correct treatment of under-determined problems.

A key to understanding the situation is *Singular Value Decomposition* (SVD). In the following, it will be shown that SVD can be interpreted as a linear analogy of a neural network with one hidden layer and that it can be used for generating a good initial solution for optimizing nonlinear multi-layer neural networks. Saxe et. al [SMG14] support this theory from the opposite direction: they showed empirically that the optimized results from a nonlinear network are very similar to the results coming from an SVD. McCloone et al. [McL+98] have an interesting approach on how to support optimization by SVD in various network architectures (namely *Multi-Layer Perceptron* (MLP) and RBF networks). However, our interest is in pointing out the direct relationship between SVD and a shallow multi-layer neural network with a low dimensional hidden layer. The low dimensionality of the hidden layer has the function of feature extraction.

The work of Xue et al. [XLG13] is decreasing the number of parameters within a neural network significantly by replacing a layer's weight matrix by two layers which weight matrices are

constructed using SVD. Similar to our approach, this results effectively in initializing the two layers using the resulting matrices from SVD (2.1). Still, it is used to decrease the model size rather than showing that this initialization is already good guess for finding the (near-) optimal solution.

2.2 Singular Value Decomposition

SVD is a powerful concept of linear algebra. It is a decomposition of an arbitrary matrix A of size $m \times n$ into three factors:

$$A = USV^T \tag{2.1}$$

where U and V are orthonormal and S is of identical size as A, consisting of a diagonal matrix D_0 and a zero matrix. For $m < n$, it is $\begin{bmatrix} S_0 & 0 \end{bmatrix}$, for $m > n$ it is $\begin{bmatrix} S_0 & 0 \end{bmatrix}^T$. In the further discussion, only the case of $m < n$ will be considered as the opposite case is analogous.

SVD is then simplified to

$$A = USV^T = U\begin{bmatrix} S_0 & 0 \end{bmatrix}\begin{bmatrix} V_0 & V_x \end{bmatrix}^T = US_0V_0^T \tag{2.2}$$

by omitting redundant zero terms. This form is sometimes called *economical*.

For the economical form (2.2), the decomposition with $r = \min(m,n)$ has $mr + r + nr = (m+n+1)r$ nonzero parameters. The orthonormality of U and V imposes $2r$ unity norm constraints, and $r(r-1)$ orthogonality constraints, resulting in a total number of

$$2r + r(r-1) = r^2 + r \tag{2.3}$$

constraints.

The number of free parameters amounts to

$$(m+n+1)r - r^2 - r \tag{2.4}$$

which is

$$(m+n+1)m - m^2 - m = mn, \quad \text{for } m < n \tag{2.5}$$

and, analogically,

$$(m+n+1)n - n^2 - n = mn, \quad \text{for } m > n \tag{2.6}$$

So, the economical form of SVD possesses the same number of free parameters as the original matrix A.

The number of nonzero singular values in S_0 is equal to the rank r of matrix A. An interesting case arises if the matrix A is not full rank, that is, if $r < \min(m,n)$. Then, some diagonal elements of S_0 are zero. Reordering the diagonal elements of S_0 (and, correspondingly the columns of U and V_0) so that its nonzero elements are in the field S_1 and zero elements in S_2, the decomposition further collapses to

$$A = US_0V_0^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix}\begin{bmatrix} S_1 & 0 \\ 0 & S_2 \end{bmatrix}\begin{bmatrix} V_1 & V_2 \end{bmatrix}^T = U_1S_1V_1^T \tag{2.7}$$

Then, with the help of orthogonality of U_1 and V_1, the matrix can be decomposed into the sum

$$A = USV^T = \sum_{k=1}^{r} s_k u_k v_k^T \tag{2.8}$$

An important property of SVD is its capability for a matrix approximation by a matrix of lower rank. In analogy to the partitioning the singular values with the help of S_1 and S_2 to nonzero and zero ones, they can be partitioned to large and small ones. Selecting the $\hat{r}$ largest singular values makes (2.8) to an approximation $\hat{A}$ of matrix A. This approximation has the outstanding property of being that with the minimum L_2 matrix norm of the difference $\hat{A} - A$

$$\left\|\hat{A} - A\right\|_2 \tag{2.9}$$

out of all matrices of rank $\hat{r}$.

The L_2 matrix norm of M is defined as an induced norm by the L_2 vector norm, so that it is defined as

$$\left\|M\right\|_2 = \max_x \frac{\left\|Mx\right\|_2}{\left\|x\right\|_2} \tag{2.10}$$

In many practical cases, a relatively small number $\hat{r}$ leads to approximations very close to the original matrix. Equation 2.8 shows that this property can be used for an economical representation of a $m \times n$ matrix A by only $\hat{r}\,(m + n + 1)$ numerical values. The optimum approximation property is shown below to be relevant for the mapping approximation discussed below.

A further important application of SVD is an explicit formula for a matrix pseudo-inverse. Pseudo-inverse A^+ is the analogy of an inverse matrix for the case of non-square matrices, with the property

$$AA^+A = A \tag{2.11}$$

It can be easily computed with the help of SVD:

$$X^+ = VS_{\text{inv}}^T U^T \tag{2.12}$$

with S_{inv} being a matrix of the same dimension as S with inverted non-zero elements $\frac{1}{s_{ii}}$ on the diagonal.

2.3 SVD and Linear Regression

One of the applications of the pseudo-inverse (2.11) is a computing scheme for least squares. The linear regression problem is specified by input/output column vector pairs (x_i, y_i), seeking the best possible estimates

$$\hat{y}_i = Bx_i + a \tag{2.13}$$

in the sense of least squares.

The bias vector a can be received by extending the input patterns x_i by a unity constant. For simplicity, it will be omitted in the ongoing discussion.

The solution amounts to solving the equation

$$Y = BX \tag{2.14}$$

with matrices Y and X made of the corresponding column vectors. The optimum is found with the help of the pseudo-inverse X^+ of X. In the over-determined case (typical for linear regression), the least squares solution is

$$B = YX^+ = YX^T\left(XX^T\right)^{-1} \tag{2.15}$$

In the under-determined case, there is an infinite number of solutions with zero approximation error. The following solution has the minimum matrix norm of B:

$$B = YX^+ = Y\left(X^TX\right)^{-1}X^T \tag{2.16}$$

Both (2.15) and (2.16) use the pseudo-inverse that can be easily computed with help of SVD according to (2.12).

2.4 SVD and Mappings of a Given Rank

Both the full SVD (2.1) and its reduced rank form (2.7) are products of a dense matrix U, a partly or fully diagonal matrix S, and a dense matrix V^T. This suggests the possibility of viewing them as a product of two dense matrices US and V^T, or U and SV^T. All these matrices are full rank, even if the original matrix B was not due to the under-determination.

The product US and V^T is the sequence of two linear mappings. The latter matrix maps the n-dimensional input space to an intermediary space of dimension $\hat{r}$, the former the intermediary space to the m-dimensional output space. Since $n > \hat{r}$ and $m > \hat{r}$, the intermediary space represents a bottleneck similar to a hidden layer of a neural network. The orthogonal columns of V can be viewed as hidden features compressing the information in the input space. This relationship to neural networks be followed in Section 2.3.

The reasons to search for such a compressed mapping are different for the over-determined and the under-determined problems.

2.4.1 Over-determined Problems

Suppose for an over-determined problem with input matrix X and output matrix Y, the best linear solution is sought. The columns of X and Y correspond to the training examples. The least-square-optimum solution is the linear regression

$$y = Bx + a \tag{2.17}$$

with matrix B from (2.15). The bias vector a can be received by extending the matrix X by a unit row and applying the pseudo-inversion of such an extended matrix. The last column of such an extended regression matrix corresponds to the column bias vector a.

The linear regression matrix is $m \times n$ for input dimension n and output dimension m, its SVD is as in (2.1).

With more than n independent training examples, the regression matrix B and also the matrix S are full rank with singular values on the diagonal of S.

There may be reasons for assuming that there are random data errors, without which the rank of B would not be full. This would amount to the assumption that some of the training examples are, in fact, linearly dependent or even identical and only the random data errors make them different. To ensure correct generalization, it would then be appropriate to assume a lower rank of the regression matrix. This will suggest using the approximating property of SVD with a reduced singular value set. Leaving out the components with small singular values may be equivalent to removing the data noise. Taking a matrix S_{mod} with $\hat{r}$ largest singular values while zeroing the remaining ones (see, e.g., [TB97]) results in a matrix according to (2.7):

$$B_{\text{mod}} = U_{\text{mod}} S_{\text{mod}} V_{\text{mod}}^T \tag{2.18}$$

that has the least matrix L_2 norm

$$\left\| B - B_{\text{mod}} \right\|_2 \tag{2.19}$$

out of all existing matrices B_{mod} with rank $\hat{r}$. The L_2 matrix norm is induced by the L_2 vector norm, as defined in Equation 2.10. The definition (2.10) of the L_2 matrix norm has an implication for the accuracy of the forecasts with help of R and R_{mod}:

$$\left\| B - B_{\text{mod}} \right\|_2 = \max_x \frac{\left\| (B - B_{\text{mod}})\, x \right\|_2}{\left\| x \right\|_2} = \max_x \frac{\left\| y - y_{\text{mod}} \right\|_2}{\left\| x \right\|_2} \tag{2.20}$$

The vector norm of the forecast error equal to the square root of the mean square error is obviously minimal for a given norm of the input vector. In other words, the modified, reduced-rank regression matrix has the least maximum forecast deviation from the original regression matrix relative to the norm of the input vector x.

2.4.2 Under-determined Problems

A different situation is if the linear regression is under-determined. This is frequently the case in high-dimensional applications such as CV and corpus-based semantics—the number of training examples may be substantially lower than the dimensions of the input. The training examples span a subspace of the input vector space. Using this training information, new patterns can only be projected onto this subspace. The projection operator, using the same definition of the input matrix X as above, is given as:

$$\hat{x} = X X^+ x = X \left(X^T X \right)^{-1} X^T x \tag{2.21}$$

This can be viewed as a pattern-specific weighting of training examples by a weight vector w_x

$$\hat{x} = X w_x^T \tag{2.22}$$

To recall the corresponding output, the same weight vector can be used:

$$\hat{y} = Y w_x^T = Y X^+ x = Y \left(X^T X \right)^{-1} X^T x \tag{2.23}$$

This is equivalent to solving the regression problem

$$Y = RX \tag{2.24}$$

with help of the pseudo-inverse (see, e.g., [Koh89]) of X, which is (2.16) in the under-determined case.

The regression matrix R is, as usual, of size $m \times n$. If the input dimension m exceeds the number of training examples the regression matrix R solving Equation 2.24 is not full rank. Its SVD will exhibit some zero singular values and can be reduced, without a loss of information, to a reduced form:

$$B_{\text{red}} = U_{\text{red}} S_{\text{red}} V_{\text{red}}^T \tag{2.25}$$

2.5 SVD and Linear Networks

Before establishing the relationship between SVD and nonlinear neural networks, let us consider hypothetical multi-layer networks with linear units of the form $g(x) = x$ in the hidden layer.

Suppose a network with one hidden layer of predefined size p is used to represent a mapping from input x to output y. Suppose now that the best linear mapping from input x to output y is

$$y = Bx \tag{2.26}$$

The best approximation with a rank limitation to $\hat{r}$ and is, according to (2.7):

$$y = U_1 S_1 V_1^T x \tag{2.27}$$

This expression can be viewed as a network with one linear hidden layer of width $p = \hat{r}$. The weight matrix between the input and the hidden layers is

$$V_1^T \tag{2.28}$$

and that between the hidden and the output layers is

$$U_1 S_1 \tag{2.29}$$

This network has the property of being the best approximation of the mapping from the input to the output between all networks of this size with orthonormal (in the hidden layer) and orthogonal (in the output layer) weight vectors.

This optimality is not strictly guaranteed to be reached if relaxing the orthogonality constraints. The difference between the orthogonal and the non-orthogonal solutions depends on the ratio between the input and the output widths, and on the relative width of the hidden layer in the following way.

How serious this optimality gap may be can be assessed observing the fraction of the number of orthogonality constraints to the number of parameters. If this fraction is small, the number of independent parameters is close to the number of all parameters and the influence of the orthogonality constraints is small.

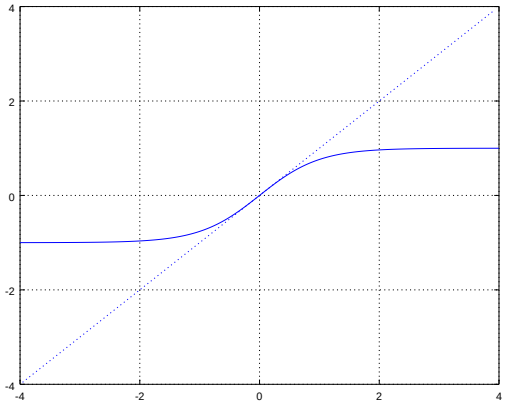


Figure 2.1: Plot of Equation 2.32

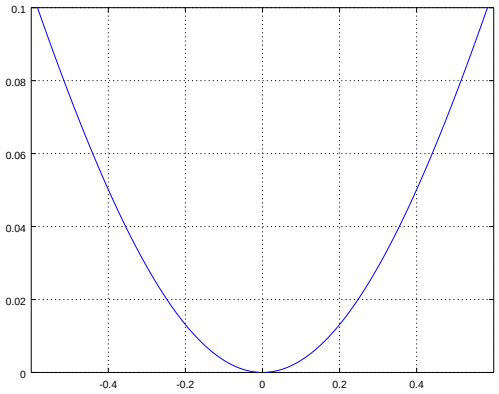


Figure 2.2: Plot of relative deviation between a linear function and sigmoid

With hidden layer size r (equal to the rank of the linear mapping), the total number of constraints is $r^2 + r$. With $m < n$ and $n = qm, q \geq 1$, the total number of parameters is $(m + qm + 1)\,r$. The fraction, and its approximation for realistic values of $r \gg 1$ is then

$$\frac{r^2 + r}{(m + qm + 1)\,r} = \frac{r + 1}{m + qm + 1} \approx \frac{r}{(1 + q)\,m} \tag{2.30}$$

This fraction decreases with the ratio $\frac{m}{r}$ (the degree of *feature compression* by the network) and the ratio q. Since both ratios will usually be large in practical problems of the mentioned domain, the distance to the optimality after relaxing the orthogonality constraints can be expected to be small.

