UNIVERSITÄT ZU LÜBECK

From the Institute of Signal Processing
of the University of Lübeck
Director: Prof. Dr.-Ing. Alfred Mertins

Shaping Sparse Training Methods for Deep Neural Networks

Dissertation
for Fullfilment of
Requirements
for the Doctoral Degree
of the University of Lübeck

from the Department of Computer Sciences and Technical Engineering

Submitted by
Paul Wimmer
from Mutlangen

Lübeck 2024

First referee: PD Dr.-Ing. habil. Alexandru Paul Condurache
Second referee: Prof. Dr.-Ing. Erhardt Barth
Date of oral examination: 12th March, 2024
Approved for printing. Lübeck, 18th March, 2024

UNIVERSITÄT ZU LÜBECK

Aus dem Institut für Signalverarbeitung
der Universität zu Lübeck
Direktor: Prof. Dr.-Ing. Alfred Mertins

Gestaltung von spärlichen Trainingsmethoden für tiefe neuronale Netze

Inauguraldissertation
zur
Erlangung der Doktorwürde
der Universität zu Lübeck

Aus der Sektion Informatik/Technik

vorgelegt von
Paul Wimmer
aus Mutlangen

Lübeck, 2024

1. Berichterstatter/Berichterstatterin: PD Dr.-Ing. habil. Alexandru Paul Condurache
2. Berichterstatter/Berichterstatterin: Prof. Dr.-Ing. Erhardt Barth
Tag der mündlichen Prüfung: 12. März 2024
Zum Druck genehmigt. Lübeck, den 18. März 2024

Abstract

Nowadays, artificial intelligence applications are often solved with the help of Deep Learning methods as they provide excellent performance. The success of deep learning is driven by big models that have a huge number of parameters and therefore require a lot of resources. In order to ensure a good generalization capability of the models for a given task, their parameters must be trained on large amounts of data. Altogether, training such big models requires a lot of time and energy – which is costly.

Reducing the number of parameters helps to lower the memory footprint as well as the computational complexity of a model with positive effects on monetary costs. Commonly, parameters in a deep neural network are set to zero to get both, storage and computation time reduction. This is called pruning. However, many pruning methods require that a dense model is trained before its dimensionality can be reduced. Such a procedure decreases costs for inferring the model but has an even higher training cost than a dense model. The high training-time expenses for deep learning can be lowered significantly by pruning a model before training begins while keeping the model sparse during the whole training process. Up to now, these sparse training methods are not able to achieve similar inference-time results as pruning a densely pre-trained model. This thesis presents three methods that boost sparse training of deep neural networks.

The first proposed approach improves sparse training by freezing untrained weights at their random initial position instead of fixing them at zero. Especially for high compression rates, this helps guaranteeing a high information flow whereas pruning at initialization tends to generate models with a low information flow. By initializing the model with a pseudo random number generator, the frozen weights do not need to be stored but can be recovered by a single 32bit random seed number. Moreover, gradients for the frozen weights do not have to be computed which also decreases computational costs during training.

Second, COPS combines two pruning scores to generate a more powerful pruning at initialization strategy. Pruning weights based on a single score works well for some

architectures and compression rates but may also fail for other ones. Optimizing for more than one score, however, prevents the pruning criterion to overly specialize on a single situation – like a certain range of pruning rates. Hereby, the weaknesses of each single pruning method is overcome by carefully balancing between two of them. This is achieved by solving a constrained combinatorial optimization problem efficiently via a relaxation on a linear program. As a consequence, COPS helps to generate pruning methods applied at initialization which are tailored for the given application, network architecture and compression rate.

The third presented method is called interspace pruning which improves pruning methods in general, including pruning methods applied before training starts. Instead of representing $K \times K$ convolutional filters by their spatial coordinates, a basis for the filters is trained jointly with the corresponding coefficients. By pruning the filter basis coefficients, the dedicated sparse models better approximate dense filters than pruning the spatial coordinates to the same sparsity. This leads to better training of sparse models, especially for high compression rates. To keep the number of trained parameters low, filter bases are shared among many filters. Hereby, sharing filter bases between convolutional filters regularizes training and therefore enhances the generalization ability of the models – also for low compression rates.

The improvements of the proposed methods compared to the state-of-the-art are shown in extensive experimental evaluations. The analyzed deep learning tasks are the popular image classification problems on ImageNet, Tiny ImageNet, CIFAR-10, CIFAR-100 and MNIST solved with numerous modern deep learning architectures.

Kurzfassung

Anwendungen der künstlichen Intelligenz werden heutzutage meist teilweise, oder sogar ganz, von Deep Learning Methoden aufgrund ihrer herausragenden Leistungen gelöst. Der Erfolg von Deep Learning wird durch die Verwendung von sehr großen Modellen die viele Ressourcen benötigen vorangetrieben. Um eine gute Generalisierungsfähigkeit dieser Modelle zu gewährleisten müssen deren Parameter auf einer großen Menge an Trainingsdaten trainiert werden. Insgesamt ist das Trainieren solch großer Modelle kostspielig und verbraucht viel Zeit und Energie.

Die Verringerung der Anzahl der Netzwerkparameter hilft, den Speicherplatzbedarf und Rechenaufwand eines Modells zu reduzieren. Dies spart Zeit, Energie und damit Geld. In der Regel werden in einem tiefen neuronalen Netz die Parameter auf Null gesetzt, um beides, Speicherplatz und Rechenzeit zu reduzieren. Dies wird als Pruning bezeichnet. Allerdings benötigen viele Pruningmethoden ein vollständig trainiertes, dichtes Modell bevor dessen Parameter auf Null gesetzt werden können. So ein Vorgehen senkt die Kosten für die Inferenz des Modells, hat jedoch höhere Trainingskosten als ein dichtes Modell. Diese hohen Trainingskosten können überwunden werden, indem ein Modell vor Beginn des Trainings geprunt wird und die Anzahl der auf Null gesetzten Parameter während des gesamten Trainings gleich bleibt. Jedoch sind Pruningmethoden die vor dem Training angewendet werden bislang nicht in der Lage, genauso gute Ergebnisse zu erzielen wie das Pruning eines dicht vortrainierten Modells. In dieser Arbeit werden drei Methoden vorgestellt, die das spärliche Training von tiefen neuronalen Netzen verbessern.

Der erste vorgeschlagene Ansatz verbessert das spärliche Training durch Einfrieren der nicht trainierten Gewichte an ihrer zufällig gezogenen Startposition, anstatt sie auf Null zu setzen. Dies gewährleistet einen hohen Informationsfluss, auch bei hohen Kompressionsraten. Pruning das direkt bei der Initialisierung angewendet wird neigt hingegen dazu, Modelle mit einem geringen Informationsfluss zu erzeugen. Durch die Initialisierung des Modells mit einem Pseudozufallszahlengenerator, müssen die eingefrorenen Gewichte nicht gespeichert werden, sondern können durch eine einzige 32-Bit-Zufallszahl wiederhergestellt werden. Des Weiteren müssen die Gradienten für die eingefrorenen Gewichte nicht berechnet werden was auch die Rechenkosten während des Trainings senkt.

Zweitens kombiniert COPS zwei Bewertungskriterien für eine spärliche Architektur um eine leistungsfähigere Strategie für das Pruning vor dem Training zu erhalten. Wenn Gewichte nur auf Grund einer einzelnen Pruningbewertung auf Null gesetzt werden, funktioniert dies in manchen Situationen gut, in anderen hingegen schlecht. Wird mehr als ein Bewertungskriterium zu Rate gezogen, sind die spärlichen Architekturen nicht mehr auf eine bestimmte Situation besonders fokussiert. Hierdurch können die Schwächen jeder einzelner Pruningmethode durch sorgfältige Balance zwischen zwei Kriterien überwunden werden. Dies wird durch die effiziente Lösung eines kombinatorischen Optimierungsproblems mit Nebenbedingungen mit Hilfe einer Relaxation auf ein lineares Programm erreicht.

Die dritte Methode wird als Interspace Pruning bezeichnet und verbessert allgemeine Pruningmethoden, also auch Pruningmethoden die vor Beginn des Trainings angewendet werden. Anstatt $K \times K$ Faltungsfilter durch ihre räumlichen Koordinaten darzustellen, wird eine Basis für die Filter zusammen mit den entsprechenden Koeffizienten trainiert. Durch das Pruning der Filterbasiskoeffizienten nähern sich die spärlichen Modelle den dichten Filtern besser an als durch das Pruning der räumlichen Koordinaten. Dies führt zu einem besseren Training von spärlichen Modellen, insbesondere bei hohen Kompressionsraten. Um die Anzahl der trainierten Parameter gering zu halten, wird eine einzelne Filterbasis von vielen Filtern geteilt. Durch die gemeinsame Nutzung der Filterbasen wird das Training regularisiert, was die Ergebnisse auch bei niedrigen Kompressionsraten verbessert.

Die Verbesserungen der vorgeschlagenen Methoden verglichen mit dem Stand der Technik werden in umfangreichen experimentellen Auswertungen gezeigt. Die evaluierten Deep Learning Aufgaben sind die populären Bildklassifikationsaufgaben ImageNet, Tiny ImageNet, CIFAR-10, CIFAR-100 und MNIST welche mit zahlreichen modernen Deep Learning Netzwerkarchitekturen gelöst werden.

Acknowledgements

I would like to thank my supervisors Dr. rer. nat. Jens Mehnert and PD Dr.-Ing. habil. Alexandru Paul Condurache for their support, guidance, fruitful discussions, helpful ideas and constructive critics. Moreover, I would like to thank my Bosch colleagues, in particular Ali, Julia, Matthias, Max and Rinor, for their support, encouragement and feedback. Finally, I want to thank my family, Doro, Wello, Max and Melissa, for their support and encouragement during the time of my PhD.

Table of Contents

Chapter 1
Introduction

1.1 Motivation for Improving Sparse Deep Neural Network Training

In recent years, deep neural networks (DNNs) have shown state-of-the-art (SOTA) performance in many artificial intelligence applications, like image classification [141], speech recognition [137] or object detection [185]. To master these tasks with outstanding quality, DNNs usually have a parameter count in the millions or even billions. Not only the number of parameters is huge, but also the number of floating point operations (`FLOPs`) needed to train and evaluate such a model. Moreover, training these DNNs requires a massive amount of data to harness the power of the huge number of parameters [31, 35, 167, 170].

Training and inferring such large models has been made possible by the technological progress over the last decade. In order to do this in a reasonable runtime, specialized hardware is used as for example GPUs or TPUs. To accelerate training, models are usually trained on more than one GPU/TPU. For example, XLNet [200] requires 5.5 days of training on 512 TPU v3 chips. Deep learning (DL) therefore consumes a lot of energy. For natural language processing, running a neural architecture search for a large transformer network and training it afterwards has a carbon footprint of 284 tons [169]. This is equivalent to the CO_2 emissions of 315 passengers flying from New York City to San Francisco – corresponding to one person flying 32.5 times around the globe. Of course, storing, transferring and inferring such large DNNs is costly as well.

Figure 1.1 shows the increase of `FLOPs` required to train a SOTA artificial intelligence (AI) system since the 1960's. The plot shows two eras of AI, where the modern era is dominated by DL methods. As can be seen, the computational demands are growing much faster nowadays than forecasted by Moore's Law [58]. This growth in computational requirements is, on the one hand, induced by new innovative DNN architectures which

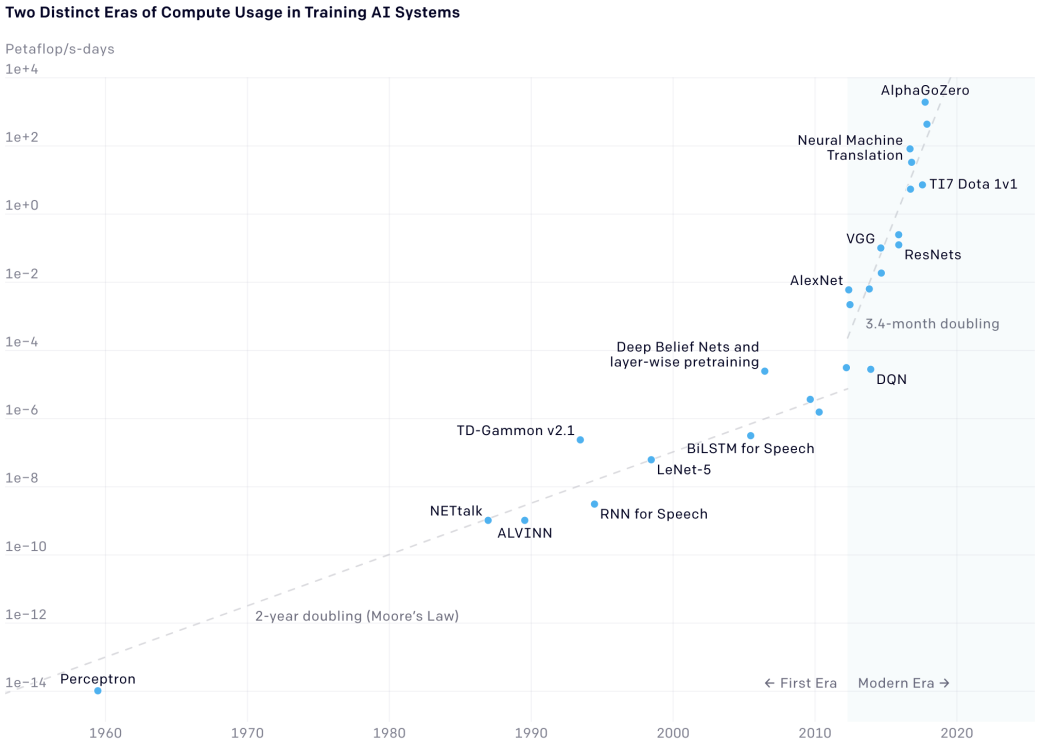


Figure 1.1: (From the OpenAI blog post [3]) Evolution of `FLOPs` per day required for training SOTA AI models over the last 60 years.

increase the number of parameters and computations [35, 64, 70] but, on the other hand, by adding more training data. The growth in computational, memory and data complexity for DL models helps to improve the SOTA for AI systems. At the same time improving the computational, parameter and sample efficiency of DNNs is an orthogonal, yet equally important, research topic.

Compressing a model reduces memory and, depending on the compression technique, also computational costs for DNNs. Popular methods to compress DL models are given by *quantization*, *low rank tensor approximation*, *tensor decomposition*, *weight sharing*, *freezing weights* and *network pruning*. These methods can be applied *after* or *before* training. The former makes *inference* of models more efficient. The latter additionally lowers costs for training the models and is called *dimensionality reduced training* (DRT) [195]. Quantization [24, 59, 75, 99, 196, 206, 209] lowers the number of bits for encoding

weights in a DNN from 32bits floating points to integers with usually 8 or less bits. Some quantization methods even use only 1bit to represent network parameters [32, 146]. By using low precision integers as network weights, the memory requirements are reduced massively while the computational time can be sped up drastically. Tensor decomposition [90, 133, 198], *e.g*. low rank tensor approximation [29, 102, 159], represent tensors as a product of smaller tensors which can also reduce the `FLOP` count of the model. Weight sharing [20, 134, 178] clusters the parameters of the model to shrink the number of different weights in a network. By choosing parameters from a small set of possible values, they can be stored more efficiently which lowers the memory footprint of the model.

Network *pruning* [4, 43, 59, 78, 82, 91, 94, 126, 212] reduces the network size by fixing parameters at the value 0. Since layers in DNNs are evaluated by dot products, pruning also reduces the number of computations required to process the corresponding layer. Pruning can be divided in unstructured [43, 59, 78, 82, 91, 94, 126] and structured [4, 18, 73, 212] pruning. Structured pruning sets whole blocks of parameters to zero at the same time which generates leaner architectures and immediately reduces the computation time for the model. Unstructured pruning sets weights individually to zero which allows more fine-grained sparsity patterns. Compared to structured pruning, unstructured pruning with the same number of non-zero parameters leads to better performing models. However, unstructured pruning requires soft- and hardware that supports sparse tensor calculations to accelerate computations as well [38, 50, 60, 136, 188].

Freezing [68, 71, 97, 157, 171, 192] parts of a DNN helps to reduce the number of trained parameters which lowers memory and computational costs during training. By using pseudo-random number generators, the frozen weights do not need to be stored but can be recovered by a single random seed number [192]. Moreover, frozen weights help to overcome catastrophic forgetting for continual- or multi task learning [81, 149]. Also, communication costs for distributed training can be reduced by freezing parts of the model [171].

In recent years, training already compressed models *from scratch* became of interest to the DL community, providing the benefits of reduced memory, transfer, communication and computation costs not only for inference but also for training [40, 43, 94, 122, 184, 192, 193, 194, 195]. This thesis concentrates on reducing the number of trained parameters by pruning or freezing parameters *before* training begins. Such trainings

are called dimensionality reduced training [195] since models are trained on a low dimensional submanifold in the parameter space. While DRT methods have the advantage of reducing the memory footprint and the potential to reduce the computational costs for training as well, they show inferior results compared to classical pruning techniques applied on densely pre-trained models until now [45, 194, 195].

1.2 Contributions of this Thesis

This thesis proposes three techniques to improve the SOTA for dimensionality reduced training methods. They are given by:

- *FreezeNet* [192] overcomes the problem of weak information flow which occurs in models where only a small subset of the weights is trained. To get rid of the weak information flow, FreezeNet freezes the untrained weights at their random initialization instead of setting them to zero. By using pseudo-random number generators, a FreezeNet has the same memory requirements as a pruned model. Moreover, results are improved greatly in situations where information flow is a bottleneck for training, *i.e.* for high compression rates. For lower compression rates, weight decay can be applied on the frozen weights to use the randomly initialized weights at the beginning of training, but smoothly shrinks them to zero during training. After training, such a model is sparse and achieves better results than pruning the model at initialization.

- *COPS* [193] combines two pruning methods which are applied at initialization. SOTA methods that prune the model at initialization either work well for lower compression rates or for higher ones. COPS performs well in both regimes by combining the best of two pruning methods, while being adaptive to the given situation. This is realized by efficiently solving a constrained, combinatorial optimization problem. The solution is a projection of a solution to a linear relaxation of the problem. Hereby, the tightness of the constraint balances between the two pruning methods. It is an additional hyperparameter for training which can be found efficiently and robustly by using the mathematical formulation of the COPS problem. COPS is *not* restricted to two specific pruning methods, but can be applied to arbitrary ones.

- *Interspace pruning* [194] is a method to better represent $K \times K$ convolutional filters of convolutional neural networks (CNNs) with sparse coefficient vectors than pruning the standard, *i.e.* spatial, coefficients. This is achieved by jointly training an adaptive basis, the so called *filter basis*, and corresponding coefficients for a set of convolutional filters that share the same filter basis. The representation is called the interspace representation of convolutional filters. Using such a representation increases the expressibility of sparse models compared to the standard representation. Consequently, dense filters can be better approximated by a sparse coefficient vector. Hereby, general DRT methods – but also pruning methods applied during training or on a densely pre-trained model – are improved, especially for high compression rates. Moreover, sharing filter bases regularizes training and therefore interspace representations also improve the generalization ability of CNNs trained with lower compression rates or no compression at all.

1.3 Outline of this Thesis

The remainder of this work is structured as follows. First, Section 1.4 summarizes the mathematical notation used in this thesis. Chapter 2 gives an introduction into deep learning and further defines dimensionality reduced training as optimizing the parameters of a DNN on a low dimensional submanifold. Moreover, current SOTA methods for DRT are presented. Their shortcomings, which will be tackled by the three proposed methods in Chapters 3 - 5, are discussed in Section 2.3.3. In Chapter 3, FreezeNet – the method for freezing weights at their random initialization – is described. Then, Chapter 4 introduces the COPS method which combines two pruning methods applied before training begins to generate a more powerful one. Chapter 5 describes pruning of interspace coefficients w.r.t. a trainable filter basis – interspace pruning. Finally, Chapter 6 summarizes the main conclusions of this thesis and collects promising research directions for future work.

The setup for the experimental evaluations is provided in Appendix, Section A. The mathematical statements proposed in Chapters 4 and 5 are proven in Section B in the Appendix.

1.4 Mathematical Notation

Let $\mathbb{R}$ denote the real numbers and $\mathbb{N} = \{0, 1, 2, \dots\}$ the natural numbers including 0. For $\alpha, \beta \in \mathbb{R}$ and a set $\mathcal{B} \subset \mathcal{A}$ and $a \in \mathcal{A}$ let

$$\delta_{\alpha,\beta} = \begin{cases} 1, & \alpha = \beta \\ 0, & \alpha \neq \beta \end{cases} \text{ and } \chi_{\mathcal{B}}(a) = \begin{cases} 1, & a \in \mathcal{B} \\ 0, & a \notin \mathcal{B} \end{cases} . \tag{1.1}$$

As common, the i-th element of a vector $\mathbf{a} \in \mathbb{R}^D$ is given by $\mathbf{a}_i$, *i.e.* $\mathbf{a} = (\mathbf{a}_i)_{i=1}^D$. The same holds for tensors, thus $\mathbf{A} = (\mathbf{A}_{i_1,\dots,i_n})_{i_1 \in \{1,\dots,D_1\},\dots,i_n \in \{1,\dots,D_n\}} \in \mathbb{R}^{D_1 \times \dots \times D_n}$. The one tensor $\mathbf{1} \in \mathbb{R}^{D_1 \times \dots \times D_n}$ of arbitrary order n is defined by $\mathbf{1}_{i_1,\dots,i_n} = 1$.

The *support* $\operatorname{supp} \mathbf{A}$ for a tensor $\mathbf{A} \in \mathbb{R}^{D_1 \times \dots \times D_n}$ is given as the set of indices where $\mathbf{A}$ is not equal to zero, *i.e.*

$$\operatorname{supp} \mathbf{A} = \{(i_1, \dots, i_n) : \mathbf{A}_{i_1,\dots,i_n} \neq 0\} . \tag{1.2}$$

The *standard basis* $\mathcal{B} \subset \mathbb{R}^D$ of the vector space $\mathbb{R}^{D_1 \times \dots \times D_n}$ is given by

$$\mathcal{B} = \{\mathbf{E}^{(1,\dots,1)}, \dots, \mathbf{E}^{(D_1,\dots,D_n)}\} \text{ with } \mathbf{E}^{(i_1,\dots,i_n)}_{j_1,\dots,j_n} = \delta_{i_1,j_1} \cdot \ldots \cdot \delta_{i_n,j_n} . \tag{1.3}$$

Furthermore, 1 dimensional *slices* of tensors in dimension j at position $(i_1, \dots, i_{j-1}, i_{j+1}, \dots, i_n)$ are defined as

$$\mathbf{A}_{i_1,\dots,i_{j-1},:,i_{j+1},\dots,i_n} = (\mathbf{A}_{i_1,\dots,i_{j-1},k,i_{j+1},\dots,i_n})_{k=1}^{D_j} \in \mathbb{R}^{D_j} . \tag{1.4}$$

Inductively, m dimensional slices for all $m \leq n$ can be defined in such a way. Therefore, $n-1$ dimensional slices in dimensions $1, \dots, j-1, j+1, \dots, n$ at position k_j are given by

$$\mathbf{A}_{:,\dots,:,k_j,:,\dots,:} = (\mathbf{A}_{i_1,\dots,i_{j-1},k_j,i_{j+1},\dots,i_n})_{i_1 \in \{1,\dots,D_1\},\dots,i_{j-1} \in \{1,\dots,D_{j-1}\},i_{j+1} \in \{1,\dots,D_{j+1}\},i_n \in \{1,\dots,D_n\}} \tag{1.5}$$

$$\in \mathbb{R}^{D_1 \times \dots \times D_{j-1} \times D_{j+1} \times \dots \times D_n} . \tag{1.6}$$

Especially, the i-th *row* and j-th *column* of a matrix $\mathbf{A} \in \mathbb{R}^{D_1 \times D_2}$ are given by $\mathbf{A}_{i,:} \in \mathbb{R}^{D_2}$

and $\mathbf{A}_{:,j} \in \mathbb{R}^{D_1}$, respectively. The *transpose* of a matrix $\mathbf{A} \in \mathbb{R}^{D_1 \times D_2}$ is given by $\mathbf{A}^T \in \mathbb{R}^{D_2 \times D_1}$ with $(\mathbf{A}^T)_{i,j} = \mathbf{A}_{j,i}$.

The *Hadamard product* $\odot$ of two tensors $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{D_1 \times \ldots \times D_n}$ of the same size is given as their componentwise multiplication, *i.e.*

$$(A \odot B)_{i_1,\ldots,i_n} = A_{i_1,\ldots,i_n} \cdot B_{i_1,\ldots,i_n} \,. \tag{1.7}$$

The *outer product* $\otimes$ of two tensors $\mathbf{A} \in \mathbb{R}^{D_1 \times \ldots \times D_n}$ and $\mathbf{B} \in \mathbb{R}^{d_1 \times \ldots \times d_l}$ of arbitrary orders is defined as $A \otimes B \in \mathbb{R}^{D_1 \times \ldots \times D_n \times d_1 \times \ldots \times d_l}$ with

$$(A \otimes B)_{i_1,\ldots,i_n,j_1,\ldots,j_l} = A_{i_1,\ldots,i_n} \cdot B_{j_1,\ldots,j_l} \,. \tag{1.8}$$

The standard L_p-norm $\|\cdot\|_p$ on $\mathbb{R}^D$ for $p \in [0,\infty)$ and an arbitrary dimension D is given by

$$\|\mathbf{a}\|_p = \begin{cases} \left(\sum_{i=1}^{D} |\mathbf{a}_i|^p\right)^{\frac{1}{p}} & \text{, if } p \in (0,\infty) \\ \#\operatorname{supp}\mathbf{a} & \text{, if } p = 0 \end{cases} \tag{1.9}$$

for $\mathbf{a} \in \mathbb{R}^D$. The *Frobenius* norm $\|\cdot\|_F$ for matrices $\mathbf{A} \in \mathbb{R}^{D_1 \times D_2}$ is given by

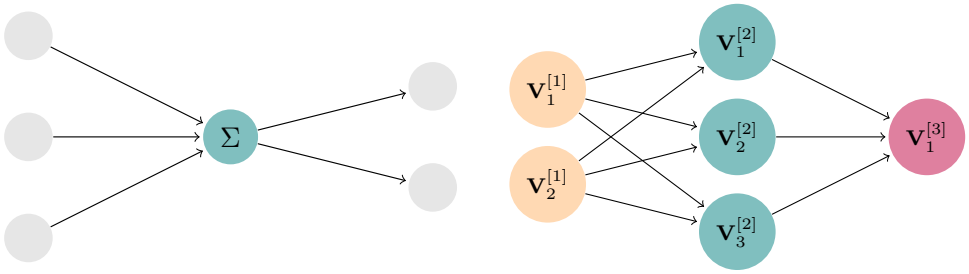
$$\|\mathbf{A}\|_F := \left(\sum_{i=1}^{D_1}\sum_{j=1}^{D_2} |\mathbf{A}_{i,j}|^2\right)^{\frac{1}{2}} \,. \tag{1.10}$$

Chapter 2
Sparse Training Methods

This Chapter is an introduction to the underlying methods used in this thesis. Section 2.1 gives an overview of deep learning in general. This is followed by an introduction into dimensionality reduced training in Section 2.2. Section 2.3 gives a detailed survey on SOTA dimensionality reduced training methods and discusses their drawbacks – which are tackled by the three proposed methods in the following Chapters 3 - 5.

2.1 Deep Learning Needs Sparse Training

First, this Section gives an overview of DNNs and DL in general. Later on, the advantages of training a model with reduced dimensionality are discussed in Section 2.1.3.



(a) (b)

Figure 2.1: (a) A single neuron with 3 input and 2 output connections. (b) A deep neural network with 1 hidden layer, 2 input nodes and 1 output node.

2.1.1 Deep Neural Networks

2.1.1.1 Neurons

Inspired by information transmission in biological neurons – the elementary building block of human brains – McCulloch and Pitts [119] introduced a first mathematical model of neurons. In the following years, this model was refined [121, 156, 190]. A mathematical neuron obtains c_{in} input signals and passes its output to c_{out} following neurons. These signals can be modeled as binary values. However, nowadays continuous signals in $\mathbb{R}$ are preferred as they enable differentiability of neurons. The neuron itself weights and sums up all incoming signals, see 2.1(a). The weighted sum is then passed through a non-linear *gate* whose output is then sent to all wired neurons which again weight, sum and gate all incoming signals. Note, the connection graph of neurons is assumed to be *directed* therefore the incoming and outgoing connections of a neuron are not equal in general, see Figure 2.1(b). Wiring neurons together is called *neural network* and is inspired by human brains which consist of a large number of connected neurons [41].

Activation Functions. The weighted sum computed in a neuron is a linear map. The non-linear gate function, also called *activation function*, applied after the weighted sum helps to increase the expressibility of neural networks. A popular activation function is the *rectified linear unit* $\text{ReLU} : \mathbb{R} \to \mathbb{R}$, $\alpha \mapsto \max\{0, \alpha\}$ [128].

2.1.1.2 Multi Layer Perceptron

A feed-forward deep neural network is a function $f_\Theta : \mathcal{A}_{\text{in}} \to \mathcal{A}_{\text{out}}$ consisting of stacked layers where each layer contains a certain amount of neurons. All neurons in one layer are connected to the same incoming neurons and the same outgoing neurons. The DNN is parameterized by $\Theta = (\Theta^{[1]}, \ldots, \Theta^{[L]}) \in \mathbb{R}^D$ where $\Theta^{[l]}$ defines the parameters in layer l used to weight the incoming signals. Since DNNs are optimized with gradient descent, the input- and output domains are assumed to be real valued, *i.e.* $\mathcal{A}_{\text{in}} \cong \mathbb{R}^{d_{\text{in}}}$ and $\mathcal{A}_{\text{out}} \cong \mathbb{R}^{d_{\text{out}}}$. Each layer $l \in \{1, \ldots, L\}$ again defines a function $f^{[l]}_{\Theta^{[l]}}$ parameterized by $\Theta^{[l]}$ where the input of the layer only depends on the output of all previous layers

$1, \ldots, l-1$. The input $\mathbf{X}$ of the network is processed recursively by

$$f_\Theta(\mathbf{X}) = \mathbf{V}^{[L]} \text{ where } \mathbf{V}^{[l+1]} = f^{[l+1]}_{\Theta^{[l+1]}}(\mathbf{V}^{[l]}, \mathbf{V}^{[l-1]}, \ldots, \mathbf{V}^{[0]}) \text{ and } \mathbf{V}^{[0]} = \mathbf{X}\,. \tag{2.1}$$

Therefore, the total number of layers L is called *depth* of the network.

Fully Connected Layer. A fully connected layer has $\mathbf{V}^{[l-1]} \in \mathbb{R}^{c_{\text{in}}}$ as input. Each entry of $\mathbf{V}^{[l-1]}$ corresponds to a neuron in layer $l-1$, see Figure 2.1(b). The neurons of layer l now combine information of *all* neurons of layer $l-1$ by first applying an affine linear map followed by a non-linear activation function. This corresponds to passing a weighted sum through a non-linear gate for each neuron. The whole layer can be computed via

$$\mathbf{V}^{[l]} = \phi^{[l]}\left(\mathbf{W}^{[l]}\right) \text{ and } \mathbf{W}^{[l]} = \Theta^{[l]} \cdot \mathbf{V}^{[l-1]} + \mathbf{b}^{[l]}\,, \tag{2.2}$$

where $\Theta^{[l]} \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}}}$ denotes the weights of layer l, the *bias* $\mathbf{b} \in \mathbb{R}^{c_{\text{out}}}$ is the affine part and $\phi^{[l]} : \mathbb{R} \to \mathbb{R}$ is the non-linear activation function which is applied to the output of each neuron. A network that only contains fully connected layers is called a *multi layer perceptron* (MLP). For DL, the weights and the biases of each layer are optimized to better represent a usually unknown target function f^* on a dataset $\mathcal{X}_{\text{train}}$ – for more details see Section 2.1.2.

It was proven that any continuous function $h : \mathcal{A}_{\text{in}} \to \mathcal{A}_{\text{out}}$ can be approximated on a compact domain $\mathcal{K} \subset \mathcal{A}_{\text{in}}$ as close as desired by a single-layer MLP that is wide enough, if and only if the activation function ϕ is non-polynomial [25, 69, 96, 142]. This is the so-called *universal approximation Theorem*. Widening MLPs is not necessary to achieve universal approximation ability if their depth is increased [5, 111]. It was shown in practice that adding more depth instead of width to neural networks is essential for good generalization ability [5, 37]. Therefore, DNNs, *i.e.* neural networks with many layers, are used nowadays to approximate desired target functions of high complexity. Such models are required to solve the high dimensional and highly complex problems in the field of artificial intelligence, such as image classification [141], speech recognition [137] or object detection [185].

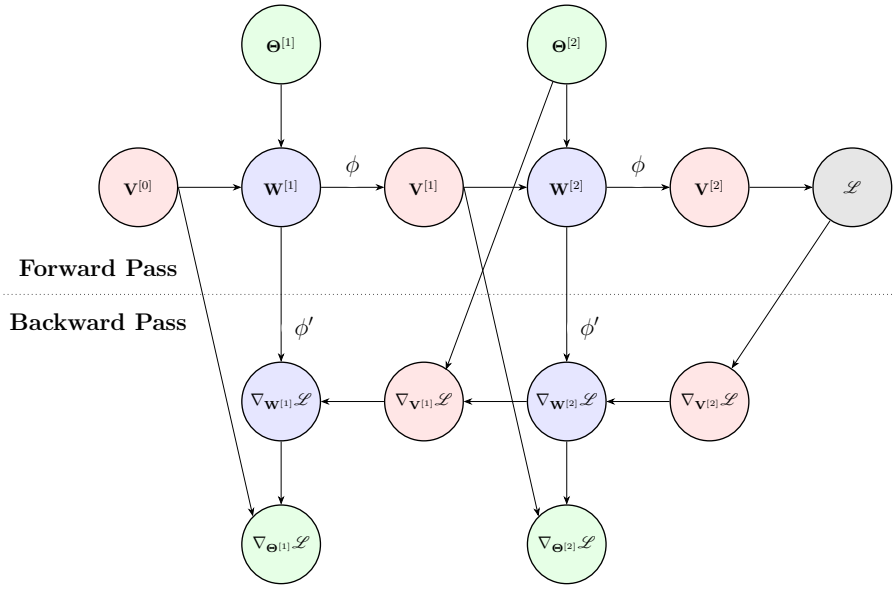


Figure 2.2: The computational graph of a DNN with depth $L = 2$ decomposed into the forward and backward pass.

2.1.1.3 Backpropagation

DNNs can be optimized on a given task by applying *gradient descent* w.r.t. a loss function $\mathscr{L}$. This requires the knowledge of the gradients for each parameter Θ_i,

$$\nabla_{\Theta_i}\mathscr{L} := \left(\nabla_{\Theta}\mathscr{L}\left(\Theta\right)\right)_i \text{ with } \nabla_{\Theta}\mathscr{L}\left(\Theta\right) = \frac{\partial \mathscr{L}(\Theta)}{\partial \Theta}\,, \tag{2.3}$$

for an almost everywhere differentiable function $\mathscr{L} : \mathbb{R}^D \to \mathbb{R}$. These gradient values can be computed efficiently layer by layer with the *backpropagation* procedure [158]. The backpropagation simply uses the chain rule. For layer l it holds

$$\mathbf{W}^{[l]} = \Theta^{[l]} \cdot \mathbf{V}^{[l-1]} \text{ and } \mathbf{V}^{[l]} = \phi^{[l]}\left(\mathbf{W}^{[l]}\right) \tag{2.4}$$

where the bias term is set to 0 w.l.o.g. For $\mathbf{b}^{[l]} \neq 0$, setting $\hat{\Theta}^{[l]} \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}}+1}$ with $\hat{\Theta}^{[l]}_{[:,j]} = \Theta^{[l]}_{[:,j]}$ for $j = 1,\ldots,c_{\text{in}}$, $\hat{\Theta}^{[l]}_{[:,c_{\text{in}}+1]} = \mathbf{b}^{[l]}$ and $\hat{\mathbf{V}}^{[l-1]} \in \mathbb{R}^{c_{\text{in}}+1}$ with $\hat{\mathbf{V}}^{[l-1]}_j = \mathbf{V}^{[l-1]}_j$, $j = 1,\ldots,c_{\text{in}}$ and $\hat{\mathbf{V}}^{[l-1]}_{c_{\text{in}}+1} = 1$, is an equivalent formulation for the fully connected layer which now can be computed by (2.4).

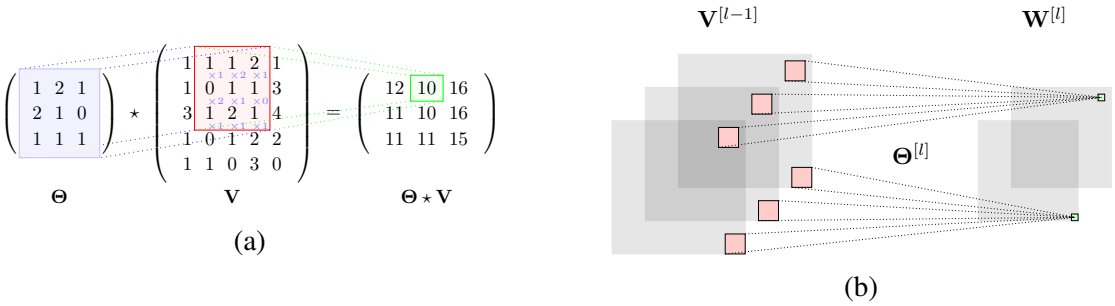


Figure 2.3: (a) A 2D convolution with one output channel and an single channel input image. (b) A convolutional layer with 3 input and 2 output channels. In both plots, the receptive field of a pixel in the output feature map (green) is shown by the connected red patch in the input feature maps.

Therefore, the gradient $\nabla_{\Theta^{[l]}}\mathscr{L}$ for the parameters $\Theta^{[l]}$ in layer l can be computed via

$$\nabla_{\Theta^{[l]}}\mathscr{L} = \nabla_{\mathbf{W}^{[l]}}\mathscr{L} \otimes \mathbf{V}^{[l-1]} \;,\; l \in \{1,\dots L\} \;, \tag{2.5}$$

$$\nabla_{\mathbf{W}^{[l]}}\mathscr{L} = \nabla_{\mathbf{V}^{[l]}}\mathscr{L} \odot \phi^{[l]\prime}\left(\mathbf{W}^{[l]}\right) \;,\; l \in \{1,\dots L\} \;, \tag{2.6}$$

$$\nabla_{\mathbf{V}^{[l]}}\mathscr{L} = (\nabla_{\mathbf{W}^{[l+1]}}\mathscr{L} \cdot \Theta^{[l+1]})^T \;,\; l \in \{1,\dots L-1\} \;. \tag{2.7}$$

In (2.6), $\phi^{[l]\prime}$ corresponds to the derivative of $\phi^{[l]}$. The computational graph describing the forward and backward propagation of a MLP is shown in Figure 2.2. By storing activations $\mathbf{V}^{[l]}$ and pre-activations $\mathbf{W}^{[l]}$ during the forward propagation of the network, the gradients can be computed in a single *backward pass*.

2.1.1.4 Convolutional Neural Networks

For image processing applications like object detection or classification, equivariance of the model's output w.r.t. translations of the input is desired [150, 151]. For example, a car should be detected as a car, regardless of its position in an image. Moreover, neurons in the visual cortex of monkeys respond selectively to certain geometrical shapes [48]. The responses of these neurons are locally restricted and mostly unaffected by the position of the geometrical shape [48]. *Convolutions* provide both, equivariance w.r.t. translations of the input and a local receptive field by using small convolutional kernels. By stacking layers of convolutions on top of each other, local features can be combined in order to provide global context of the scene. This stack of convolutional layers is

called a *convolutional neural network*. Nowadays, CNNs show SOTA results in image recognition tasks [108].

A 2D convolution of an image $\mathbf{V} \in \mathbb{R}^{d_1 \times d_2}$ with a kernel $\Theta \in \mathbb{R}^{K \times K}$ is given by $\Theta \star \mathbf{V} \in \mathbb{R}^{d_1-K+1 \times d_2-K+1}$ with

$$(\Theta \star \mathbf{V})_{i,j} = \sum_{k,l=1}^{K} \Theta_{k,l} \cdot \mathbf{V}_{i+k,j+l} \,,\; i \in \{1,\dots,d_1-K+1\} \,,\; j \in \{1,\dots,d_2-K+1\} \,, \quad (2.8)$$

graphically shown in Figure 2.3(a). Like a MLP, a CNN again consists of multiple layers. Each layer has nodes which are connected to nodes from the previous and subsequent layer, respectively. However, the feature of a node is now a 2D tensor in $\mathbb{R}^{d_1 \times d_2}$. Stacking these features in one layer is called a *feature map* and generates a 3D tensor $\mathbf{V}^{[l]} \in \mathbb{R}^{c_{\text{out}} \times d_1 \times d_2}$. This feature map is passed as input to the subsequent layer. Note, the input feature map of the first layer is an image $\mathbf{X} = \mathbf{V}^{[0]} \in \mathbb{R}^{c \times d_1 \times d_2}$ where $c = 1$ for a grayscale image and $c = 3$ for a RGB image. Analogue to the color channels of images, the nodes in CNNs are called *channels*. The nodes of the previous layer are called input channels as they provide the input features while the nodes in this layer are called output channels as their feature map is passed to the subsequent layer. Each connection between an input and output channel consists of a trainable convolutional kernel. Stacked together, the weights in a 2D convolutional layer are tensors $\Theta^{[l]} \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}} \times K \times K}$ of order 4. The *kernel size* $K \times K$ of the convolution is for simplicity assumed to be quadratic. The feature of an output channel is then computed as the sum of each input channel convolved with a corresponding convolutional kernel, see 2.3(b). The output of the layer is given by

$$\mathbf{V}^{[l]} = \phi^{[l]}\left(\mathbf{W}^{[l]}\right) \text{ and } \mathbf{W}^{[l]} = \Theta^{[l]} \,\text{✲}\, \mathbf{V}^{[l-1]} + \mathbf{b}^{[l]} \otimes \mathbf{1} \,, \quad (2.9)$$

where $\mathbf{b} \in \mathbb{R}^{c_{\text{out}}}$ is again the bias, $\mathbf{1}$ has the same dimension as $\mathbf{W}^{[l]}_{i,:,:}$ and $\phi^{[l]} : \mathbb{R} \to \mathbb{R}$ is the non-linear activation function applied componentwise. The convolution ✲ of the whole layer is computed for each output channel i via

$$\left(\Theta^{[l]} \,\text{✲}\, \mathbf{V}^{[l-1]}\right)_{i,:,:} = \sum_{j=1}^{c_{\text{in}}} \Theta^{[l]}_{i,j,:,:} \star \mathbf{V}^{[l-1]}_{j,:,:} \,. \quad (2.10)$$

Note, most CNNs have a fully connected layer as last layer to obtain a 1D output. However, each fully connected layer can equivalently be written as a convolutional layer

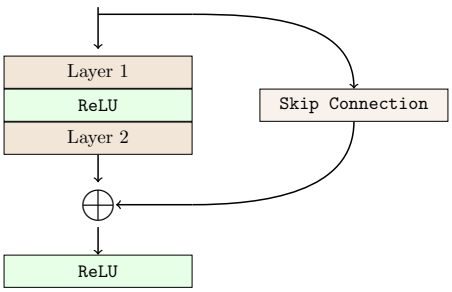


Figure 2.4: A residual block with two layers and a skip connection which are summed up.

with kernel size 1×1. Similar to the fully connected layer, all weights and biases of a convolutional layer are optimized during training.

Backpropagation for Convolutional Neural Networks. The backpropagation formulas for the pre-activations, $\nabla_{\mathbf{W}^{[l]}}\mathscr{L}$, is similar for MLPs and CNNs, *i.e.* given by (2.6). Moreover,

$$\nabla_{\Theta^{[l]}_{i,j,:,:}}\mathscr{L} = \nabla_{\mathbf{W}^{[l+1]}_i}\mathscr{L} \star \mathbf{V}^{[l]}_j \,, \tag{2.11}$$

whereas $\nabla_{\mathbf{V}^{[l]}}\mathscr{L}$ can be computed as the transpose of the convolution ✡ between $\nabla_{\mathbf{W}^{[l+1]}}\mathscr{L}$ and a rotated version of $\Theta^{[l]}$ [56].

2.1.1.5 Special Layers Used to Improve Trainability of Deep Neural Networks

Residual Connections. Residual connections are used to guarantee a high information flow in deep neural networks. This is achieved by a *skip* connection that fuses the input feature map of a *residual block* (usually two or three consecutive convolutional layers) with the output of the block by a simple summation [65], see Figure 2.4. The skip connection is either the identity if the residual block does not change spatial resolution and channel numbers or a simple convolution with kernel size one and appropriate stride to be able to adapt channel number and spatial resolution of the feature map. By using residual connections, deep convolutional networks with more than $1,000$ layers can be trained [66].

Batch Normalization Layers. For varying input data, normalization layers harmonize the inputs of a layer to stabilize the training of DNNs. The most prominent normalization layer is the so called *batch normalization* layer [74]. During training time, the output of a node (MLP) or feature map (CNN) is normalized by the mean and standard deviation of its values in the used *batch*. A batch is a set of input data which is processed in parallel by a DNN. Moreover, mean and standard deviation are computed over the training set which is used for normalization during inference. The batch normalization approximately generates outputs with zero mean and unit standard deviation which lets training converge faster [191]. However, the normalization step restricts the layer's output. This restriction is circumvented by adaptive, trainable scale parameters. Adding batch normalization layers regularizes training and thus leads to models with improved generalization ability [74].

2.1.2 Training Deep Neural Networks

Up to now, DNNs were introduced and the forward and backward-propagation of DNNs was discussed. However, an essential part of deep learning still misses: the *learning* part.

2.1.2.1 Problem Formulation for Deep Learning

A general formulation for deep learning is to optimize the following statistical *empirical risk minimization* problem [180]

$$\min_{f \in \mathcal{H}} \mathbb{E}_{\mathbf{X} \sim \mathcal{X}} A\left(f(\mathbf{X}), f^*(\mathbf{X})\right) \tag{2.12}$$

where $\mathbb{E}$ is the expected value operator, $\mathcal{H} = \{f : \mathcal{A}_{\text{in}} \to \mathcal{A}_{\text{out}}\}$ the *hypothesis class* of potential DNNs, $(\mathcal{X}, f^*)$ the dataset defining the learning task together with the corresponding quality measure $A : \mathcal{A}_{\text{out}} \times \mathcal{A}_{\text{out}} \to \mathbb{R}$. The dataset consists of two parts, the *data points* $\mathcal{X} \subset \mathcal{A}_{\text{in}}$ and the corresponding ground truth *target function* $f^* : \mathcal{X} \to \mathcal{Y}$ where $\mathcal{Y} \subset \mathcal{A}_{\text{out}}$ is the target domain.

For example if the task is to classify handwritten, digitized numbers from 0 to 9 like MNIST [92], a data point $\mathbf{X} \in \mathcal{X} \subset \mathbb{R}^{28 \times 28}$ corresponds to the digital image whereas the corresponding target value $f^*(\mathbf{X}) \in \mathcal{Y} = \{0, 1 \ldots, 9\}$ is the actual digit shown in the image.

Given a desired network configuration, *i.e.* a choice of activation functions, size c_{out} for each layer and convolutional kernel size K if a layer is convolutional, and by restricting the DNNs on this configuration, the hypothesis class equals $\mathcal{H} = \{f_\Theta : \mathcal{A}_{\text{in}} \to \mathcal{A}_{\text{out}} : \Theta \in \mathbb{R}^D\}$ with the model's weights Θ. In the following, we assume $\mathcal{A}_{\text{in}} \cong \mathbb{R}^{d_{\text{in}}}$ and $\mathcal{A}_{\text{out}} \cong \mathbb{R}^{d_{\text{out}}}$. For CNNs, the input domain is usually a subset of $\mathbb{R}^{3\times d_1 \times d_2} \cong \mathbb{R}^{3d_1 d_2}$ if the input images are RGB or a subset of $\mathbb{R}^{d_1 \times d_2} \cong \mathbb{R}^{d_1 d_2}$ if they are grayscale pictures.

2.1.2.2 Splitting the Data for Training, Validation and Testing

Real-life datasets usually consist of a subset $\mathcal{X}_l \subset \mathcal{X}$ of labeled data points as collecting and curating data is expensive. The labels are provided by a label function $f_l^* = f^*|_{\mathcal{X}_l}$.[1] Since the labeled data is usually a sparse subset of the total dataset, it might not suffice to find the $f_{\bar{\Theta}} \in \mathcal{H}$ that approximates f_l^* best. Even if $f_{\bar{\Theta}}$ approximates f_l^* perfectly on $\mathcal{X}_l$, it might show bad *generalization ability* on the unseen part of the dataset $\mathcal{X} \setminus \mathcal{X}_l$. The generalization ability of a model f_Θ is defined in this thesis as the ability to approximate f^* well on unknown data $\mathcal{X} \setminus \mathcal{X}_l$ [179].

To overcome discrepancies between performance on the known and unknown part of the dataset, the drawn samples are split in two disjoint parts, the *train-and-validation* dataset $(\mathcal{X}_l \setminus \mathcal{X}_{\text{test}}, f^*|_{\mathcal{X}_l \setminus \mathcal{X}_{\text{test}}})$ and the *test* dataset $(\mathcal{X}_{\text{test}}, f^*_{\text{test}})$ with $f^*_{\text{test}} = f^*|_{\mathcal{X}_{\text{test}}}$. Thus, all labeled datapoints are either in the train-and-validation *or* test dataset. If the set of test data points $\mathcal{X}_{\text{test}}$ is chosen well enough to cover the most important aspects of the data $\mathcal{X}$, the generalization ability of a network trained on the train-and-validation dataset can be determined close enough by its performance on the test dataset.

To be able to actually measure the generalization ability of a DNN, a training algorithm *must not* have access to the test dataset. During training, however, the generalization ability of the model should be tracked as well, otherwise the model most likely *overfits* on the training data. Overfitting means a good performance on the dataset used for training but at the same time a bad generalization ability [9, 47, 189]. To prevent overfitting, the train-and-validation data is again split in two parts, resulting in the *train* dataset $(\mathcal{X}_{\text{train}}, f^*_{\text{train}})$ and the *validation* dataset $(\mathcal{X}_{\text{val}}, f^*_{\text{val}})$ where $\mathcal{X}_{\text{train}}, \mathcal{X}_{\text{val}}$ and $\mathcal{X}_{\text{test}}$ are disjoint, $f^*_{\text{train}} = f^*|_{\mathcal{X}_{\text{train}}}$ and $f^*_{\text{val}} = f^*|_{\mathcal{X}_{\text{val}}}$. Now, the model can be optimized on the train dataset while its performance on the validation dataset can be used as a proxy measure for its

[1] The experimental evaluation in this thesis covers supervised learning only. However, unlabeled data can be used for training as well. To simplify the notation, unlabeled data is not included here.

generalization ability. Finally, the model with the best performance on the validation dataset should be chosen as the final, optimized model. This yields the deep learning problem which will be dealt with in the remaining of this thesis, given by

$$f_{\bar{\Theta}} = f_{\Theta^{(\bar{t})}} \text{ with } \bar{t} = \arg\min_{t \in \mathbb{N}} \left\{ \mathbb{E}_{\mathbf{X} \sim \mathcal{X}_{\text{val}}} A \left(\mathscr{T}_{\mathcal{Z}, \mathcal{X}_{\text{train}}, f^*_{\text{train}}} (f_{\Theta^{(0)}}, t)(\mathbf{X}), f^*_{\text{val}}(\mathbf{X}) \right) \right\} . \quad (2.13)$$

Here, $f_{\Theta^{(0)}}$ denotes the initial DNN which is then optimized by a *training algorithm* $\mathscr{T}_{\mathcal{Z}, \mathcal{X}_{\text{train}}, f^*_{\text{train}}} : \mathcal{H} \times \mathbb{N} \to \mathcal{H}$ for $t \in \mathbb{N}$ training steps. The training algorithm $\mathscr{T}_{\mathcal{Z}, \mathcal{X}_{\text{train}}, f^*_{\text{train}}}$ will be discussed later on in this Section. For now, it is important that it optimizes an initial DNN $f_{\Theta^{(0)}}$ with the help of the labeled data for a number of training steps $t \in \mathbb{N}$. The output of the training algorithm is the model at training step $\bar{t}$ which shows the best generalization ability on the validation set. This choice of the optimal model is called *early stopping* and prevents overfitting on the training data.

2.1.2.3 Quality Measures and the Loss Function

Deep learning tasks are widespread. Each task has a different set of input data, but also an individual type of decision or prediction attached to it. Therefore, there is not an all-embracing quality measure for the performance of a model, but it has to be carefully engineered for each AI task individually. In this thesis, DNNs are trained for image classification tasks.

Accuracy for Classification. Classification tasks for a C-class problem have data points $\mathcal{X}$ with corresponding labels $f^*(\mathbf{X}) \in \{0, \ldots, C-1\}$, $\mathbf{X} \in \mathcal{X}$. The task for a DNN is to predict the correct class for a given $\mathbf{X} \in \mathcal{X}$. Therefore, the quality of the DNN can be measured by its *accuracy*

$$A : \{0, \ldots, C-1\} \times \{0, \ldots, C-1\} \to \mathbb{R}, \ (\bar{y}, f^*(\mathbf{X})) \mapsto \delta_{\bar{y}, f^*(\mathbf{X})} , \quad (2.14)$$

with the actual label $f^*(\mathbf{X})$, and the class $\bar{y}$ predicted by the DNN. By using the average of the accuracy over the whole test dataset, the generalization ability of a network can be approximated via [9]

$$\mathbb{E}_{\mathbf{X} \sim \mathcal{X}_{\text{test}}} A(f(\mathbf{X}), f^*(\mathbf{X})) . \quad (2.15)$$

Other tasks of course have different quality measures, see for example [120] for an overview of different machine learning metrics.

The accuracy (2.14) has the drawback of being defined on a discrete subset only and therefore it is non-differentiable. This problem can be solved by:

- Training the network with a differentiable function, the so called *loss function* [56]. Optimizing DNNs w.r.t. a loss function should provide DNNs which also have a high performance w.r.t. the corresponding quality measure, like the accuracy.
- If the targets $\mathcal{Y}$ are discrete, the DNNs should be designed to have a continuous output since a differentiable loss function requires continuous inputs. For the classification task, these inputs can be interpreted as the estimated conditional probability vector for each output class given the network input [56] and are given by

$$f_\Theta(\mathbf{X}) = \hat{p}(\mathbf{y}|\mathbf{X}) = (\hat{p}(f^*(\mathbf{X}) = c|\mathbf{X}))_{c=0}^{C-1} \in [0,1]^C \text{ , with } \sum_{c=0}^{C-1} \hat{p}(\mathbf{y}|\mathbf{X})_c = 1 \text{ .} \tag{2.16}$$

  The actual, discrete predictions required for inference or for measuring the model's accuracy are then generated by an additional step, *e.g.* applying arg max over the entries.

Note, for some problems as for example regression tasks, the loss function can directly be chosen as the quality measure.

The Loss Function. As seen above, classification tasks have a discrete output in $\{0,\dots,C-1\}$. Optimizing a proper loss function for this task should simultaneously optimize the accuracy (2.14).

The output of a DNN used for classification tasks can be seen as the estimated conditional distribution of the class probabilities given the input data, see (2.16). Therefore, a loss function for the classification task can be designed as a measure of the discrimination between two conditional probability distributions. The first distribution is defined by the labels of the dataset and the second one is the distribution learned by the DNN, see (2.19). The difference between two discrete conditional probability distributions q_1 and

q_2 can be measured by the *Kullback-Leibler divergence* [87]

$$\sum_{\mathbf{X}\in\mathcal{X}} q_1(\mathbf{y}|\mathbf{X}) \cdot \log \frac{q_1(\mathbf{y}|\mathbf{X})}{q_2(\mathbf{y}|\mathbf{X})} = \sum_{\mathbf{X}\in\mathcal{X}} q_1(\mathbf{y}|\mathbf{X}) \log q_1(\mathbf{y}|\mathbf{X}) - \sum_{\mathbf{X}\in\mathcal{X}} q_1(\mathbf{y}|\mathbf{X}) \log q_2(\mathbf{y}|\mathbf{X}) \tag{2.17}$$

$$= -\underbrace{\mathbb{E}_{\mathcal{X}}\left(-\log q_1\right)}_{\text{entropy of } q_1} + \underbrace{\mathbb{E}_{q_1}\left(-\log q_2\right)}_{\text{cross-entropy between } q_1, q_2} . \tag{2.18}$$

As shown, the Kullback-Leibler divergence can be split into the *entropy* of q_1 and the *cross-entropy* between q_1 and q_2.

In our case, the distribution defined by the labels is fixed. Therefore, only the cross-entropy between the conditional distribution learned by the network and the ground truth labels needs to be minimized in order to align both distributions. This leads to the cross-entropy loss function

$$\mathscr{L} : \mathbb{R}^C \times \{0,\dots,C-1\} \to \mathbb{R}, (\hat{\mathbf{y}}, f^*(\mathbf{X})) \mapsto -\sum_{c=0}^{C-1} \log\left(\frac{\exp(\hat{\mathbf{y}}_c)}{\sum_{s=0}^{C-1}\exp(\hat{\mathbf{y}}_s)}\right) \cdot \delta_{c,f^*(\mathbf{X})} . \tag{2.19}$$

Here, $f^*(\mathbf{X})$ is the actual label of the data point $\mathbf{X}$ which leads to the ground truth conditional distribution $\delta_{\cdot,f^*(\mathbf{X})}$. The output of the DNN is given by $\hat{\mathbf{y}} = f_\Theta(\mathbf{X}) \in \mathbb{R}^C$. As can be seen, the output $\hat{\mathbf{y}}$ is normalized with a *softmax* function to be a probability vector $\hat{p}(\mathbf{y}|\mathbf{X})$. Therefore, the predicted class of the DNN can simply be determined by

$$\bar{y} := \arg\max_c \frac{\exp(\hat{\mathbf{y}}_c)}{\sum_{s=0}^{C-1}\exp(\hat{\mathbf{y}}_s)} = \arg\max_c \hat{p}(\mathbf{y}|\mathbf{X})_c = \arg\max_c f_\Theta(\mathbf{X})_c \in \{0,\dots,C-1\} . \tag{2.20}$$

Since $-\log$ is a strictly monotonic decreasing function, $\mathscr{L}(\hat{\mathbf{y}}, f^*(\mathbf{X}))$ reaches its minimum for $\frac{\exp(\hat{\mathbf{y}}_y)}{\sum_{s=0}^{C-1}\exp(\hat{\mathbf{y}}_s)} \to 1$, *i.e.* a perfect classification. If the prediction $\bar{y}$ is inaccurate, the cross-entropy loss is high. Therefore, there is an equivalence between a DNN minimizing the loss function $\mathscr{L}$ and a DNN that maximizes the accuracy A.

Deep Neural Networks Learn by Minimizing the Loss Function. The cross-entropy loss function (2.19) reaches its minimum for perfect classification. Therefore, the deep learning task can be formulated as *minimizing the loss function on the training set while*

maintaining a high performance on the validation set. As shown in problem (2.13), deep learning means optimizing a network on a dataset, not a single example. However, computing the loss function on the whole dataset might be infeasible due to restricted computation and/or storage capacity in real-life applications. Therefore, the loss function is minimized over a sample of the whole dataset [83, 93, 154], a so called batch of training data. The loss function for one batch of training data equals

$$\mathscr{L}(\Theta; \mathcal{X}^B_{\text{train}}, f^*_{\text{train}}) := \frac{1}{B} \sum_{i=1}^{B} \mathscr{L}\left(\Theta; \mathbf{X}^{(i)}, f^*_{\text{train}}\right) \tag{2.21}$$

$$\xrightarrow{B \to \#\mathcal{X}_{\text{train}}} \mathbb{E}_{\mathbf{X} \sim \mathcal{X}_{\text{train}}} \mathscr{L}\left(\Theta; \mathbf{X}, f^*_{\text{train}}\right) \tag{2.22}$$

where the data batch $\mathcal{X}^B_{\text{train}} = \{\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(B)}\} \sim \mathcal{X}_{\text{train}}$ is sampled from the training data without replacement and

$$\mathscr{L}\left(\Theta; \mathbf{X}, f^*_{\text{train}}\right) := \mathscr{L}\left(f_\Theta(\mathbf{X}), f^*_{\text{train}}(\mathbf{X})\right) \ . \tag{2.23}$$

2.1.2.4 Optimizing Deep Neural Networks

Curve fitting defines the task of finding a function $\bar{f} : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}$ which fits a given set of *data points* $(\mathcal{X}_{\text{train}}, f^*_{\text{train}}(\mathcal{X}_{\text{train}}))$ best, *i.e.* $\bar{f}$ should be as close as possible to f^*_{train} on the domain $\mathcal{X}_{\text{train}}$ [14]. The measure for the *closeness* of two curves depends on the application. Often, the *mean-squared* distance between $f^*_{\text{train}}(\mathcal{X}_{\text{train}})$ and $\bar{f}(\mathcal{X}_{\text{train}})$ is used since minimizing the mean-squared error yields a maximum likelihood estimator if the measurement uncertainty of f^*_{train} is Gaussian [14]. Usually, the estimated curve $\bar{f}$ follows some constraints, *e.g.* being parameterized as a polynomial or a DNN. In some cases, $\bar{f}$ can be directly computed as a closed form solution. If this is not possible, the distance can also be optimized step-by-step through minimization of $\bar{f}$ in the direction of the gradient [14].

For classification tasks, a DNN f_Θ defines an almost everywhere differentiable function $f_\Theta : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}$ parameterized by Θ. As discussed in Section 2.1.2.3, this function should best fit a given set of data points provided by $(\mathcal{X}_{\text{train}}, f^*_{\text{train}}(\mathcal{X}_{\text{train}}))$. The distance to the ground truth function is measured by the cross-entropy loss (2.19). Therefore, training a DNN can be seen as a curve fitting method. Since DNNs are deep and contain non-linear activation functions, generally there is no closed form solution to this curve fitting

problem. To actually minimize the loss function over the whole training data, deep learning therefore optimizes the weights Θ of a network f_Θ via minimization in direction of the gradient on the loss surface. Since datasets are usually quite large, the gradient is not computed over the whole dataset but a subset of the dataset – a data batch – see (2.21). The gradient can be computed efficiently via backpropagation, see Section 2.1.1.3. The weights are then updated with this gradient. Successively, the loss function is minimized in such a way over the whole dataset. Due to the stochasticity induced by optimizing over sampled batches of data instead of the whole dataset, this procedure is called *stochastic gradient descent* (SGD) [93].

The plain SGD update at training iteration $t \in \mathbb{N}_{\geq 1}$ applied to the weights $\Theta^{(t-1)}$ from iteration $t-1$ equals

$$\Theta^{(t)} = \Theta^{(t-1)} - \eta^{(t)} \underbrace{\nabla_{\Theta^{(t-1)}} \mathscr{L}(\Theta^{(t-1)}; \mathcal{X}^B_{\text{train}}, f^*_{\text{train}})}_{=:\mathbf{g}^{(t-1)}}, \tag{2.24}$$

where B is the batch size, $\mathcal{X}^B_{\text{train}} = \{\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(B)}\} \sim \mathcal{X}_{\text{train}}$ the randomly drawn batch of training data for iteration t and $\eta^{(t)} > 0$ the learning rate used in training step t. For $t = 0$, a so called *initialization* $\Theta^{(0)}$ of the network is required. Initializing DNNs can be done randomly [55, 64] but also by using *pre-trained* weights obtained by a DNN trained on a related task [31, 33, 115, 170].