2.6 SVD and Initializing Nonlinear Neural Networks

Most popular hidden units possess a linear or nearly linear segment. A *sigmoid* unit

$$s(x) = \frac{1}{1 + e^{-x}} \tag{2.31}$$

is nearly linear around the point $x = 0$ where its derivative is equal to 0.25. Rescaling this unit to the symmetric form

$$f(x) = 2s(2x) - 1 = \frac{2}{1 + e^{-2x}} - 1 \tag{2.32}$$

we obtain a nonlinear function the derivative of which is unity around $x = 0$, plotted in Figure 2.1.

Its relative deviation from the linear function $g(x) = x$ is below ten percent for $q|x| < 0.58$ (see Fig. 2.2). So, a neural network with one hidden layer using the sigmoid activation function (2.1) behaves like a linear network for small activation values of the hidden layer.

This fact can be used for finding a good initial guess of parameters of a nonlinear neural network with a single hidden layer. The in-going weights into the hidden and output layers are (2.28) and (2.29), respectively.

2.7 Computing Experiments

A series of computing experiments have been carried out to assess the real efficiency of using SVD as a generator of the initial state of neural network parameters. To provide a meaningful interpretation of the mean square figures attained, all problems have been deliberately defined to have a minimum at zero. To justify the use of the hidden layer as a feature extractor, its width should be smaller than the minimum of the input and output sizes. The dimensions have been chosen so that the full regression would be under-determined (as typical for the application types mentioned above), but the use of a hidden layer with a smaller width makes it slightly over-determined. So, the effect of overfitting, harmful for generalization, is excluded.

The software used for SVD computation was the Python module *SciPy* [OPJ+01]. Neural networks were optimized by several methods implemented in the popular framework *Keras* [Cho+15]: SGD, selected because of its widespread use, as well as *Adadelta* and RMSprop, which seem to be the most efficient ones for the problems considered. Typical Keras-methods are first order and there is a widespread opinion in the neural network community that second-order methods are not superior to the first order ones. However, there are strong theoretical and empirical arguments in favor of the second-order methods from numerical mathematics. So the CG method, as implemented in SciPy, has also been applied. Since the SciPy/Keras interface failed to work for this method[1], efficient Keras-based network evaluation procedures could not be used. So, for the largest problems, the CG method had to be omitted.

The performance of the optimization methods has been compared with the help of the number of gradient calls. All methods have been used with the default settings of Keras and SciPy.

Three problem sizes denoted as *A*, *B*, and *C* have been used. Using different size classes will make it possible to discern possible dependencies on the problem size if there are any. The largest size of class *C* is still substantially below that of huge networks such as *VGG-19* [SZ15] used in image classification. The computing effort for making method comparison with such huge sizes would be excessive for the goals of this study. However, we believe the size is sufficient for showing trends relevant for very large network sizes.

The three size classes are characterized by their input and output dimensions as well as by the size of the training set. The concrete network sizes, parameter numbers, and numbers of constraints (output values to be reached times the number of training examples) are given in Table 2.1. The column "# constraints" shows the number of constraints imposed by the reference outputs to be fitted. It is the product of the output dimension and the training set size. Comparing the number of constraints with the number of parameters defines the over-determination or the under-determination of the problem (e.g., a problem with more constraints than parameters is over-determined).

The results for the different size classes are given in Table 2.2. For each network architecture, three different parametrizations with corresponding training sets have been generated, all with a known mean square error minimum of zero. For every variant, an SVD has been computed and used to determine the network initialization. For comparison, five different random network initializations have been generated. The results below are geometric means of minima reached (means from three optimization runs for SVD initializations, and means from $3 \cdot 5 = 15$ runs for random initializations).

The results of four optimization methods are given in the randomly initialized variant and in the variant initialized with help of the SVD solution.

[1] Since we use TensorFlow as Keras' backend execution engine, the resulting computation graph would have been cut into two different executions for each optimization step which causes a too high computational overhead.

Table 2.1: Test problem definitions

Type	# input	# output	# hidden	# training	# parameters	# constraints
A	100	50	20	80	3,070	4,000
B	300	150	60	240	27,210	36,000
C	1,000	50	200	800	300,700	400,000

Table 2.2: Mean square minima reached by various optimization methods with random and SVD-based initial parameter sets

Algorithm	Init.	size class A		size class B		size class C	
		# iter.	$F_{\text{opt}} \times 10^{-3}$	# iter.	$F_{\text{opt}} \times 10^{-3}$	# iter.	$F_{\text{opt}} \times 10^{-3}$
SVD	—	—	10.200	—	10.559	—	10.814
SGD	Random	2,000	30.361	2,000	90.402	2,000	177.095
RMSprop	Random	2,000	0.040	2,000	0.096	2,000	0.260
Adadelta	Random	2,000	1.290	2,000	6.748	2,000	31.076
CG	Random	637	0.002	821	0.012	—	—
SGD	SVD	2,000	1.779	2,000	4.145	2,000	7.254
RMSprop	SVD	2,000	0.030	2,000	0.085	2,000	0.248
Adadelta	SVD	2,000	0.062	2,000	0.511	2,000	2.086
CG	SVD	316	0.000	233	0.021	—	—

The first row, labeled with algorithm "SVD", shows the minima reached by the SVD solution without any subsequent optimization. It is obvious that the SVD-based initialization is pretty good. Its mean square error minimum is substantially better than the weakest Keras-method SGD with random initialization. For the largest problem size class, SVD without optimization is also superior to Adadelta with random initialization.

An SVD-based initialization with a subsequent optimization lets SGD reach an acceptable minimum, with even better results using *Adadelta*. The best Keras-method, RMSprop, was clearly inferior to the CG, although CG stopped the optimization substantially earlier that the fixed iteration number of RMSprop. For both these methods, the improvement by SVD-based initialization was weak (for CG only in the number of iterations). This is not unexpected: good optimization methods are able to find the representations similar to the SVD by themselves, solving a closely related problem with a different numerical procedure.

2.8 Conclusion and Discussion

SVD constitutes a bridge between the linear algebra concepts and multi-layer neural networks—it is their linear analogy. Besides this insight, it can be used as a good initial guess for the network parameters. The quality of this initial guess may be, for large problems, better than weakly performing (but widely used) methods such as SGD ever reach.

It has to be pointed out that as long as the network uses nonlinear hidden units, simply using this initial guess as ultimate network parameters makes little sense: it would be preferable to

make the units linear, and use the SVD matrices directly to represent the desired input-output mapping.

Unfortunately, there seems to be no analogous generalization for networks with multiple hidden layers. With a hidden layer sequence of monotonically decreasing width (for example, from the input towards the output) it would be possible to proceed iteratively, by successively adding hidden layers of decreasing width.

The procedure would start by defining the first hidden layer z_1 (the one with the largest dimension) and initializing its weights with the help of SVD. Then, the following iterations over the desired number of hidden layers would be performed:

1. Analyzing the mapping between the output of the last hidden layer considered and the output layer $z_i \to y$ (with $z_0 = x$) using SVD.
2. Finding an initial guess of parametrization for the incoming weights to z_{i+1}.
3. Optimizing the weights of such extended nonlinear network by some appropriate optimization method.

This is a formal generalization of the procedure for the network with a single hidden layer z_1, as presented above.

However, it is difficult to find a founded justification for this procedure, as it is equally difficult to find a founded justification for using multiple fully connected hidden layers at all—although there seems to be empirical evidence in favor of this. Of course, there are good justifications for using special architectures such as convolutional networks, which are motivated, e.g., by spatial operators in image processing.

References

[Cho+15] François Chollet et al. *Keras*. 2015. URL: https://keras.io.

[Koh89] Teuvo Kohonen. *Self-Organization and Associative Memory*. Red. by Thomas S. Huang, Teuvo Kohonen, Manfred R. Schroeder, and Helmut K. V. Lotsch. Vol. 8. Springer Series in Information Sciences. Berlin, Heidelberg: Springer, 1989. ISBN: 978-3-540-51387-2. DOI: 10.1007/978-3-642-88163-3.

[McL+98] S. McLoone, M.D. Brown, G. Irwin, and A. Lightbody. "A Hybrid Linear/Nonlinear Training Algorithm for Feedforward Neural Networks." In: *IEEE Transactions on Neural Networks* 9.4 (July 1998), pp. 669–684. ISSN: 1941-0093. DOI: 10.1109/72.701180.

[OPJ+01] Travis E. Oliphant, Pearu Peterson, Eric Jones, et al. *Scipy: Open Source Scientific Tools for Python*. 2001.

[SMG14] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. *Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks*. Feb. 19, 2014. URL: http://arxiv.org/abs/1312.6120. preprint.

[SZ15] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Apr. 10, 2015. DOI: 10.48550/arXiv.1409.1556. preprint.

[TB97] Lloyd N. Trefethen and David Bau Iii. *Numerical Linear Algebra*. Philadelphia, PA: Society for Industrial and Applied Mathematics, Jan. 1997. ISBN: 978-0-89871-361-9. DOI: 10.1137/1.9780898719574.

[XLG13] Jian Xue, Jinyu Li, and Yifan Gong. "Restructuring of Deep Neural Network Acoustic Models with Singular Value Decomposition." In: *Proc. Interspeech 2013*. Interspeech. Jan. 2013, pp. 2365–2369. DOI: 10.21437/Interspeech.2013-552.

Chapter 3

Number of Attention Heads vs. Number of Transformer-encoders in Computer Vision

Abstract Determining an appropriate number of attention heads on one hand and the number of transformer-encoders, on the other hand, is an important choice for *Computer Vision* (CV) tasks using the Transformer architecture. Computing experiments confirmed the expectation that the total number of parameters has to satisfy the condition of overdetermination (i.e., number of constraints significantly exceeding the number of parameters). Then, good generalization performance can be expected. This sets the boundaries within which the number of heads and the number of transformers can be chosen. If the role of context in images to be classified can be assumed to be small, it is favorable to use multiple transformers with a low number of heads (such as one or two). In classifying objects whose class may heavily depend on the context within the image (i.e., the meaning of a patch being dependent on other patches), the number of heads is equally important as that of transformers.

3.1 Introduction

Architecture based on the concept of Transformers became a widespread and successful neural network framework. Originally developed for *Natural Language Processing* (NLP), it has been recently also used for applications in CV [Dos+21].

The key concept of a Transformer is *(self-) attention*. The attention mechanism picks out segments (or words, tokens, image patches, etc.) in the input data that are building relevant context for a given segment. This is done by means of segment weights assigned according to the similarity between the segments. The similarity assignment can be done within multiple *attention heads*. Each of these attention heads evaluates similarity in its own way, using its own similarity matrices. All these matrices are learned through fitting to training data. In addition to similarity matrices, a transformer (-encoder) adds the results of attention heads and processes this sum through a nonlinear perceptron whose weights are also learned. Transformer layers are usually stacked so that the output of one transformer layer is the input of the next one. Among the most important choices for implementing a transformer-based processing system are

1. the number of attention heads per transformer-encoder and
2. the number of transformer-encoders stacked.

The user has to select these numbers and the result substantially depends on them but it is difficult to make recommendations for these choices. Following the general recommendation to avoid underdetermined configurations (where the number of parameters exceeds the number

of constraints) and thus overfitting leading to poor generalization, there are still two above-mentioned numbers to configure: approximately the same number of network parameters can be reached by taking more attention heads in fewer transformer-encoders or vice versa. The decision in favor of one of these alternatives may be substantial for the success of the application. The goal of the present work is to investigate the effect of both numbers on learning performance with the help of several CV applications.

3.2 Parameter structure of a multi-head transformer

The parameters of a multi-head transformer (in the form of only encoders and no decoders) consist of:

1. matrices transforming token vectors to their compressed form (*value* in the transformer terminology);
2. matrices transforming token vectors to the feature vectors for similarity measure (*key* and *query*), used for context-relevant weighting;
3. matrices transforming the compressed and context-weighted form of tokens back to the original token vector length;
4. parameters of a feedforward network with one hidden layer;

All these matrices can be concatenated (e.g., column-wise) to a single parameter-vector. Each transformer-encoder contains the same number of parameters. The total parameter count is thus proportional to the number of transformer-encoders. Varying the number of heads affects the parameter count resulting from the transformation matrices of the attention mechanism, the remaining ones being the parameters of the feedforward network. The total parameter count is thus less than proportional to the number of heads.