A single SGD update should reduce the loss function $\mathscr{L}$ in order to find its global minimum. By using the first order Taylor expansion, it holds

$$\mathscr{L}(\Theta^{(t)}; \mathcal{X}^B_{\text{train}}, f^*_{\text{train}}) - \mathscr{L}(\Theta^{(t-1)}; \mathcal{X}^B_{\text{train}}, f^*_{\text{train}}) = -\eta^{(t)} \left\| \mathbf{g}^{(t-1)} \right\|_2^2 + O\left(\left(\eta^{(t)} \right)^2 \right). \tag{2.25}$$

If $\Theta^{(t-1)}$ is not a local minimum, SGD updates therefore minimize the loss over the *whole* training data if two conditions are fulfilled:

- The usage of a sufficiently small learning rate.
- Choice of a big enough batch size such that minimizing $\mathscr{L}$ over a batch also minimizes it over the whole dataset.

However, using a *constant* learning rate η will either (i) converge extremely slow if η is small or (ii) force $\Theta^{(t)}$ to oscillate around a local minimum if it is chosen bigger [127].

Thus, learning with SGD requires a careful choice of learning rates $\eta^{(t)}$ [11, 12], as for example using a step-wise decaying learning rate [51].

Training Algorithm. The *training algorithm* $\mathscr{T}_{\mathcal{Z},\mathcal{X}_{\text{train}},f^*_{\text{train}}} : \mathcal{H} \times \mathbb{N} \to \mathcal{H}$ with the training dataset $(\mathcal{X}_{\text{train}}, f^*_{\text{train}})$ contains the information of the used stochastic gradients based optimization method, *e.g.* SGD or Adam [84]. Moreover, important *hyperparameters* like the learning rate schedule, batch size, maximal number of training steps T and additional required hyperparameters for SGD with momentum [143, 145, 158] or Adam are combined in the set of hyperparameters $\mathcal{Z}$ and encoded in $\mathscr{T}_{\mathcal{Z},\mathcal{X}_{\text{train}},f^*_{\text{train}}}$. On top of that, the sampling procedure for training batches $\mathcal{X}^B_{\text{train}} \sim \mathcal{X}_{\text{train}}$ is also contained in $\mathscr{T}_{\mathcal{Z},\mathcal{X}_{\text{train}},f^*_{\text{train}}}$. Often ${}^{\#\mathcal{X}_{\text{train}}}/_{B} \ll T$, in this case the training data is simply revisited more than once – like learning with flashcards. Going once through the whole training dataset is called an *epoch* while going once through a batch of training data is called a *training step*.

As discussed, the number of training steps, the learning rate and batch size are so called hyperparameters. In contrast to the network parameters Θ, the hyperparameters $\mathcal{Z}$ are *not* optimized by stochastic gradient descent methods, but set before training starts. As a consequence, they have to be tuned manually, the so called hyperparameter *tuning* or *search*. Adding hyperparameter tuning to the deep learning task finally leads to

$$\min_{\mathcal{Z}} \left\{ \min_{t_{\mathcal{Z}} \in \{0,\dots,T\}} \left\{ \mathbb{E}_{\mathbf{X} \sim \mathcal{X}_{\text{val}}} A\left(\mathscr{T}_{\mathcal{Z},\mathcal{X}_{\text{train}},f^*_{\text{train}}}(f_{\Theta^{(0)}}, t_{\mathcal{Z}})(\mathbf{X}), f^*_{\text{val}}(\mathbf{X}) \right) \right\} \right\} . \tag{2.26}$$

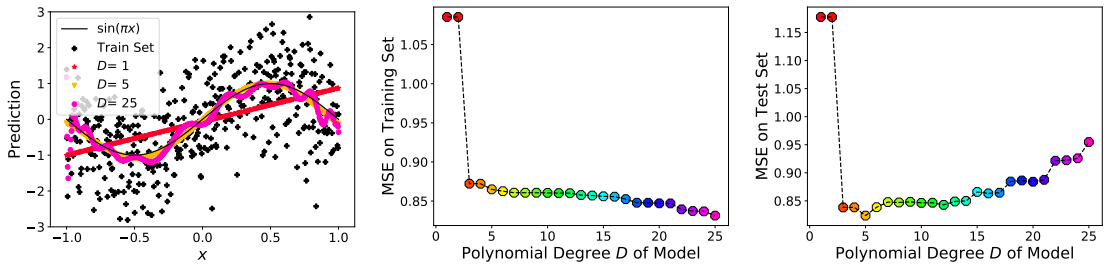
Searching optimal hyperparameters might be expensive as a model has to be trained with $\mathscr{T}_{\mathcal{Z},\mathcal{X}_{\text{train}},f^*_{\text{train}}}$ and be evaluated on the validation set for each tested $\mathcal{Z}$.

2.1.2.5 Regularization Methods

Using stochastic gradient descent, or an adapted version of it, has the advantage that the corresponding training algorithm is defined recursively, *i.e.*

$$\mathscr{T}_{\mathcal{Z},\mathcal{X}_{\text{train}},f^*_{\text{train}}}(f_{\Theta^{(0)}}, t) = \mathscr{T}_{\mathcal{Z}^{(t)},\mathcal{X}_{\text{train}},f^*_{\text{train}}}\left(\mathscr{T}_{\mathcal{Z},\mathcal{X}_{\text{train}},f^*_{\text{train}}}(f_{\Theta^{(0)}}, t-1), 1 \right) \tag{2.27}$$

where $\mathcal{Z}^{(t)}$ is the appropriate set of hyperparameters shifted to the training step t. Therefore, $\mathscr{T}_{\mathcal{Z},\mathcal{X}_{\text{train}},f^*_{\text{train}}}(f_{\Theta^{(0)}}, t)$ does not have to be computed for each t from scratch, but can



(a) Predictions on train set. (b) MSE on train set. (c) MSE on test set.

Figure 2.5: Fitting noisy labels of $f^*(x) = \sin(\pi \cdot x)$ with white noise $\nu \sim \mathcal{N}(0,1)$ using polynomial models $\hat{f}_\Theta$ with degree D and $\hat{f}_\Theta(x) = \sum_{j=1}^{D} \Theta_j \cdot x^j$ for $\Theta \in \mathbb{R}^D$ trained via least square regression. (a): Shows the ground truth f^* together with predictions of polynomials with different degrees on the training set. (b) and (c): Show the MSE on the train and test set, respectively.

be updated from the model trained for $t-1$ steps. The generalization ability measured on the validation set, *i.e.* the inner part of (2.26), can also be evaluated step by step. For reasons of efficiency it is usually evaluated after each epoch. By tracking the validation performance of the model and storing a copy of the best performing model, the best generalizing model can easily be chosen after training for T steps. Often, models tend to overfit for t close to T due to *overtraining*. Therefore, choosing the best generalizing model with this technique is called early stopping [201].

In general, methods to improve the generalization ability of DNNs are called *regularization methods* [205]. Despite early stopping, which is an implicit ingredient in the formulation of the deep learning task (2.26), there exist other generalization methods.

Bias-Variance Trade Off. A performance measure A for a regression task is given by the *squared error*

$$\mathrm{SE}\left(\hat{f}_\Theta(\mathbf{x}), f^*(\mathbf{x})\right) = \left(\hat{f}_\Theta(\mathbf{x}) - (f^*(\mathbf{x}) + \nu)\right)^2 \tag{2.28}$$

where $f^*(\mathbf{x}) + \nu$ is the noisy target of the input $\mathbf{x} \sim \mathcal{X}$, $f^* : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}$ the target function, ν a zero mean, *i.e.* *white*, noise with variance σ_ν^2 and $\hat{f}_\Theta : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}$ an estimation of f^* parameterized by $\Theta \in \mathbb{R}^D$. Optimizing the *mean-squared error*

$$\mathrm{MSE}_{\mathcal{X}}(\hat{f}_\Theta) := \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left(\mathrm{SE}\left(\hat{f}_\Theta(\mathbf{x}), f^*(\mathbf{x})\right)\right) \tag{2.29}$$

over the training data $\mathcal{X}_{\text{train}}$ therefore defines a deep learning task. The quality of the used training algorithm $\mathcal{T}_{\mathcal{Z}}$ independent of the choice of training data $\mathcal{X}_{\text{train}} \sim \mathcal{X}$ can be determined by its corresponding *learning error*, given by

$$\text{LE}(\mathcal{T}_{\mathcal{Z}}) := \mathbb{E}_{\mathcal{X}_{\text{train}}} \left(\text{MSE}_{\mathcal{X}_{\text{test}}}(\hat{f}_{\Theta\,\mathcal{T}_{\mathcal{Z}},\mathcal{X}_{\text{train}},f^*_{\text{train}}}) \right) \tag{2.30}$$

with $\hat{f}_{\Theta\,\mathcal{T}_{\mathcal{Z}},\mathcal{X}_{\text{train}},f^*_{\text{train}}}$ defined as the learned solution of (2.26) with training algorithm $\mathcal{T}_{\mathcal{Z},\mathcal{X}_{\text{train}},f^*_{\text{train}}}$ and A set as (2.28). By taking the expectation over all valid choices for the training set $\mathcal{X}_{\text{train}}$, the learning error is independent of a particular training set. Using a single training set for comparison might favor an algorithm that performs bad on all other possible sets of training data. The learning error of $\mathcal{T}_{\mathcal{Z}}$ can be decomposed as [77]

$$\text{LE}(\mathcal{T}_{\mathcal{Z}}) = \sigma_v^2 + \mathbb{E}_{\mathbf{x}\sim\mathcal{X}_{\text{test}}} \left(\text{Bias}(\mathcal{T}_{\mathcal{Z}},\mathbf{x}) + \text{Var}(\mathcal{T}_{\mathcal{Z}},\mathbf{x}) \right) . \tag{2.31}$$

The components in (2.31) are the variance of the inevitable label noise σ_v^2, the bias term

$$\text{Bias}(\mathcal{T}_{\mathcal{Z}},\mathbf{x}) = \left(\mathbb{E}_{\mathcal{X}_{\text{train}}} \left(\hat{f}_{\Theta}(\mathbf{x}) \right) - f^*(\mathbf{x}) \right)^2 \tag{2.32}$$

and the variance part

$$\text{Var}(\mathcal{T}_{\mathcal{Z}},\mathbf{x}) = \mathbb{E}_{\mathcal{X}_{\text{train}}} \left(\hat{f}_{\Theta}(\mathbf{x}) - \mathbb{E}_{\mathcal{X}_{\text{train}}} \left(\hat{f}_{\Theta}(\mathbf{x}) \right) \right)^2 . \tag{2.33}$$

The bias term can be seen as the error introduced by estimating the complex target function f^* by a simplified model $\hat{f}_{\Theta}$ [77]. An example for this would be the estimation of a non-linear model by a linear model. Moreover, the variance measures how a learned estimator $\hat{f}_{\Theta}$ would change if the training set changes [77]. Therefore, a high variance corresponds to models that adapt well to new training data but therefore might only memorize it and consequently not generalize well.

The data noise σ_v^2 should not be learned. Therefore, the optimal learning algorithm would minimize the sum of the bias and variance term. By increasing the model capacity, the bias term can be reduced since the DNN gains expressibility. However, decreasing the bias usually increases the variance due to overfitting on the training data [77]. Therefore, the best DNN topology $f_{\Theta^{(0)}}$ and hyperparameter setup $\mathcal{Z}$ finds the sweet-spot between model expressibility and overfitting which is called *bias-variance* trade off.

In real life applications, sampling all possible $\mathcal{X}_{\text{train}}$ is infeasible. The learning error $\text{LE}(\mathcal{T}_{\mathcal{Z}})$ can be estimated by using a single sample of the training set, *i.e.* *the* training set, or different splittings of the train-and-validation set. Also the more expensive *cross-validation* methods can be used [53].

Dimensionality Reduction Improves Generalization. Figure 2.5 shows the bias-variance trade off for a regression task with noisy labels. The polynomials $\hat{f}_\Theta$ vary in their degree D. A model with small degree yields a large bias term since it can not express the target function $f^*(x) = \sin(\pi \cdot x)$ well enough, see $D = 1$ in 2.5(a). The larger the degree, the more variance of the training set can be fitted by the model. However, most variance is induced by the white noise ν and should *not* be learned by the model. If D is chosen too big, the model does not generalize well anymore. This can be seen in 2.5(b) and (c) where the MSE on the train set is a monotonic decreasing function of D whereas the MSE on the test set increases for $D > 5$. For this model, the best bias-variance trade off is given by $D = 5$.

This experimentally shows one aspect of the bias-variance trade off, the model size. Using too small models yields a high bias term whereas too big models have a big variance term. For DNNs, the model size is usually chosen big enough such that a model can be trained to have 0 training loss. Therefore, the generalization ability of the learned model can often be improved by reducing the expressibility of the hypothesis class of DNNs. In practice, this is done by *network pruning* or additional regularization terms in the loss function which restrict the parameter's norm.

The variance term can also be reduced by increasing the size of the train set [53], *e.g.* by using data augmentation, or by averaging results of many networks [77]. The bias-variance trade off can also be formulated for other tasks than regression, *e.g.* for classification tasks [118] or reinforcement learning [42].

Regularization terms in the loss function. Regularizing DNNs directly via the loss function is done by adding a regularization term

$$\mathscr{R} : \mathbb{R}^D \to \mathbb{R} \, , \, \Theta \mapsto \mathscr{R}(\Theta) \tag{2.34}$$

to the loss function resulting in the total loss

$$\mathscr{L}(\Theta; \mathcal{X}^B_{\text{train}}, f^*_{\text{train}}) + \lambda_{\mathscr{R}} \mathscr{R}(\Theta) \tag{2.35}$$

which is optimized during training [109]. Here, $\lambda_{\mathscr{R}} \geq 0$ specifies the strength of the penalty. A regularization term forces the model to minimize the loss function $\mathscr{L}$ and the regularization term $\mathscr{R}$ at the same time via stochastic gradient methods. Note, $\lambda_{\mathscr{R}}$ is an additional hyperparameter that needs to be optimized.

Prominent regularizations which penalize a high parameter norm are *weight decay*, which is L_2 regularization [86], L_1 regularization [19, 176] or smooth approximations of the L_0 regularization [110, 123, 207]. Both, L_1 and L_0 regularizations force the parameters Θ to become sparse, whereas weight decay smoothly shrinks all elements in Θ uniformly. Beyond L_p regularizations, there exist other penalties that induce sparsity in a DNN [114, 199, 203].

Other regularization methods. As mentioned in Section 2.1.1.5 batch normalization layers regularize training. Randomly setting values of nodes, feature maps or single weights to zero during training also regularizes training and is called *Dropout* [67, 183]. Dropout prevents units from co-adapting too much. Moreover, the generalization ability of a DNN can be improved by enabling the model to use geometrical prior knowledge about the scene, like invariances in the data [22, 23, 76, 150, 151], or by shifting the model back to an area where it generalizes well [112, 113, 152, 165]. Also, data augmentation during training can be used to improve the model's generalization ability [166].

2.1.3 Training Sparse Models

As discussed in Section 2.1.2.5, the generalization ability of DNNs can be increased by limiting their capacity. Due to the bias-variance trade off, however, reducing the capacity of a model only improves the generalization ability of the DNN up to a certain extent. If the capacity is shrunken lower than this sweat spot, the expressibility of the model is too low which again decreases the generalization ability of the model.

2.1.3.1 Lowering Training Costs by Dimensionality Reduction

While restricting the L_2 norm of weights is a valid and popular choice to limit the capacity of models during training, *reducing* the number of trained parameters has more advantages than the possibility to improve generalization.

Freeze Weights. Generally, all D parameters $\Theta \in \mathbb{R}^D$ are updated in order to optimize the model f_Θ during training. *If* some of the parameters in Θ are not trained, less gradient values need to be updated. This reduces the computational cost during the backward pass, see (2.5).

A special case of this is to drop gradients of some parameters for *all* training steps $t \in \{1, \dots, T\}$. Therefore, the parameters are frozen at their value before the training begins. Now, if the frozen parameters have an easy to recover value, like 0 or a number generated by a pseudo random number generator, the frozen values are *free* to store. Thus, freezing weights does not only reduce computations during backpropagation, but it also reduces the memory requirements for the network during *and* after training.

Prune Weights. Moreover, assume all untrained weights to be frozen at zero. This is called pruning. By looking at the forward- and backward passes of a MLP, (2.2) and (2.7), respectively, it can be seen that all multiplications by zero can be skipped. For both forward- and backward propagation, the theoretical number of computations is d/D times the one required for the dense layer, where d is the number of non-zero parameters. Similar results hold for CNNs. Note, the theoretical number of computations is only achieved if the zeroed parameters follow some structure, like pruning whole channels at the same time. For unstructured sparsity patterns, additional work has to be done to match the indices of the non-zero parameters with the layer's input [60, 136, 177].

Summary. In summary, training only a sparse part of a model has at least three advantages: the possibility to improve a model's generalization ability, a decreased number of gradient computations during training and reduced memory requirements if the initial frozen parameters are chosen as 0 or by a pseudo random number generator. Moreover, if the parameters are pruned, the forward- and backward passes of the model can be accelerated during training. Reduced memory and computational costs speed

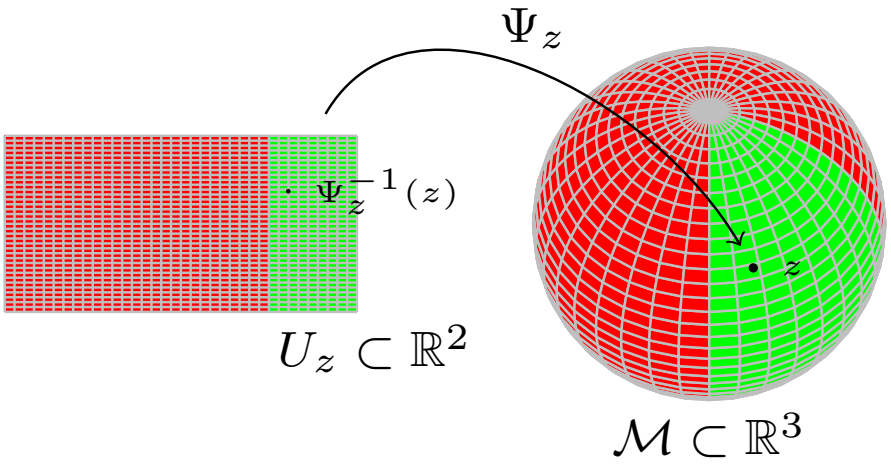


Figure 2.6: Local representation of a 2 dimensional submanifold $\mathcal{M} \subset \mathbb{R}^3$ around $z \in \mathcal{M}$ by an embedding $\Psi_z : U_z \to \mathcal{M}$, where $U_z \subset \mathbb{R}^2$ and $z \in \Psi_z(U_z)$.

up training *and* lower the energy and hardware costs. This thesis covers three methods that improve existing approaches to train only a sparse part of a model. A mathematical model which describes the sparse training of a network is introduced in the following Section 2.2.

2.2 Train Efficiently with Reduced Dimensionality

The following introduction to *dimensionality reduced training methods* is adapted from Wimmer et al. [195] and provides a mathematical foundation for DRT.

2.2.1 Formulating Dimensionality Reduced Training

A straight forward way to reduce the dimensionality of a DNN is to restrict the parameter space on a submanifold $\mathcal{M} \subset \mathbb{R}^D$. A submanifold $\mathcal{M}$ can be described locally around each point $z \in \mathcal{M}$ by an *embedding* $\Psi_z : U_z \to \mathcal{M}$, where $d \leq D$ is the *dimension* of the submanifold, $z \in \Psi_z(U_z)$ and $U_z \subset \mathbb{R}^d$ an open set [88]. Thus, $\mathcal{M}$ locally behaves like the lower dimensional space $\mathbb{R}^d$ but embedded in $\mathbb{R}^D$, *i.e.* $\mathcal{M}$ has reduced dimensionality and $q := 1 - d/D$ defines the model's *compression rate*. A graphical example of a 2D submanifold in $\mathbb{R}^3$ is given in Figure 2.6.

Therefore, submanifolds provide a powerful mathematical tool to describe dimensionality reduction for parameter spaces. Note, the embedding Ψ_z usually varies for different points in the submanifold. However, this work only deals with *global* embeddings, *i.e.* a single embedding Ψ describing the whole submanifold $\mathcal{M} = \Psi(\mathbb{R}^d)$. To be precise, global, affine linear embeddings, *i.e.* d-dimensional affine subspaces of $\mathbb{R}^D$, are investigated. In this case, the submanifold is *smooth* and stochastic gradient based optimization can be applied as usual.

Dimensionality reduced training means that for each training step t, the parameters are given by $\Theta^{(t)} = \Psi^{(t)}\left(\vartheta^{(t)}\right)$ with the embedding $\Psi^{(t)} : \mathbb{R}^d \to \mathbb{R}^D$, and $d \ll D$. Here, the same transformation as introduced in Mostafa and Wang [125] to embed $\mathbb{R}^d$ into $\mathbb{R}^D$ is used. Note, this formulation would in theory allow for arbitrary smooth transformations $\Psi^{(t)}$, not only affine linear transformations.

In contrast to training on the whole parameter space $\mathbb{R}^D$, DRT only trains on a d-dimensional smooth submanifold $\mathcal{M}^{(t)}$ by optimizing the *trainable parameters* $\vartheta^{(t)}$ and, if desired, the embedding $\Psi^{(t)}$. All methods discussed in this work are restricted to fulfill $\vartheta^{(t)} \in \mathbb{R}^d$ for all training iterations $t \in \{0, \dots, T\}$, *i.e.* reduce dimensionality throughout training. The corresponding model with reduced dimensionality is $f_{\Psi^{(t)}\left(\vartheta^{(t)}\right)}$.

Restricting the training on an adaptive submanifold reduces the parameter count $d \ll D$ significantly. However, the affine linear mapping $\Psi^{(t)}$ has parameters which need to be stored as well. In the worst case, storing the affine linear mapping $\Psi^{(t)} : \mathbb{R}^d \to \mathbb{R}^D$ would require $D \cdot (d+1)$ parameters which are, of course, much more parameters than the original DNN has. Therefore, not only $d \ll D$ but also $\text{size}(\Psi^{(t)}) + d \ll D$ is assumed, where $\text{size}(\cdot)$ computes the minimal number of parameters needed to express the affine linear transformation $\Psi^{(t)}$.

2.2.2 Pruning and Freezing Formulated as Dimensionality Reduced Training Methods

This work covers and compares three different DRT approaches. The methods use different embeddings for the small parameter vector $\vartheta^{(t)}$ into the larger DNN. A graphical overview of the three methods is given in Figure 2.7.

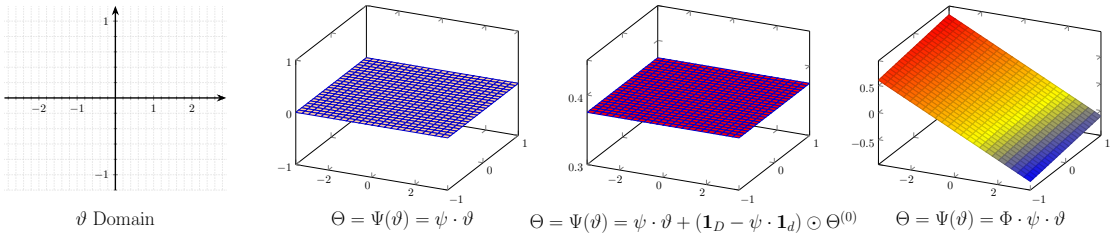


Figure 2.7: Different DRT methods with $d = 2$, $D = 3$ visualized by the corresponding geometry of the 2 dimensional submanifolds in $\mathbb{R}^3$. The left plot shows the low dimensional domain where the dimensionality reduced $\vartheta^{(t)}$ is trained. The second left plot shows *pruning* where the submanifold is spanned by the x_1, x_2 axes. Freezing is shown in the second right, a 2 dimensional affine plane in $\mathbb{R}^3$. The right plot shows pruning with an additional basis transformation $\Phi^{(t)} \in \mathbb{R}^{D \times D}$ which results in a rotated plane.

2.2.2.1 Pruning

Pruning corresponds to setting parameters in a DNN to zero. The d un-pruned elements in a DNN at training step t are determined by the *pruning mask* $\mathbf{m}^{(t)} \in \{0,1\}^D$ with $\left\|\mathbf{m}^{(t)}\right\|_0 = d$, where $\Theta_i^{(t)}$ is un-pruned if and only if $\mathbf{m}_i^{(t)} = 1$. Indices with mask values equal to zero are pruned.

In the set-up of submanifolds, $\Psi^{(t)}$ encodes the positions of the non-zero entries whereas $\vartheta^{(t)}$ stores the values of those parameters. A corresponding embedding can be easily constructed by setting

$$\Psi^{(t)}\left(\vartheta^{(t)}\right) = \psi^{(t)} \cdot \vartheta^{(t)}\,, \tag{2.36}$$

with the *pruning transformation* $\psi^{(t)} \in \mathbb{R}^{D \times d}$ defined via

$$\psi_{i,j}^{(t)} = \begin{cases} 1, & \text{if } \mathbf{m}_i^{(t)} = 1 \text{ and } \sum_{k \le i} \mathbf{m}_k^{(t)} = j \\ 0, & \text{else} \end{cases}\,. \tag{2.37}$$

In this case, the submanifold $\Psi^{(t)}\left(\mathbb{R}^d\right)$ corresponds to a d-dimensional subspace of $\mathbb{R}^D$ spanned by the $x_{i_1^{(t)}}, \ldots, x_{i_d^{(t)}}$ axes where $\operatorname{supp}\mathbf{m} = \{i_1^{(t)}, \ldots, i_d^{(t)}\}$.

Pruning leads to sparse embeddings of $\mathbb{R}^d$ into $\mathbb{R}^D$ since it corresponds to projecting the parameters on a d dimensional subspace. Having so many fixed zeros in the spatial representation can be sub-optimal. To overcome this, the pruning transformation $\psi^{(t)} \in$

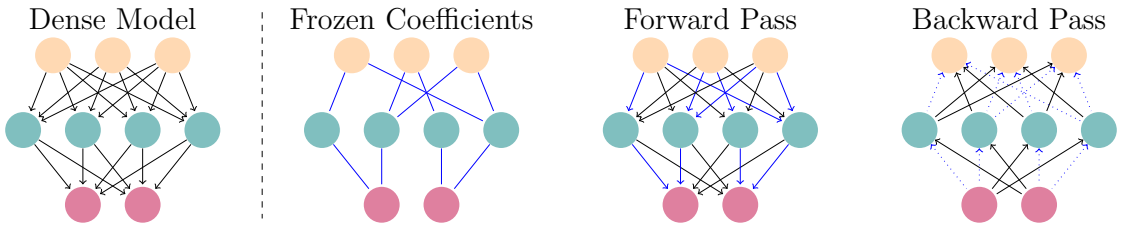


Figure 2.8: (Figure 4 in Wimmer et al. [195]) Left: Standard dense model. Middle left: Topology of the frozen weights. Middle right: The forward pass is exactly the same for the dense and the frozen model. Right: The frozen weights are used for the backward pass. However, they are not updated and therefore drawn with dotted arrows.

$\mathbb{R}^{D\times d}$ (2.37) can be be combined with a basis transformation, *i.e.* $\Phi^{(t)} \cdot \psi^{(t)} \in \mathbb{R}^{D\times d}$ with $\Phi^{(t)} \in \mathbb{R}^{D\times D}$. For arbitrary $\Phi^{(t)} \cdot \psi^{(t)}$, the model is still trained on a d dimensional subspace but without the restriction of the embedded parameter to be filled with zeros. However, the transformation matrix $\Phi^{(t)} \cdot \psi^{(t)}$ is not sparse anymore and therefore has higher memory requirements than (2.37). A possibility to overcome this is to share parameters in $\Phi^{(t)}$ [195].

2.2.2.2 Freezing

The pruning transformation can be generalized to also allow for untrained parameters to be frozen at a pre-defined value [157, 171, 192]. In this case, the parameters are located in a d dimensional affine subspace. Although the embedding $\Psi^{(t)}$ can be chosen adaptive here as well, the approach presented in Chapter 3 is restricted to a fixed affine subspace for all training steps t. The resulting transformation is given by

$$\Psi^{(0)}\left(\vartheta^{(t)}\right) = \psi^{(0)} \cdot \vartheta^{(t)} + (\mathbf{1}_D - \psi^{(0)} \cdot \mathbf{1}_d) \odot \Theta^{(0)} \tag{2.38}$$

with the pruning transformation $\psi^{(0)} \in \mathbb{R}^{D\times d}$ from (2.37), $\mathbf{1}_D$ and $\mathbf{1}_d$ the ones in $\mathbb{R}^D$ and $\mathbb{R}^d$, respectively, and the random initialization $\Theta^{(0)} \in \mathbb{R}^D$. It holds

$$(\mathbf{1}_D - \psi^{(0)} \cdot \mathbf{1}_d)_i = \begin{cases} 0, & \text{if } i \in \operatorname{supp} \mathbf{m}^{(0)} \\ 1, & \text{else} \end{cases}, \tag{2.39}$$

which means that all untrained weights $\Theta_i^{(t)}$ with $i \notin \operatorname{supp} \mathbf{m}^{(0)}$ are frozen at their random initial value $\Theta_i^{(0)}$. Here, the *freezing mask* $\mathbf{m}^{(0)} \in \{0,1\}^D$ with $\left\|\mathbf{m}^{(0)}\right\|_0$ determines the trained and frozen parameters. A graphical overview of freezing parts of a network is given in Figure 2.8.

2.2.2.3 Storage Costs for Dimensionality Reduced Training

As already mentioned, DRT yields models trained on a d dimensional submanifold. To be able to recover the model, the trained parameters $\vartheta^{(t)}$ and the transformation $\Psi^{(t)}$ have to be stored. The trained weights and the pruning transformation $\psi^{(t)}$ can be stored together efficiently by the *compressed sparse row* (CSR) format [177]. As discussed, using an additional basis transformation $\Phi^{(t)}$ on top of the pruning transformation only makes sense if its size is small. This can be guaranteed by parameter sharing [195].

If the pruning transformation $\psi^{(t)}$ is known, the indices of the frozen weights, *i.e.* $(\mathbf{1}_D - \psi^{(t)} \cdot \mathbf{1}_d)$, can be determined immediately. Moreover, if pseudo-random number generators are used to initialize the network, the frozen weights can be recovered with a single 32bit random seed [192].

2.2.2.4 Computational Costs for Dimensionality Reduced Training

This work covers *unstructured* pruning and freezing methods only. Unstructured pruning/freezing means that the pruning transformation embeds the trainable parameters $\vartheta^{(t)}$ in an unstructured manner into $\mathbb{R}^D$, *i.e.* $\operatorname{supp} \mathbf{m}^{(t)} = \{i_1^{(t)}, \ldots, i_d^{(t)}\}$ can have an arbitrary structure. In contrast to that, structured pruning would set whole structures like neurons or channels to zero at the same time [4, 18, 73, 80, 186, 211, 212].

Unstructured pruning only reduces the computational costs of a network if the used soft- and hardware supports sparse tensor computations [60, 136]. In this case, training a pruned network reduces computations in both, forward and backward propagation during training. However, if the soft or hardware does not support sparse computations, an unstructuredly pruned network has the same cost for evaluation as a dense network.

For a frozen network, the forward pass is exactly the same as for a dense network, see Figure 2.8 middle right. During training, however, the frozen weights do not need to be updated which reduces the number of required gradient values. Consequently, the number of gradient computations is reduced if parameters of a model are frozen.

2.2.3 Determining Embeddings of a Sparse Model

In the following, the important ingredients to determine the embedding $\Psi^{(t)}$ are discussed.

2.2.3.1 The Compression Rate

A hyperparameter required to determine $\Psi^{(t)}$ is the dimension of the trained submanifold d. In practice, this is done by setting the compression rate $q = 1 - d/D$. Compression rates can be either set globally, *i.e.* on all parameters $\Theta^{(t)}$ together, or for each layer individually.

2.2.3.2 Criterion to Choose the Trainable Parameters

The pruning transformation $\psi^{(t)}$ can easily be constructed by knowing $\mathbf{m}^{(t)}$. In the following, the most common criteria to determine the trainable weights are presented. The criteria are discussed in the context of pruning. However, they can be adapted easily to freeze parts of a network, see Wimmer et al. [192] and Zhou et al. [210].

Random Criterion. The most easy way to determine the trainable weights is by choosing them randomly. Random pruning is often used as first comparison for a newly developed pruning method. *Dynamic sparse training* continuously adapts the pruning transformation $\psi^{(t)}$ during training. Therefore, initially trained parameters are often chosen by random [40, 122] as the dynamics of the sparse topology will find a well performing architecture anyway. Also for DRT methods where information flow is not a limiting factor, like freezing [97, 157], trainable weights are often chosen randomly.

Magnitude Criterion. Pruning Θ can be viewed as solving the minimal distortion problem

$$\min_{\bar{\Theta}\in\mathbb{R}^D, \|\bar{\Theta}\|_0 \leq d} \|\Theta - \bar{\Theta}\|_2 \,, \tag{2.40}$$

which bounds the overall distortion of a network on a compact domain since the non-linearities are usually Lipschitz continuous and the layers are affine linear. The solution

to (2.40) is given by

$$\bar{\Theta}_i = \begin{cases} \Theta_i, & \text{if } \Theta_i \text{ belongs to the top } d \text{ magnitudes of } \{\Theta_1, \dots, \Theta_D\} \\ 0, & \text{else} \end{cases} . \tag{2.41}$$

This is also called *magnitude pruning*, *i.e.* pruning the weights with the smallest magnitudes while training those with big magnitudes.

Since magnitude pruning solves (2.40), the sparse networks approximate dense networks well if the pruned parameters are not too big. In practice, this is achieved by pruning only small fractions of the parameters in one step [43, 44, 59, 107].

Gradient Based Criterion. Pruning can also be interpreted as a disturbance of the loss function

$$\mathscr{L}(\Theta - \mathbf{r}) = \mathscr{L}(\Theta) - \langle \underbrace{\nabla_\Theta \mathscr{L}}_{\mathbf{g}}, \mathbf{r} \rangle + \mathcal{O}(\|\mathbf{r}\|^2) . \tag{2.42}$$

Here, $\mathbf{r} = \Theta - \bar{\Theta}$ is the distortion caused by pruning where $\bar{\Theta} = \Psi(\vartheta)$ is the pruned version of Θ with $d \ll D$ non-zero parameters. Ignoring the higher order terms, the distortion of the loss function induced by pruning can be minimized by

$$\max_{\mathbf{r}} - \langle \mathbf{g}, \mathbf{r} \rangle = \underbrace{- \sum_{i=1}^{D} g_i \cdot \Theta_i}_{const} + \max_{\vartheta} \sum_{i=1}^{D} g_i \cdot \bar{\Theta}_i = const + \max_{\vartheta} \sum_{i=1}^{D} g_i \cdot \bar{\Theta}_i . \tag{2.43}$$

In most applications [27, 63, 94, 95, 160, 181, 192], $g_i \cdot \bar{\Theta}_i$ is exchanged by its absolute value. Therefore, *gradient based* pruning sets those weights with the lowest $|g_i \cdot \bar{\Theta}_i|$ to zero since that maximizes (2.43), *i.e.*

$$\bar{\Theta}_i = \begin{cases} \Theta_i, & \text{if } |\mathbf{g}_i \cdot \Theta_i| \text{ belongs to the top } d \text{ values of } \{|\mathbf{g}_1 \cdot \Theta_1|, \dots, |\mathbf{g}_D \cdot \Theta_D|\} \\ 0, & \text{else} \end{cases} . \tag{2.44}$$

The gradient based criterion for pruning at initialization can lead to a loss of information flow for high pruning rates since some layers are pruned too aggressively [174, 184, 192]. However, information flow is essential for training since the gradient vanishes otherwise and then training is not possible anymore. Networks fulfilling the layerwise *dynamic*

isometry property [144, 162, 163, 197] – which can be achieved by a rescaling of the random initialization – will lead to an improved information flow in the pruned network [95]. Also, adding a basis transformation $\Phi^{(t)}$ on top of $\psi^{(t)}$ improves the information flow and consequently the sparse network [195].

Conserving Information Flow. Sufficient information flow can also be guaranteed directly by the criterion for selecting the trainable parameters. Wang et al. [184] train the sparse network with the highest total gradient L_2 norm

$$\Delta\mathscr{L}(\Theta) := \langle \nabla_\Theta \mathscr{L}, \nabla_\Theta \mathscr{L} \rangle = \langle \mathbf{g}, \mathbf{g} \rangle \tag{2.45}$$

after pruning. This is optimized by evaluating the first order Taylor expansion of $\Delta\mathscr{L}(\Theta)$ at the sparse model, *i.e.* for the pruning distortion $\mathbf{r} = \Theta - \bar{\Theta}$, via

$$\Delta\mathscr{L}(\Theta) - \Delta\mathscr{L}(\Theta - \mathbf{r}) = -2 \cdot \mathbf{r}^T \cdot \nabla^2_\Theta \mathscr{L} \cdot \nabla_\Theta \mathscr{L} + \mathcal{O}(\|\mathbf{r}\|^2) \,. \tag{2.46}$$

Therefore, maximizing $\bar{\Theta}_i \cdot (\mathbf{H} \cdot \mathbf{g})_i$ will lead to sparse networks with high gradient flow, where $\mathbf{H}$ is the Hessian of the loss evaluated at Θ. Consequently, the information flow is not the bottleneck for training. To compute these scores, not the whole Hessian $\mathbf{H}$ needs to be computed but only the Hessian-vector product $\mathbf{H} \cdot \mathbf{g}$ which is cheaper to calculate.

Synaptic saliency of weights is introduced in Tanaka et al. [174]. A synaptic saliency is defined as

$$\nabla_\Theta \mathcal{R} \odot \Theta \,, \tag{2.47}$$

where $\mathcal{R} : \mathbb{R}^D \to \mathbb{R}$ is an almost everywhere smooth function which depends on the weights Θ. In Tanaka et al. [174], a conservation law for a layer's total synaptic saliency is shown. By iteratively pruning a small fraction of parameters based on a synaptic saliency score (2.47), the conservation law guarantees a faithful information flow in the sparse network. Examples for a synaptic flow based pruning criterion are setting $\mathcal{R}(\Theta)$ as the L_1 path norm [131, 132] [174] or the L_2 path norm [52, 139] of the DNN f_Θ.

2.2.3.3 Updating the Transformation

For dimensionality reduced training, the network's parameters $\Theta^{(0)}$ are compressed at the beginning of training. This generates the initial embedding $\Psi^{(0)}$ as well as the

corresponding dimensionality reduced coordinates $\vartheta^{(0)}$. Therefore, training starts with $\Psi^{(0)}(\vartheta^{(0)})$.

Training the network corresponds to updating $\vartheta^{(t)}$ as discussed in Section 2.1.2. Moreover, the transformation $\Psi^{(t)}$ can be adapted as well. This results in an adaptive submanifold where training takes places. However, $\Psi^{(t)}$ is restricted in this thesis to always define a d dimensional submanifold to guarantee a fully dimensionality reduced training. Note, most discussed methods actually do not change $\Psi^{(t)}$ but keep $\Psi^{(0)}$ fixed.

All transformations used in this work share the pruning transformation $\psi^{(t)}$ which is an element of $\{0,1\}^{D\times d}$. As such, $\psi^{(t)}$ can not be optimized naively via stochastic gradient based optimization. On way to overcome this is presented by Wimmer et al. [195] whom add an additional basis transformation $\Phi^{(t)}$ on top of the pruning transformation, see Section 2.2.2.1. In this way, $\Phi^{(t)}$ can be adapted easily via backpropagation. This so called *interspace pruning* will be discussed later on in this work.

Another way is to frequently update the pruning transformation $\psi^{(t)}$ by pruning new parameters while giving other parameters the chance to re-grow. This is called *dynamic sparse training* [7, 30, 40, 122, 125]. The parameters are usually pruned each t_p iterations while the same number of parameters is re-grown afterwards. The re-growing is based on purely random processes [122, 125] or on the size of the gradient of formerly untrained weights [30, 40].

2.3 State-of-the-art in Dimensionality Reduced Training

This Section gives an overview of the SOTA methods for DRT achieved by pruning or freezing networks before training begins. The best *pruning* methods applied before training are presented in Section 2.3.1 while Section 2.3.2 presents an overview of freezing approaches. Finally, Section 2.3.3 discusses problems of the SOTA pruning methods applied before training which are tackled by the three methods presented in this work.

2.3.1 Pruning Models before Training Begins

Since all three proposed methods in this thesis are closely related to network pruning, this Section gives a detailed overlook in the related pruning literature.

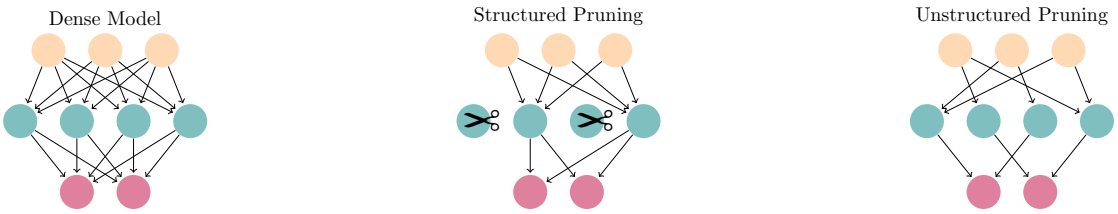


Figure 2.9: (Figure 5 in Wimmer et al. [195]) Graphical comparison between structured and unstructured pruning for a MLP.

Goals for Pruning Methods. Pruning is used to reduce the number of parameters in a DNN [59, 91, 126] which is achieved by setting parameters to zero. This reduces the complexity of the model and, by the bias-variance trade off, may even improve the generalization ability of the network [6, 62, 91, 194]. Also, forward- and backward propagation of the DNN can be accelerated due to sparse tensor computations [10].

Structured and Unstructured Pruning. Pruning is divided in *structured* and *unstructured* pruning, see Figure 2.9. Structured pruning sets whole structures, *e.g.* channels, neurons or even coarser structures like whole layers, to zero. Therefore, structured pruning generates leaner and/or smaller structures which immediately reduces the runtime of the model. Unstructured pruning sets single weights to zero, without the restriction of zeroing a group of weights together [4, 73, 101, 175, 187, 211]. Compared to structured pruning, unstructured pruning leads to better results since it is more flexible [98]. However, specialized soft- and hardware is required to exploit the sparse tensors for accelerating computations in the forward and backward propagation [38, 50, 60, 136].

Pruning a Pre-trained Model. This work addresses dimensionality reduction applied before training. However, networks can be pruned at any time in training, not only before it begins. The historically first approaches [61, 78, 82, 91, 126] prune a pre-trained model by an iterative prune and fine-tune approach. The pruning criteria are often based on the Hessian of the weight vector Θ w.r.t. the loss function but also magnitude based pruning can be used [49, 59, 98].

Pruning During Training. Moreover, sparse networks can be enforced by adding sparsity imposing regularizations to the loss function [16, 17, 61, 110, 199], like a

smoothed version of the L_0 regularization. After training, many parameters are shrunken to zero, or are almost zero, due to the sparsity enforcing regularization. These weights can therefore be pruned without damaging the network's performance. Since the number of zeros is determined by the strength of the regularization term, the compression rate of such methods can not directly be set but needs to be found by testing different strengths of the regularization term.

Pruning can also be applied during training by gradually increasing the sparsity in the network, the so called *gradual magnitude pruning* [49].

Lottery Ticket Hypothesis. The *Lottery Ticket Hypothesis* (LTH) [43] successfully showed that it is possible to train extremely sparse, randomly initialized networks to the same performance as their dense counterpart while converging faster – or at least with the same speed. To find these sparse networks, so-called *lottery tickets*, an iterative pruning procedure has to be applied. First, a dense network is trained to convergence. Then, weights are pruned based on their magnitude. For the best results, only 20% of the weights are pruned in one step. The non-zero weights are reset to their initial value and again trained to convergence. After $\left\lceil \frac{\log(1-q)}{\log 4 - \log 5} \right\rceil$ trainings, the final sparse model with compression rate q is found. The lottery ticket with sparsity q is then trained a last, final time.

Resetting weights to their initial value only shows good results for small datasets and small CNN architectures [43, 44, 49]. This problem can be overcome by resetting weights to values reached early in training [44]. The high cost for finding lottery tickets with the iterative approach can be reduced by using early stopping or low precision training [202], sharing lottery tickets for different datasets and optimizers [124] or iteratively reducing the dataset together with the number of non-zero parameters [208].

Pruning at Initialization. Finding well trainable sparse architectures at initialization without expensive pre-training – which is required to find lottery tickets – is the main motivation for pruning at initialization (PaI). The first method SNIP [94] only trains the weights with the highest influence in changing the loss function. The influence of a single weight on changing the loss function can be measured by the gradient criterion (2.43). For high compression rates, however, this generates sparse architectures with low information flow [174, 184, 192]. The information flow can be increased by adapting

the random initializations [63, 95] or by iteratively pruning the network [27, 174, 181]. Moreover, the sparse networks can directly be optimized to have a high information flow [174, 184]. To do this, GraSP [184] only keeps those weights which increase the gradient flow in the sparse model via (2.46). SynFlow [174] conserves a synaptic flow generated by the L_1 path norm in the sparse model which also guarantees a strong information flow. The trained weights for SynFlow are chosen according to (2.47).

Theoretical works about the so-called *strong lottery tickets* show that a randomly initialized DNN contains, if it is wide or deep enough, a sparse, untrained network that performs equally well as a trained network with the same number of non-zero parameters [15, 117, 135, 140, 148]. Ramanujan et al. [148] propose an algorithm to extract these strong lottery tickets without training any parameter by optimizing a pruning score for each weight.

A single PaI method usually performs well for some compression rates but also shows bad results for other ones [45]. Especially, there is not a clear SOTA PaI method [45]. Moreover, the proposed PaI methods are sometimes even outperformed by random or magnitude pruning at initialization [45, 194].

Dynamic Sparse Training. A complete sparse training can also be performed if the topology of the sparse network is adaptive. This is called *dynamic sparse training* (DST). For DST, the pruning transformation $\psi^{(t)}$ is adaptive. After each t_p training iterations, the pruning transformation is updated. The initial sparse model, *i.e.* $\psi^{(0)}$, is usually determined by random pruning [40, 122]. During training, the weights are pruned based on their magnitude. Since weights are pruned, new weights can be re-grown without increasing the dimension d of the space of trainable parameters. Weights are re-grown randomly [122, 125] or based on their gradient [30, 40]. Moreover, weights might be redistributed over different layers [30, 125]. Liu et al. [103] explain the success of DST methods by the *in-time over-parametrization*. This means that DST shows good results if enough different sparse architectures are tested out during training while at the same time each tested architecture has enough time to be optimized. Such a training can be seen as an overparametrization in the space-time manifold. Therefore, results for DST can be improved by either using more training steps or by reducing the batch size for training while training with the same number of data points until $\psi^{(t)}$ is updated. Evci et al. [40] also show empirically that spending more time on training DST models significantly

improves their performance.

2.3.2 Freezing Models at their Random Initialization

Models with random weights can be used to generate meaningful features [54, 161]. Randomly initialized convolutional pooling architectures are inherently frequency selective [161]. Moreover, randomly initialized ReLU networks shrink the euclidean distance between two input data points inversely proportional to their euclidean angle as shown by Giryes et al. [54]. Assuming that data points from the same class have smaller euclidean angles than data points from different classes, random ReLU networks can be seen as an *universal system that separates any data* [54]. Deeper random networks tend to separate data better since each layer shrinks the distance between data points from the same class even more. However, in real applications, not all data points behave well. Therefore, the network needs training – at least for a fraction of the parameters – to ensure good generalization ability [54].

Li et al. [97] use frozen weights in order to measure the *intrinsic dimension* of a given deep learning problem. The intrinsic dimension is the minimal dimension d of the affine subspace spanned by randomly freezing a part of the parameters which still leads to similar results as dense training. By measuring these dimensions, they show that frozen networks can reach competitive results. In contrast to FreezeNet [192], Li et al. [97] use a dense random orthogonal transformation instead of the pruning transformation $\psi^{(0)}$.

Extreme versions of frozen networks, freeze the whole DNN except the classification layer [71, 72, 147]. On the other hand, freezing the classification layer while only training the remaining network, together with a temperature parameter which scales the random layer, can lead to similar results as dense training [68].

Zhou et al. [210] compare pruning and freezing weights in the context of the lottery ticket hypothesis. Therefore, they use magnitude based pruning/freezing and reset the trainable weights to their random initialization. Similar to FreezeNet [192], they show that freezing weights improves results if the compression rate is high whereas pruning leads to better results for lower compression rates. Finally, they show that a combination of freezing and pruning leads to the best results. In the combined version, they prune the weights which moved towards zero during training the dense model while those who moved away from zero are frozen.

Frozen weights are also used in the context of transfer learning [171] and multi-task learning [182]. Hereby, freezing some weights prevents catastrophic forgetting, *i.e.* forgetting of already learned tasks or representations which are still beneficial for the network but are forgotten since they are not represented in the current data anymore. Moreover, Sung et al. [171] show that freezing weights helps to significantly reduce the communication costs for distributed training.

Rosenfeld and Tsotsos [157] freeze weights in a structured way – mainly they freeze whole channels in CNNs. The frozen weights are chosen randomly. In this set-up, freezing outperforms pruning for almost all compression rates.

Finally, Rosenfeld and Tsotsos [157] and Frankle et al. [46] investigate networks which are frozen *completely* while only the batch normalization layers are trained. Only training the batch normalization layers leads to surprisingly well performing networks which are far better than random guessing.

2.3.3 Problems of State-of-the-Art Sparse Training Methods

This Section discusses drawbacks of existing DRT methods. Overcoming these problems is a comprising motivation for the proposed DRT methods FreezeNet (Chapter 3), COPS (Chapter 4) and interspace pruning (Chapter 5). Of course, none of the three methods solves all problems for DRT methods at once, yet they improve existing SOTA.

2.3.3.1 High Costs to Find Sparse Architecture

As already mentioned, finding and training a lottery ticket with sparsity q requires $\left\lceil \frac{\log(1-q)}{\log 4 - \log 5} \right\rceil + 1$ trainings. Training a model with sparsity $q = 0.9$ with this procedure requires 11 trainings to find the lottery ticket and a final, 12^{th}, sparse training. Of course, this is quite expensive for obtaining a single model with sparsity q. Lottery tickets can be found up to high sparsity [43, 44]. However, the high costs for finding these tickets make them unpractical for big scale datasets and big DNN architectures.

Contrarily to iteratively cutting the lottery ticket from the dense initialization, dynamic sparse training only requires one fully sparse training. The idea is to allow the sparse topology to adapt during training. This is done by frequently pruning weights and letting the same number of parameters re-grow. DST leads to well trainable sparse models, also for high compression rates. However, many hyperparameters like layerwise compression

rates, update frequency t_p of the pruning transformation $\psi^{(t)}$ and the re-growing rate for different layers have to be set which in the end requires many training runs to find the best performing hyperparameters [195].

2.3.3.2 Missing Information Flow

An intuitive DRT approach is *pruning at initialization* which, similarly to the lottery ticket hypothesis, trains a sparse randomly initialized network from scratch. In contrast to the lottery ticket hypothesis, the networks are pruned directly after initialization, without any pre-training of the weights.[2] SNIP by Lee et al. [94] was the first method which successfully found sparse networks in a random initialization by using the gradient criterion, see Section 2.2.3.2.

However, pruning weights only based on their influence on the loss function might lead to weak information flow in the sparse architecture [184, 192] and even lead to the pruning of whole layers [174]. The vanishing gradient flow for SNIP for high compression rates is shown in Figure 3.1. A weak information flow in the network of course hinders a successful training. Therefore, SNIP leads to inferior results for high compression rates [174, 184, 192].

2.3.3.3 Inferior Results for Denser Structures

Overcoming the weak information flow of SNIP was a motivation for GraSP [184] and SynFlow [174] to directly optimize the network to have a high information flow – even for extremely high compression rates. By doing so, both methods successfully train networks in the extreme sparsity regime [174, 184]. However, Wimmer et al. [193] showed that only optimizing the network on having a high information flow is not optimal for sparse training. This can be seen for example by GraSP's inferior results to SNIP for denser network structures, *i.e.* lower compression rates, see Figure 4.1. Ideally, a well performing DRT method would not only work well for high compression rates, but also for low ones (and vice-versa).

[2] Even though the lottery ticket hypothesis starts the final sparse training from a random initialization, it uses many trainings of a model with more than d parameters to find the sparse architecture – which is a form of pre-training.

2.3.3.4 Summary

Altogether, DRT methods have several drawbacks. First, they are often quite expensive since they require many training runs to (a) find the sparse topology for the randomly initialized network and (b) optimize the hyperparameters required to successfully train the sparse model with an adaptive topology. Furthermore, the cheaper PaI methods do usually not perform well for both, low and high compression rates. Moreover, it was shown that they are not able to perform on par with lottery tickets or DST [45, 194] and are sometimes even outperformed by random pruning [194].

Chapter 3
FreezeNet: Full Performance by Reduced Storage Costs

This Chapter investigates freezing parts of a DNN for the whole training process and compares it to the more widely used network pruning. The model trained with partly frozen weights is called *FreezeNet*. The frozen weights can be recovered by a single 32bit random seed number if the model is initialized with a pseudo random number generator. Frozen weights guarantee a high information flow in the network – even for high compression rates. Therefore, using frozen weights yields better trainable models with higher capacity than setting the parameters to zero. In particular, FreezeNet shows good results for high compression rates. The following Chapter reflects *FreezeNet: Full Performance by Reduced Storage Costs* by Wimmer et al. [192], follows the structure of this publication and puts FreezeNet in the context of general DRT methods.

3.1 Motivation for Freezing Parts of a Network

The main motivation for FreezeNet is to maintain a high information flow while updating only a sparse part of the network. The PaI method SNIP [94] finds the most dominant weights by using gradient based pruning, see Section 2.2.3.2. As already discussed, this might lead to a weak gradient flow in the sparse network. Figure 3.1 shows the weak gradient flow for SNIP [94] for high compression rates. It also shows that even GraSP [184], which optimizes sparse DNNs to have the highest possible gradient flow, loses information flow for high compression rates. On the other hand, freezing weights instead of pruning them does not generate sparse models which results in superior gradient flow for high compression rates.

FreezeNet aims to improve pruning at initialization methods, *i.e.* methods that prune

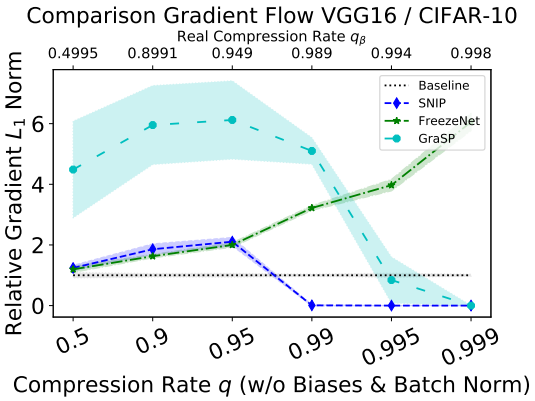


Figure 3.1: (Figure 4 left in Wimmer et al. [192]) Compares the relative gradient norm of the trained weights for FreezeNet [192], SNIP [94] and GraSP [184] normalized by the gradient flow of a densely trained baseline for a VGG16 on CIFAR-10.

randomly initialized networks without any pre-training and which fix the pruning transformation $\psi^{(0)}$. Training a network pruned at initialization leads to sparse gradient computations. But the weak gradient flow of such networks might hinder the DNN to train successfully [184, 192]. FreezeNet uses the same strategy as SNIP to find the trainable weights. However, the untrained weights are frozen at their random initial value. A graphical overview of FreezeNet is given in Figure 3.2. By freezing the weights, the weight tensors are not sparse anymore. Therefore, a proper gradient flow in the network is always guaranteed – even for an extremely small number of trained parameters. The dimensionality reduced model is trained on a fixed d dimensional affine hyperplane $\psi^{(0)} \cdot \vartheta + (\mathbf{1}_D - \psi^{(0)} \cdot \mathbf{1}_d) \odot \Theta^{(0)}$ in $\mathbb{R}^D$, see Figure 2.7 second right. The frozen weights are not updated and therefore do not require a gradient during training. As a consequence, the frozen weights improve the expressiveness of the DNN trained with reduced dimensionality without adding any gradient computations. Finally, the frozen weights can be stored with a single random seed number if the network is initialized with a pseudo-random number generator.

3.1.1 Contributions of FreezeNet: Full Performance by Reduced Storage Costs

The main contribution of Wimmer et al. [192] is given by introducing FreezeNet which can be applied to arbitrary DNNs. In detail, FreezeNet has the following benefits:

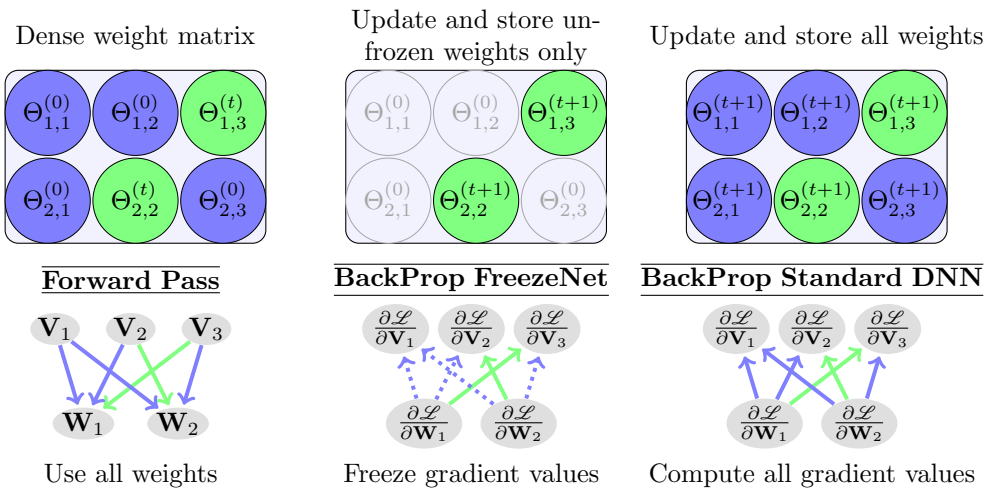


Figure 3.2: (Figure 1 in Wimmer et al. [192]) FreezeNet compared with a standard, dense MLP with neurons $\mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3$ and $\mathbf{W}_1, \mathbf{W}_2$, and corresponding weights and gradient values.

- Improving SNIP [94] by achieving better results with a smaller parameter count.
- Preserving the gradient flow even for extremely high compression rates.
- Storing the frozen weights by using a single random seed number and thereby having similar memory requirements as pruning.
- More efficient training than a standard dense network since the same number $d \ll D$ of gradients as for pruned networks has to be computed.

The major improvement achieved by FreezeNet – compared to pruning – is a faithful gradient flow for all compression rates. Moreover, weight decay can be applied to the frozen weights. By doing so, the frozen weights are smoothly shrunken to zero during training. This leads to a sparse network in the end of training while the information flow of the dense structure can be used at early training stages to improve results. In all situations where SNIP shows better results than FreezeNet, applying weight decay on the frozen weights leads to better results than pruning while both methods yield the same number of zeroed parameters.

Since frozen weights do not need to be updated, FreezeNet is perfectly suitable for applications which spend much time for training the network and less time for inferring

it, as for example networks trained for research or prototyping. Also, DNNs which have to be re-trained many times due to changing data, like transfer learning or online learning, profit from the reduced training costs for FreezeNet. Finally, FreezeNet is cheap to store and transfer since it only trains a fraction of the model's parameters. This could be beneficial for applications which require frequent updates of the used DNNs, like autonomous vehicle fleets or internet services, but also for applications which have strict restrictions on the maximal DNN size. Finally, freezing weights massively reduces the communication costs for distributed training [171] and helps to overcome catastrophic forgetting in the domain of multi task and transfer learning [171, 182].

3.2 Related Work for FreezeNet

The related work for FreezeNet can be divided in *pruning* and *DNNs with frozen weights*. The related pruning literature is provided in Section 2.3.1 while models with frozen weights are covered in Section 2.3.2.

3.3 Freezing Parts of the Weights During Training

Assume a DNN $f_\Theta : \mathbb{R}^{d_{\text{in}}} \to [0,1]^C$ with parameters $\Theta \in \mathbb{R}^D$ to be given. Since batch normalization and bias parameters are so little compared to the weights of a DNN, SOTA unstructured pruning methods only prune weights and keep the bias and batch normalization parameters trainable [43, 94, 184]. To have a fair comparison, FreezeNet also freezes only weights while bias and batch normalization parameters are always trained.

3.3.1 Freezing Criterion

FreezeNet relies on the same criterion to choose the trainable weights as SNIP [94] – the gradient based criterion which is presented in Section 2.2.3.2. Let $\Theta^{(0)} \in \mathbb{R}^D$ be the randomly initialized weights of a DNN. For simplicity, assume that $\Theta^{(0)}$ only contains the weights, not the bias and batch normalization parameters. Now the so-called *saliency*

score is computed for each weight as the gradient criterion according to (2.43), *i.e.*

$$\mathbf{s}_i := \left| \frac{\partial \mathscr{L}(\Theta^{(0)})}{\partial \Theta_i^{(0)}} \cdot \Theta_i^{(0)} \right| . \tag{3.1}$$

Weights $\Theta_i^{(0)}$ with high scores $\mathbf{s}_i$ are trained whereas weights with low scores are frozen. The gradient $\nabla_{\Theta^{(0)}} \mathscr{L}$ is approximated with one batch of training data. However, for bigger datasets with many classes $C \gg B$, like ImageNet, more accurate scores can be obtained by using more than one batch of training data for determining $\mathbf{s}_i$.

An accessible interpretation of the saliency score is given by Lee et al. [94]. It holds by the quotient rule [94, 95]

$$\mathbf{s}_i = \left| \left. \frac{\partial \mathscr{L}(\Theta^{(0)} \odot \mathbf{m})}{\partial \mathbf{m}_i} \right|_{\mathbf{m}_i=1} \right| \tag{3.2}$$

where $\mathbf{m} \in \mathbb{R}^D$ models the activation of a weight. If $\mathbf{m}_i = 1$, the corresponding weight $\Theta_i^{(0)}$ is trainable whereas it is not trainable if $\mathbf{m}_i = 0$. Now, $\mathbf{s}_i$ measures how changing the activation of a weight influences the loss function. If this influence is high, the corresponding weight is beneficial for training whereas it is not too important if the saliency score is low. After the saliency score has been computed, $\mathbf{m}_i$ is set to zero if the score is low and to one if $\mathbf{s}_i$ is high. A threshold depending on the compression rate determines which weights are frozen and which ones are trained, see Algorithm 3.1. The so found $\mathbf{m}$ is the freezing mask of the model, which was introduced in Section 2.2.2.2.