3.3 Measuring the degree of overdetermination

Fitting a parameterized structure to a data set can be viewed as an equation system. M outputs to be fitted for K training examples constitute MK equations. P free parameters whose values are sought for the best fit correspond to P variables. Consequently, we have a system of MK equations with P variables. Since it is not certain that these equations can be satisfied, it is more appropriate to speak about constraints instead of equations. In the case of linear constraints, there are well-known conditions for solvability. Assuming mutual linear independence of constraints, this system has a unique solution if $MK = P$. The solution is then *exactly determined*. With $MK < P$, the system has an infinite number of solutions—it is *underdetermined*. In the case of $MK > P$, the system is *overdetermined* and cannot be exactly solved—the solution is only approximate. One such solution is based on the least-squares, i.e., minimizing the *Mean Square Error* (MSE) of the output fit. Usually, the real system on which the training data have been measured is assumed to correspond to a model (e.g., a linear one) with additional noise. The noise may reflect measurement errors but also the inability of the model to describe the reality perfectly. It is desirable that the assumed model is identified as exactly as possible while fitting the parameters to the noise in the training set is to be avoided. The latter requirement is justified by the fact that novel patterns not included in the training set will be loaded by different noise values than those from the training set. This undesirable fitting to the training set noise is frequently called overfitting. For exactly determined or underdetermined configurations, the fit

to the training set outputs including the noise is perfect and thus overfitting is unavoidable. For overdetermined configurations, the degree of overfitting depends on the ratio of the number of constraints to the number of parameters. This ratio can be denoted as

$$Q = \frac{MK}{P} \tag{3.1}$$

For a model with a parameter structure corresponding to the real system, it can be shown that the proportion of noise to which the model is fitted is equal to $1/Q$. With increasing the number of training cases, this number is diminishing, with a limit of zero. Asymptotically, the MSE corresponds, in the case of white Gaussian noise, to the noise variance. In other words, overfitting decreases with a growing number of training cases. The dependency of MSE on the number of training samples is

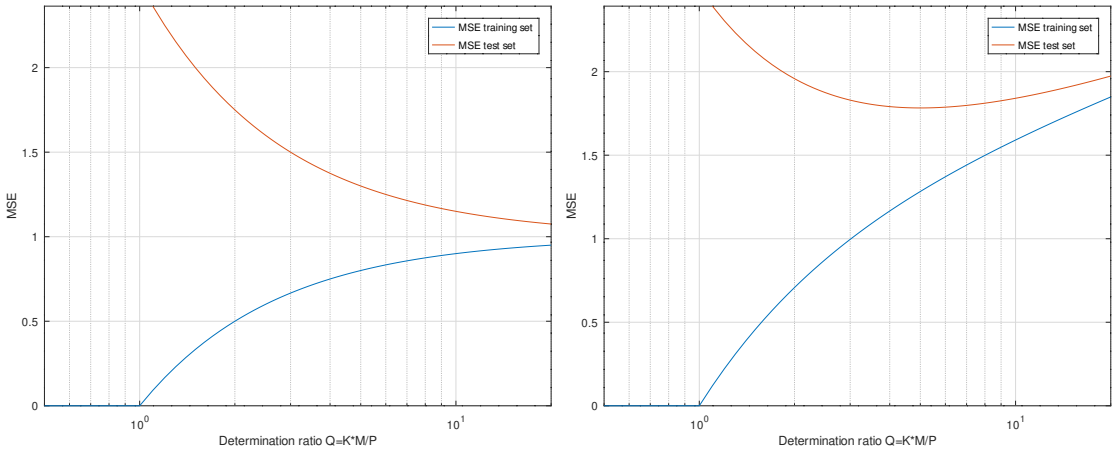
$$E = \sigma^2 \left(1 - \frac{1}{Q}\right) = \sigma^2 \left(1 - \frac{P}{MK}\right) \tag{3.2}$$

The genuine goal of parameter fitting is to receive a model corresponding to the real system so that novel cases are correctly predicted. The prediction error consists of an imprecision of the model and the noise. For a linear regression model, the former component decreases with the size of the training set since the term $(X'X)^{-1}$ determines the variability of estimated model parameters (with X being the input data matrix) develops with c_1/K. The prediction is a linear combination of model parameters that amount on average to a constant c_2. The noise component is inevitable—its level is identical to that encountered in the training set (if both sets are representative of the statistical population). The resulting dependency is, with constants P and M,

$$\begin{aligned} E &= c_2\sigma^2(X'X)^{-1} + \sigma^2 \\ &= \frac{c_1 c_2 \sigma^2}{K} + \sigma^2 \\ &= \frac{P}{M}\frac{c_1 c_2 \sigma^2}{Q} + \sigma^2 \\ &= \sigma^2\left(\frac{c}{Q} + 1\right) \end{aligned} \tag{3.3}$$

The shape of dependencies of training and test set MSE is exemplified in Figure 3.1a. The coefficient of determination on the x-axis varies as the number of training samples grows. The output dimension M and the number of model parameters P are kept constant.

In summary, the MSEs for both the training set and for the novel cases converge to the same level determined by the variance of noise if the number of training samples grows. The condition for this is that the model structure is sufficiently expressive to capture the input/output dependence of the real system. With nonlinear systems, these laws can be approximately justified by means of linearization. Additionally, nonlinear systems such as layered neural networks exhibit dependencies between the parameters, the best known of which are the permutations of hidden layer units. In the Transformer architecture, another source of redundancy are the similarity matrices of the attention mechanism. This makes the number of genuinely free parameters P (as used above) to be below the total number of parameters. However, the number of free parameters is difficult to assess and thus the total number can be used for a rough estimate (or as an upper bound). Systems with $Q > 1$ are certain to be overdetermined while those with



(a) Fixed parameter set, varying training set

(b) Fixed training set, varying parameter set

Figure 3.1: Training and test set MSE in dependence on determination ratio.

$Q < 1$ are not necessarily underdetermined. Nevertheless, the ratio Q is the best we have in practice.

Figure 3.1a corresponds to the situation where the parameter set is kept constant while the size of the training set varies. Frequently, the situation for choice is inverse. There is a fixed training set and an appropriate parameter set is to be determined. Varying (in particular, reducing) the parameter set (and maybe also the model architecture) will probably violate the condition of the model being sufficiently expressive to capture the properties of the real system. Reducing the parameter set represents an additional source of estimation error—the model would not be able to be perfectly fitted to training data even in the case of zero noise. Then, the training and test set MSE will develop with an additional term growing with ratio Q (and decreasing P). The shape of this term is difficult to assess in advance without knowledge of the real system. The typically encountered dependence is depicted in Figure 3.1b (with arbitrary scaling of the MSE).

3.4 Computing results

To show the contribution of the number of heads and that of the number of transformer-encoders, a series of model fitting experiments has been performed, for several CV classification tasks. The data sets used have been popular collections of images, frequently used for various benchmarks. The data sets have been chosen particularity for their match of determination ratio for the experimental networks. Bigger data sets are deliberately left out. For every task, a set of tasks with various pairs (h, t), the number of heads being h and the number of transformer-encoders being t, have been optimized. Some combinations with high numbers of both heads and transformer-encoders had too many parameters and have thus been underdetermined. The consequence has been a poor test set performance. In the following, a cross-section of the results is presented:

- four transformer-encoders and any number of attention heads;
- four attention heads and any number of transformer-encoders.

These cross-sections contain mostly overdetermined configurations with acceptable generalization properties. The performance has been measured by mean categorical cross-entropy on training and test sets (further referred to as *loss*). The x-axis of Figures 3.1 to 3.6 is the

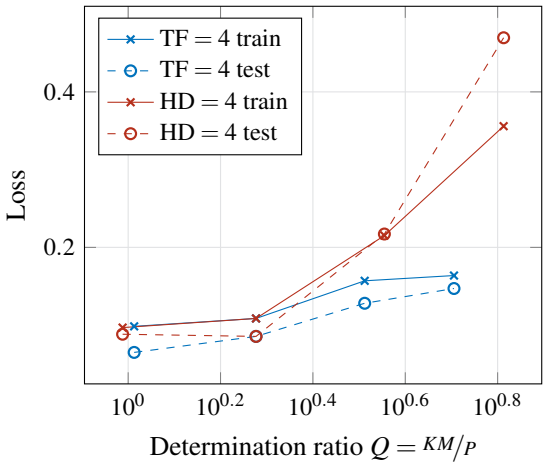


Figure 3.2: Training and test set losses of model variants for dataset *MNIST*.

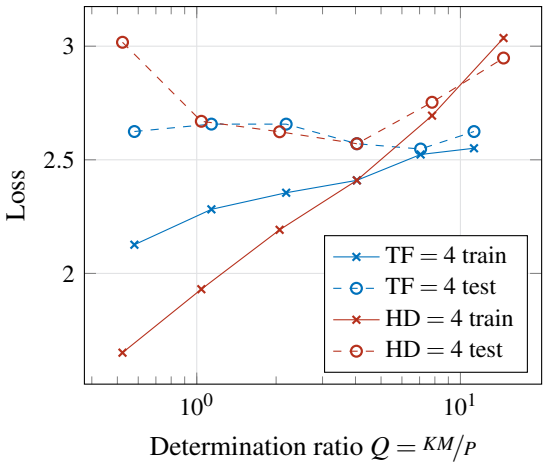


Figure 3.3: Training and test set losses of model variants for dataset *CIFAR-100*.

determination ratio Q of (Equation (3.1)), in logarithmic scale (so that the value 10^0 corresponds to $Q = 1$). This presentation makes the dependence of the generalization performance (as seen in the convergence of the training and the test set cross-entropy) on the determination ratio clear. This ratio grows with the decreasing number of parameters, that is, with the decreasing number of heads if the number of transformer-encoders is kept to four and the decreasing number of transformer-encoders if the number of heads is kept to four. The rightmost configuration is that with a single head or a single transformer-encoder, respectively, followed to the left with two heads or two transformer-encoders, etc..

The optimization was done exclusively with single precision (*float32*) over a fixed number of 100 epochs by *AdamW* [LH19] with a learning rate of 10^{-3} and a weight decay of 10^{-4}. For consistency, the batch size was set to 256 for all experiments.

As a simple regularization during training, standard image augmentation techniques were applied: random translation by a factor of $(0.1, 0.1)$, random rotation by a factor of 0.2, and random cropping to 80 %.

The patches are flattened and their absolute position is added in embedded form to each patch before entering the first encoder.

All experiments were individually conducted on one *Tesla V100-SXM3-32GB* GPU for a total number of 60 GPU days.

3.4.1 Dataset MNIST

The MNIST [LeC+98] dataset consists of pixel images of digits. All pairs (h, t) with number of heads $h \in \{1, 2, 4, 8\}$ and number of transformer-encoders $t \in \{1, 2, 4, 8\}$ have been optimized. The results in the form of loss depending on the determination ratio Q are given in Figure 3.2.

The gray-scale images were resized to 32×32 and the patch size was set to 2. All internal dimensions (keys, queries, values, feedforward, and model size) are set to 64.

The cross-entropies for the training and the test sets are fairly consistent, due to the determination ratio $Q > 1$. The results are substantially more sensitive to the lack of transformer-encoders: the rightmost configurations with four heads but one or two transformer-encoders have a poor

performance. By contrast, using only one or two heads leads only to a moderate performance loss. In other words, it is more productive to stack more transformer-encoders than to use many heads. This is not surprising for simple images such as those of digits. The context-dependency of image patches can be expected to be rather low and to require only a simple attention mechanism with a moderate number of heads.

3.4.2 Dataset CIFAR-100

The dataset *CIFAR-100* [Kri09] is a collection of images of various object categories such as animals, household objects, buildings, people, and others. The objects are labeled into 100 classes. The training set consists of 50,000, the test set of 10,000 samples. With $M = 100$ and $K = 50\,000$, the determination coefficient Q is equal to unity (100 on the plot x-axis) for 5 million free parameters ($M \times K$). The results are given in Figure 3.3. All pairs (h, t) with number of heads $h \in \{1, 2, 4, 8, 16, 32\}$ and number of transformer-encoders $t \in \{1, 2, 4, 8, 16, 32\}$ have been optimized.

The colored images were up-scaled to 64×64 and the patch size was set to 8. All internal dimensions (keys, queries, values, feedforward, and model size) are set to 128.

The cross-entropies for the training and the test sets converge to each other for about $Q > 4$, with a considerable generalization gap for $Q < 1$. This can be expected taking theoretical considerations mentioned in Section 3.3 into account. The results are more sensitive to the lack of transformer-encoders than to that of heads. How far a high number of transformer-encoders would be helpful, cannot be assessed because of getting then into the region of $Q < 1$. With this training set size, a reduction of some transformer parameters such as key, query, and value width would be necessary.

3.4.3 Dataset CUB-200-2011

The training set of the dataset *CUB-200-2011* [Wah+11] (*birds*) used for the image classification task consists of 5,994 images of birds of 200 species. All pairs (h, t) with number of heads $h \in \{1, 2, 4, 8\}$ and number of transformer-encoders $t \in \{1, 2, 4, 8\}$ have been optimized (Figure 3.4).

The colored images were resized to 128×128 and the patch size was set to 8. All internal dimensions (keys, queries, values, feedforward, and model size) are set to 32.

The cross-entropies for the training and the test sets are mostly consistent due to the high determination ratio Q. There are relatively small differences between small numbers of heads and transformer-encoders. Both categories seem to be comparable. This suggests, in contrast to the datasets treated above, a relatively large contribution of context to the classification performance—multiple heads are as powerful as multiple transformer-encoders. This is not unexpected in the given domain: the habitat of the bird in the image background may constitute a key contribution to classifying the species.

3.4.4 Dataset places365

The training set of dataset *places365* [Zho+18] consists of 1,803,460 images of various places in 365 classes (Figure 3.5). Pairs (h, t) with number of heads $h \in \{1, 2, 4, 8, 16, 32\}$ and number of transformer-encoders $t \in \{1, 2, 4, 8, 16, 32\}$ have been optimized.

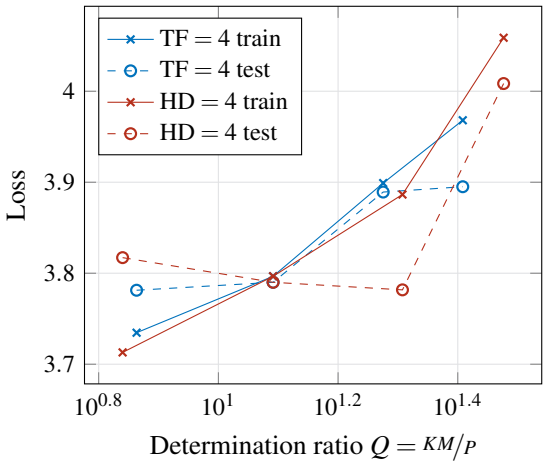


Figure 3.4: Training and test set losses of model variants for dataset *birds*.