3.3.2 Backpropagation in Sparse Models

The main motivation for FreezeNet is to overcome the weak gradient flow which PaI methods have for high compression rates. Therefore, the gradients of the DNN have to be analyzed. For simplicity, the following backpropagation formulas are restricted to fully connected layers. As shown in Section 2.1.1.4, similar formulas hold for convolutional

Algorithm 3.1: FreezeNet

Input: Compression rate q,
Initial parametrization $\Theta^{(0)} \in \mathbb{R}^D$,
Corresponding network $f_{\Theta^{(0)}}$,
Loss function $\mathscr{L}$

Output: Trained DNN $f_{\mathbf{m}\odot\tilde{\Theta}+(\mathbf{1}-\mathbf{m})\odot\Theta^{(0)}}$ with partly frozen weights

1 Compute saliency score $\mathbf{s} \in \mathbb{R}^D$ with one batch of training data according to (3.1)
2 Define freezing mask $\mathbf{m} \in \{0,1\}^D$
3 Set freezing threshold $\varepsilon_{q,D}$ as the $\lfloor (1-q)\cdot D \rfloor$-highest value of $\mathbf{s}$
4 Set $\mathbf{m}_i = 0$ if $\mathbf{s}_i < \varepsilon_{q,D}$ else $\mathbf{m}_i = 1$
5 Train with forward propagation as usual but use gradient $\mathbf{m} \odot \nabla_{\Theta^{(t)}}\mathscr{L}$ for weights

layers. The backpropagation formulas in layer l have the following form

$$\nabla_{\Theta^{[l]}}\mathscr{L} = \nabla_{\mathbf{W}^{[l]}}\mathscr{L} \otimes \mathbf{V}^{[l-1]} \tag{3.3}$$

$$\nabla_{\mathbf{W}^{[l]}}\mathscr{L} = \nabla_{\mathbf{V}^{[l]}}\mathscr{L} \odot \phi^{[l]\prime}\left(\mathbf{W}^{[l]}\right) \tag{3.4}$$

$$\nabla_{\mathbf{V}^{[l]}}\mathscr{L} = (\nabla_{\mathbf{W}^{[l+1]}}\mathscr{L} \cdot \Theta^{[l+1]})^T\,. \tag{3.5}$$

For sparse weight matrices $\Theta^{[l+1]}$, equation (3.5) can lead to small gradients $\nabla_{\mathbf{V}^{[l]}}\mathscr{L}$ which will also lead to small weight gradients $\nabla_{\Theta^{[l]}}\mathscr{L}$ in the previous layer, due to (3.4) and (3.3). In the extreme case, where $\nabla_{\mathbf{W}^{[l]}}\mathscr{L} = 0$ for a layer l, all previous layers $\tilde{l} \leq l$ have a gradient equal to zero and therefore learn nothing. If the untrained parts of the model are kept at their random position instead of setting them to zero, this problem does not occur. Overcoming the drying up of the gradient while training only a sparse part of a DNN was the main motivation for freezing the untrained weights instead of pruning them.

3.3.3 FreezeNet

Algorithm 3.1 summarizes training a FreezeNet. FreezeNet only requires one additional hyperparameter, the compression rate q. Before training, the saliency score $\mathbf{s}$ is computed according to (3.1). For the required gradient, one batch of training data is used to estimate it. However, more than one batch might be used to get an even better estimate of $\nabla_{\Theta^{(0)}}\mathscr{L}$. If $\mathbf{s}$ is computed, the freezing threshold $\varepsilon_{q,D}$ can simply be set as the $\lfloor (1-q)\cdot D \rfloor$-highest

value of $\mathbf{s}$. Then, the freezing mask is defined as $\mathbf{m} \in \{0,1\}^D$ by

$$\mathbf{m}_i = \begin{cases} 0 & \text{if } \mathbf{s}_i < \varepsilon_{q,D} \\ 1 & \text{else} \end{cases} . \tag{3.6}$$

Contrarily to pruning, the corresponding weight values $\Theta_i^{(0)}$ with $\mathbf{m}_i = 0$ are not set to zero but frozen at their randomly initialized value. To guarantee this, the gradient of the network is multiplied element-wise by the freezing mask, *i.e.* the gradient used to update the parameters is given by $\mathbf{m} \odot \nabla_{\Theta^{(t)}} \mathscr{L}$. However, for computational and storage reasons, it is more advantageous to not set the gradient values of the frozen weights to zero after having computed them, but to not compute them in the first place.

3.3.3.1 Memory Requirements

The memory requirements for a FreezeNet are similar to standard pruning. PyTorch [138] and TensorFlow [1] provide pseudo random number generators. By initializing the network with these pseudo random numbers, only the random seed for reproducing the pseudo random initialization has to be stored to recover the frozen weights. As discussed in Section 2.2.2.3, the trained weights together with the structure of the trained weights, *i.e.* $\vartheta^{(t)}$ and $\psi^{(0)}$, can be stored together by using the CSR format [177] or entropy encoding [36]. In total, FreezeNet has only 32bit more storage requirements than a pruned network which comes from the 32bit random seed. This tiny additional memory requirement is neglectable.

Batch normalization and bias parameters represent only a fraction of the network's total parameters.[3] In the literature it is therefore common to not prune/freeze the bias and batch normalization parameters, but only the weights of the network. Moreover, the compression rate is usually computed w.r.t. weights only whereas batch normalization and bias parameters are ignored. If not mentioned otherwise, this thesis also reports compression rates w.r.t. the weights only.

Since FreezeNet performs well up to extreme compression rates for weights, the small number of trainable bias and batch normalization parameters become significant again.

[3] For the models tested in Wimmer et al. [192], the bias and batch normalization parameters contribute around $0.1 - 0.2\%$ to the network's overall parameters.

Table 3.1: (Table 1 in Wimmer et al. [192]) Comparing costs for standard training, pruning at initialization and a FreezeNet.

Method	# Non-zero Weights	# Weights to Store	Sparse Gradients	Sparse Tensor Computations	Faithful Gradient Flow
Standard	D	D	✗	✗	✓
Pruned	$D \cdot (1-q)$	$D \cdot (1-q)$	✓	✓	✗
FreezeNet	D	$D \cdot (1-q)$	✓	✗	✓

Therefore, all plots in this Chapter provide a second x-axis, showing the *real compression rate* q_β which takes all parameters into account, *i.e.* the weights, batch normalization and bias parameters. Therefore, $q \geq q_\beta$ where the biggest differences can be seen for extreme compression rates $q \approx 1$.

3.3.3.2 Computational Cost

Pruned networks use sparse weight tensors $\psi^{(0)} \cdot \vartheta^{(t)} = \mathbf{m} \odot \Theta^{(t)}$ in the forward- and backward pass. In theory, the number of computations required for the forward pass of a pruned model can therefore be reduced by q_β. FreezeNet uses *all* weights in the forward pass in order to increase the information flow in the model. Therefore, FreezeNet has the same computational cost as the dense model. However, if the used soft- or hardware does not support sparse computations, FreezeNet and pruned networks have the same computational cost.

By looking at (3.5), computing $\nabla_{\mathbf{V}^{[l]}} \mathscr{L}$ requires $\mathbf{m} \odot \Theta^{[l+1]}$ for a pruned model, whereas frozen networks compute $\nabla_{\mathbf{V}^{[l]}} \mathscr{L}$ with the help of the dense weights $\Theta^{[l+1]}$. Again, this results in higher computational requirements for FreezeNet – if sparse tensor computations are supported – while the information flow for FreezeNets remains strong even for high compression rates. On the other hand, only the gradients $\mathbf{m} \odot \nabla_{\Theta^{[l]}} \mathscr{L}$ are required for updating a FreezeNet. Therefore, training a model with frozen weights reduces the computational time, but not as much as training a pruned model. Moreover, the number of gradients needed to be stored for updating the weights via SGD can be reduced by q_β. A qualitative comparison of the computational costs for training a dense, pruned and frozen model is given in Table 3.1.

3.4 Experimental Results and Discussion for FreezeNet

In this Section, experimental results for FreezeNet are presented and discussed. The experiments are conducted on MNIST [92], CIFAR-10/100 [85] and Tiny ImageNet [89]. Freezing parts of a model at initialization is compared with pruning it by using SNIP [94] as PaI method. This means that both, pruning and freezing use the same criterion to choose the trainable weights.

The experiments for MNIST are conducted on the fully connected LeNet-300-100 [92] and a small CNN, LeNet-5-Caffe [92]. Results for CIFAR-10/100 are provided with a VGG-16 [168]. Moreover, experiments for Tiny ImageNet are run with a ResNet34 [64].

If not mentioned otherwise, all models are initialized with a Xavier-normal initialization [55] and weight decay is applied on the trainable parameters only. The experiments are run 5 times for MNIST and CIFAR as well as 3 times for Tiny ImageNet with different random seeds, resulting in different network initializations, data orders and data augmentations. The plots show the mean $\pm$ one standard deviation. All hyperparameters are equal as in Lee et al. [94] and are listed in Table A.9 in the Appendix, Section A.3. The used DNN architectures are provided in Section A.2 in the Appendix.

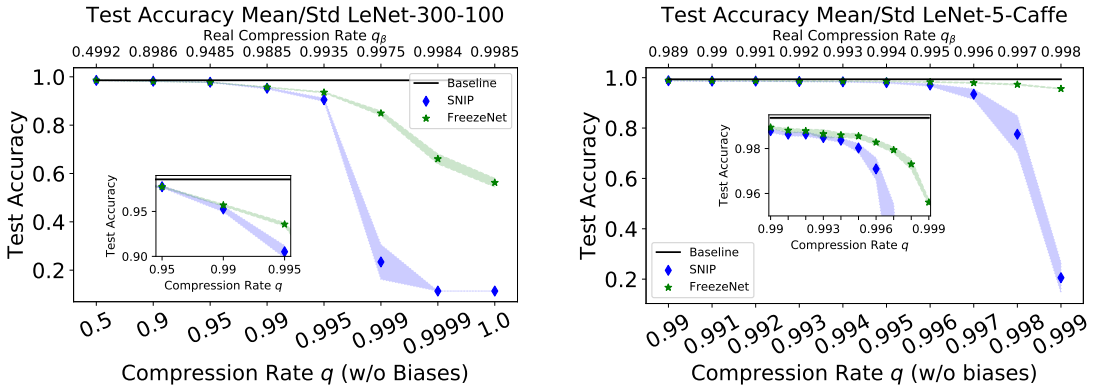
3.4.1 Experimental Results for FreezeNet

Results for SNIP and FreezeNet on the image classification tasks MNIST, CIFAR-10/100 and Tiny ImageNet are compared and discussed in the following.

3.4.1.1 MNIST

Often, a first baseline to compare pruning techniques is the classification task of handwritten digits, MNIST [92]. MNIST is tested on a small fully connected network, LeNet-300-100, and a small CNN, LeNet-5-Caffe. Moreover, FreezeNet's results on LeNet-5-Caffe are compared with other SOTA pruning methods.

LeNet-300-100. A comparison between SNIP and FreezeNet for a LeNet-300-100 is given in Figure 3.3(a). As shown, both SNIP and FreezeNet perform similar for compression rates $q \leq 0.95$. Moreover, they match the dense baseline for $q \leq 0.9$. For



(a) (b)

Figure 3.3: (Figure 2 in Wimmer et al. [192]) Test accuracy for SNIP, FreezeNet and the dense baseline for (a) LeNet-300-100 and (b) LeNet-5-Caffe for the MNIST classification task. Inserted are zoomed versions of the plots for better visualization.

high compression rates $q > 0.95$, FreezeNet exceeds its pruning counterpart. Remarkably, training only bias parameters while freezing *all* weights for $q = 1$ shows 56.2% test accuracy while SNIP degenerates to random guessing.

LeNet-5-Caffe. Similar to LeNet-300-100, results for FreezeNet and SNIP are equivalent for moderate compression rates $q \leq 0.95$. Therefore, Figure 3.3(b) only shows results for high compression rates $q \geq 0.99$. As suspected, FreezeNet performs better than SNIP in the regime of high compression rates. The performance gap widens with increasing q. For $q \geq 0.995$, SNIP's performance drops steeply whereas FreezeNet is able to keep a satisfying performance. As shown in Table 3.2, FreezeNet is able to save parameters w.r.t. both, SNIP pruned models and the dense baseline.

FreezeNet's improved performance can be explained by an improved trainability for the networks. To test this, a LeNet-5-Caffe is trained without applying weight decay to the parameters. As a result, the model overfits on the training data if its capacity is big enough. Figure 3.4(a) shows that FreezeNet is able to overfit up to higher compression rates than SNIP. Thus, it has more capacity for high q. However, for $q \geq 0.992$ FreezeNet does not perfectly memorize the training data anymore. Therefore, using enough trainable weights is important to ensure enough capacity in a FreezeNet.

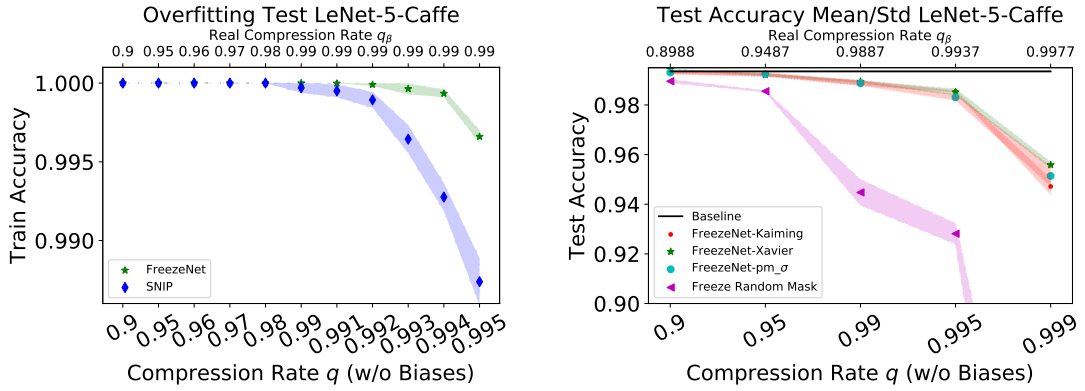
Table 3.2: (Table 2 in Wimmer et al. [192]) Comparison of FreezeNet, SNIP and the LeNet-5-Caffe baseline. Results for different *compression rates* q with corresponding *real compression rates* q_β are displayed. The *network sizes* are calculated without compression, *i.e.* all weights are stored as 32bit floats. The *compression factor* of FreezeNet/SNIP is the ratio of the *network sizes* of the baseline and the frozen/pruned network.

q	q_β	Network Size	Compression Factor	Test Acc. SNIP	Test Acc. FreezeNet	$\frac{\text{FreezeNet Acc.}}{\text{Baseline Acc.}}$
0 (Baseline)		1,683.9kB	1		99.36%	1.000
0.9	0.899	170.7kB	**9.9**	99.24%	**99.37%**	1.000
0.99	0.989	19.1kB	**88.2**	98.80%	**98.94%**	0.996
0.995	0.994	10.7kB	**157.4**	98.02%	**98.55%**	0.992
0.999	0.998	3.9kB	**431.8**	20.57%	**95.61%**	0.962

Comparison With Other Pruning Methods. SNIP degenerates to random guessing for high compression rates. Table 3.3 compares the results of FreezeNet with various SOTA pruning methods. The comparison is done for a LeNet-5-Caffe trained on MNIST. The compared methods are SNIP [94], GraSP [184], the dynamic sparse training method *Sparse-Momentum* and three pruning methods applied during training or on a pre-trained model, *Connection-Learning* [59], *Dynamic-Network-Surgery* [57] and *Learning-Compression* [16]. The results for FreezeNet, GraSP and SNIP are reported from Wimmer et al. [192] whereas the remaining results are reported from the corresponding papers.

The best performance is shown by Connection-Learning and Sparse-Momentum with 99.2% test accuracy. FreezeNet performs slightly worse than these two methods with 99.1% accuracy. FreezeNet trains only 1% of the weights while Connection-Learning trains 8.3% of the weights. As shown in Table 3.2, FreezeNet has 99.4% test accuracy when training 10% of the weights. The dynamic sparse training method Sparse-Momentum updates the pruning transformation $\psi^{(t)}$ frequently during training. For this, the gradient momentum is computed which requires the knowledge of the whole gradient, *i.e.* also for the untrained weights, for each training step. Moreover, hyperparameters as the number of newly pruned weights or the frequency of updating $\psi^{(t)}$ has to be set for Sparse-Momentum which makes it a more complicated training method than FreezeNet.

Similar results than FreezeNet are achieved by Dynamic-Network-Surgery which pre-trains a dense model before starting with the pruning procedure. Moreover, FreezeNet



(a) (b)

Figure 3.4: (Figure 3 in Wimmer et al. [192]) (a): Training accuracy for SNIP and FreezeNet, both trained without regularization. (b): FreezeNet with different initializations as well as a randomly frozen FreezeNet with Xavier-normal initialization. Both plots are for a LeNet-5-Caffe trained on MNIST.

shows better results than SNIP, GraSP and Learning-Compression.

3.4.1.2 CIFAR

MNIST is a relatively easy task, both models achieve a test accuracy around 99%. To test FreezeNet on more challenging datasets, the CIFAR-10 and CIFAR-100 classification tasks are used. On top of the standard FreezeNet, now weight decay is also applied on the frozen parameters. This is called FreezeNet-WD. Results for a VGG-16 on CIFAR-10 and CIFAR-100 are summarized in Table 3.4.

CIFAR-10. For moderate compression rates $q \leq 0.95$, SNIP performs slightly worse than the dense baseline while showing better results than FreezeNet. Still, using the randomly initialized weights at the beginning of training can help to improve results. Smoothly setting the randomly initialized weights to zero during training by applying weight decay to them, FreezeNet-WD, helps to improve results for $q \leq 0.95$. In this regime, all untrained weights for FreezeNet-WD can be safely pruned at the early stopping epoch which makes FreezeNet-WD a pruning method that smoothly sets parts of its weights to zero while always training on a d dimensional submanifold. Therefore, FreezeNet-WD improves SNIP without adding any additional gradient computations

Table 3.3: (Table 4 in Wimmer et al. [192]) Comparison of different pruning methods with FreezeNet on LeNet-5-Caffe trained on MNIST. For the FreezeNet, SNIP and GraSP results, a train/validation split of 19/1 is used.

Method	Sparse Gradients in Training	Additional Hyperparameters	Percent of trainable parameters	Test Accuracy
Baseline [92]	–	–	100%	99.4%
SNIP [94]	✓	✗	1.0%	98.9%
GraSP [184]	✓	✗	1.0%	98.9%
Connection-Learning [59]	✗	✗	8.3%	99.2%
Dynamic-Network-Surgery [57]	✗	✗	0.9%	99.1%
Learning-Compression [16]	✗	✓	1.0%	98.9%
Sparse-Momentum [30]	✗	✓	1.0%	99.2%
FreezeNet [192]	✓	✗	1.0%	99.1%

while generating the same sparse architecture after training.

For high compression rates, FreezeNet without weight decay outperforms SNIP. For $q \geq 0.99$, SNIP has similar results as random guessing, while FreezeNet shows decent performance. In this regime, FreezeNet-WD still exceeds SNIP but shows worse results than FreezeNet.

CIFAR-100. CIFAR-100 is more complex than CIFAR-10 since the classes are more fine-grained. As can be seen in Table 3.4, FreezeNet exceeds SNIP for all compression rates. Random weights help to separate distinct classes for ReLU networks [54]. Therefore, features which are generated by the random, frozen parameters might help to improve the model's performance. For more sophisticated tasks, frozen weights seem to be even more beneficial since more complex information has to propagate through the network during training. Using dense weight tensors in the forward- and backward pass helps to provide enough information for the gradients to guarantee a successful training.

3.4.1.3 Tiny ImageNet

FreezeNet is also tested on a ResNet34 [64] on the Tiny ImageNet classification task [89]. Figure 3.5(a) shows the results for FreezeNet, FreezeNet-WD and SNIP. FreezeNet

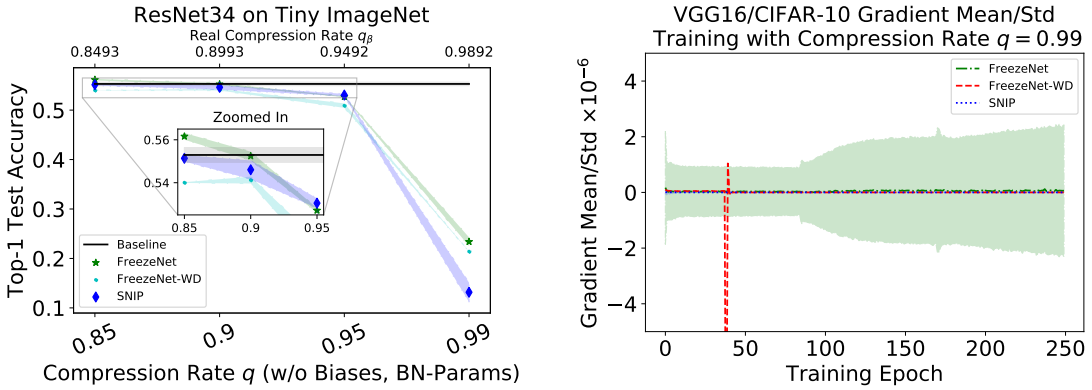
Table 3.4: (Table 3 in Wimmer et al. [192]) Comparison of results for the CIFAR-10/100 tasks with a VGG-16 baseline.

			CIFAR-10	**CIFAR-100**
Method	Compression Rate	Trained Parameters	Mean ± Std	Mean ± Std
Baseline	0	15.3mio	$\mathbf{93.0 \pm 0.1}\%$	$\mathbf{71.6 \pm 0.6}\%$
SNIP	0.9	1.5mio	$92.9 \pm 0.1\%$	$53.9 \pm 1.7\%$
	0.95	780k	$92.5 \pm 0.1\%$	$48.6 \pm 6.6\%$
	0.99	169k	$10.0 \pm 0.0\%$	$1.0 \pm 0.0\%$
	0.995	92k	$10.0 \pm 0.0\%$	$1.0 \pm 0.0\%$
FreezeNet	0.9	1.5mio	$92.2 \pm 0.1\%$	$\mathbf{70.7 \pm 0.3}\%$
	0.95	780k	$91.7 \pm 0.1\%$	$\mathbf{69.0 \pm 0.2}\%$
	0.99	169k	$\mathbf{88.6 \pm 0.1}\%$	$\mathbf{59.8 \pm 0.3}\%$
	0.995	92k	$\mathbf{86.0 \pm 0.1}\%$	$\mathbf{53.4 \pm 0.1}\%$
FreezeNet-WD	0.9	1.5mio	$\mathbf{93.2 \pm 0.2}\%$	$53.1 \pm 1.8\%$
	0.95	780k	$\mathbf{92.8 \pm 0.2}\%$	$44.5 \pm 5.4\%$
	0.99	169k	$76.1 \pm 1.0\%$	$13.1 \pm 1.8\%$
	0.995	92k	$74.6 \pm 1.1\%$	$11.9 \pm 1.4\%$

has similar performance as the dense baseline while training only 10% of its weights. For all q, FreezeNet achieves better or equal results than SNIP. Moreover, FreezeNet's results are more stable, shown by less variance for different random seeds. Similar to CIFAR-100, FreezeNet-WD shows worse results than SNIP for lower compression rates.

3.4.2 Performance Differences between SNIP, FreezeNet and FreezeNet-WD

As shown for MNIST, CIFAR-10/100 and Tiny ImageNet, freezing weights reduces the network's capacity. However, the models still have higher capacity then their pruned counterparts, see Figure 3.4(a). The bias-variance trade-off, Section 2.1.2.5, explains why SNIP performs better for lower compression rates than FreezeNet while FreezeNet exceeds SNIP for high compression rates. For lower q, the model's complexity is big enough, even if the parameters are pruned. In this regime, a lower complexity improves the generalization ability while a higher complexity, *i.e.* more variance, leads to overfitting on the training data. For high compression rates, the expressibility of



(a) (b)

Figure 3.5: (a): (Figure S1 in Wimmer et al. [192]) Comparison of FreezeNet, FreezeNet-WD and SNIP for a ResNet34 trained on Tiny ImageNet. (b): (Figure 4 right in Wimmer et al. [192]) Gradient mean $\pm$ one standard deviation for a VGG-16 trained on CIFAR-10 tracked for one training run for compression rate $q = 0.99$.

the hypothesis class is low. Therefore, the bias term in the bias-variance trade off is high. Reducing the bias term by increasing the model's complexity through freezing the weights instead of pruning them leads to better generalizing models.

In the regime where the compressed model has enough capacity, using frozen weights at the beginning of training and smoothly reducing them to zero during training – as done with FreezeNet-WD – improves results compared to prune them directly after initialization (SNIP). An explanation for this is that the continuous change of the untrained weights in the course of training add an additional regularization to the training procedure.

3.4.2.1 Freezing Weights Improves the Gradient Flow

For higher compression rates, the pruned models have shown worse results than FreezeNet. Often, the performance is comparable to random guessing. Figure 3.5(b) shows that SNIP's problem for high q is a weak information flow for the whole training – here measured by the mean $\pm$ one standard deviation of the model's combined gradient values. On the other hand, FreezeNet has a high information flow for the whole training procedure by using the dense tensors in the forward- and backward pass. Therefore, FreezeNet performs well for high q. Figure 3.5(b) also explains the performance drop of

FreezeNet-WD for high compression rates. At the beginning of training, the untrained weights are still big and guarantee a strong information flow during this time. At some point in training, they become close to zero since weight decay shrinks them. Now, the topology of FreezeNet-WD and SNIP is similar. In Figure 3.5(b) this happens around epoch 40. Afterwards, FreezeNet-WD is not able to learn anymore. It should be highlighted that in this case, the model found with early stopping for FreezeNet-WD is not sparse. At least some untrained weights are non-zero since that is necessary for the information to flow in this case.

As can be seen in Figure 3.5(b), freezing weights improves the gradient flow for high compression rates. However, there exist pruning methods, like GraSP [184] or SynFlow [174], which directly optimize the sparse model to have a high information flow. Figure 3.1 shows a comparison of the mean L_1 gradient norm of FreezeNet, SNIP and GraSP measured after the dimensionality was reduced. For compression rates $q \leq 0.95$, freezing or pruning weights does not significantly change the gradient flow. However, GraSP shows a much higher gradient flow since the network is directly optimized for it. For high compression rates $q \geq 0.99$, setting weights to zero significantly reduces the gradient flow compared to freezing them. Both pruning methods, SNIP and GraSP, reach zero gradient flow for a given compression rate and are therefore not able to train in this sparsity regime.

In comparison, FreezeNet does not suffer from a loss of information flow since the tensors used for the forward- and backward pass are dense. The saliency score (3.1) used to choose the trained weights is biased towards picking weights with high initial gradient values. Therefore, the mean gradient flow of the trainable weights even increases with increasing compression rate.

3.4.3 Influence of Mask Criterion and Parameter Initialization

As Figure 3.4(b) shows, randomly choosing the freezing mask lowers the performance of a model with frozen weights significantly. Therefore, choosing the trained parameters is an important factor for FreezeNet. In Wimmer et al. [192], frozen weights are chosen by the SNIP criterion. Of course, other criteria could also be used to successfully freeze parts of a model during training.

Moreover, Figure 3.6 shows that reinitializing the weights *after* the freezing mask has

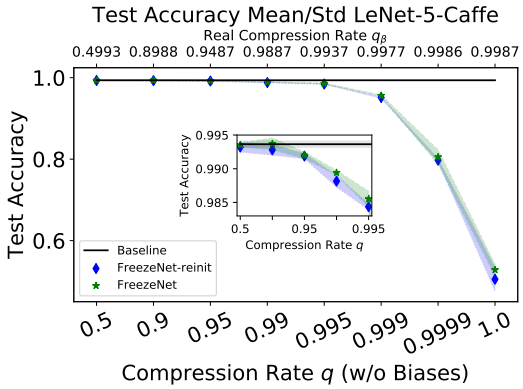


Figure 3.6: FreezeNet-reinit is a reinitialized version of FreezeNet, where the reinitialization is done *after* the freezing mask is computed. FreezeNet-reinit is compared with FreezeNet on a LeNet-5-Caffe for MNIST.

been computed does not harm performance. This reinitialization after freezing can be interpreted as randomly freezing weights with a layerwise compression rate determined by the gradient based criterion. Comparing the reinitialized model with the randomly frozen model shows that the choice of layerwise compression rates is important for the final performance.[4] Similar findings are discussed in Frankle et al. [45] for the PaI methods SNIP, GraSP and SynFlow. For them, randomly pruning the network with the same layerwise pruning rate determined by the SNIP/GraSP/SynFlow criterion leads to similar results. Therefore, Frankle et al. [45] argue that these PaI methods might not find the best sparse architecture but only a well performing layerwise sparsity distribution.

By looking at the results for reinitializing FreezeNets, these findings can be transferred to models trained with frozen parameters – at least if the frozen weights are determined at initialization by the saliency criterion (3.1).

3.4.3.1 Role of the Initialization

As discussed, the performance of FreezeNet heavily relies on the distribution of the frozen parameters. Therefore, the initialization of the model plays an important role. Figure 3.4(b) shows different initializations for the FreezeNet. All compared initializations are so-called *variance scaling* initializations [55, 64]. These initializations prevent the model

[4] Random freezing means that all weights are frozen with the same probability q which leads to a layerwise compression rate of approximately q.

from exploding and vanishing gradients by properly scaling the randomly initialized layers according to the size of their weight tensors. The tested initializations are the Xavier-normal [55], the Kaiming-uniform [64] and the $\texttt{pm}_\sigma$ initialization. The $\texttt{pm}_\sigma$ initialization initializes each weight with $\text{sgn} \cdot \sigma$ where $\text{sgn} \in \{-1, 1\}$ follows a binomial distribution with probability $1/2$ and σ^2 is the variance of the Xavier-normal initialization. Therefore, the $\texttt{pm}_\sigma$ and the Xavier-normal initialization define distributions with the same first (0 mean) and second (variance σ^2) moment. However, the $\texttt{pm}_\sigma$ initialization is more simple since the initialized – and therefore also frozen – weights only have two distinct values $\pm\sigma$ in each layer.

As shown in Figure 3.4(b), the three random initializations perform almost equivalent, where the Xavier-normal initialization shows the best results for high compression rates. However, freezing random weights with the $\texttt{pm}_\sigma$ initialization does not lead to significantly worse results. This can be useful in situations where the frozen weights can not be generated with a pseudo random number generator. For the $\texttt{pm}_\sigma$ initialization, only the sign sgn has to be saved for each frozen weight. In this case, frozen weights need only 1bit to be stored. Also, if a FreezeNet should be quantized, restricting the frozen weights to two possible values simplifies the quantization task.

3.5 Conclusions for FreezeNet

FreezeNet is able to significantly reduce the dimension of the trained parameter space while at the same time guaranteeing a high performance. FreezeNet matches SOTA pruning methods, while outperforming its pruning counterpart for high compression rates. The main advantage of freezing weights instead of setting them to zero is a guaranteed high information flow in the dimensionality reduced model. This helps to increase the capacity of the networks which leads to better models if only a few parameters are trained.

Compared to its pruning counterpart SNIP, FreezeNet reaches similar or better results while training fewer parameters. This lowers memory requirements and reduces the cost for transferring the model. For lower compression rates, however, using frozen weights might perform worse than pruning the model. In this case, smoothly shrinking the untrained parameters to zero by applying weight decay to the frozen weights helps to combine the best of both worlds: frozen weights at the beginning of training to increase

the information flow while the model is pruned at the end of training due to weight decay. For higher compression rates on the other hand, applying weight decay to the frozen weights yields inferior results since the non-zero frozen weights help to guarantee a strong information flow in the model trained with reduced dimensionality.

FreezeNet has shown to be robust w.r.t. the choice of the random initialization. Especially, a simple 1bit initialization achieves similar performance as using SOTA network initialization methods. Such an initialization helps to reduce the memory cost for the frozen weights in situations where no pseudo random number generator can be used.

Reinitializing the weights after the freezing mask has been computed shows similar results to the standard FreezeNet. This suggests that FreezeNet is able to find a well performing distribution of frozen weights in each layer but there might exist a better frozen architecture which is not invariant to a reinitialization of the weights.

Chapter 4
COPS: Controlled Pruning Before Training Starts

COPS: Controlled Pruning Before Training Starts by Wimmer et al. [193] addresses the problem that the SOTA pruning at initialization methods perform well either for low compression rates *or* high compression rates. To see this, compare SNIP and GraSP on the right hand side of Figure 4.1. PaI methods usually rely on a single pruning criterion as presented in Section 2.2.3.2. These pruning criteria have in common that they are all based on a pruning *score*. However, these criteria only work well in some situations while failing for other ones. The **CO**mbined **P**runing **S**core (COPS) overcomes this problem by combining two pruning criteria by melding the corresponding pruning scores. By doing so, COPS combines the best of the two pruning criteria and is thereby able to create more powerful pruning methods from already existing ones. The following Chapter reproduces Wimmer et al. [193] and therefore follows the structure of this paper.

4.1 Motivation for Combining Two Pruning Methods

Training sparse networks from scratch has become more and more popular in recent years [7, 43, 94, 174, 184, 193]. Training with a sparse model lowers memory requirements also during training while the forward- and backward propagation can be accelerated if the soft- and hardware supports sparse computations. However, as discussed in Section 3.4.2, training sparse models often has the problem of weak gradient signals if many parameters are pruned. By using pruning criteria which directly optimize for a high gradient flow, like GraSP [184] or SynFlow [174], the information flow can be strengthened.

The presented work by Wimmer et al. [193] focuses on one-shot pruning at initializa-

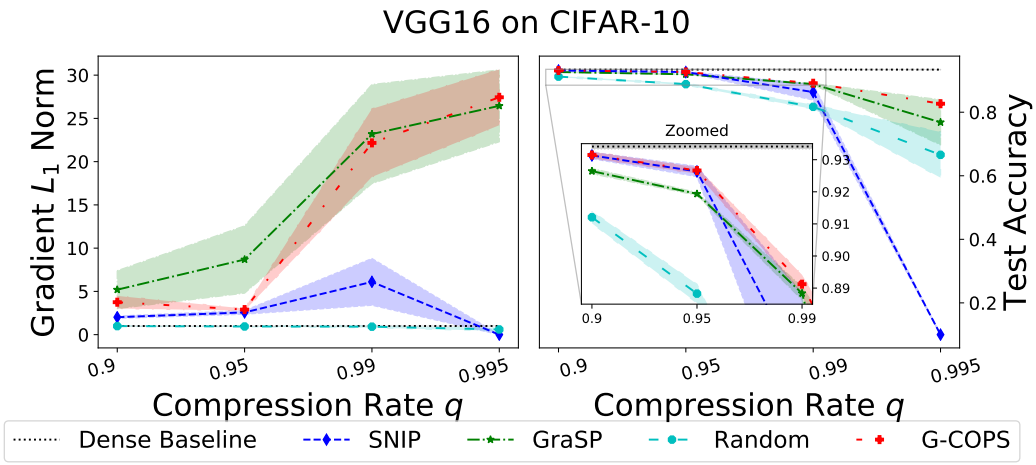


Figure 4.1: (Figure 1 in Wimmer et al. [193]) Left: Compares the normalized gradient flow for SNIP, GraSP, random PaI and G-COPS. The pruning masks are computed with 100 batches of training data, not with 1 as done in Figure 3.1. Right: Compares performance of these methods. Experiments are conducted for a VGG-16 on CIFAR-10.

tion, *i.e.* no iterative pruning methods like [27, 174, 181]. In this regime, SNIP [94] and GraSP [184] are SOTA. As shown in Figure 4.1, SNIP achieves better results than GraSP for lower compression rates whereas GraSP outperforms SNIP for high compression rates. As already discussed in Section 3.4.2, the worse results for SNIP are caused by missing information flow for high q. Again, this can be seen in Figure 4.1, left hand side. Note, in contrast to the experiments for FreezeNet in Section 3, experimental evaluations for COPS use more batches to compute the pruning score. As a consequence, the found sparse architectures become better for high q, thus SNIP can train up to higher sparsity.

GraSP directly optimizes models to have a high gradient flow. As Figure 4.1 shows, optimizing a sparse architecture only to have a strong gradient flow does not yield the best results for lower compression rates, *i.e.* in situations where information flow is *not* the bottleneck for training. For such q, SNIP shows better performance since it finds sparse models with the highest influence in improving the loss function. In summary, a higher gradient flow does not directly guarantee a better performance. However, having a gradient flow bigger than zero is a *necessary* condition for a sparse model to train well. Getting the best of these two methods, SNIP's focus on the weights which have a high influence on improving the loss together with a strong gradient flow guaranteed by GraSP, was the motivation to combine them. Thereby, a new, more powerful pruning

method applied at initialization is created which reaches good performance for both, low and high compression rates. This combination of two (or more) pruning scores is called COPS.

In Wimmer et al. [193], all tested COPS scores use SNIP as one of the two combined scores. Therefore, the combination of SNIP and GraSP is called G-COPS. In Figure 4.1, G-COPS is compared to SNIP, GraSP and random pruning at initialization. As can be seen, G-COPS shows better results than SNIP, GraSP and random PaI for both, low and high compression rates. Moreover, G-COPS is able to guarantee a strong gradient flow, also for high compression rates, while at the same time training those weights which influence the loss function the most.

In general, COPS is able to combine arbitrary pruning methods, as long as they are based on a pruning score – as the presented criteria in Section 2.2.3.2 – which is called a *generalized synaptic score* (GSS).

4.1.1 Contributions of COPS: Controlled Pruning Before Training Starts

The main contributions presented in this Chapter are given by:

- COPS combines arbitrary GSS based pruning methods.
- The combination of GSS' leads to a constrained, combinatorial optimization problem. This problem is relaxed on a linear program (LP) and solved analytically. The solution of the relaxed problem is projected to an approximate solution of the un-relaxed, combinatorial problem.
- An algorithm is proposed for solving the relaxed LP which has lower complexity than the best known general LP solver by Jiang et al. [79].
- Better sparse architectures than by the two SOTA methods SNIP and GraSP are generated by combining and balancing the corresponding pruning scores properly.
- Thereby, the proposed method for combining two pruning scores is shown to be more efficient than naively combining the two scores linearly.

4.2 Related Work for COPS

The related work for COPS [193] can be segmented in *pruning* and *linear programming*. The related pruning literature is discussed in Section 2.3.1.

4.2.1 Linear Programming

The pruning scores are combined by solving a combinatorial, constrained optimization problem. It has a non-smooth and non-convex L_0 constraint since the solution of this problem is a sparse pruning mask. Moreover, optimizing over pruning masks $\mathbf{m} \in \{0,1\}^D$ results in a combinatorial optimization problem. This constrained, combinatorial optimization problem is relaxed which results in a linear program. The dual problem of this LP is solved with the help of convex optimization methods [13]. In contrast to the standard Simplex methods [26], the solution is obtained not by walking between vertices of the resulting polytope of feasible solutions, but by using the simple and robust bisection of intervals. As a consequence, no pivoting rules for overcoming worst case scenarios are required. Another way to solve LPs fast and robustly is given by the so-called *interior point methods* [79].

4.3 Combined Pruning Scores

The main goal of Wimmer et al. [193] is to find a pruning mask that combines the best parts of two pruning methods. The found pruning mask is called COPS mask. As shown in (2.37), having a pruning mask $\mathbf{m} \in \{0,1\}^D$ is equivalent to knowing the corresponding pruning transformation $\psi^{(t)} \in \{0,1\}^{D\times d}$. Therefore, it suffices to find a fitting COPS pruning mask to embed a low dimensional parameter $\vartheta \in \mathbb{R}^d$ sparsely into $\mathbb{R}^D$.

The COPS mask is found by combining two (or more) pruning scores. The *target score* function $S_0 : \mathbb{R}^D \to \mathbb{R}$ is primarily optimized over all feasible pruning masks $\mathbf{m} \in \{0,1\}^D$ satisfying the sparsity constraint $\|\mathbf{m}\|_0 \leq d$. Moreover, the target score is controlled via the *control score* function $S_1 : \mathbb{R}^D \to \mathbb{R}$ by a constraint $S_1(\mathbf{m}) \leq \kappa$.

Note, by adding more than one control function, an arbitrary number of pruning scores can be combined while Wimmer et al. [193] focus on combining only two of them. But by using standard techniques of convex optimization, the presented results for COPS can

be generalized to more than one control score. From now on, assume only one control score function S_1 to be given.

4.3.1 Basic Assumptions and Problem Formulation for COPS

As always let f_Θ be a DNN with weights $\Theta \in \mathbb{R}^D$. Now, pruning is modeled with a pruning mask $\mathbf{m} \in \{0,1\}^D$ where the pruned weights are given by $\mathbf{m} \odot \Theta$. In contrast to the freezing mask, the pruning mask sets the weights to zero in the forward and the backward pass. Whereas the freezing mask only sets the gradients to zero during training.[5]

By looking at the presented pruning criteria in Section 2.2.3.2, they can all be described by optimizing a corresponding pruning score function under a sparsity constraint. They can be written as

$$S_{\text{Magnitude}}(\mathbf{m}) = -\langle |\Theta|, \mathbf{m} \rangle = \langle \mathbf{s}^{(\text{Magnitude})}, \mathbf{m} \rangle \tag{4.1}$$

$$S_{\text{Random}}(\mathbf{m}) = -\langle |\mathbf{z}|, \mathbf{m} \rangle = \langle \mathbf{s}^{(\text{Random})}, \mathbf{m} \rangle \ , \ \mathbf{z}_i \sim \mathcal{N}(0,1) \text{ i.i.d.} \tag{4.2}$$

$$S_{\text{SNIP}}(\mathbf{m}) = -\langle |\nabla_\Theta \mathscr{L} \odot \Theta|, \mathbf{m} \rangle = \langle \mathbf{s}^{(\text{SNIP})}, \mathbf{m} \rangle \tag{4.3}$$

$$S_{\text{GraSP}}(\mathbf{m}) = -\left\langle \left(\nabla^2_\Theta \mathscr{L} \cdot \nabla_\Theta \mathscr{L}\right) \odot \Theta, \mathbf{m} \right\rangle = \langle \mathbf{s}^{(\text{GraSP})}, \mathbf{m} \rangle \tag{4.4}$$

$$S_{\text{SynFlow}}(\mathbf{m}) = -\langle \nabla_\Theta \mathcal{R} \odot \Theta, \mathbf{m} \rangle = \langle \mathbf{s}^{(\text{SynFlow})}, \mathbf{m} \rangle \ . \tag{4.5}$$

In Section 2.2.3.2, the corresponding pruning scores are all *maximized*. To be consistent with the convex optimization literature, score functions S are *minimized* in this Chapter. Therefore, (4.1) - (4.5) all have a minus sign in front of the middle term, while the minus is assumed to be encoded in the corresponding *score vectors* $\mathbf{s} \in \mathbb{R}^D$ on the right hand side of the equations. As can be seen, all score functions $S : \mathbb{R}^D \to \mathbb{R}$ are *linear*, *i.e.* they can be evaluated by a scalar product with a corresponding pruning score vector $\mathbf{s} \in \mathbb{R}^D$ to generate the pruning score of a mask $\mathbf{m} \in \{0,1\}^D \subset \mathbb{R}^D$. Such pruning scores are called *generalized synaptic scores*.

From now on, it is assumed that the pruning criterion is based on the evaluation of a GSS. Moreover, assume the pruning score vector $\mathbf{s}$ to be known. In this case,

[5] Which also holds true for the experimental evaluation of COPS since it is designed for pruning at initialization methods which never update the pruned weights, *i.e.* the pruned weights are frozen at zero.

the dependency of $\mathbf{s}$ on Θ, $\mathscr{L}$ or other factors can be ignored and only the evaluation $\mathbf{m} \mapsto \langle \mathbf{s}, \mathbf{m} \rangle$ will be analyzed.

Note, for a GSS S it holds by linearity

$$S(\mathbf{m}) = \langle \mathbf{s}, \mathbf{m} \rangle = \sum_{i=1}^{D} \mathbf{s}_i \cdot \mathbf{m}_i \, . \tag{4.6}$$

If Θ_i is active, *i.e.* $\mathbf{m}_i = 1$, the corresponding score value $\mathbf{s}_i$ can be seen as the contribution of Θ_i to the overall score $S(\mathbf{m})$. Since score functions are minimized, pruning a weight with high score value $\mathbf{s}_i$ is assumed to lead to better results than pruning a weight with low score value, see also Section 2.2.3.2 for an interpretation of this assumption.

4.3.1.1 Pruning with a Single Score Function

For pruning with only one pruning score function S_0 and no constraint, the following optimization problem has to be solved

$$\min_{\mathbf{m} \in \mathcal{V}_{0,d}} S_0(\mathbf{m}) \tag{4.7}$$

where

$$\mathcal{V}_{0,d} := \{\mathbf{m} \in \{0,1\}^D : \|\mathbf{m}\|_0 \leq d\} \, . \tag{4.8}$$

For a criterion with a GSS function S_0 and corresponding score vector $\mathbf{s}^{(0)}$, (4.7) is solved by $\mathbf{m}^\dagger \in \{0,1\}^D$ with

$$\operatorname{supp} \mathbf{m}^\dagger \in d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} : i \in \{1,\ldots,D\}, \mathbf{s}_i^{(0)} < 0\} \, . \tag{4.9}$$

Note, $\mathbf{a} \in \{0,1\}^D$ is uniquely defined by its support $\operatorname{supp} \mathbf{a}$ since all entries which are not zero must be equal to one. For a finite index set $\mathcal{I}$, $\{\alpha_i : i \in \mathcal{I}\} \subset \mathbb{R}$ and $k \in \mathbb{N}$, $k - \operatorname{idx}\min\{\alpha_i\}$ is defined as

$$k - \operatorname{idx}\min\{\alpha_i\} := \{\mathcal{J} \subset \mathcal{I} : \#\mathcal{J} = \min\{k, \#\mathcal{I}\} \text{ and } \alpha_j \leq \alpha_i \, \forall j \in \mathcal{J}, i \in \mathcal{I} \setminus \mathcal{J}\} \, , \tag{4.10}$$

i.e. a set consisting of all subsets $\mathcal{J} \subset \mathcal{I}$ with $\#\mathcal{J} = k$ indices where the corresponding $\{\alpha_j : j \in \mathcal{J}\}$ have the k smallest entries. Note, $k - \operatorname{idx}\min\{\alpha_i\}$ is unique if $k = 0$, $k \geq \#\mathcal{I}$

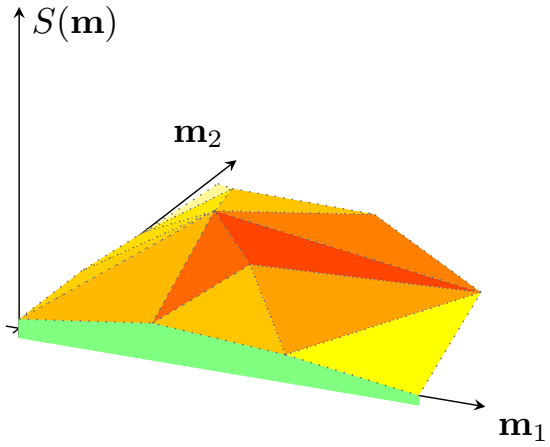


Figure 4.2: Shows a simplex defined by the graph of a convex, piecewise linear function $S : \mathbb{R}^2 \to \mathbb{R}$. Note, for better visibility, the function S is maximized, not minimized.

or $\alpha_{i_k} < \alpha_{i_{k+1}}$ where i_j sorts α_i such that $\alpha_{i_1} \leq \ldots \leq \alpha_{i_{\#\mathcal{I}}}$.

4.3.1.2 Pruning with an Additional Control Score Function

However, pruning a model w.r.t. only one GSS does not yield the best results in all situations, see Figure 4.1 right. If the weakness of the pruning criterion is known, like the weak gradient flow for models pruned with the SNIP criterion, this weakness can be controlled by constraining it. By interpreting pruning as an optimization problem (4.7), this yields a constraint $\kappa \in \mathbb{R}$ on a control score function $S_1 : \mathbb{R}^D \to \mathbb{R}$. Combining a target score function S_0 and a control score function S_1 with constraint κ finally results in the optimization problem

$$\min_{\mathbf{m} \in \mathcal{V}_{0,d}} S_0(\mathbf{m}) \text{ , such that } S_1(\mathbf{m}) \leq \kappa \,. \tag{4.11}$$

The set $\mathcal{V}_{0,d}$ of all masks satisfying the sparsity constraint is discrete. Therefore, (4.11) is a combinatorial, constrained optimization problem. In this form, the solution is expensive to compute since the problem can not be solved by linear programming or other efficient convex optimization methods.

4.3.2 Relaxing the Combinatorial Problem

A standard technique used for solving computationally expensive non-convex optimization problems is to first solve a relaxation of the problem and afterwards project the relaxed solution on an approximate solution of the original problem [13, 21, 34].

The sparsity constraint $\|\mathbf{m}\|_0 \leq d$ is highly non-convex. To find analytical or numerical solutions for problems with a L_0 constraint, it is therefore common to relax the constraint to a convex one, like constraining the L_1 norm of the solution [13, 21, 34]. Inspired by that, the sparsity constraint is relaxed to $\|\mathbf{m}\|_1 \leq d$. If the feasible solutions are restricted to fulfill $\mathbf{m}_i \geq 0$ for all i, this even yields a linear constraint

$$\|\mathbf{m}\|_1 = \langle \mathbf{1}, \mathbf{m} \rangle \leq d \; . \tag{4.12}$$

If $\mathbf{m}_i \geq 0$ for all i and both pruning score functions are GSS', the target function and all constraints will be linear. Pruning masks $\mathbf{m} \in \{0,1\}^D$ define vertices of a D dimensional hypercube. For LPs, the solutions are always given by a vertex of the simplex forming all feasible solutions [26]. Therefore, the feasible points are restricted to lie anywhere on the D dimensional hypercube $[0,1]^D$. It will be shown that a solution to the relaxed problem can be generated by a convex combination of two vertices $\mathbf{m}^-, \mathbf{m}^+ \in \mathcal{V}_{0,d}$ which have $D-2$ similar entries. The solution of the relaxed problem can therefore be projected on either $\mathbf{m}^-$ or $\mathbf{m}^+$ to obtain an approximate solution to (4.11).

Using these motivations, the relaxed version of the optimization problem (4.11) is given as the LP

$$\min_{\mathbf{m} \in \mathcal{V}_{1,d}} S_0(\mathbf{m}) \text{ , such that } S_1(\mathbf{m}) \leq \kappa \tag{4.13}$$

with

$$\mathcal{V}_{1,d} := \{\mathbf{m} \in [0,1]^D : \|\mathbf{m}\|_1 \leq d\} \; . \tag{4.14}$$

A graphical example of a LP is given in Figure 4.2.

4.3.3 Projecting the Relaxed Solution on a COPS Mask

Section 4.3.2 motivates the relaxation from the combinatorial optimization problem (4.11) to the LP (4.13). As outlined, solving the relaxed problem should yield an approximate solution for the original problem. Analytically, (4.13) is solved with the help of its dual

problem. The corresponding solution of the primal, relaxed problem is then projected on an approximate solution of the un-relaxed discrete optimization problem. Furthermore, error bounds for the approximate solution are determined.

4.3.3.1 Dual Function and Dual Problem

For the dual problem, the Lagrangian of (4.13) has to be defined. It is given by [13]

$$\Lambda(\mathbf{m},\lambda) := S_0(\mathbf{m}) + \lambda \cdot (S_1(\mathbf{m}) - \kappa) \tag{4.15}$$

where $\lambda \in [0,\infty)$. The Lagrangian (4.15) can now be used to define the *dual function* w.r.t. (4.13) [13]

$$g : [0,\infty) \to \mathbb{R}, \lambda \mapsto \inf_{\mathbf{m}\in\mathcal{V}_{1,d}} \Lambda(\mathbf{m},\lambda) \; . \tag{4.16}$$

Note, $g(\lambda) = -\infty$ is possible if $\Lambda(\cdot,\lambda)$ is not bounded from below. The dual function g can be used to solve the dual problem of (4.13). The dual problem is given by

$$\max_{\lambda\geq 0} g(\lambda) \; . \tag{4.17}$$

To solve the dual problem, first the dual function g is determined in Lemma 4.3.1. A graphical explanation for the dual function is given in the left part of Figure 4.3.

Lemma 4.3.1. *For every $\lambda \geq 0$, the dual function $g(\lambda)$ is given by*

$$g(\lambda) = \sum_{i\in\mathcal{I}_\lambda} \mathbf{s}_i^{(0)} + \lambda \cdot \mathbf{s}_i^{(1)} - \lambda\kappa = S_0(\mathbf{m}_\lambda) + \lambda \cdot (S_1(\mathbf{m}_\lambda) - \kappa) \tag{4.18}$$

with $\mathcal{I}_\lambda \in d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda \cdot \mathbf{s}_i^{(1)} : i \in \{1,\ldots,D\}, \mathbf{s}_i^{(0)} + \lambda \cdot \mathbf{s}_i^{(1)} < 0\}$, $\mathbf{m}_\lambda \in \mathcal{V}_{0,d}$ with $\operatorname{supp}\mathbf{m}_\lambda = \mathcal{I}_\lambda$.

Proof of Lemma 4.3.1 is given in Appendix, Section B.1.1. ■

The solution of the COPS problem (4.11) is obtained by optimizing the dual function. Lemma 4.3.1 shows that g is a piecewise linear function, *i.e.* differentiable almost everywhere, which is shown to be concave in Lemma 4.3.2. Its derivatives can be generalized to *superderivatives* which are defined everywhere for concave functions, see

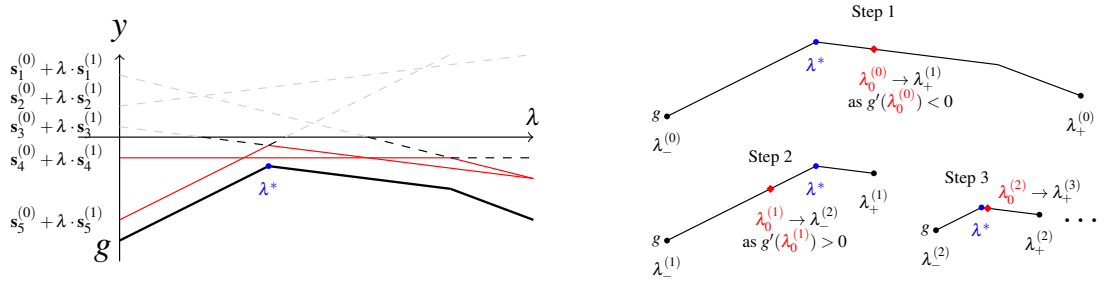


Figure 4.3: (Figure 3 in Wimmer et al. [193]) Left: Graphical example of g and λ^* with sparsity constraint $d = 2$ and underlying affine functions $s_i^{(0)} + \lambda \cdot s_i^{(1)}$. For clearness, $\kappa = 0$ is used. Affine parts with active $i \in \mathcal{I}_\lambda$ are colored red. Gray coloring of $s_i^{(0)} + \lambda \cdot s_i^{(1)}$ for a given λ highlights $s_i^{(0)} + \lambda \cdot s_i^{(1)} \geq 0$. Right: Shows first steps of Algorithm 4.1 to obtain a solution λ^*.

Definition 4.3.1. Superderivatives help to define a necessary condition for an optimum of g in Corollary 4.3.1.

Definition 4.3.1 (Superderivative)**.** A *superderivative* of a function $h : [0,\infty) \to \mathbb{R}$ at $x_0 \in [0,\infty)$ is a $\xi \in \mathbb{R}$ such that

$$h(x) - h(x_0) \leq \xi \cdot (x - x_0) \tag{4.19}$$

holds for all $x \in [0,\infty)$.

The definition 4.3.1 is analogous to the definition of *subderivatives* for convex functions [155]. Analogue to convex functions [155], concave functions h have a non-empty set of superderivatives $\hat{\partial} h(x_0) := \{\xi : \xi \text{ is a superderivative of } h \text{ at } x_0\}$ for each $x_0 \in [0,\infty)$. Moreover, $\hat{\partial} h(x_0) = \{h'(x_0)\}$ if and only if h is differentiable in x_0. With a slight abuse of notation, $h'(x_0)$ denotes both, the standard derivative and a single element of the superderivative of a function h in x_0 in this Section. All superderivatives $h'(x_0)$ of h in x_0 are characterized by [13]

$$\xi_+(x_0) := \lim_{t \searrow 0} \frac{h(x_0+t) - h(x_0)}{t} \leq h'(x_0) \leq \lim_{t \nearrow 0} \frac{h(x_0+t) - h(x_0)}{t} =: \xi_-(x_0)\,. \tag{4.20}$$

Here, $\xi_+(x_0)$ and $\xi_-(x_0)$ are the right and left derivative of h in x_0 which are both well-defined for all x_0 and monotonic decreasing functions in x_0. Superderivatives and their

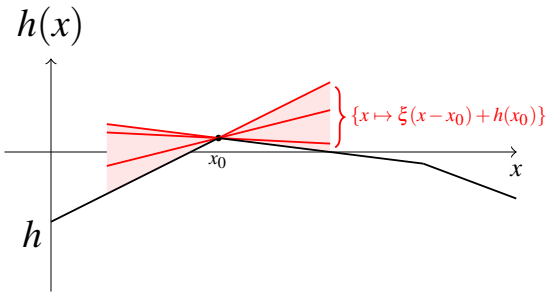


Figure 4.4: Graphical example for superderivatives of a concave, piecewise linear $h : [0,\infty) \to \mathbb{R}$ at a non-differentiable point x_0. The set of superderivatives $\hat{\partial} h(x_0)$ is shaded red. For a concave function, $\xi_+(x_0) \leq \xi \leq \xi_-(x_0)$ defines all superderivatives ξ at x_0.

relationship to left and right derivatives are shown in Figure 4.4.

Lemma 4.3.2. *The dual function $g : [0,\infty) \to \mathbb{R}$ defined by* (4.18) *is concave. A superderivative of g at $\lambda \geq 0$ is defined by*

$$g'(\lambda) = \sum_{i \in \mathcal{I}_\lambda} \mathbf{s}_i^{(1)} - \kappa \,. \tag{4.21}$$

Proof of Lemma 4.3.2 is given in Appendix, Section B.1.2. ■

Note, the superderivative of g is not unique for λ where $\mathcal{I}_\lambda$ changes, *i.e.* $\mathcal{I}_\lambda \neq \mathcal{I}_{\lambda + t \cdot \mathrm{sgn}}$ for all $t > 0$ small enough and a $\mathrm{sgn} \in \{\pm 1\}$.

Corollary 4.3.1. *Let $\xi \in \mathbb{R}$ be a superderivative of the dual function g in $\lambda_0 \geq 0$. Then it holds*

- $\xi = 0 \Rightarrow \lambda_0$ *maximizes g,*
- $\xi > 0 \Rightarrow \lambda^* \geq \lambda_0$ *for all λ^* which maximize g,*
- $\xi < 0 \Rightarrow \lambda^* \leq \lambda_0$ *for all λ^* which maximize g.*

Proof of Corollary 4.3.1 is given in Appendix, Section B.1.3. ■

4.3.3.2 Using Strong Duality to Find the COPS Mask

By Slater's condition [13], strong duality holds for convex problems *if* there exists a strictly feasible point $\mathbf{m}^\circ$. A strictly feasible point is an interior point that satisfies all

constraints with strict inequality. Since (4.13) is linear, it is also convex. For the problem (4.13), a strictly feasible point $\mathbf{m}^\circ$ therefore must satisfy

$$\mathbf{m}^\circ \in (0,1)^D \ , \ \|\mathbf{m}^\circ\|_1 < d \text{ and } S_1(\mathbf{m}^\circ) < \kappa \tag{4.22}$$

to fulfill Slater's condition. If Slater's condition holds, strong duality guarantees [13]

$$p^* = d^* \tag{4.23}$$

where p^* is the minimum of the primal problem (4.13) and d^* is the optimum of the dual problem (4.17).

Lemma 4.3.3. *Let* $\mathbf{m}^\circ$ *be a strictly feasible point,* i.e. $\mathbf{m}^\circ$ *satisfies* (4.22)*. Then there exists a* $\lambda^* \geq 0$ *which solves the dual problem* (4.17)*. Furthermore, there exists* $\mathbf{m}^* \in \mathcal{V}_{1,d}$ *solving the primal problem* (4.13)*, given by a convex combination of* $\mathbf{m}^-, \mathbf{m}^+ \in \mathcal{V}_{0,d}$ *with* $\operatorname{supp}\mathbf{m}^-, \operatorname{supp}\mathbf{m}^+ \in d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} : i \in \{1,\dots,D\}, \mathbf{s}_i^{(0)} + \lambda \cdot \mathbf{s}_i^{(1)} < 0\}$, i.e.

$$\mathbf{m}^* = \gamma \cdot \mathbf{m}^- + (1-\gamma) \cdot \mathbf{m}^+ \tag{4.24}$$

with $\gamma \in [0,1]$.

Proof of Lemma 4.3.3 is given in Appendix, Section B.1.4. ■

By looking at the proof of Lemma 4.3.3, it is necessary that $d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\}$ has two distinct elements if $\lambda^* > 0$ and g not differentiable in λ^*. In this case, $\mathbf{m}^-$ corresponds to the linear part of g left of λ^* with positive slope while $\mathbf{m}^+$ generates the right part with negative slope. Graphically, this is shown in the left part of Figure 4.3.

If the conditions for Lemma 4.3.3 are fulfilled, first the solution $\lambda^* \geq 0$ of the dual problem needs to be found. Then, an approximate solution of the un-relaxed problem (4.11) is given by a $\mathbf{m} \in \mathcal{V}_{0,d}$ with $\operatorname{supp}\mathbf{m} \in d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\}$. Determining $d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\}$ can be done by sorting all $\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)}$ which requires $O(D \cdot \log D)$ steps.

Theorem 4.3.1. *Let* $\kappa > \min_{\mathbf{m} \in \mathcal{V}_{0,d}} S_1(\mathbf{m})$*. Then, there exists a* $\lambda^* \geq 0$ *which solves the dual problem* (4.17)*, a* $\mathbf{m}^* \in \mathcal{V}_{1,d}$ *that solves the relaxed, primal problem* (4.13) *and a* $\bar{\mathbf{m}} \in \mathcal{V}_{0,d}$ *solving the un-relaxed discrete problem* (4.11)*.*

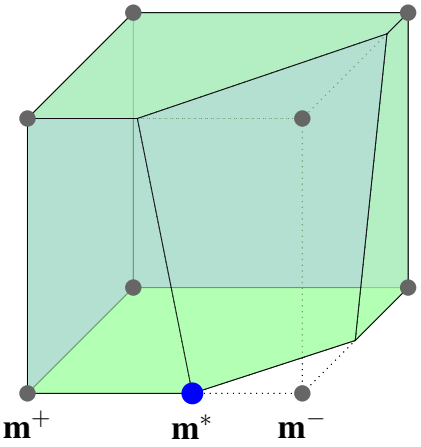


Figure 4.5: The hypercube where all points $\mathbf{m} \in \mathcal{V}_{0,d}$ lie on. The green part shows the feasible area. The solution of the relaxed problem $\mathbf{m}^*$ lies on the edge between $\mathbf{m}^-$ and $\mathbf{m}^+$. Note, $\mathbf{m}^+$ is in the feasible domain while $\mathbf{m}^-$ slightly violates the constraint.

If $\lambda^* = 0$, *all* $\mathbf{m} \in d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} < 0\}$ *solve the relaxed, primal problem* (4.13) *and the un-relaxed discrete problem* (4.11).