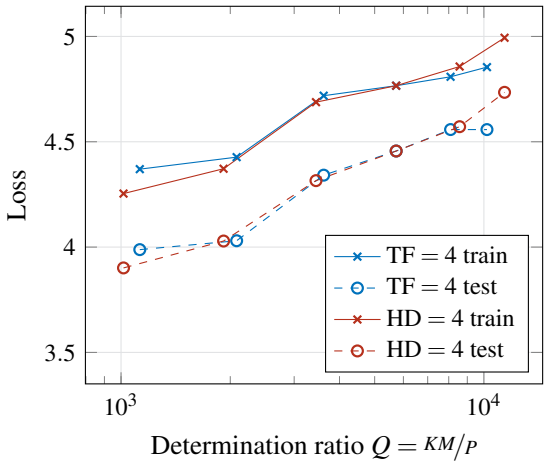


Figure 3.5: Training and test set losses of model variants for dataset *places*.

The colored images were resized to 128×128 and the patch size was set to 16. All internal dimensions (keys, queries, values, feedforward, and model size) are set to 32.

The cross-entropies for the training and the test sets are parallel. Surprisingly, test set losses are lower than those for the training set. This can be caused by an inappropriate test set containing only easy-to-classify samples. The reason for this training to test consistency is the very high determination ratio Q (over 1,000). This would allow even larger numbers of transformer-encoders and heads without worry about generalization, with a corresponding high computing expense.

There are hardly any differences between variants with varying heads and those varying transformer-encoders. With a given total number of parameters (and thus a similar ratio Q), both categories seem to be equally important. It can be conjectured that there is a relatively strong contribution of context to the classification performance can be assumed.

3.4.5 Dataset imagenet

The training set of the popular *imagenet* [KSH12] dataset contains 1,281,167 images of 1,000 different classes of current everyday objects (like airplanes, cars, different types of animals, etc.)

For this dataset, the pairs (h, t) with number of heads $h \in \{1, 2, 4, 8\}$ and number of transformer-encoders $t \in \{1, 2, 4, 8\}$ have been optimized.

Analog to the *places* experiment, the colored images were resized to 128×128 and the patch size was set to 16. All internal dimensions (keys, queries, values, feedforward, and model size) are set to 64.

For this experiment, it can be seen in the determination ratios in Figure 3.6 that it behaves similarly to the dataset *places* (in Figure 3.5). Again, the test loss is consistently lower than the training loss. The lowest cross-entropies are comparable which means, analog to *places*, that increasing the number of attention heads and the number of transformer-encoder layers is beneficial to the performance. Compared to the other experiments, the determination ratio is very high (10^3 to 10^4) which means that the number of parameters in the classification network is too small and even larger stacks of transformer-encoders with more attention heads could decrease the loss even further.

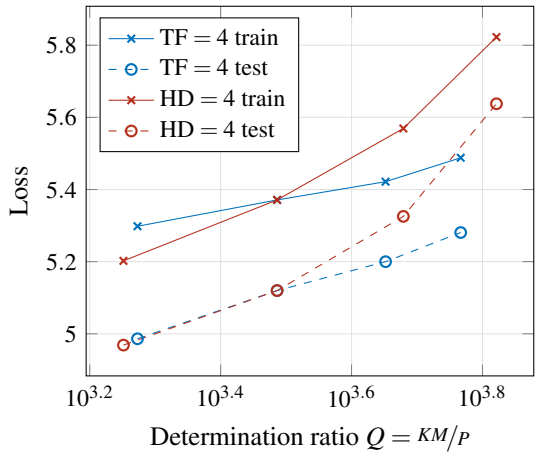


Figure 3.6: Training and test set losses of model variants for dataset *imagenet*.

Looking at the varying number of attention heads, it can be seen that their number has a low impact on the performance.

3.5 Conclusions

Determining the appropriate number of self-attention heads on one hand and, on the other hand, the number of transformer-encoder layers is an important choice for CV tasks using the Transformer architecture.

A key decision concerns the total number of parameters to ensure good generalization performance of the fitted model. The determination ratio Q, as defined in Section 3.3, is a reliable measure: values significantly exceeding unity (e.g., $Q > 4$) lead to test set loss similar to that of the training set. This sets the boundaries within which the number of heads and the number of transformer-encoders can be chosen.

Different CV applications exhibit different sensitivity to varying and combining both numbers.

- If the role of context in images to be classified can be assumed to be small, it is favorable to "invest" the parameters into multiple transformer-encoders. With too few transformer-encoders, the performance will rapidly deteriorate. Simultaneously, a low number of attention heads (such as one or two) is sufficient.
- In classifying objects whose class may heavily depend on the context within the image (i.e., the meaning of a patch being dependent on other patches), the number of attention heads is equally important as that of transformer-encoders.

This seems to be consistent with other experiments like [Li+22] where the optimal number of attention heads depends on the dataset.

Future work Although this study provides a systematic comparison between the number of attention heads and number of consecutive transformer-encoders, the sheer number of different hyperparameters is still underrepresented. The hyperparameters in this study were chosen for the task at hand, e.g. the patch size was chosen accordingly to the input image size. However, the patch size is on its own a crucial hyperparameter which might lead to different results if

chosen differently. Any of the listed hyperparameters in the experiments (Section 3.4) need the same systematic analysis as the current study. This is left out for future work.

References

[Dos+21] Alexey Dosovitskiy et al. "An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale." In: *International Conference on Learning Representations*. ICLR. Vienna, Austria, 2021, p. 21. URL: https://openreview.net/forum?id=YicbFdNTTy.

[Kri09] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Dataset. University of Toronto, 2009, p. 60.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[LeC+98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition." In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791.

[Li+22] Feimo Li et al. "Structural Attention Enhanced Continual Meta-Learning for Graph Edge Labeling Based Few-Shot Remote Sensing Scene Classification." In: *Remote Sensing* 14.3 (Jan. 2022), p. 485. ISSN: 2072-4292. DOI: 10.3390/rs14030485.

[LH19] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. Jan. 4, 2019. DOI: 10.48550/arXiv.1711.05101. preprint.

[Wah+11] Catherine Wah et al. *The Caltech-UCSD Birds-200-2011 Dataset*. Dataset CNS-TR-2011-001. Pasadena, CA: California Institute of Technology, 2011, p. 8. URL: https://resolver.caltech.edu/CaltechAUTHORS:20111026-120541847.

[Zho+18] Bolei Zhou et al. "Places: A 10 Million Image Database for Scene Recognition." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.6 (June 2018), pp. 1452–1464. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2017.2723009.

Chapter 4

Training Neural Networks in Single vs. Double Precision

Abstract The commitment to single-precision floating-point arithmetic is widespread in the deep learning community. To evaluate whether this commitment is justified, the influence of computing precision (single and double precision) on the optimization performance of the *Conjugate Gradient* (CG) method (a second-order optimization algorithm) and *Root Mean Square Propagation* (RMSprop) (a first-order algorithm) has been investigated. Tests of neural networks with one to five fully connected hidden layers and moderate or strong nonlinearity with up to 4 million network parameters have been optimized for *Mean Square Error* (MSE). The training tasks have been set up so that their MSE minimum was known to be zero. Computing experiments have disclosed that single-precision can keep up (with superlinear convergence) with double-precision as long as line search finds an improvement. First-order methods such as RMSprop do not benefit from double precision. However, for moderately nonlinear tasks, CG is clearly superior. For strongly nonlinear tasks, both algorithm classes find only solutions fairly poor in terms of mean square error as related to the output variance. CG with double floating-point precision is superior whenever the solutions have the potential to be useful for the application goal.

4.1 Introduction

In the deep learning community, the use of single precision computing arithmetic (the *float32* format) became widespread. This seems to result from the observation that popular first-order optimization methods for deep network training (steepest gradient descent methods) do not sufficiently benefit from a precision gain if the double-precision format is used. This has even led to a frequent commitment to hardware without the capability of directly performing double-precision computations. For convex minimization problems, the second-order optimization methods are superior to the first-order ones in convergence speed. As long as convexity is given, their convergence is superlinear—the deviation from the optimum in decimal digits decreases as fast as or faster than the number of iterations. This is why it is important to assess whether and how far the accuracy of the second-order methods can be improved by using double precision computations (that are standard in many scientific and engineering solutions).

4.2 Second-order optimization methods: factors depending on machine precision

Second-order optimization methods are a standard for numerical minimization of functions with a single local minimum. A typical second-order method is the CG algorithm [FR64].

(There are also attempts to develop dedicated second-order methods, e.g., Hessian-free optimization [Mar10].) In contrast to the first-order methods, it modifies the actual gradient in a way such that the progress made by previous descent steps is not spoiled in the actual step. The algorithm is stopped if the gradient norm is smaller than some predefined small constant. CG has the property if previous descent steps have been optimal in their descent direction, that is if a precise minimum has been reached in this direction. This is reached by a one-dimensional optimization subroutine along the descent direction, called *line search*. Line search successively maintains three points along this line, the middle of which has a lower objective function value than the marginal ones. The minimization is done by shrinking the interval embraced by the three points. The stopping rule of line search consists in specifying the width of this interval at which the minimum has been reached with sufficient precision. The precision at which this can be done is limited by the machine precision. Second-order methods may suffer from insufficient machine precision in several ways related to the imprecision of both gradient and objective function value computation:

- The lack of accuracy of the gradient computation may lead to distorted descent direction.
- It may also lead to a premature stop of the algorithm since the vanishing norm can be determined only with limited precision.
- It may lead to wrong embracing intervals of line search (e.g., with regard to the inequalities between the three points).
- It may also lead to a premature stop of line search if the interval width reduction no longer succeeds.

There are two basic parameters to control the computation of the CG optimization: There is a threshold for testing the gradient vector for being close to zero. This parameter is usually set at a value close to the machine precision, for example, 10^{-7} for single-precision and 10^{-14} for double-precision. The only reason to set this parameter to a higher value is to tolerate a less accurate solution for economic reasons. Another threshold defines the width of the interval embracing the minimum of line search. Following the convexity arguments presented in [Pre+92], this threshold should not be set to a lower value than a square root of machine precision to prevent useless line search iterations hardly improving the minimum. Its value is above 10^{-4} for single precision and 3×10^{-8} for double precision [Pre+92]. There may be good reasons to use even higher thresholds: low values lead to more line search iterations per one gradient iteration. Under overall constraints of computing resources such as limiting the number of function calls, it may be more efficient to accept a less accurate line search minimum, gaining additional gradient iterations. The experiments have shown that the influence of the tolerance parameter is surprisingly low. There is a weak preference for large tolerances. This is why the tolerance of 10^{-1} has been used for both single and double precision in the following experiments. The actual influence in the typical neural network optimization settings can be evaluated only experimentally. To make the results interpretable, it is advantageous to use training sets with known minimums. They can be generated in the following way:

1. Define a neural network with specified (e.g., random) weights.
2. Define a set of input values.
3. Determine the output values resulting from the forward pass of the defined network.
4. Set up a training set consisting of the defined input values and the corresponding computed results.

This training set is guaranteed to have a minimum error of zero.

4.3 Controlling the extent of nonlinearity

It can be expected that the influence of machine precision depends on the problem. The most important aspect is the problem size. Beyond this, the influence can be different for relatively easy problems close to convexity (nearly linear mappings) on one hand and strongly non-convex problems (nonlinear mappings). This is why it is important to be able to generate problems with different degrees of non-convexity. The tested problems are feedforward networks with one, two, or five consecutive hidden layers, and a linear output layer. The bias is ignored in all layers. All layers are fully connected. For the hidden layers, the symmetric sigmoid with unity derivative at $x = 0$ has been used initially as the activation function:

$$\frac{2}{1 + e^{-2x}} - 1 \tag{4.1}$$

Both input pattern values and network weights used for the generation of output patterns are drawn from the uniform distribution. The distribution of input values is uniform on the interval $\langle a, b\rangle = \langle -1, 1\rangle$ whose mean value is zero and variance is

$$\frac{b^3 - a^3}{3\,(b - a)} = \frac{1}{3} \tag{4.2}$$

To control the degree of nonlinearity during training data generation, the network weights are scaled by a factor c so that they are drawn from the uniform distribution $\langle -c, c\rangle$. The variance of the product of an input variable x and its weight w is

$$\begin{aligned} \frac{1}{4c}\int_{-c}^{c}\int_{-1}^{1}(wx)^2 dx dw &= \frac{1}{4c}\int_{-c}^{c} w^2 \frac{2}{3} dw \\ &= \frac{1}{6c}\int_{-c}^{c} w^2 dw \\ &= \frac{1}{6c}\frac{2c^3}{3} = \frac{c^2}{9} \end{aligned} \tag{4.3}$$

For large N, the sum of N products $w_i x_i$ converges to the normal distribution with variance $N\frac{c^2}{9}$ and standard deviation $\sqrt{N}\frac{c}{3}$. This sum is the argument of the sigmoid activation function of the hidden layer. The degree of nonlinearity of the task can be controlled by a normalized factor d such that $c = \frac{d}{\sqrt{N}}$, resulting in the standard deviation $\frac{d}{3}$ of the sigmoid argument. In particular, it can be evaluated which share of activation arguments is larger than a certain value. Concretely, about twice the standard deviation or more is expected to occur in 5 % of the cases.

If a sigmoid function of the form Equation (4.1) is directly used, its derivative is close to zero with values of input argument x approaching 2. For normalizing factor $d = 2$, the derivative is lower than 0.24 at 5 % of the cases, compared to the derivative of unity for $x = 0$. For normalizing factor $d = 4$, the derivative is lower than 0.02 at 5 % of the cases. The vanishing gradient problem is a well-known obstacle to the convergence of the minimization procedure. The problem can be alleviated if the sigmoid is supplemented by a small linear term defining the guaranteed minimal derivative.

$$(1 - h)\left(\frac{2}{1 + e^{-2x}} - 1\right) + hx \tag{4.4}$$

For such sigmoid without saturation with $h = 0.05$, the derivatives are more advantageous: For normalizing factor $d = 2$, the derivative is lower than 0.28 at 5 % of the cases, compared to the derivative of unity for $x = 0$. For normalizing factor $d = 4$, the derivative is lower than 0.07 at 5 % of the cases.

This activation function Equation (4.4) is used for the strongly nonlinear training scenarios.

4.4 Comparison with RMSprop

In addition to comparing the performance of the CG algorithm (as a representative of second-order optimization algorithms) with alternative computing precisions, it is interesting to know how competitive the CG algorithm is compared with other popular algorithms (mostly first-order). Computing experiments with the packages *TensorFlow/Keras* [Aba+15; Cho+15] and various default optimization algorithms suggest a clear superiority of one of them: RMSprop [Hin12]. In fact, this algorithm was the only one with performance comparable to CG. Other popular algorithms such as *Stochastic Gradient Descent* (SGD) were inferior by several orders of magnitude. This makes a comparison relatively easy: CG is to be contrasted to RMSprop. RMSprop modifies the simple fixed-step-length gradient descent by adding a scaling factor $\sqrt{d_{t,i}}$ depending on the iteration t and the network parameter element index i.

$$\begin{aligned} w_{t+1,i} &= w_{t,i} - \frac{c}{\sqrt{d_{t,i}}} \frac{\partial E\left(w_{t,i}\right)}{\partial w_{t,i}} \\ d_{t,i} &= g d_{t-1,i} + (1-g) \left(\partial \frac{E\left(w_{t-1,i}\right)}{\partial w_{t-1,i}} \right)^2 \end{aligned} \tag{4.5}$$

This factor corresponds to the weighted norm of the derivative sequence of the given parameter vector element. In this way, it makes the steps of parameters with small derivatives larger than those with large derivatives. If the convex error function is imagined to be a “bowl”, it makes a lengthy oval bowl more circular and thus closer to a normalized problem. It is a step toward the normalization done by CG but only along the axes of individual parameters, not their linear combinations.

4.5 Computing results

The CG method [Pre+92] with *Brent* line search has been implemented in *C* and applied to the following computing experiments. It has been verified by form published in [NW06] (implemented in the scientific computing framework *SciPy* [Vir+20]), with line search algorithm from [Wol69].

All training runs are optimized with a limit of 3,000 epochs for tasks with up to four million parameters. Smaller tasks had around 30,000, 300,000, and 1 million parameters. The configuration of the reported largest networks with one or five hidden layers can be seen in Table 4.1. The mentioned epoch limit cannot be satisfied exactly since the CG algorithm always stops after a complete conjugate gradient iteration and thus a complete line search could consist of multiple function/gradient calls. The number of gradient calls is generally variable per one optimization iteration, however, during the experiments they were always evaluated as often as forward passes.

Table 4.1: The two different network configurations with four million parameters.

Name	# Inputs	# Outputs	# Hidden Layers	Hidden Layer Size	# parameters
4mio-1h	4,000	2,000	1	680	4,080,000
4mio-5h			5	510	4,100,400

The concept of an *epoch* in both types of experiments corresponds to one optimization step through the full training data with exactly one forward and one backward pass. For CG, the number of forward/backward passes can vary independently and the number of *equivalent epochs* is adapted accordingly: a forward pass alone (as used in line search) counts as one equivalent epoch while forward and backward pass (as used in gradient computations) counts as two. This is conservative with regard to the CG computing expense. The ratio between the computing expense for a backward pass and that for a forward pass are varying in dependence on the number of hidden layers: it is one with no hidden layer and approaches two with the number of hidden layers growing. With this definition, CG can be handicapped by up to 33 % in the following reported results. In the further text, *epochs* refer to *equivalent epochs*.

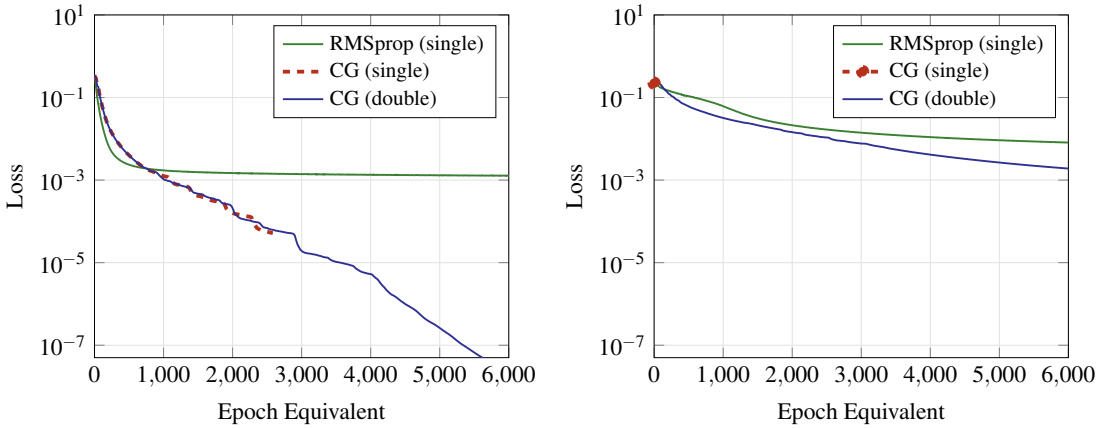
Using higher machine precision with first-order methods, including RMSprop, brings about no significant effect. Rough steps in the direction of the gradient, modified by equally rough scaling coefficients, whose values are strongly influenced by user-defined parameters such as g in (Equation (4.5)) do not benefit from high precision. In none of our experiments with both precisions, there was a discernible advantage by double-precision. This is why the following comparison is shown for

- single and double precision CG method and
- single precision RMSprop.

To assess the optimization performance, statistics over large numbers of randomly generated tasks would have to be performed. However, resource limitations of the two implementation frameworks do not allow such a consequent approach for large networks with at least millions of parameters. And it is just such large networks for which the choice of the optimization method is important. This is why several tasks of progressively growing size have been generated for networks with depths of one, two, and five hidden layers, one for each combination of size and depth. Every task has been run with single and double precision. In the following, only the results for the largest network size are reported since no significant differences have been observed for smaller networks. The networks with two hidden layers behave like a compromise between a single hidden layer and five hidden layers and are thus also omitted from the presentation.

To interpret the differences between such attempts, random influences affecting even such well-defined algorithms as CG are to be taken into account. The convexity condition can be (and frequently is) violated so that better algorithms may be set to a suboptimal search path for some time. For the final result to be viewed as better, the difference must be significant. Intuitively, differences by an order of magnitude or more can be taken as significant while factors of three, two, or less are not so—they may turn to the opposite if provided some additional iterations.

To make the minimum MSE reached practically meaningful, the results are presented as a quotient Q of the finally attained MSE and the training set output variance. In this form, the quotient corresponds to the complement of the well-known coefficient of determination R^2,



(a) One hidden layer (*4mio-1h*), moderate nonlinearity.

(b) Five hidden layers (*4mio-5h*), moderate nonlinearity.

Figure 4.1: The largest one/five-hidden-layer networks (four million parameters) with moderate nonlinearity, loss progress (in log scale) in dependence on the number of epochs.

which is the ratio of the variability explained by the model to the total variability of output patterns. The relationship between both is

$$Q = \frac{\text{MSE}}{\text{Var}\,(y)} = 1 - R^2 \tag{4.6}$$

If Q is, for example, 0.01, the output can be predicted by the trained neural network model with an MSE corresponding to 1 % of its variance.

4.5.1 Moderately nonlinear problems

Networks with weights generated with nonlinearity parameter $d = 2$ (see Section 4.3) can be viewed as *moderately* nonlinear.

Optimization results with shallow (single-hidden-layer) neural networks have shown that the minimum MSE (known to be zero due to the task definitions) can be reached with a considerable precision of 10^{-6} to 10^{-20} even for the largest networks. While double-precision computation is not superior to single precision for smaller networks (in a range of one order of magnitude), the improvements for the networks with one and four million parameters are in the range of two to six orders of magnitude. The optimization progress for the largest network, comparing the dependence on the number epoch equivalents for single and double-precision arithmetic is shown in Figure 4.1a.

With both precisions, CG exhibits superlinear convergence property: between epochs 1,000 and 6,000, the logarithmic plot is approximately a straight line. So every iteration leads to an approximately fixed multiplicative gain of precision of the minimum actually reached. The single-precision computation, however, stops after 2,583 epochs (156 iterations) because line search can't find a better result given the low precision boundary. In other words, the line search in single precision is less efficient. This would be an argument in favor of double precision.

For the network with a single hidden layer, the CG algorithm is clearly superior to RMSprop. The largest network attains a minimum error precision better by five orders of magnitude, and possibly more when increasing the number of epochs. The reason is the superlinear convergence

of CG obvious from (Figure 4.1a). It is interesting that in the initial phase, RMSprop descends faster, quickly reaching the level that is no longer improved in the following iterations.

However, with a growing number of hidden layers, the situation changes. For five hidden layers, single-precision computations lag behind by two orders of magnitude. The reason for this lag is different from that observed with the single-hidden-layer tasks: the single-precision run is prematurely stopped after less than five CG iterations, because of no improvement in the line search. This is why the line of the loss for single-precision CG can hardly be discerned in Figure 4.1b. By contrast, the double-precision run proceeds until the epoch limit is reached (655 iterations).

As seen in Figure 4.1b, the superlinear convergence property with the double-precision computation is satisfied at least segment-wise: a faster segment until about 1,000 epoch equivalents and a slower segment from epoch 4,000. Within each segment, the logarithmic plot is approximately a straight line. (Superlinearity would have to be rejected if the precision gain factors would be successively slowing down, particularly within the latter segment.)

For RMSprop, a lag behind the CG can be observed with five hidden layers, but the advance of CG is minor—one order of magnitude. However, the plot suggests that CG has more potential for further improvement if provided additional resources. Once more, RMSprop exhibits fast convergence in the initial optimization phase followed by weak improvements.

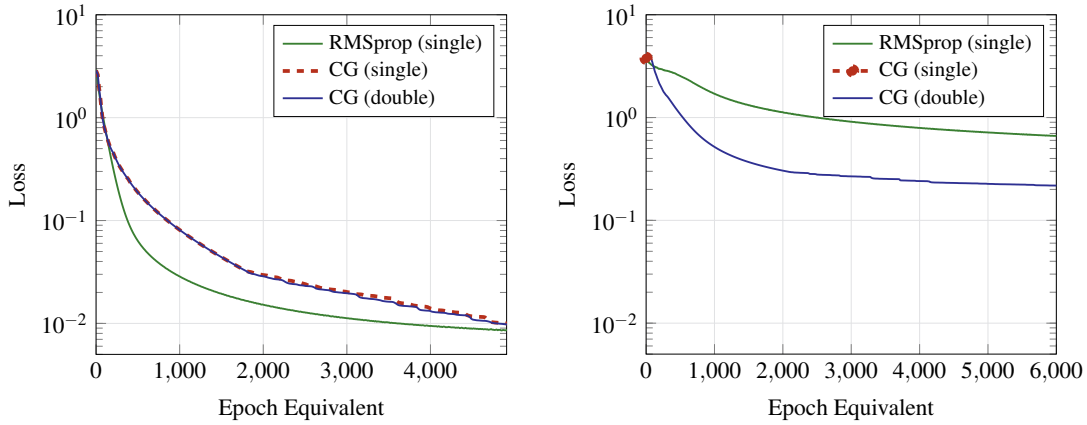
4.5.2 Strongly nonlinear problems

Mapping tasks generated with nonlinearity parameter $d = 4$ imply *strong* nonlinearities in the sigmoid activation functions with 5 % of activations having an activation function derivative of less than 0.02. With a linear term avoiding saturation (Equation (4.4)), this derivative grows to 0.07, a still very low value compared to the unity derivative in the central region of the sigmoid. With CG, the parameter optimization of networks of various sizes with one hidden layer and two hidden layers shows no significant difference between single and double-precision computations. The attainable accuracy of the minimum has been, as expected, worse than for moderately nonlinear tasks but still fairly good: almost 10^{-5} for a single hidden layer and 10^{-2} for two hidden layers.

The relationship between both CG and RMSprop is similar for single-hidden-layer networks (Figure 4.2a)—CG is clearly more efficient. The attainable precision of error minimum is, as expected, worse than for moderately nonlinear tasks.

For five hidden layers, a similar phenomenon as for moderately nonlinear tasks can be observed: the single-precision computation stops prematurely because line search fails to find an improved value (see Figure 4.2b). The minimum reached is in the same region as the initial solution.

The convergence of both algorithms (CG and RMSprop) is not very different with five hidden layers—superiority of CG is hardly significant. Here, the superlinear convergence of CG is questionable. The reason for this may be a lack of convexity of the MSE with multiple hidden layers and strongly nonlinear relationships between input and output. It is important to point out that the quality of the error minimum found is extraordinarily poor: the MSE is about 10 % of the output variance. This is 30 % in terms of standard deviation, which may be unacceptable for many applications.



(a) One hidden layer (*4mio-1h*), strong nonlinearity. (b) Five hidden layers (*4mio-5h*), strong nonlinearity.

Figure 4.2: The largest one/five-hidden-layer networks (four million parameters) with strong nonlinearity, loss progress (in log scale) in dependence on the number of epochs.