For $\lambda^* > 0$ *there exist* $\mathbf{m}^-, \mathbf{m}^+ \in d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\} \subset \mathcal{V}_{0,d}$ *with* $S_1(\mathbf{m}^+) \leq \kappa \leq S_1(\mathbf{m}^-)$,

$$S_0(\mathbf{m}^-) \leq S_0(\mathbf{m}^*) \leq S_0(\bar{\mathbf{m}}) \leq S_0(\mathbf{m}^+) \tag{4.25}$$

and further

$$S_0(\mathbf{m}^+) - S_0(\mathbf{m}^-) \leq \mathbf{s}_{i_+}^{(0)} - \mathbf{s}_{i_-}^{(0)} \text{ and } S_1(\mathbf{m}^-) - S_1(\mathbf{m}^+) \leq \mathbf{s}_{i_-}^{(1)} - \mathbf{s}_{i_+}^{(1)}\,, \tag{4.26}$$

where $\operatorname{supp}\mathbf{m}^+ \setminus \operatorname{supp}\mathbf{m}^- = \{i_+\}$ *and* $\operatorname{supp}\mathbf{m}^- \setminus \operatorname{supp}\mathbf{m}^+ = \{i_-\}$.

Proof of Theorem 4.3.1 is given in Appendix, Section B.1.5. ■

If there exists a $\mathbf{m} \in d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\}$ with $S_1(\mathbf{m}) = \kappa$, the proof of Theorem 4.3.1 guarantees that $\mathbf{m} \in \mathcal{V}_{0,d}$ solves both, the relaxed and un-relaxed problem by setting $\mathbf{m}^- = \mathbf{m}^+ = \mathbf{m}$.

In general, an approximation of the true solution $\bar{\mathbf{m}}$ of the discrete problem (4.11) can be chosen as the feasible point $\mathbf{m}^+ \in \mathcal{V}_{0,d}$ which has a slightly worse target score than the optimum *or* by the point $\mathbf{m}^- \in \mathcal{V}_{0,d}$ which even has a better target score than the

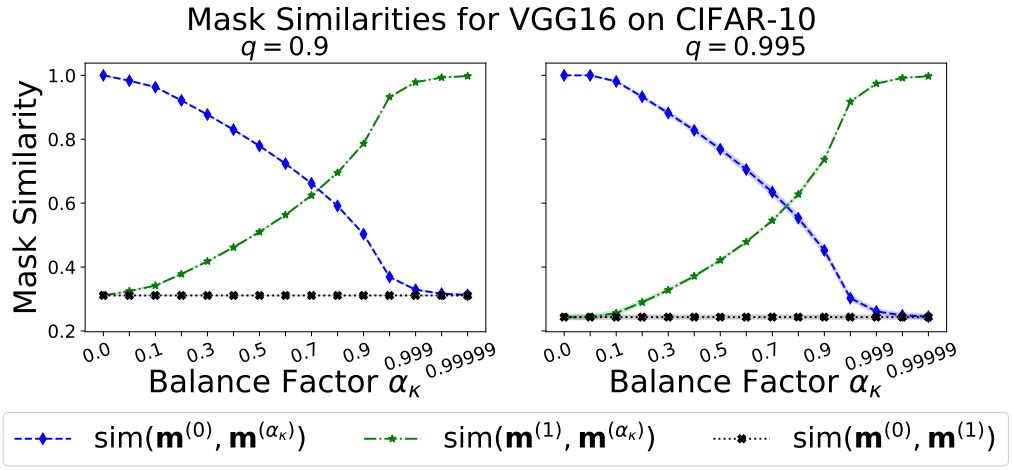


Figure 4.6: (Figure 2 in Wimmer et al. [193]) Mask similarity between SNIP ($\mathbf{m}^{(0)}$), GraSP ($\mathbf{m}^{(1)}$) and G-COPS ($\mathbf{m}^{(\alpha_K)}$), where the G-COPS mask is computed for different balance factors $\alpha_K \in [0,1)$, tested for a VGG-16 on CIFAR-10.

optimum but slightly violates the constraint of the optimization problem. A graphical explanation of the relationship between $\mathbf{m}^-$, $\mathbf{m}^+$ and $\mathbf{m}^*$ is shown in Figure 4.5.

Assuming the target and control score vectors to be sampled i.i.d. from a continuous distribution, the probability of $d - \mathrm{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\}$ having more than two elements is equal to zero. Moreover, $d - \mathrm{idx}\min\{\mathbf{s}_i^{(0)} + \lambda \cdot \mathbf{s}_i^{(1)} < 0\}$ has with probability 0 more than one element for a randomly chosen $\lambda \geq 0$. Therefore, Algorithm 4.1 will choose an arbitrary $\mathbf{m} \in d - \mathrm{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\}$ since λ^* itself is only determined up to a certain numerical precision and the corresponding $d - \mathrm{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\}$ has almost surely only one element.

4.3.4 Effect of Control on the COPS Mask

Theorem 4.3.1 requires $\kappa > \min_{\mathbf{m} \in \mathcal{V}_{0,d}} S_1(\mathbf{m})$ to be fulfilled for guaranteeing a projected solution to the discrete problem (4.11). For the experiments in Section 4.4, a constraint $\kappa > 0$ has shown to be fulfilled trivially by the solution of the unconstrained problem. Therefore, $\kappa \leq 0$ is assumed in the following.

Now,

$$\min_{\mathbf{m} \in \mathcal{V}_{0,d}} S_1(\mathbf{m}) = S_1(\mathbf{m}^\dagger) =: \kappa_{\min} \,, \tag{4.27}$$

where $\operatorname{supp}\mathbf{m}^\dagger \in d - \operatorname{idx}\min\{\mathbf{s}_i^{(1)} < 0\}$ can be computed by sorting the entries $\mathbf{s}_i^{(1)}$ of the control score vector $\mathbf{s}_i^{(1)}$. Since $\kappa_{\min}$ is the minimal possible constraint on the control score, setting a constraint $\kappa < \kappa_{\min}$ will yield no feasible solution. On the other hand, Slater's condition is not fulfilled with $\kappa = \kappa_{\min}$. However, if $\kappa = \kappa_{\min}$ is desired, the unconstrained pruning problem (4.7) w.r.t. the control score function S_1 can be solved (*i.e.* pruning with only one score).

Therefore, $\kappa > \kappa_{\min}$ does not restrict the solution space and is a reasonable assumption. If κ is chosen close to $\kappa_{\min}$, the constraint is tight. Therefore, the solution to (4.11) will be restricted towards fulfilling the constraint. As a consequence, the solution will be biased towards optimizing the control score function S_1. For $\kappa \gg \kappa_{\min}$, the control is loosened. This allows the solution more freedom to optimize the target score function S_0.

Knowing the minimal satisfiable constraint $\kappa_{\min}$ gives the opportunity to define a more descriptive parameter for setting the constraint κ, namely the *balance factor* α_κ. For $\alpha_\kappa \in [0,1)$, the corresponding constraint is defined as

$$\kappa = \alpha_\kappa \cdot \kappa_{\min} \,. \tag{4.28}$$

For $\alpha_\kappa = 0$, the constraint is 0 and therefore inactive. For $\alpha_\kappa \to 1$ on the other hand, $\kappa \to \kappa_{\min}$ which tightens the constraint. The balance factor α_κ therefore is an interpretable measure for the tightness of the constraint for arbitrary control- and target scores, datasets and network architectures.

To confirm that $\alpha_\kappa = 0$ corresponds to the unconstrained problem while $\alpha_\kappa \to 1$ yields optimizing mainly the control score, a similarity metric for different pruning masks is required. In this way, the distance between masks obtained by different optimization problems can be measured. Let $\mathbf{m}, \tilde{\mathbf{m}} \in \{0,1\}^D$ be two pruning masks. Their similarity is defined as

$$\operatorname{sim}(\mathbf{m}, \tilde{\mathbf{m}}) := \frac{\|\mathbf{m} \odot \tilde{\mathbf{m}}\|_0}{\max\{\|\mathbf{m}\|_0, \|\tilde{\mathbf{m}}\|_0\}} \in [0,1] \,. \tag{4.29}$$

Two equal pruning masks have a similarity of 1 while masks with a completely distinct support have a similarity of 0.

To check the effect of the balance factor α_κ on the resulting COPS mask, the similarity between the COPS mask and the underlying pruning masks is shown in Figure 4.6. Here, SNIP is used as target score and GraSP as control score which are combined to form the

G-COPS mask. The G-COPS mask $\mathbf{m}^{(\alpha_\kappa)}$ corresponds to the solution of problem (4.11) with balance factor α_κ. Therefore, the SNIP mask is denoted as $\mathbf{m}^{(0)}$ (no constraint, *i.e.* $\alpha_\kappa = 0$) while the GraSP mask is $\mathbf{m}^{(1)}$ (strongest constraint $\kappa_{\min}$, *i.e.* $\alpha_\kappa = 1$). As suspected, α_κ indeed balances the G-COPS mask to lie between the SNIP and GraSP mask. For $\alpha_\kappa \to 0$, G-COPS equals the SNIP mask, while the G-COPS mask reaches the GraSP mask for tight constraints $\alpha_\kappa \to 1$. The similarity between G-COPS and GraSP monotonically increases with an increasing balance factor. On the other hand, $\text{sim}\left(\mathbf{m}^{(0)}, \mathbf{m}^{(\alpha_\kappa)}\right)$ decreases monotonically for growing α_κ. Therefore, the COPS mask can be seen by an interpolation between the target pruning mask and the control pruning mask where the balance between these two masks is controlled by α_κ.

Moreover, all tested pruning masks are never completely distinct as $\text{sim}\left(\mathbf{m}^{(0)}, \mathbf{m}^{(1)}\right) > 0.2$ for both tested compression rates q. However, the similarity between SNIP and GraSP decreases with higher compression rates as can be seen in Figure 4.6 by a lower $\text{sim}\left(\mathbf{m}^{(0)}, \mathbf{m}^{(1)}\right)$ for $q = 0.995$ than for $q = 0.9$. Therefore, the most important weights for SNIP are not the most important ones for GraSP and vice-versa. This also explains SNIP's loss of information flow for high compression rates, since weights which guarantee a strong gradient flow – the ones which are kept by GraSP – are pruned by SNIP.

4.3.5 Algorithm to Compute COPS

By (4.18) and (4.21), the dual function and a superderivative of g can be evaluated by knowing $\mathcal{I}_\lambda$. Moreover, Corollary 4.3.1 yields a criterion for localizing an optimum λ^* via a bisection of intervals. For this, a lower bound $\lambda_- = 0$ and an upper bound $\lambda_+ > 0$ with $g'(\lambda_+) \leq 0$ are required. If $g'(\lambda_-) \leq 0$, the lower bound $\lambda_- = 0$ optimizes the dual function. Otherwise, evaluate g' at a point $\lambda_0 \in (\lambda_-, \lambda_+)$, for example $\lambda_0 = \frac{\lambda_- + \lambda_+}{2}$. If $g'(\lambda_0) = 0$, of course an optimum is found. For $g'(\lambda_0) < 0$, g has a maximum in $[\lambda_-, \lambda_0]$ whereas it lies in $[\lambda_0, \lambda_+]$ if $g'(\lambda_0) > 0$. Iteratively, this procedure can be continued until λ_0 lies close enough to an actual optimizer λ^* of the dual function, *i.e.* $|\lambda_+ - \lambda_-| \leq 2 \cdot \varepsilon_n$ for the desired *numerical precision* $\varepsilon_n > 0$. This procedure is summarized in Algorithm 4.1. The first three steps of Algorithm 4.1 are visualized in the right side of Figure 4.3.

Algorithm 4.1: COPS

Input: Sparsity constraint $d \in \{1, \dots, D\}$,
Balance Factor $\alpha_\kappa \in [0, 1)$,
Target score vector $\mathbf{s}^{(0)} \in \mathbb{R}^D$,
Control score vector $\mathbf{s}^{(1)} \in \mathbb{R}^D$,
Numerical precision $\varepsilon_n > 0$,
Upper bound $\lambda_+ > 0$ with $g'(\lambda_+) \leq 0$

Output: Optimal mask $\bar{\mathbf{m}} \in \{0,1\}^D$

Set constraint $\kappa = \alpha_\kappa \cdot \kappa_{\min}$ // $\kappa_{\min}$ according to (4.27)
Calculate $\mathcal{I}_0$ as in Lemma 4.3.1 // $\mathcal{I}_0$ yields $g'(0)$ via (4.21)
if $g'(0) \leq 0$ **then** // $g \searrow$ for $\lambda \geq 0$
  $\lambda^* \leftarrow 0$
else // $\lambda^* \geq 0$, $\lambda_- = 0$ is lower bound
  Calculate $\mathcal{I}_{\lambda_+}$ // $\mathcal{I}_{\lambda_+}$ gives $g'(\lambda_+)$
  $\lambda_-^{(0)} \leftarrow 0$, $\lambda_+^{(0)} \leftarrow \lambda_+$, $k \leftarrow 0$ // k counts steps
  repeat // Bisection
    $\lambda_0^{(k)} \leftarrow \frac{\lambda_-^{(k)} + \lambda_+^{(k)}}{2}$ // Midpoint $\lambda_0^{(k)}$
    Calculate $\mathcal{I}_{\lambda_0^{(k)}}$ // $\mathcal{I}_{\lambda_0^{(k)}}$ yields $g'(\lambda_0^{(k)})$
    if $g'(\lambda_0^{(k)}) = 0$ **then** // $g(\lambda_0^{(k)})$ is optimal
      $\lambda^* = \lambda_0^{(k)}$
      break
    else if $g'(\lambda_0^{(k)}) > 0$ **then** // $\lambda^* \geq \lambda_0^{(k)}$
      $\lambda_-^{(k+1)} \leftarrow \lambda_0^{(k)}$, $\lambda_+^{(k+1)} \leftarrow \lambda_+^{(k)}$
    else // $\lambda^* \leq \lambda_0^{(k)}$
      $\lambda_+^{(k+1)} \leftarrow \lambda_0^{(k)}$, $\lambda_-^{(k+1)} \leftarrow \lambda_-^{(k)}$
    $k \leftarrow k + 1$ // Old $k+1$ now k
  until $\lambda_+^{(k)} - \lambda_0^{(k)} \leq \varepsilon_n$ // Up to precision ε_n
  $\lambda^* \leftarrow \lambda_0^{(k-1)}$
Set optimal mask $\bar{\mathbf{m}} \in \{0,1\}^D$ with $\bar{\mathbf{m}}_i = 1$ if $i \in \mathcal{I}_{\lambda^*}$ else $\bar{\mathbf{m}}_i = 0$

4.3.5.1 Convergence Speed and Complexity of Algorithm 4.1

Since Algorithm 4.1 is based on the bisection method, it has linear order of convergence. Especially, it holds

$$|\lambda_0^{(k)} - \lambda^*| \leq \lambda_+^{(k)} - \lambda_0^{(k)} = 2^{-(k+1)} \cdot (\lambda_+ - \lambda_-) \,. \tag{4.30}$$

Since g is piecewise linear, it is Lipschitz continuous with Lipschitz constant $L_g > 0$. Therefore, the distance from $g(\lambda_0^{(k)})$ to the real optimum d^* can be bounded by

$$|g(\lambda_0^{(k)}) - g(\lambda^*)| \leq L_g \cdot |\lambda_0^{(k)} - \lambda^*| \leq L_g \cdot 2^{-(k+1)} \cdot (\lambda_+ - \lambda_-) \,. \tag{4.31}$$

If desired, the numerical precision in Algorithm 4.1 can be lifted from bounding the maximal distance between $(\lambda_0^{(k)}, \lambda^*)$ to bounding the distance between $(g(\lambda_0^{(k)}), d^*)$ by using (4.31).

Furthermore, a superderivative in λ_0 is computed by sorting the scores $\{\mathbf{s}_i^{(0)} + \lambda_0 \cdot \mathbf{s}_i^{(1)}\}$ which requires $O(D \cdot \log D)$ steps. By (4.30), the number of bisections k must be $O(\log 1/\varepsilon_n)$ to guarantee precision ε_n. Altogether, Algorithm 4.1 needs $O(D \cdot \log D \cdot \log 1/\varepsilon_n)$ steps to converge deterministically. The convergence rate depends on the size of the network D, the numerical precision $\varepsilon_n > 0$ but not on the chosen sparsity constraint $1 \leq d \leq D$.

The *faster dynamic matrix inverses* (FDMI) method [79] shows the best *expected* runtime for solving *general* LPs in $O\left(D^{2.055} \log D/\varepsilon_n\right)$. Since $D^{2.055} \gg D \cdot \log D$ and λ^* immediately determines a projected solution of the un-relaxed discrete problem (4.11), the COPS Algorithm 4.1 is much faster in finding the COPS mask than using a standard LP solver like FDMI. FDMI is slower than Algorithm 4.1 since it needs to invert matrices which is expensive while Algorithm 4.1 only sorts values of a vector which is cheaper. Of course, the proposed method to solve COPS is specialized on solving problems like (4.13) and projecting the solution on $\mathcal{V}_{0,d}$, and not on solving general LPs.

4.4 Experimental Results and Discussion for COPS

In this Section, experimental results for COPS are proposed and discussed. Furthermore, the stability of COPS, the efficiency of the balance factor α_κ as hyperparameter and

the difference between solutions found by COPS and the underlying pruning scores are analyzed.

Similar to FreezeNet, only weights are pruned while bias and batch normalization parameters are kept trainable. The training setups are chosen from the literature for other pruning at initialization methods. As for FreezeNet, five training runs are evaluated for results on MNIST and CIFAR while three runs are used for Tiny ImageNet results. The training setups with used hyperparameters are given in Table A.10 in the Appendix, Section A.4.

In contrast to the FreezeNet experiments, 100 batches of training data are used to compute the pruning scores $\mathbf{s}$. Using more than 1 batch yields a better estimation of the gradient over the whole dataset and therefore improves results – especially for high pruning rates.

Pruning Methods and Combined Pruning Scores. The evaluation focuses on one-shot pruning at initialization methods since they have the biggest problem with missing information flow for high pruning rates [174, 184, 192]. For this, the SOTA methods SNIP [94] and GraSP [184] together with random PaI and magnitude based PaI are used. Moreover, a combination of SNIP as target and GraSP as control score, G-COPS, and a combination of SNIP as target score and magnitudes as control score, M-COPS, are compared with these methods. Also, an unsupervised version of SNIP, USNIP [95], is used to further show the advantages of using more than one pruning criterion. Here, USNIP is combined with SNIP, where SNIP acts as target score and USNIP as control score. By controlling the pruning mask with an unsupervised score, the influence of missing or wrong labels in the dataset is reduced. The combination of these two scores is called U-COPS.

Tasks and Used DNN Architectures. The experimental evaluations are done on the image classification tasks MNIST [92], CIFAR-10, CIFAR-100 [85] and Tiny ImageNet [89]. For these tasks, the CNNs LeNet-5-Caffe [92], VGG-16 [168], ResNet18 [64] and Wide-ResNet18-2 [204] are used as DNN architectures.

Optimizing the Hyperparameter α_κ. The COPS masks rely on the balance factor α_κ between the target and control score. This adds an additional hyperparameter to the

Table 4.1: (Table III in Wimmer et al. [193]) G-COPS and M-COPS compared with other pruning at initialization methods on CIFAR-10 with VGG16. The optimal hyperparameter choice for α_κ for COPS is given in parentheses.

Pruning Criterion	$q = 0.9$	$q = 0.95$	$q = 0.99$	$q = 0.995$
Dense Baseline ($q = 0$)		$93.41 \pm 0.07\%$		
Random	91.21%	88.83%	81.70%	66.59%
SNIP	93.13%	92.63%	86.34%	10.00%
GraSP	92.64%	91.94%	88.84%	76.82%
Magnitude	92.94%	92.16%	10.00%	10.00%
G-COPS	**93.15**%	92.66%	**89.12**%	**82.67**%
(α_κ)	(0.1)	(0.05)	(0.99999)	(0.9999)
M-COPS	**93.15**%	**92.68**%	87.71%	22.30%
(α_κ)	(0.1)	(0.3)	(0.5)	(0.4)

training procedure. Inspired by the combination of SNIP and GraSP, see Figure 4.6, a grid search for $\alpha_\kappa \in (0,1)$ is used to find the best balance between the two underlying pruning scores. For high compression rates, the gradient flow is a limiting factor for successful training. Therefore, the balance factors are more fine-grained towards 1. Altogether, the hyperparameter α_κ is chosen from $\{0.05, 0.1, 0.2, \ldots, 0.9, 0.99, \ldots, 0.99999\}$, *i.e.* 14 choices in total. Even though M-COPS and U-COPS have different boundary conditions, the same grid search for α_κ is used for them as well.

4.4.1 Experimental Results for COPS

The performance of COPS is evaluated on the datasets CIFAR-10, CIFAR-100 and Tiny ImageNet. The results are compared with the underlying pruning methods SNIP, GraSP and magnitude pruning as well as with random PaI.

First, COPS is tested on a VGG-16 on CIFAR-10. Two versions for COPS are tested in this scenario, namely G-COPS and M-COPS. The results are summarized in Table 4.1.

As suspected, SNIP shows better results than GraSP for lower compression rates $q \leq 0.95$ while GraSP outperforms SNIP for higher q. Combining both methods via G-COPS shows better results than both pruning criteria for all compression rates q. Especially for high q, where SNIP significantly drops performance, controlling the gradient flow via GraSP while also training a few highly salient weights – obtained by

Table 4.2: (Table IV in Wimmer et al. [193]) Comparison of G-COPS, SNIP and GraSP. Here, A_Δ denotes the accuracy difference to G-COPS' result. The similarity sim equals $\text{sim}\left(\mathbf{m}^{(0)}, \mathbf{m}^{(\alpha_\kappa)}\right)$ and $\text{sim}\left(\mathbf{m}^{(1)}, \mathbf{m}^{(\alpha_\kappa)}\right)$ for SNIP and GraSP, respectively, where $\mathbf{m}^{(\alpha_\kappa)}$ is the G-COPS mask. The chosen balance factor α_κ for the G-COPS mask is given in the last row.

	CIFAR-100 ResNet18			Tiny ImageNet ResNet18			Tiny ImageNet Wide-ResNet18-2		
	Compression rate q								
	0.85	0.95	0.99	0.85	0.95	$q = 0.99$	0.85	0.95	0.99
Baseline	$74.53 \pm 0.33\%$			$55.97 \pm 0.37\%$			$59.38 \pm 0.25\%$		
Random	70.71%	64.54%	50.70%	52.59%	46.96%	33.87%	56.52%	52.94%	42.63%
SNIP	71.61%	68.39%	56.69%	56.10%	54.50%	37.00%	58.73%	57.71%	51.07%
A_Δ	**−0.06%**	**−0.13%**	**−2.56%**	**−0.06%**	**−0.25%**	**−7.27%**	**−0.03%**	**−0.06%**	**−1.94%**
sim	0.988	0.938	0.302	0.985	0.990	0.277	0.982	0.981	0.271
GraSP	71.16%	68.40%	58.67%	54.77%	52.83%	43.81%	56.87%	57.22%	52.69%
A_Δ	**−0.51%**	**−0.12%**	**−0.58%**	**−1.39%**	**−1.92%**	**−0.46%**	**−1.89%**	**−0.55%**	**−0.32%**
sim	0.351	0.350	0.994	0.343	0.306	0.998	0.355	0.303	0.994
G-COPS	**71.67%**	**68.52%**	**59.25%**	**56.16%**	**54.75%**	**44.27%**	**58.76%**	**57.77%**	**53.01%**
(α_κ)	(0.05)	(0.2)	(0.9999)	(0.05)	(0.05)	(0.99999)	(0.05)	(0.05)	(0.9999)

optimizing the SNIP target score – improves results significantly. For lower q, however, results for G-COPS and SNIP are almost the same. In summary, gradient flow is a necessary condition for successful training while it should not be the only criterion to choose a pruning mask – even for high compression rates. On the other hand, constraining the gradient flow in situations where it is not a limiting factor does not improve results significantly. Therefore, the chosen balance factor for G-COPS is close to zero in these situations.

Similarly, combining the SNIP and magnitude pruning criteria to generate M-COPS improves results, especially for high compression rates. This is surprising since the magnitude criterion alone yields untrainable architectures in these situations. *E.g.* controlling the magnitudes of the un-pruned weights improves performance more than 1.3% for $q = 0.99$. A comparison of M-COPS and G-COPS shows that both perform equally well for lower q while G-COPS has superior results than M-COPS for high q. Therefore, experiments for CIFAR-100 and Tiny ImageNet are limited to G-COPS which overall performs better.

Table 4.2 shows results for CIFAR-100 and Tiny ImageNet with the residual networks ResNet18 and Wide-ResNet18-2. Similar to CIFAR-10, controlling the gradient flow for lower compression rates only improves results slightly compared to SNIP. As shown, the G-COPS pruning mask is very similar to the SNIP mask in these cases. Again, GraSP has inferior performance than SNIP and G-COPS for lower compression rates. However, if the compression rate is increased, G-COPS shows significant improvements compared to SNIP *and* GraSP. This underlines the fact that sparse architectures should not be optimized to have a high information flow alone. For example, G-COPS shows 0.5% better test accuracy than GraSP for $q = 0.99$ with a ResNet18 on Tiny ImageNet. In this scenario, a tight constraint $\alpha_\kappa = 0.99999$ is required to guarantee a high information flow for the G-COPS mask. This results in a mask similarity between G-COPS and GraSP of 0.998. However, the tiny difference between G-COPS and GraSP results in a significant improvement in performance. For CIFAR-10, G-COPS surpasses GraSP by 5.85% for $q = 0.995$ while the corresponding masks have a similarity of 0.992.

In summary, using G-COPS improves both, SNIP and GraSP significantly for high compression rates q. In these scenarios, G-COPS is able to guarantee a high information flow in the sparse network by setting a tight constraint on the GraSP score. Moreover, highly salient key weights are trained as well which leads to the improved performance. However, for lower q, controlling the gradient flow of the sparse architecture does not yield significantly better trainable models. Therefore, using SNIP instead of G-COPS in these situations is the more economic choice since SNIP does not need to tune the balance factor. Finally, COPS is not restricted on combining SNIP and GraSP, but can also improve other combinations of pruning scores, as shown by M-COPS.

4.4.1.1 COPS Is Not a Binary Decision Between Two Scores

As can be seen in Tables 4.1 and 4.2, COPS tends to find a mask with high similarity to either the SNIP or the GraSP mask. However, in most situations, either SNIP or GraSP performs significantly better than the other one. Therefore, it is most likely that the best possible combined pruning mask for the given compression rate lies closer to the better performing one. As shown in Tables 4.1 and 4.2, G-COPS is able to find a better performing pruning mask, especially for high compression rates. Consequently, even if G-COPS is close to either SNIP or GraSP, it is *not* a binary decision between those

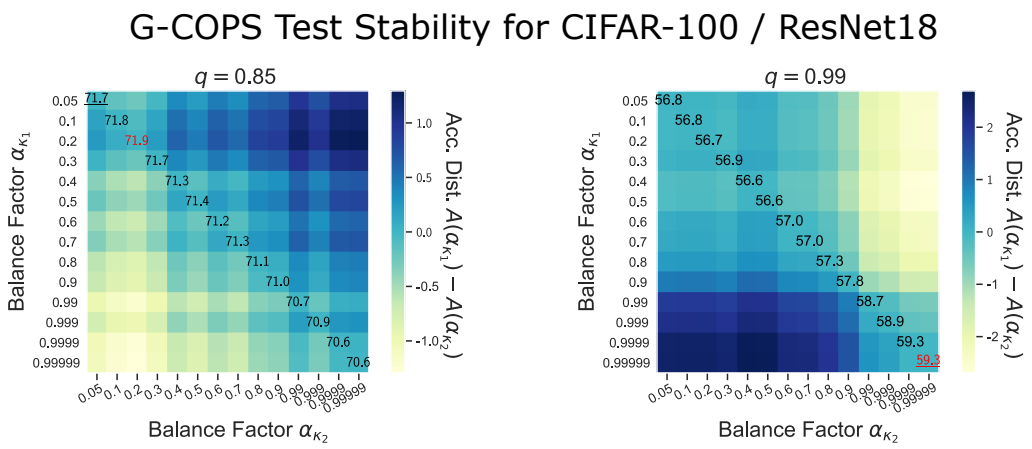


Figure 4.7: (Figure 5 in Wimmer et al. [193]) Stability analysis for G-COPS w.r.t. the hyperparameter α_κ for a ResNet18 on CIFAR-100 with compression rates $q = 0.85$ (left) and $q = 0.99$ (right). The diagonal shows the test accuracies $A(\alpha_\kappa)$ for the given choice of α_κ. The best test accuracy is highlighted in red while the accuracy for α_κ with the best corresponding validation accuracy is underlined.

pruning methods. Preferably, COPS is esteemed as a method that combines the better parts of two pruning methods.

4.4.2 Stability of COPS

COPS requires the balance factor between the two pruning scores α_κ as an additional hyperparameter which needs to be tuned. Figure 4.7 analyzes the stability of the found G-COPS pruning mask w.r.t. changing balance factors α_κ. For the analysis, two compression rates $q = 0.85$ and $q = 0.99$ are used. The results are shown on a heat map where the corresponding cell corresponds to the test accuracy difference between the models with G-COPS masks computed with balance factor α_{κ_1} (rows) and α_{κ_2} (columns), respectively. The test accuracy of a model pruned with balance factor α_κ is denoted by $A(\alpha_\kappa)$.

Analogous to the results discussed in Section 4.4.1, G-COPS has better results for a loose constraint, *i.e.* low α_κ, for the lower compression rate $q = 0.85$. On the other hand, a tight constraint is required if more parameters are pruned. Moreover, the results for G-COPS are stable for lower compression rates. Here, stability means that $|A(\alpha_{\kappa_1}) - A(\alpha_{\kappa_2})|$ is small if $|\alpha_{\kappa_1} - \alpha_{\kappa_2}|$ is small. Stability w.r.t. hyperparameter choices

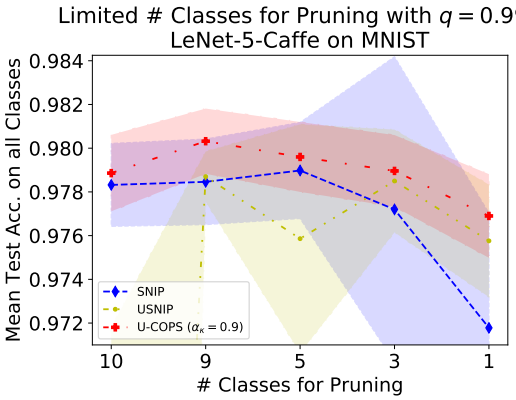


Figure 4.8: (Figure 4 in Wimmer et al. [193]) Comparing SNIP, USNIP and the combined version U-COPS for a LeNet-5-Caffe trained on MNIST. The x-axis shows the number of available classes for the score computation. USNIP is unstable if all classes are used which results in a test accuracy of $89.8 \pm 7.4\%$.

is important since hyperparameters are rather optimized to lie close to an optimum than exactly being one. For high q on the other hand, there is a performance gap in the heat map. The test accuracy differs around 0.9% between $\alpha_\kappa = 0.9$ and $\alpha_\kappa = 0.99$. If many parameters are pruned, information flow becomes a bottleneck for training. Therefore, a tight constraint on the gradient flow of the sparse networks guarantees a well trainable model whereas too less focus on the gradient flow leads to worse results. On both sides of the performance gap, G-COPS is again stable. For high pruning rates it is therefore important to find a α_κ which guarantees enough gradient flow.

The α_κ with the best validation accuracy does not always yield the best test accuracy as shown for $q = 0.85$ in Figure 4.7. However, due to the stability of the G-COPS pruned models w.r.t. balance factors around optima, the results with the optimal α_κ for the validation accuracy are close enough to the best results regarding test accuracy.