4.6 Summary and discussion

With the CG optimization algorithm, double precision computation is superior to single-precision in two cases:

1. For tasks relatively close to convexity (single-hidden-layer networks with moderate nonlinearity), the optimization progress with double-precision seems to be faster due to a smaller number of epochs necessary to reach a line search minimum with a given tolerance. This allows the algorithm to perform more CG iterations with the same number of epochs. However, since both single and double precision have the superlinear convergence property, the gap can be bridged by allowing slightly more iterations with single precision to reach a result equivalent to that of double precision.

2. For difficult tasks with multiple hidden layers and strong nonlinearities, a more serious flaw of single-precision computation occurs: a premature stop of the algorithm because of failing to find an objective function improvement by line search. This may lead to unacceptable solutions.

In summary, it is advisable to use double precision with the second-order methods.

The CG optimization algorithm (with double precision computation) is superior to the first-order algorithm RMSprop in the following cases:

1. Tasks with moderate nonlinearities. The advance of CG is large for shallow networks and less pronounced for deeper ones. Superlinear convergence of CG seems to be retained also for the latter group.

2. Tasks with strong nonlinearities modeled by networks with a single hidden layer. Also here, superlinear convergence of CG can be observed.

For tasks with strong nonlinearities and multiple hidden layers, both CG and RMSprop (which has been by far the best converging method from those implemented in the popular *TensorFlow/Keras* frameworks) show very poor performance. This is documented by the square error attained, whose minimum is known to be zero in our training examples. In practical terms,

such tasks can be viewed as "unsolvable" because the forecast error is too large in relation to the output variability—the model gained does not really explain the behavior of the output.

The advance of RMSprop, if any, in some of the strongly nonlinear cases is not very significant (factors around two). By contrast, for tasks with either moderate nonlinearity or shallow networks, the CG method is superior. In these cases, the advance of CG is substantial (sometimes several orders of magnitude). So in the typical case where the extent of nonlinearity of the task is unknown, CG is the safe choice.

It has to be pointed out that tasks with strong nonlinearities in individual activation functions are, strictly speaking, intractable by any local optimization method. Strong nonlinearities sum up to strongly non-monotonous mappings. But square errors of non-monotonous mappings are certain to have multiple local minima with separate attractors. For large networks of sizes common in today's data science, the number of such separate local minima is also large. This reduces the chance of finding the global minimum to a practical impossibility, whichever optimization algorithms are used. So the cases in which the CG shows no significant advantage are just those "hopeless" tasks.

Next to the extent of nonlinearity, the depth of the network is an important category where the alternative algorithms show different performances. The overall impression that the advance of the CG method over RMSprop shrinks with the number of hidden layers, that is, with the depth of the network, may suggest the conjecture that it is not worth using CG with necessarily double-precision arithmetic for currently preferred deep networks [Hea18].

However, the argument has to be split into two different cases:

1. networks with fully connected layers
2. networks containing special layers, in particular convolutional ones.

In the former case, the question is how far it is useful to use multiple fully connected hidden layers at all. Although there are theoretical hints that in some special tasks, deep networks with fully connected layers may provide a more economical representation than those with shallow architectures [Mon+14] or [DB11], the systematic investigation of [BHH19a] has disclosed no usable representational advantage of deep networks. In addition to it, deep networks are substantially harder to train and thus exploit their representational potential. This can also be seen in the results presented here. Networks with five hidden layers, although known to have a zero error minimum, have not been able to be trained to a square error of less than 10 % of the output variability. Expressed in standard deviation, the standard deviation of the output error is more than 30 % of the standard deviation of the output itself. These 30 % do not correspond to noise inherent to the task (whose error minimum is zero on the training set) but to the error caused by the inability of local optimization methods to find a global optimum. This is a rather poor forecast. In the case of the output being a vector of class indicators, the probability of frequently confusing the classes is high. In this context, it has to be pointed out that no exact methods exist for finding a global optimum of nonconvex tasks of sizes typical for data science with many local minima. The global optimization of such tasks is an NP-complete problem with solution time exponentially growing with the number of parameters. This documents the infeasibility of tasks with millions of parameters.

Limitation to fully connected networks The conjectures of the present work cannot be simply extrapolated to networks containing convolutional layers—this investigation was concerned only with fully connected networks. The reason for this scope limitation is that it is difficult to select a meaningful prototype of a network with convolutional layers, even more one

with a known error minimum—the architectures with convolutional layers are too diversified and application-specific. So the question is which optimization methods are appropriate for training deep networks with multiple convolutional layers but a low number of fully connected hidden layers (maybe a single one). This question cannot be answered here, but it may be conjectured that convolutional layers are substantially easier to train than fully connected ones, for two reasons:

1. Convolutional layers have only a low number of parameters (capturing the dependence within a small environment of a layer unit).
2. The gradient with regard to convolutional parameters tends to be substantially larger than that of fully connected layers since it is a sum over all unit environments within the convolutional layer. In other words, convolutional parameters are "reused" for all local environments that make their gradient grow.

This suggests a meaningful further work: to find some sufficiently general prototypes of networks with convolutional layers and to investigate the performance of alternative optimization methods on them, including the influence of machine precision for the second-order methods.

References

[Aba+15] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: https://www.tensorflow.org/.

[BHH19a] Bernhard Bermeitinger, Tomas Hrycej, and Siegfried Handschuh. "Representational Capacity of Deep Neural Networks: A Computing Study." In: *Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 1: KDIR*. 11th International Conference on Knowledge Discovery and Information Retrieval. Vienna, Austria: SCITEPRESS, 2019, pp. 532–538. ISBN: 978-989-758-382-7. DOI: 10.5220/0008364305320538.

[Cho+15] François Chollet et al. *Keras*. 2015. URL: https://keras.io.

[DB11] Olivier Delalleau and Yoshua Bengio. "Shallow vs. Deep Sum-Product Networks." In: *Advances in Neural Information Processing Systems*. Vol. 24. Curran Associates, Inc., 2011. URL: https://proceedings.neurips.cc/paper/2011/hash/8e6b42f1644ecb1327dc03ab345e618b-Abstract.html.

[FR64] R. Fletcher and C. M. Reeves. "Function Minimization by Conjugate Gradients." In: *The Computer Journal* 7.2 (Feb. 1, 1964), pp. 149–154. ISSN: 0010-4620. DOI: 10.1093/comjnl/7.2.149.

[Hea18] Jeff Heaton. "Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep Learning." In: *Genetic Programming and Evolvable Machines* 19.1 (June 1, 2018), pp. 305–307. ISSN: 1573-7632. DOI: 10.1007/s10710-017-9314-z.

[Hin12] Geoffrey Hinton. "Neural Networks for Machine Learning" (Toronto). 2012. URL: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[Mar10] James Martens. "Deep Learning via Hessian-free Optimization." In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Madison, WI, USA: Omnipress, June 21, 2010, pp. 735–742. ISBN: 978-1-60558-907-7.

[Mon+14] Guido F Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. "On the Number of Linear Regions of Deep Neural Networks." In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 2924–2932. URL: http://papers.nips.cc/paper/5422-on-the-number-of-linear-regions-of-deep-neural-networks.pdf.

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. 2nd ed. Springer Series in Operations Research. New York: Springer, 2006. 664 pp. ISBN: 978-0-387-30303-1.

[Pre+92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*. USA: Cambridge University Press, 1992. ISBN: 978-0-521-43108-8.

[Vir+20] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[Wol69] Philip Wolfe. "Convergence Conditions for Ascent Methods." In: *SIAM Review* 11.2 (Apr. 1969), pp. 226–235. DOI: 10.1137/1011036.

Chapter 5

Make Deep Networks Shallow Again

Abstract Deep neural networks have a good success record and are thus viewed as the best architecture choice for complex applications. Their main shortcoming has been, for a long time, the vanishing gradient which prevented the numerical optimization algorithms from acceptable convergence. A breakthrough has been achieved by the concept of residual connections—an identity mapping parallel to a conventional layer. This concept is applicable to stacks of layers of the same dimension and substantially alleviates the vanishing gradient problem. A stack of residual connection layers can be expressed as an expansion of terms similar to the Taylor expansion. This expansion suggests the possibility of truncating the higher-order terms and receiving an architecture consisting of a single broad layer composed of all initially stacked layers in parallel. In other words, a sequential deep architecture is substituted by a parallel shallow one. Prompted by this theory, we investigated the performance capabilities of the parallel architecture in comparison to the sequential one. The *Computer Vision* (CV) datasets MNIST and CIFAR10 were used to train both architectures for a total of 6,912 combinations of varying numbers of convolutional layers, numbers of filters, kernel sizes, and other meta parameters. Our findings demonstrate a surprising equivalence between the deep (sequential) and shallow (parallel) architectures. Both layouts produced similar results in terms of training and validation set loss. This discovery implies that a wide, shallow architecture can potentially replace a deep network without sacrificing performance. Such substitution has the potential to simplify network architectures, improve optimization efficiency, and accelerate the training process.

5.1 Introduction

Deep neural networks (i.e., networks with many nonlinear layers) are widely considered to be the most appropriate architecture for mapping complex dependencies such as those arising in Artificial Intelligence tasks. Their potential to map intricate dependencies has advanced their widespread use.

For example, the study [Mei+23] compares the first deep convolutional network for image classification with two sequential convolutional layers *LeNet* [LeC+89] to its deeper evolution *VGG16* [SZ15] with 13 sequential convolutional layers. While the performance gain in this comparison was significant, further increasing the depth resulted in very small performance gains. Adding three additional convolutional layers to *VGG16* improved the validation error slightly from 25.6 % to 25.5 % on the ILSVRC-2014 competition dataset [Rus+15], while increasing the number of trainable parameters from 138M to 144M.

However, training these networks remains a significant challenge, often navigated through numerical optimization methods based on the gradient of the loss function. In deeper networks,

the gradient can significantly diminish particularly for parameters distant from the output, leading to the well-documented issue known as the "vanishing gradient".

A breakthrough in this challenge is the concept of *residual connections*: using an identity function parallel to a layer [He+16]. Each residual layer consists of an identity mapping copying the layer's input to its output and a conventional weighted layer with a nonlinear activation function. This weighted layer represents the residue after applying the identity. The output of the identity and the weighted layer are summed together, forming the output of the residual layer. The identity function plays the role of a bridge—or "highway" [SGS15]—transferring the gradient w.r.t. layer output into that of the input with unmodified size. In this way, it increases the gradient of layers remote from the output.

The possibility effectively training deep networks led to the widespread use of such residual connection networks and to the belief that this is the most appropriate architecture type [MLP17]. However, extremely deep networks such as *ResNet-1000* with ten times more layers than *ResNet-101* [He+16] often demonstrate a performance decline.

Although there have been suggestions for wide architectures like *EfficientNet* [TL19], these are still considered "deep" within the scope of this paper.

This paper questions the assumption that deep networks are inherently superior, particularly considering the persistent gradient problems. Success with methods like residual connections can be mistakenly perceived as validation of the superiority of deep networks, possibly hindering exploration into potentially equivalent or even better-performing "shallow" architectures.

To avoid such premature conclusions, we examine in this paper the relative performance of deep networks over shallow ones, focusing on a parallel or "shallow" architecture instead of a sequential or "deep" one. The basis of the investigation is the mathematical decomposition of the mapping materialized by a stack of convolutional residual networks into a structure that suggests the possibility of being approximated by a shallow architecture. By exploring this possibility, we aim to stimulate further research, opening new avenues for AI architecture exploration and performance improvement.

5.2 Decomposition of stacked residual connections

A layer of a conventional multilayer perceptron can be thought of as a mapping $y = F_h(x)$. With the residual connection concept [He+16], this mapping is modified to

$$y = Ix + F_h(x) \tag{5.1}$$

For the h-th hidden layer, the recursive relationship is

$$z_h = Iz_{h-1} + F_h(z_{h-1}) \tag{5.2}$$

For example, the second and the third layers can be expanded to

$$z_2 = Iz_1 + F_2(z_1) \tag{5.3}$$

and

$$\begin{aligned} z_3 &= Iz_2 + F_3(z_2) \\ &= I(Iz_1 + F_2(z_1)) + F_3(Iz_1 + F_2(z_1)) \\ &= Iz_1 + F_2(z_1) + F_3(Iz_1 + F_2(z_1)) \end{aligned} \tag{5.4}$$

In the operator notation, it is

$$z_h = z_{h-1} + F_h * z_{h-1} = (I + F_h) * z_{h-1} \tag{5.5}$$

For linear operators, the recursion up to the final output vector y can be explicitly expanded [see Hry+23, Section 6.7.3.1]

$$y = \quad I * x + \sum_{h=1}^{H} F_h * x + \sum_{h=1}^{H} \sum_{k=1,k>h}^{H} F_k * F_h * x \cdots \tag{5.6}$$

with all combinations of operator triples, quadruples, etc. up to the product of all H layer operators.

Typically, these layer mappings are not linear due to their activation functions such as *sigmoid*, *tanh*, or *ReLU*. As a result, it does not satisfy the condition $F_h(x + z) = F_h(x) + F_h(z)$. However, their gradient is a linear operator. In a multilayer perceptron with a residual connection, the error gradient w.r.t. the output of the h-th layer is

$$\begin{aligned} \frac{\partial E}{\partial z_h} &= \left(\prod_{k=h+1}^{H} \frac{\partial z_k}{\partial z_{k-1}} \right) \frac{\partial E}{\partial z_H} \\ &= \left(\prod_{k=h+1}^{H} \left(I + W_k^T \nabla F_k \right) \right) \frac{\partial E}{\partial z_H} \end{aligned} \tag{5.7}$$

The error gradient w.r.t. the weights is, for both standard layers and those with residual connection

$$\frac{\partial E}{\partial W_h} = \nabla F_h \frac{\partial E}{\partial z_h} z_{h-1}^T \tag{5.8}$$

and w.r.t. biases

$$\frac{\partial E}{\partial b_h} = \nabla F_h \frac{\partial E}{\partial z_h} \tag{5.9}$$

This shows that the expansion given in Equation (5.6) is valid for an approximation linearized with the help of the local gradient. In particular, it is valid around the minimum.

In an analogy to Taylor expansion, it can be hypothesized that the first two terms

$$y = I * x + \sum_{h=1}^{H} F_h * x \tag{5.10}$$

may be a reasonable approximation of the whole mapping in Equation (5.6).

In terms of implementation as neural networks, the stack of layers with residual connections (as exemplified in Figure 5.1) could be approximated by the parallel architecture such as that illustrated in Figure 5.2.

Of course, this hypothesis has to be confirmed by tests on real-world problems. If acceptable, it would be possible to substitute a deep residual network of H sequential layers with a "shallow" network with a single layer consisting of H individual modules in parallel, summing their output vectors. Each of these modules would be equivalent to one layer in the deep architecture. The main objective is not to prove that both networks are nearly equivalent with the same parameter set, as this is unlikely to be the case. Rather, the goal is to demonstrate that both shallow and deep architectures can effectively learn and attain comparable performances on the given task. The consequence would be that the shallow architecture can reach the same

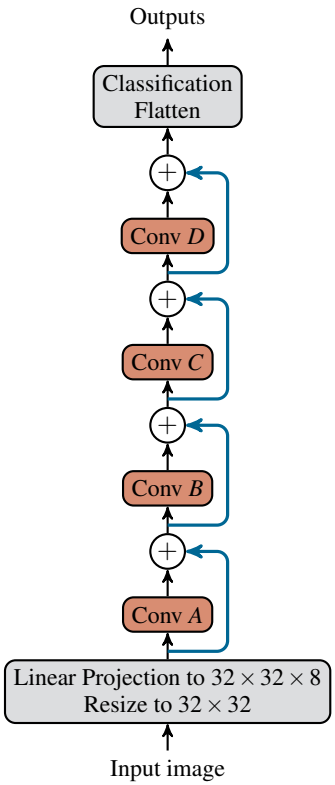


Figure 5.1: Overview of the sequential architecture with four consecutive convolutional layers with eight filters each and their skip connections.

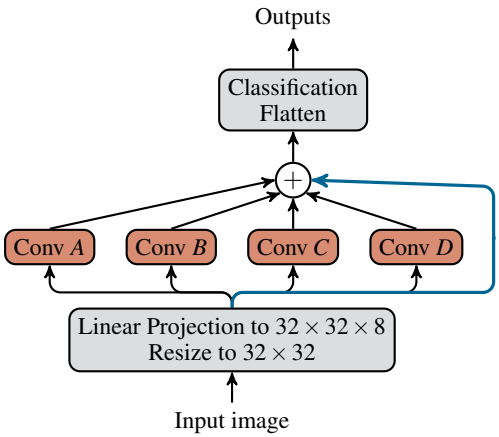


Figure 5.2: Overview of the parallelized architecture of Figure 5.1 with four convolutional layers with eight filters each and one skip connection.

performance as the deep one, with the same number of parameters. This may be relevant for the preferences in setting up neural networks for particular tasks since shallow networks suffer less from numerical computing problems such as vanishing gradient.

5.3 Setup of computing experiments

The analysis of Section 5.2 suggests that the expressive power of a network architecture in which stacked residual connection layers of a deep network are reorganized into a parallel operation in a single, broad layer, may be close to that of the original deep network. This hypothesis is to be tested on practically relevant examples.

It is important to point out that residual connection layers are restricted to partial stacks of equally sized layers (otherwise the unity mapping could not be implemented). A typical use of such networks is image classification where an image is processed by consecutive layers of size equal to the (possibly reduced) pixel matrix. The output of this network is usually a vector of class probabilities that differ in dimensionality from that of the input image. This is the reason for one or more non-residual layers at the output and some preprocessing non-residual layers at the input.

Residual connections can be used for any stack of layers of the same dimensions. However, in domains such as image processing, the layers are mostly of the *convolutional* type. This is a layer concept in which the same, relatively small weight matrix, is applied to the neighbor environment of every position in the input. They are implementing a local operator (such as edge detection) shifted over the extension of the image. The following benchmark applications are using convolutional layers.

Filters are a concept in convolutional layers which consist of a multiplicity of such convolution operators. Each filter convolves individually with the input matrix for generating the output. Multiple filters in a layer operate independently from each other, building a parallel structure. The computing experiments reported here were done both with and without multiple filters. The possibility of making the consecutive layer stack parallel concerns only the middle part with residual connections of identically sized layers.

For the experiments, the two well-known image classification datasets MNIST [LeC+98] and CIFAR10 [Kri09] were used. MNIST contains black and white images of handwritten digits (0–9) while CIFAR10 contains color images of exclusively ten different mundane objects like “horse”, “ship”, or “dog”. They contain 60,000 (MNIST) and 50,000 (CIFAR10) training examples. Their respective preconfigured test split of each 10,000 examples are used as validation sets. While CIFAR10 is evenly distributed among all classes, MNIST is roughly evenly distributed with a standard deviation of 322 for the training set and 59 for the validation set. We took no special treatment for this small class imbalance.

A series of computing experiments of all the following possible architectures were run:

- Number of convolutional layers: 1, 2, 4, 8, 16, 32
- Number of filters per convolutional layer: 1, 2, 4, 8, 16, 32
- Kernel size of a filter: 1×1, 2×2, 4×4, 6×6, 8×8, 16×16
- Activation function of each convolutional layer: *sigmoid*, *ReLU*

Figure 5.1 shows the sequential architecture with depth 4 and 8 filters per convolutional layer. For comparison, the parallelized version is shown in Figure 5.2. The sizes of the filters' kernels are not shown because they don't interfere with the layout.

The images are resized to 32×32 pixels to match the varying kernel sizes. For the summation of the skip connection and the convolutional layer to work out, they need to have the same dimensionality. Therefore, for preprocessing, the images are linearly mapped to match the convolutional layers' output dimensions. To keep the architecture simple and reduce the possibility of additional side effects, the input is flattened into a one-dimensional vector before the dense classification layer with ten linear output units. These linear layers are initialized with the same set of fixed random values throughout all experiments.

The same configuration setup was used for the number of parallel filters per layer. Parallel filters are popular means of extending a straightforward convolution layer architecture: instead of each layer being a single convolution of the previous layer, it consists of multiple convolution filters in parallel. In all well-performing image classifiers based on convolutional layers, multiple filters are used [Fuk80; KSH12; SZ15].

Throughout all experiments, the parameters of the layers at the same depths were always initialized with the same random values with a fixed seed. For example, the two layers labeled *A* in Figures 5.1 and 5.2 started their training from the same parameter set.

The categorical cross-entropy loss was employed as the loss function due to its suitability for multi-label classification problems. This loss served also as the main assessment of the training performance. An alternative would have been the most popular (and the most meaningful from the application point of view) metric: classification accuracy. However, it would be a methodological fault to use a metric that is different from the loss function that is genuinely optimized. The relationship between cross-entropy loss and classification accuracy is loaded with random effects and is frequently not even monotonic. This justifies the selection of cross-entropy loss for performance review.

The batch size was set to 512. The datasets were not shuffled between epochs or experiments, leading to identical batches throughout all experiment runs.

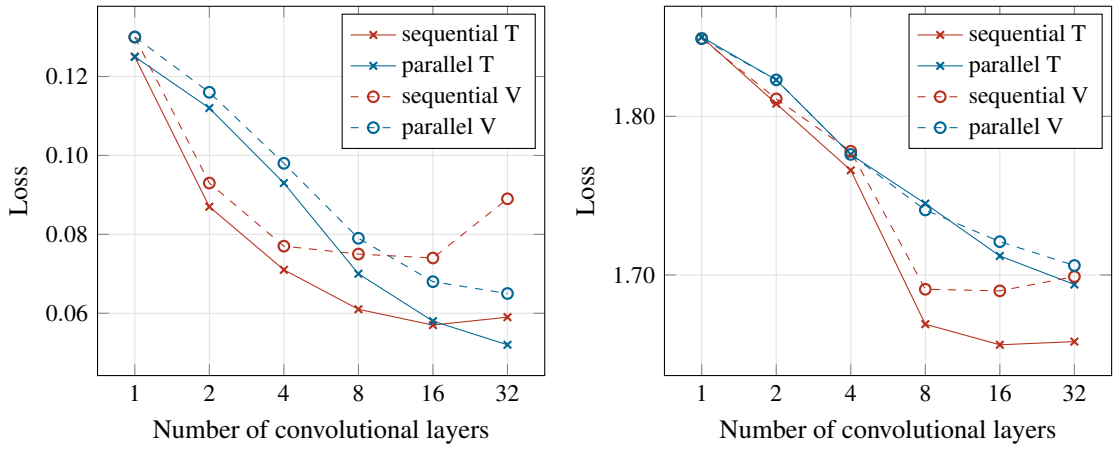
As the optimizer, *Root Mean Square Propagation* (RMSprop) [Hin12] was chosen with a fixed learning rate. All experiments were duplicated for the learning rates 10^{-2}, 10^{-3}, 10^{-4}, and 10^{-5}. Different learning rates had only a marginal effect on the results. The figures and tables show the results obtained with a learning rate of 10^{-4}.

Each experiment ran for 100 epochs, which resulted in 11,800 optimization steps for MNIST, and 9,800 steps for CIFAR10. The 6,912 experiments were run individually on *NVIDIA Tesla V100* GPUs for a total run time of 79 days. The results are reported for kernel size 16×16 which showed the best average classification performance although not significantly different.

5.4 Computing experiments

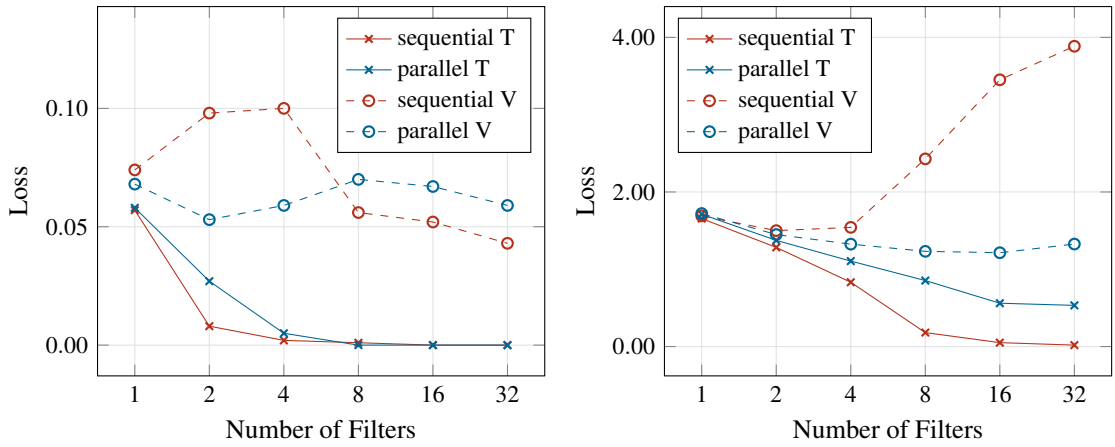
5.4.1 With a single filter

The losses after the 100 epochs for the training set (T) and the validation set (V) are given in Figure 5.3. The performance of both architectures can be observed by the points on the red (sequential architecture) and blue (parallel variant) points. The solid lines represent the training loss and the dashed lines the validation loss.



(a) MNIST (b) CIFAR10

Figure 5.3: Sequential vs. parallel architecture: loss dependence on the number of residual convolutional layers (with a single filter per layer) for the two datasets MNIST (left) and CIFAR10 (right)



(a) MNIST (b) CIFAR10

Figure 5.4: Sequential vs. parallel architecture: loss dependence on the number of filters (with 16 convolutional layers) for the two datasets MNIST (left) and CIFAR10 (right)

Due to their identical layout and equal random initialization, training the two networks with one convolutional layer and one filter each resulted consequently in equal loss values.

It can be observed that both architectures perform similarly, in particular for the largest depths of 16 and 32. For MNIST, the shallow, parallel architecture slightly outperforms the original, sequential one, while the relationship is inverse for the CIFAR10 dataset.

5.4.2 With multiple filters

A single-filter architecture is the most transparent one but it is scarcely used. It is mostly assumed that more filters are necessary to reach the desired classification performance. Therefore, experiments with multiple (1 to 32) filters per convolutional layer are included.

Same as before, the results after training for 100 epochs are shown in Figures 5.4a and 5.4b.

Table 5.1: Overdetermination ratios for both datasets and different model sizes based on the number of filters per convolutional layer

		Overdetermination ratio Q	
#filters	#parameters	MNIST	CIFAR10
1	14k	41.771	34.804
2	37k	16.256	13.545
4	106k	5.630	4.691
8	344k	1.743	1.453
16	1.2M	0.495	0.412
32	4.5M	0.132	0.110

They show an interesting development for CIFAR10: the training loss decreases by raising the number of filters while the validation loss largely increases for more than four filters. The validation loss considerably deteriorates for the sequential architecture. (The results for MNIST are similar for the training set but less interpretable for the validation set.)

The reason for the distinct picture on CIFAR10 is to be sought in relationships between constraints imposed by the task and the number of free trainable parameters [Hry+23, Chapter 4]. A task with $K = 50{,}000$ training examples constitutes equally many constraints (resulting from the goal to accurately match the target values) for each output value. For 10 classes, there are $M = 10$ such output values whose reference values are to be correctly predicted by the classifier. This creates KM constraints (here: $50{,}000 \times 10 = 500{,}000$). For the mapping represented by the network, there are P free (i.e., mutually independent) parameters to make the mapping satisfy the constraints.

- With $P = KM$, the system is perfectly determined and could be solved exactly.
- With $P > KM$, the system is underdetermined. A part of the parameters is set to arbitrary values so that novel examples from the validation set receive arbitrary predictions.
- With $P < KM$, the system is overdetermined, and not all constraints can be satisfied. This may be useful if the data are noisy, as it is not desirable to fit to noise.

An appropriate characteristic is the overdetermination ratio Q from [HBH22a] defined as

$$Q = \frac{KM}{P} \tag{5.11}$$

The number of genuinely free parameters is difficult to figure out. It can only be approximated by the total number of parameters, keeping in mind that the number of actually free parameters can be lower.

In training a model by fitting to data, the presence of the noise has to be considered. The model should reflect the underlying genuine laws in the data but not the noise. Fitting to the latter is undesirable and is the substance of the well-known phenomenon of *overfitting*. It was shown in [Hry+23, Chapter 4] that fitting to the additive noise and thus the influence of training set noise to the model prediction is reduced to the fraction $1/Q$. In other words, it is useful to keep the overdetermination ratio Q significantly over 1.

This supplementary information for the plotted variants is given in Table 5.1. Acceptable values of the overdetermination ratio Q are given with filter counts of 1, 2, and 4. This is consistent with the finding that overfitting did not take place in single-filter architectures presented in Section 5.4.1.

For 8 filters or more, Q is close to 1 or even below it. In this group, the validation loss can grow arbitrarily although the training loss is reduced. This is the result of arbitrarily assigned values of underdetermined parameters.

Altogether, the parallel architecture shows better performance on the validation set despite the slightly inferior loss on the training set. This can be attributed rather to the random effects of underdetermined parameters than to the superiority of one or other architecture. In this sense, both architectures can be viewed as approximately equivalent concerning their representational capacity.

5.4.3 Trade-off of the number of filters and the number of layers

As an additional view to the relationship between the depth and the width of the network, a group of experiments is analyzed in which the product of the number of filters (F) and the number of convolution layers (C) are kept constant. In this way, also "intermediary" architectures between deep and shallow ones are captured. For example, an architecture with 32 filters and a single convolutional layer has a ratio of $1/32$ while the ratio with one filter and 32 layers is $32/1$. For 16 layers with each 8 filters, it is $16/8 = 2$.

For the product of 32, there are the following combinations of $C \times F$: 1×32, 2×16, 4×8, 8×4, 16×2 and 32×1. In Figure 5.5, they are ordered along their depth-width ratio C/F: $1/32$, $2/16$, $4/8$, $8/4$, $16/2$, and $32/1$. These architectures are represented by the red curves.

As a reference, the blue curve shows their shallow counterparts. Those are all single-layer architectures. They differ only in the number of parameters, consistent with their sequential counterparts represented by the red curve. The difference in the number of parameters is due to the different sizes of the classification layer following the residual connection sequence. This classification layer is broader for more filters as its input is larger the more filters there are.

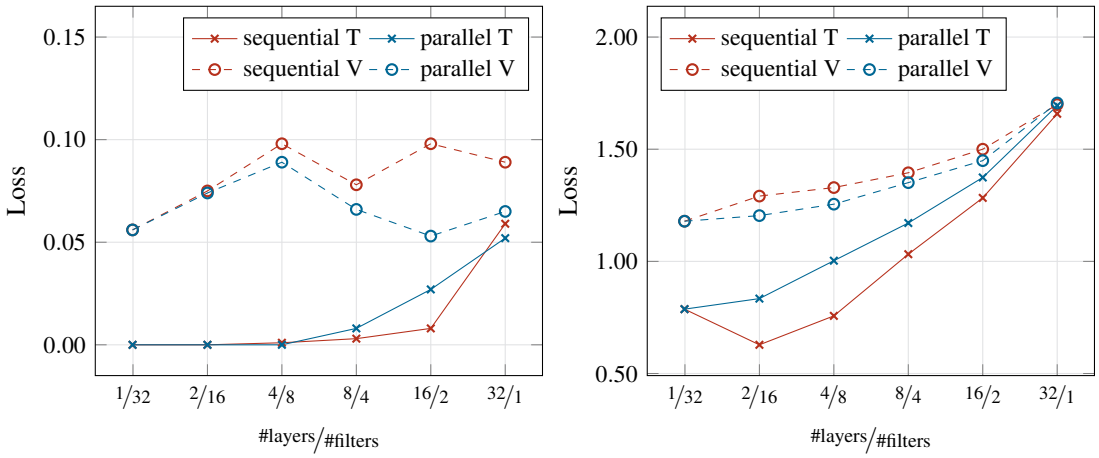
Both the training and validation losses increase with the depth-width ratio, indicating the superiority of the shallow architectures. However, it is important to note that this comparison may not be completely fair due to the inherent difference in parameter numbers. Specifically, variants with higher depth-width ratios have a diminishing number of parameters resulting from their smaller number of filters.

In Figures 5.5a and 5.5b, it can be observed that the training loss for flattened alternatives is slightly larger compared to the other architectures. However, the validation loss for flattened alternatives is smaller, albeit to a moderate extent.

In summary, the deep variants can certainly not be viewed as superior in overall terms. Both architectures are roughly equivalent, as long as the number of parameters is equal.

5.5 Statistics of experiments

In addition to experiment runs selected for the presentation in the previous sections, statistics over all 6,912 runs, partitioned into some categories, may be useful to complete the performance



(a) MNIST (b) CIFAR10

Figure 5.5: Sequential vs. parallel architecture: loss dependence on the ratio of the numbers of layers and filters (product of the number of layers and the number of filters is fixed at 32) for the two datasets MNIST (left) and CIFAR10 (right)

Table 5.2: Mean training and validation loss for sequential and parallel architectures and various determination ratios Q intervals

	$Q \in [0, 1)$		$Q \in [1, 3)$		$Q \in [3, 10)$		$Q \in [10, \infty)$	
	train	val	train	val	train	val	train	val
MNIST								
sequential	0.00013	0.05201	0.01702	0.12449	0.03620	0.11743	0.11246	0.13550
parallel	0.00009	0.07551	0.02679	0.11468	0.05238	0.11467	0.13310	0.14900
CIFAR10								
sequential	0.25326	2.03107	0.72510	1.31691	1.07333	1.34721	1.58608	1.65354
parallel	0.52658	1.32386	0.88701	1.24884	1.17085	1.34227	1.63449	1.68879

picture. Of course, averaging hundreds to thousands of experiments does not guarantee to reflect all theoretical expectations succinctly; it can only confirm rough trends.

This statistical summary is presented in Table 5.2. The losses for training and validation as well as for sequential and parallel architectures are partitioned into intervals of overdetermination ratio to show the different behavior.

According to the theory, with a growing overdetermination ratio, the discrepancy between training and validation loss becomes smaller. On the other hand, larger overdetermination ratios imply smaller numbers of free network parameters. Sometimes, this leads to increased losses from the diminished representation capacity of the network. For ratios smaller than 1, the validation loss may arbitrarily grow because of underdetermined parameters fitted to training data noise (*overfitting*). This arbitrary growth may be more or less articulated, depending mostly on random factors. However, there is always a considerable risk of such poor generalization.

As observed in the individual experiments presented, small discrepancies between training and validation loss are reached for overdetermination ratios larger than 3 for CIFAR10 and larger than 10 for MNIST. These small discrepancies testify to good generalization capability, expected for large overdetermination ratios.

With $Q < 1$, the validation loss deteriorates for CIFAR10 data if compared with the Q of the higher interval. This is the effect of arbitrary parameter values caused by underdetermination.

To summarize, there is a slight advance of shallow architectures for the validation set (five out of eight categories), and deep architectures are better on the training set. The training and validation losses are mostly closer together for the parallel architecture.

5.6 Conclusion

It is stated in Section 5.2 that a deep residual connection network can be approximately expanded into a sum of shorter (i.e., less deep) sequences of different orders. Truncating the expansion to the first two terms results in a shallow architecture with a single layer. This suggests a hypothesis that the representational capacity of such a shallow architecture may be roughly as large as that of the original deep architecture. If validated, this hypothesis could open avenues to bypass issues typically associated with deep architectures.

Subsequent computational experiments conducted on two widely recognized image classification tasks, MNIST and CIFAR10, seem to confirm this theoretically founded expectation. The performance of both architectures (in configurations with identical numbers of network parameters) is close to each other, with a slight advance of shallow architectures in terms of loss on the validation set.

While the deep architecture performed marginally better on the training set, the cause of its underperformance on the validation set remains an open question. It is plausible that the deep architecture's ability to capture abrupt nonlinearities may also make it prone to overfitting to noise. In contrast, the shallow network, due to its inherent smoothness, might exhibit a higher tolerance towards training set noise.

In conclusion, our results suggest a potential parity in the performance of deep and shallow architectures. It is important to note that the optimization algorithm utilized in this study is a first-order one, which lacks guaranteed convergence properties. Future research could explore the application of more robust second-order algorithms, which, while not commonly implemented in prevalent software packages, could yield more pronounced results. This work serves as a preliminary step towards reevaluating architectural decisions in the field of neural networks, urging further exploration into the comparative efficacy of shallow and deep architectures.

References

[Fuk80] Kunihiko Fukushima. "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position." In: *Biological Cybernetics* 36.4 (Apr. 1, 1980), pp. 193–202. ISSN: 1432-0770. DOI: 10.1007/BF00344251.

[He+16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[Hin12] Geoffrey Hinton. "Neural Networks for Machine Learning" (Toronto). 2012. URL: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[Hry+23] Tomas Hrycej, Bernhard Bermeitinger, Matthias Cetto, and Siegfried Handschuh. *Mathematical Foundations of Data Science*. Texts in Computer Science. Cham: Springer International Publishing, Mar. 13, 2023. ISBN: 978-3-031-19074-2. DOI: 10.1007/978-3-031-19074-2.

[HBH22a] Tomas Hrycej, Bernhard Bermeitinger, and Siegfried Handschuh. "Number of Attention Heads vs. Number of Transformer-encoders in Computer Vision." In: *Proceedings of the 14th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*. 14th International Conference on Knowledge Discovery and Information Retrieval. Valletta, Malta: SCITEPRESS - Science and Technology Publications, 2022, pp. 315–321. ISBN: 978-989-758-614-9. DOI: 10.5220/0011578000003335.

[Kri09] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Dataset. University of Toronto, 2009, p. 60.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html.

[LeC+98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition." In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 1558-2256. DOI: 10.1109/5.726791.

[LeC+89] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition." In: *Neural Computation* 1.4 (Dec. 1, 1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.

[Mei+23] Yuval Meir et al. "Efficient Shallow Learning as an Alternative to Deep Learning." In: *Scientific Reports* 13.1 (Apr. 20, 2023), p. 5423. ISSN: 2045-2322. DOI: 10.1038/s41598-023-32559-8.

[MLP17] Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. "When and Why Are Deep Networks Better Than Shallow Ones?" In: *Proceedings of the AAAI Conference on Artificial Intelligence* 31.1 (1 Feb. 13, 2017). ISSN: 2374-3468. DOI: 10.1609/aaai.v31i1.10913.

[Rus+15] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge." In: *International Journal of Computer Vision* 115.3 (Dec. 1, 2015), pp. 211–252. ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y.

[SZ15] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. Apr. 10, 2015. DOI: 10.48550/arXiv.1409.1556. preprint.

[SGS15] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. "Training Very Deep Networks." In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. NIPS. NIPS'15. Cambridge, MA, USA: MIT Press, Dec. 7, 2015, pp. 2377–2385.

[TL19] Mingxing Tan and Quoc Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." In: *Proceedings of the 36th International Conference on Machine Learning*. International Conference on Machine Learning. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, June 9–15, 2019, pp. 6105–6114. URL: https://proceedings.mlr.press/v97/tan19a.html.

Acknowledgments

This dissertation has been like traveling on a long road. Many people were wandering alongside me, for which I'm exceptionally grateful.

The first person to thank is my supervisor, Prof. Siegfried Handschuh. Our paths crossed nearly ten years ago when I was a Master's student at the University of Passau working on a text mining project. Your support and instant belief in me never ceased after I concluded my Master's thesis with you. Your encouragement even increased after accepting me as a doctorate student, packing up my whole life, and moving to St.Gallen with you. I'm eternally grateful for your invaluable advice, insightful comments, and tremendous support during every stage of this dissertation. This dissertation would not have been possible without your constant feedback and guidance.

The next person to thank is Dr. Tomas Hrycej. Your imaginative sparks and remarks, and very long discussions gave me the opportunity to think outside the box while still maintaining a grounded way of reasoning about various topics. I cannot thank you enough.

In addition, I would like to express my gratitude to my colleagues and co-students, current and past, from the *Digital Libraries and Web Information* chair at the University of Passau, as well as the chair for *Data Science and Natural Language Processing* from the University of St.Gallen. They shaped, in some way or format, the creation of this thesis: pushing when needed, distractions when needed, but also extraordinary help in times of stress. I would like to thank Juliano (also Ph.D. now) for sharing the office space and making it a joyful, productive, and energetic work environment.

These people helped push me forward on the road toward this doctorate. However, my parents gave me the car that enabled me to drive this rocky road (quite literally). They always provide a safe haven for me and my little family. Thank you, Mom and Dad, for your constant support and patience.

Finally, the co-driver next to me has always been the love of my life, my wife, Jessica. Thank you for the life you built for us around the unconventional working hours caused by submission deadlines suddenly appearing. Nothing proves this more than bringing our two beautiful children, Felix and Moritz, into this world and onto the backseat of our metaphorical car. Their bedtime stories consisted of neural networks.

I'm forever grateful for everything you gave. This degree is our shared success.