4.4.3 COPS Pruning with Partial Class Information

As shown in Table 4.1, COPS is not restricted to only combine SNIP and GraSP. Despite the gradient flow, other restrictions of a pruning criterion can be overcome by using COPS. An example for this is the dependency of pruning methods like SNIP or GraSP on the used data for computing the pruning score. If the data used for computing the pruning mask is a bad sample of the true data distribution, the sparse architecture might

Table 4.3: (Table V in Wimmer et al. [193]) Random search on CIFAR-10 with VGG-16 for G-COPS (α_κ) and λ for the linear combination $S_0 + \lambda S_1$ for the compression rate $q = 0.9$, each tested for 14 hyperparameters.

Method	Best Test Acc.	Worst Test Acc.	Mean Test Acc.	Std Test Acc.
G-COPS	93.08%	92.79%	92.93%	0.16%
$S_0 + \lambda S_1$	92.77%	92.46%	92.62%	0.20%

be sub-optimal for the given task. Using only a subset $\mathcal{T} \subset \{0,\dots,C-1\}$ of the classes for computing the pruning score is a simple way to model such a *bad* data sample. Consequently, the classes $\{0,\dots,C-1\} \setminus \mathcal{T}$ are not known during pruning time however they are still used for training, validation and testing.

As shown in Figure 4.8, SNIP is quite stable for $5 \leq \#\mathcal{T} \leq 10$ for a LeNet-5-Caffe trained on MNIST – which has 10 classes in total. However, if less classes are used, performance drops steeply. The sets $\mathcal{T}$ are chosen by random, but equal for all random seeds and pruning methods.

Missing information about class labels $\{0,\dots,9\} \setminus \mathcal{T}$ can for example be controlled by the *unsupervised* SNIP score, USNIP [95]. Here, random labels $y \in \{0,\dots,9\}$ are assigned to each data point used for computing the pruning mask. Therefore, all classes are backpropagated during the computation of the pruning score. Results for USNIP are also shown in Figure 4.8. As can be seen, USNIP shows worse results than SNIP for $5 \leq \#\mathcal{T} \leq 10$. However, if less class information is available, USNIP exceeds SNIP.

U-COPS controls the supervised SNIP score via USNIP to also allow class labels from $\{0,\dots,9\} \setminus \mathcal{T}$ to be used for the computation of the pruning mask. As shown in Figure 4.8, U-COPS with $\alpha_\kappa = 0.9$ improves results compared to both, SNIP and USNIP for all sizes of $\mathcal{T}$. The best results are obtained by U-COPS for $\#\mathcal{T} = 9$ which shows that using noisy labels with noise level controlled by α_κ can generate better sparse architectures. This phenomenon can be compared with *label smoothing* [173] which improves standard DNN training by slightly disturbing the true labels. This regularizes the training procedure and, by the bias-variance trade off, improves the generalization ability of a model.

4.4.4 Necessity of the Dual Problem for Finding the COPS Mask

Let $\lambda^* \geq 0$ be the solution to the dual problem (4.17). If the requirements for Theorem 4.3.1 are fulfilled, strong duality holds. This implies

$$\min_{\mathbf{m}\in\mathcal{V}_{1,d}} S_0(\mathbf{m}) + \lambda^* \cdot (S_1(\mathbf{m}) - \kappa) = g(\lambda^*) = p^* \,. \tag{4.32}$$

Therefore, instead of using α_κ as hyperparameter, $\lambda \geq 0$ can be used as hyperparameter and the corresponding optimization problem is then given as

$$\min_{\mathbf{m}\in\mathcal{V}_{1,d}} S_0(\mathbf{m}) + \lambda \cdot S_1(\mathbf{m}) \,. \tag{4.33}$$

For each $\lambda \geq 0$, $-\lambda \cdot \kappa$ is constant and is therefore dropped in this formulation. Now (4.33) can be cast to a problem on $\mathcal{V}_{0,d}$

$$\min_{\mathbf{m}\in\mathcal{V}_{0,d}} S_0(\mathbf{m}) + \lambda \cdot S_1(\mathbf{m}) \,, \tag{4.34}$$

which can be solved analogously to (4.7). The problem (4.34) can – similar to COPS – be seen as a balancing between the pruning scores S_0 and S_1 where the balance is now controlled by $\lambda \geq 0$. However, the balance factor α_κ is easier to interpret since it balances the constraint between 0 (loose) and 1 (tight), see the discussion in Section 4.3.4. For the unbounded λ it is not clear if a $\lambda = 50$ defines a tight or a loose constraint.

A natural question to ask is if using $\lambda \geq 0$ as hyperparameter and solving the intuitive problem (4.34) performs better, similar or worse than COPS. Results for the comparison are given in Table 4.3. The comparison is done via random search [8] for the hyperparameters α_κ and λ. Note, since $\lambda \geq 0$ is unbounded, the λ was constrained to lie in $\lambda \in (0, 90)$ where $90 \approx \lambda^*$ for the λ^* optimizing G-COPS for $\alpha_\kappa = 0.99999$. As before, α_κ is optimized in the bounded, open unit interval $(0, 1)$. All in all, 14 different hyperparameters are tested for both, G-COPS and optimizing λ – again averaged over 5 random seeds for each hyperparameter.

Results in Table 4.3 show that optimizing $\alpha_\kappa \in (0, 1)$ for COPS leads to much better results than optimizing λ for (4.34). COPS' superior results can be explained by the smaller interval $(0, 1)$, where the hyperparameter α_κ is chosen from, together with COPS' robustness, discussed in Section 4.4.2. By COPS' stability, a randomly chosen parameter

α_κ not too far away from the optimal one will yield good results. Directly optimizing λ on the other hand does not seem to guarantee this stability. Therefore, more training runs to find a well-performing $\lambda \geq 0$ are required than finding a good balance factor α_κ for COPS.

4.5 Conclusions for COPS

COPS is a method to combine two score based pruning masks to a more powerful one. The motivation for COPS is to fuse the strengths of the SOTA pruning methods applied at initialization SNIP [94] and GraSP [184], while at the same time overcoming their weaknesses. This is done by primarily optimizing the SNIP pruning score whilst setting a constraint on the GraSP score. Such a constraint is an intuitive choice since SNIP tends to generate sparse architectures with a weak gradient flow whereas the GraSP score optimizes the sparse model to have a high gradient flow. As a consequence, the model pruned with a COPS mask has sufficient gradient flow if the constraint is tight enough.

The COPS pruning mask is computed by solving a combinatorial, constrained optimization problem. As a discrete problem, it is expensive to solve in its raw format. Therefore, a relaxation of the problem on a LP is solved by solving its dual problem. By using Algorithm 4.1, a solution of the LP can be found more efficiently than with the fastest general LP solver. The relaxed solution can then be projected on an approximate solution to the original discrete problem.

As shown, combining the SNIP and GraSP pruning scores improves both methods for all tested compression rates. Especially for high q, combining the strengths of both pruning methods outperforms the single scores significantly. For lower q where the information flow is not a bottleneck for training, combining GraSP and SNIP does not improve results for the better performing SNIP all too much. Since COPS needs the balance factor α_κ between the two pruning scores as an additional hyperparameter, it is more economic to prune with the SNIP score in these cases.

However, the COPS method is not restricted to combine only SNIP and GraSP. Arbitrary score based pruning methods can be merged via COPS, as shown on two examples in Sections 4.4.1 and 4.4.3. By simply adding more constraints to the optimization problem solved by COPS, more than two criteria could be combined. But note, each newly added pruning criterion adds an additional balance factor which needs to be optimized.

This enlarges the search space for the COPS hyperparameters exponentially.

As shown in Section 3, dimensionality reduced training is not restricted to pruned networks. Also, the frozen weights are determined with the help of a GSS. Therefore, the COPS method can also be used for other DRT methods, as for example freezing. However, combining SNIP and GraSP might not improve results for FreezeNet significantly since FreezeNet overcomes the low information flow in the network through its frozen parameters. As shown with M-COPS and U-COPS, other combinations of GSS have improved results as well. Therefore, there might exist well combinable criteria to improve results for frozen DNNs or other dimensionality reduced training strategies. Combined pruning scores should also help to boost pruning methods applied during or after training. Here, other combinations of pruning scores might improve performance more than G-COPS does since both, SNIP and GraSP are specialized on pruning at initialization.

Chapter 5
Interspace Pruning: Using Adaptive Filter Representations to Improve Training of Sparse CNNs

Standard pruning (SP) methods prune spatial weights of DNNs, *i.e.* set entries in the weight tensors to zero. This generates a fixed subspace spanned by the corresponding pruning transformation ψ with $\psi \in \{0,1\}^{D\times d}$, $\|\psi\|_0 = d$. The sparse DNN with trainable parameters $\vartheta \in \mathbb{R}^d$ is restricted to this d dimensional, fixed subspace. Especially for pruning at initialization or other pruning methods applied before or early in training, this induces a strong bias.

Interspace Pruning: Using Adaptive Filter Representations to Improve Training of Sparse CNNs by Wimmer et al. [194] represents $K \times K$ convolutional filters by a linear combination of an underlying, adaptive basis – the *filter basis* (FB). This is the so-called *interspace representation*. For *interspace pruning* (IP), basis coefficients w.r.t. the FB are pruned. During training, a FB is adapted simultaneously with its non-zero coefficients to generate the best dictionary for representing filters sparsely. This increases the capacity of sparse models. Therefore, they can be trained better, especially for high compression rates. To maintain low memory and computational costs for sparse networks, only a few FBs are shared for the big number of convolutional filters in the network. Sharing FBs between $K \times K$ filters entangles them and therefore regularizes training. Thus, using interspace representations improves the generalization ability even for low compression rates. Altogether, using interspace representations improves results for arbitrary SOTA pruning at initialization, dynamic sparse training, lottery ticket hypothesis or gradual pruning methods – but also densely trained models – while having similar runtime and parameter costs. Moreover, the interspace representation also improves results for

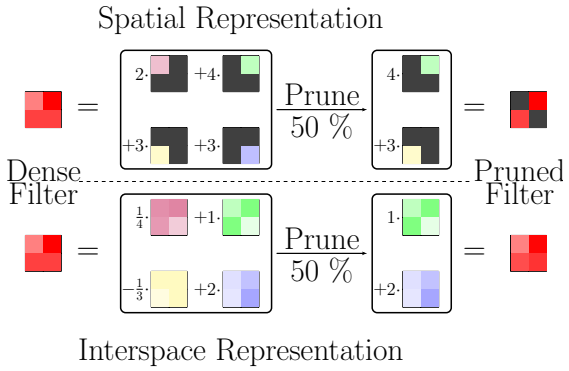


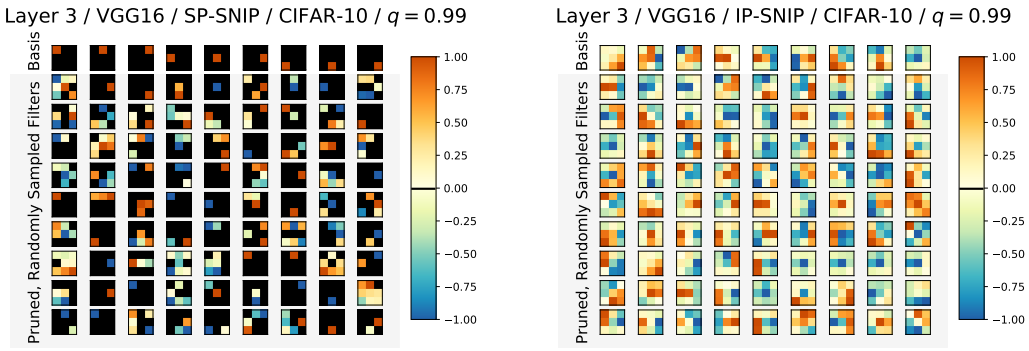
Figure 5.1: (Figure 1(a) in Wimmer et al. [194]) Comparing SP (top) and IP (bottom) on the example of 2×2 filters. Both, IP and SP, prune coefficients w.r.t. basis elements. For SP, the basis is fixed to be the standard basis $\mathcal{B}$ while for IP an adaptive basis $\mathcal{F}$ is used. The four basis elements have different colors (pink, green, yellow, blue). Dense and pruned filters (rightmost and leftmost in red) are shown in spatial representation.

other DRT methods like freezing. This Chapter presents *Interspace Pruning: Using Adaptive Filter Representations to Improve Training of Sparse CNNs* [194] and follows the structure of this publication. A graphical comparison between IP and SP is shown in Figure 5.1.

5.1 Motivation for Interspace Pruning

By definition of a pruned DNN, the sparse model with parameters $\psi \cdot \vartheta$, $\psi \in \{0,1\}^{D \times d}$ and $\|\psi\|_0 = d$, lies in a *fixed* d dimensional subspace of $\mathbb{R}^D$. As discussed in Sections 3 and 4, training pruned DNNs often leads to sub-optimal performance due to poor choices of the sparse architecture. The discussions showed that reasons for inferior results can be an architecture with bad information flow, or training weights which are less influential in optimizing the loss function. Since a single choice of a sparse DNN architecture might be sub-optimal for the whole training process, results can be improved by reducing the inductive bias of a fixed subspace.

In contrast to dynamic sparse training, which discretely updates the sparse architecture during training by adapting the pruning transformation $\psi^{(t)} \in \{0,1\}^{D \times d}$ each t_p training steps, interspace pruning continuously optimizes the low dimensional submanifold during training. This is done by additionally adding a trainable filter basis



(a) (b)

Figure 5.2: (Figure 1(b) and (c) in Wimmer et al. [194]) Contrarily to SP (a), IP (b) produces spatially dense filters after training networks with sparse coefficients. As for SP, sparsity in the interspace can be used to reduce memory requirements and, by the linearity of convolutions, also computational costs.

$\mathcal{F} = \{\mathbf{F}^{(1)}, \ldots, \mathbf{F}^{(K^2)}\} \subset \mathbb{R}^{K \times K}$ to represent convolutional filters. By combining and stacking the basis transformation matrices of all FBs in a DNN into $\Phi^{(t)}$, the corresponding adaptive submanifold at training step t can be described by $\Psi^{(t)}(\vartheta) = \Phi^{(t)} \cdot \psi^{(0)} \cdot \vartheta$. Moreover, the pruning transformation $\psi^{(t)}$ can be made adaptive as well in the interspace pruning setting. By adding an additional affine term $\Phi^{(t)} \cdot ((\mathbf{1}_D - \psi^{(t)} \cdot \mathbf{1}_d) \odot \Theta^{(0)})$, interspace representations can also be used for freezing. Note, setting a FB as the standard basis $\mathcal{B} = \{\mathbf{E}^{(1,1)}, \ldots, \mathbf{E}^{(K,K)}\} \subset \mathbb{R}^{K \times K}$ shows that standard pruning is just a special case of interspace pruning. However, general interspace representations are more flexible since bases are not fixed.

A convolutional layer consists of $c_{\text{in}} \cdot c_{\text{out}}$ many convolutional filters $\varphi \in \mathbb{R}^{K \times K}$ where $K \times K$ denotes the kernel size of the filters. The filter φ is usually represented via the standard basis by $\varphi = \sum_{i,j} \varphi_{i,j} \cdot \mathbf{E}^{(i,j)}$. Since $\mathcal{B}$ and $\mathcal{F}$ span the same K^2 dimensional vector space $\mathbb{R}^{K \times K}$, the corresponding interspace representation of φ w.r.t. the FB $\mathcal{F}$ is given by $\varphi = \sum_i \beta_i \cdot \mathbf{F}^{(i)}$. Here, $\beta \in \mathbb{R}^{K^2}$ is the *FB coefficient* vector of φ w.r.t. $\mathcal{F}$.

A $\mathcal{F}$ can be shared for an arbitrary number of $K \times K$ filters. In contrast to the fixed standard basis, the FB $\mathcal{F}$ can be trained jointly with the FB coefficients β via backpropagation. Optimizing the FBs jointly with the coefficients overcomes the lack of knowledge of bases which represent convolutional filters efficiently with only a few non-zero basis coefficients. As a result, the interspace representation yields a dynamic

submanifold. If a filter is pruned to have one non-zero FB coefficient, *i.e.* $\varphi = \beta_i \cdot \mathbf{F}^{(i)}$, it is not restricted since $\mathbf{F}^{(i)}$ can adapt. A 1-sparse filter $\varphi = \varphi_{i,j} \cdot \mathbf{E}^{(i,j)}$ w.r.t. the standard basis $\mathcal{B}$ is immediately restricted to lie in the 1 dimensional subspace span $\mathbf{E}^{(i,j)}$.

Interspace pruning is defined as setting FB coefficients β_i to zero. This generates a sparse representation of filters in the interspace. However, the spatial representation of these models is usually dense as shown in Figures 5.1 and 5.2. In particular, the density of IP models in the spatial domain helps to overcome the low information flow which would occur otherwise. As a result, IP models can be trained better than SP ones.

IP is not a pruning method by itself but is added on top of already existing ones to further improve the sparse models. The adaptivity of the underlying interspace makes IP especially interesting for DRT methods. Therefore, IP is mainly tested on the SOTA pruning at initialization methods SNIP [94], GraSP [184] and SynFlow [174], the dynamic sparse training methods SET [122] and RigL, [40] the lottery ticket hypothesis [43, 44] and on FreezeNet [192]. Moreover, improved results for other pruning methods applied during [49] or after training [153] are shown as well.

As provided in Figure 5.2, sparse DNNs in the interspace have a dense representation in the spatial domain. However, sparsity in the interspace can be used to reduce the computational costs as well. In comparison to SP, IP only adds a small, constant `FLOP` count needed for the inference of sparse models. For training models with interspace representations, additional gradient computations for the FBs are required as well.

Since FBs are shared over many filters in a DNN, the additional memory requirements for IP are usually neglectable. For example, sharing a FB in each layer increases the memory footprint of a ResNet50 by only 0.01%. Despite adding only a few extra parameters, interspace representations improve results for sparse *and* dense training significantly.

5.1.1 Contributions of Interspace Pruning: Using Adaptive Filter Representations to Improve Training of Sparse CNNs

The core contributions of Wimmer et al. [194] are given by:

- Convolutional filters $\varphi \in \mathbb{R}^{K \times K}$ are represented in the interspace, a linear space spanned by a trainable filter basis $\mathcal{F} \subset \mathbb{R}^{K \times K}$.

- Pruning FB coefficients instead of spatial weights, *i.e.* coefficients w.r.t. the standard basis.
- The FBs are trained jointly with the underlying non-zero FB coefficients for optimizing the interspace to represent sparse filters better.
- Mathematically proofing the improvements of IP over SP.
- Experiments showing that IP yields better results than SP for arbitrary pruning methods. The tested pruning methods are applied before and during training, but also on pre-trained models. Moreover, interspace representations boost the generalization ability of densely trained models.

5.2 Related Work for Interspace Pruning

The related work for Wimmer et al. [194] is mostly covered by the related pruning literature in Section 2.3.1. However, there are some special pruning methods which – similar to IP – do not prune spatial weights but coefficients w.r.t. a different basis. Liu et al. [106] prune coefficients of the filters in the frequency domain. Thus, they are able to prune different frequency bands with varying pruning rates. Liu et al. [105] prune coefficients in the Winograd-domain which leads to a particularly efficient implementation of convolutions. However, both of these methods use a fixed domain for the filters and a pruning criterion specialized for the specific domain. Interspace pruning on the other hand is flexible in the choice of the representation of filters and not restricted to a particular pruning criterion.

Also to mention is the low rank approximation of CNNs presented by Li et al. [100]. Their method approximates a dense, pre-trained model with learned, undercomplete dictionaries for 3D filters in $\mathbb{R}^{c_{\text{in}} \times K \times K}$. Interspace pruning learns the filters and their bases in one, joint training. Moreover, IP uses 2D filter bases $\mathcal{F} \subset \mathbb{R}^{K \times K}$.

5.3 Interspace Pruning in the Viewpoint of Sparse Dictionary Learning

A commonly used pruning method is magnitude based pruning, see Section 2.2.3.2. Let a dense parameter matrix $\mathbf{R} \in \mathbb{R}^{D_1 \times D_2}$, as for example weights of a linear layer or reshaped weights of a convolutional layer, be given. The parameters found by magnitude pruning solve the following optimization problem

$$\min_{\mathbf{S}^{(\mathrm{id})} \in \mathbb{R}^{D_1 \times D_2}} \left\| \mathbf{R} - \mathbf{S}^{(\mathrm{id})} \right\|_F \quad \text{such that} \quad \left\| \mathbf{S}^{(\mathrm{id})} \right\|_0 \leq d \tag{5.1}$$

where d is the sparsity constraint on the solution. *Sparse dictionary learning* is a more general class of optimization problems, where not only the sparse solution $\mathbf{S}$ is optimized but also the used representation [2, 34, 39, 116]. For this, a *dictionary* $\mathbf{T} \in \mathbb{R}^{D_1 \times D_0}$ is optimized together with the sparse coefficients $\mathbf{S}^{(\mathbf{T})} \in \mathbb{R}^{D_0 \times D_2}$ to solve the problem

$$\min_{\mathbf{T} \in \mathbb{R}^{D_1 \times D_0}, \mathbf{S}^{(\mathbf{T})} \in \mathbb{R}^{D_0 \times D_2}} \left\| \mathbf{R} - \mathbf{T} \cdot \mathbf{S}^{(\mathbf{T})} \right\|_F \quad \text{such that} \quad \left\| \mathbf{S}^{(\mathbf{T})} \right\|_0 \leq d \, . \tag{5.2}$$

The maximum number of available coefficients for describing a column of $\mathbf{R}$ is determined by D_0. If the columns of $\mathbf{T}$ form a basis of $\mathbb{R}^{D_1}$, the dictionary is called a basis. If $D_0 < D_1$, the dictionary is *undercomplete* while it is called *overcomplete* if $D_0 > D_1$. Restricting $\mathbf{T}$ to the identity matrix $\mathrm{id} \in \mathbb{R}^{D_1 \times D_1}$, shows that magnitude pruning is a special case of sparse dictionary learning.

Now, if $\mathbf{R}$ encodes a convolutional layer of size $c_{\mathrm{out}} \times c_{\mathrm{in}} \times K \times K$, the corresponding matrix representation can be given by $\mathbf{R} \in \mathbb{R}^{K^2 \times c_{\mathrm{out}} \cdot c_{\mathrm{in}}}$. Furthermore, the dictionary is given by $\mathbf{T} \in \mathbb{R}^{K^2 \times D_0}$ which should corresponds to the filter *basis* $\mathcal{F} \subset \mathbb{R}^{K \times K}$. Since $\mathcal{F}$ forms a basis, $D_0 = D_1 = K^2$. Therefore, $\mathbf{T} \in \mathbb{R}^{D_1 \times D_1}$ is assumed in the following. As a result, the coefficients $\mathbf{S}$ have the same dimension as the target matrix $\mathbf{R}$. Under these assumptions, (5.1) defines standard magnitude pruning while (5.2) gives the corresponding IP problem.

Most sparse dictionary learning methods [2, 39, 116] optimize the dictionary $\mathbf{T}$ and the coefficients $\mathbf{S}^{(\mathbf{T})}$ in an alternate fashion. SP on the other hand, fixes $\mathbf{T} = \mathrm{id}$. Moreover, $\operatorname{supp} \mathbf{S}^{(\mathrm{id})}$ is fixed for many pruning methods during training, like for PaI or the lottery ticket hypothesis. IP overcomes SP's fixed solution space by training $\mathbf{T}$ and $\mathbf{S}^{(\mathbf{T})}$ jointly

via backpropagation.

The following Theorem 5.3.1 theoretically shows that using interspace pruning indeed leads to CNNs which approximate dense models better than standard pruning. Note, Theorem 5.3.1 is restricted to SP and IP with the magnitude criterion. Moreover, only the approximation – and not the generalization – ability of the filters is evaluated. Still, results in Theorem 4.3.1 motivate to prune models in the interspace, and not the spatial, representation. Experimental results in Section 5.5 show that IP outperforms SP w.r.t. generalization ability on various tested pruning criteria.

Theorem 5.3.1. *Let* $0 < d < D_1 \cdot D_2$, $D_1 > 1$ *and* $\mathbf{R}_{i,j} \sim \mathcal{N}(0,1)$ *i.i.d. Further let* $\rho^{(\mathrm{id})}$ *be the minimum of the standard pruning problem* (5.1) *and* $\rho^{(\mathbf{T})}$ *be the infimum obtained by the interspace pruning problem* (5.2) *with* $D_1 = D_0$. *Then it holds*

1. $\rho^{(\mathbf{T})} < \rho^{(\mathrm{id})}$ *with probability* $\mathbb{P} = 1$.
2. *Let* $\bar{\mathbf{S}}^{(\mathrm{id})}$ *solve the standard pruning problem* (5.1). *Then*

$$\min_{\mathbf{T} \in \mathbb{R}^{D_1 \times D_1}, \mathbf{S}^{(\mathbf{T})} \in \mathbb{R}^{D_1 \times D_2}} \left\| \mathbf{R} - \mathbf{T} \cdot \mathbf{S}^{(\mathbf{T})} \right\|_F \quad \textit{such that } \operatorname{supp} \mathbf{S}^{(\mathbf{T})} = \operatorname{supp} \bar{\mathbf{S}}^{(\mathrm{id})} \tag{5.3}$$

is the interspace pruning problem restricted to have similar support than the solution of the standard pruning. Let $\rho^{(\mathbf{T})}$ *minimize* (5.3). *Then,* $\rho^{(\mathbf{T})} \leq \rho^{(\mathrm{id})}$ *is always true and strict inequality holds with* $\mathbb{P} \geq 1 - \varepsilon$, *where*

$$\varepsilon = \begin{cases} \frac{\binom{D_2}{d/D_2}}{\binom{D_1 \cdot D_2}{d}} , & \text{if } d \equiv 0 \mod D_1 \\ 0 , & \text{else} \end{cases} . \tag{5.4}$$

Proof of Theorem 5.3.1 is given in Appendix, Section B.2. ■

By Theorem 5.3.1 part 1, it is clear that a solution found by interspace pruning approximates the dense parameters better than SP. This holds for an arbitrary number of filters sharing *one* FB. However, the pruning transformation $\psi^{(t)}$ is often not adaptable, *e.g.* for pruning at initialization. In this case, the coefficients for IP and SP have the same support but IP can additionally train a FB. This is investigated in part 2 of Theorem 5.3.1. As shown, the solution of IP is still better or equal to the solution of SP. Moreover for 3×3 convolutions ($D_1 = 9$), the chance that both are equal is numerically equal to zero if $D_2 = c_{\text{in}} \cdot c_{\text{out}} \geq 100$, see Figure 5.3.

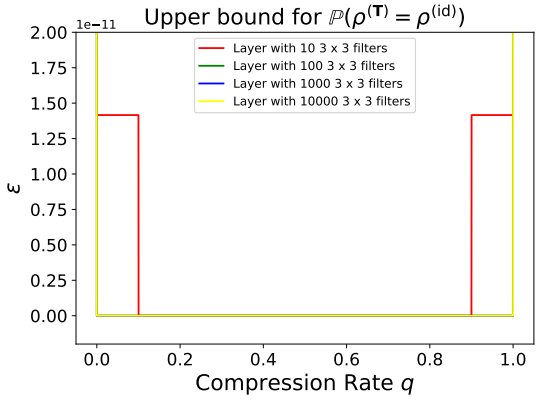


Figure 5.3: (Figure A9 in Wimmer et al. [194]) Upper bound for $\mathbb{P}(\rho^{(\mathbf{T})} = \rho^{(\mathrm{id})})$. Results are computed with kernel size $K = 3$ and $c_{\mathrm{out}} \cdot c_{\mathrm{in}}$ many filters. Therefore $D_1 = K^2 = 9$ and $D_2 = c_{\mathrm{out}} \cdot c_{\mathrm{in}}$. Note, the y-axis is scaled with 10^{-11}.

5.4 Interspace Representations and Interspace Pruning

After having discussed IP from a theoretical perspective, the procedure of IP will now be defined formally. Before diving into the pruning part, first the interspace representations and the corresponding description of 2D convolutions will be discussed.

5.4.1 Interspace Representation

Assume a convolutional layer with c_{out} and c_{in} output- and input channels, respectively. The convolutions are assumed to have a quadratic kernel size $K \times K$, no zero padding and 1×1 stride and dilation. Also, the following introduction to interspace representations is restricted to 2D convolutions. The proposed formulation can be generalized straight forward to other shapes, paddings, stride, dilations and convolutions of arbitrary dimension. Now, let $\varphi = (\varphi^{(n,m)})_{n=1,\dots,c_{\mathrm{out}},m=1,\dots,c_{\mathrm{in}}} \in \mathbb{R}^{c_{\mathrm{out}} \times c_{\mathrm{in}} \times K \times K}$ denote the 4 dimensional tensor containing all convolutional filters $\varphi^{(n,m)} \in \mathbb{R}^{K \times K}$ in this layer where $\varphi^{(n,m)}$ connects input channel m with output channel n. Further, let $\mathcal{F} = \{\mathbf{F}^{(1)}, \dots, \mathbf{F}^{(K^2)}\} \subset \mathbb{R}^{K \times K}$ be the FB used in this layer.[6] Then

$$\varphi^{(n,m)} = \sum_{i=1}^{K^2} \beta_i^{(n,m)} \cdot \mathbf{F}^{(i)} \tag{5.5}$$

[6] Using more than one FB in a layer is again straight forward but would require additional notation.

defines the interspace representation for $\varphi^{(n,m)}$ with filter base coefficients $\beta^{(n,m)} \in \mathbb{R}^{K^2}$. The interspace representations can also be derived by using the basis transformation matrix $\mathbf{A}^{(\mathcal{B},\mathcal{F})} \in \mathbb{R}^{K^2 \times K^2}$ between $\mathcal{F}$ and $\mathcal{B}$ via

$$\bar{\varphi}^{(n,m)} = \mathbf{A}^{(\mathcal{B},\mathcal{F})} \cdot \beta^{(n,m)} \,,\; \bar{\varphi}^{(n,m)} = \left(\varphi_{i_k,j_k}^{(n,m)}\right)_{k=1}^{K^2} \in \mathbb{R}^{K^2} \,,\; \mathbf{A}_{a,b}^{(\mathcal{B},\mathcal{F})} = \left\langle \mathbf{F}^{(b)}, \mathbf{E}^{(i_a,j_a)} \right\rangle \quad (5.6)$$

where $\{(i_1,j_1),\ldots,(i_{K^2},j_{K^2})\} = \{1,\ldots,K\}^2$. Thus, $\bar{\varphi}^{(n,m)}$ is a flattened version of the spatial representation $\varphi^{(n,m)}$ of the convolutional filter.

5.4.2 Convolutions with Interspace Representation

By using the interspace representation (5.5) and the linearity of the 2D convolution ✲, convolutions can be computed in the interspace representation without the need of changing the filters to the spatial representation first. Let $\mathbf{V} \in \mathbb{R}^{c_{\text{in}} \times d_1 \times d_2}$ be the input feature map of the convolutional layer. Then, the layer's output is given by $\mathbf{W} = \varphi \text{✲} \mathbf{V} \in \mathbb{R}^{c_{\text{out}} \times d_1 - K + 1 \times d_2 - K + 1}$. This can be computed for each output channel $n \in \{1,\ldots,c_{\text{out}}\}$ via

$$\mathbf{W}_{n,:,:} = \sum_{m=1}^{c_{\text{in}}} \varphi^{(n,m)} \star \mathbf{V}_{m,:,:} = \sum_{m=1}^{c_{\text{in}}} \sum_{i=1}^{K^2} \beta_i^{(n,m)} \cdot \left(\mathbf{F}^{(i)} \star \mathbf{V}_{m,:,:}\right) \,. \quad (5.7)$$

Therefore, $\mathbf{F}^{(i)} \star \mathbf{V}_{m,:,:}$ has to be computed for each combination of basis element $\mathbf{F}^{(i)}$ and channel of the input feature map $\mathbf{V}_{m,:,:}$. The output for each channel n is then simply a linear combination of the $\mathbf{F}^{(i)} \star \mathbf{V}_{m,:,:}$ where the scaling is defined by the FB coefficients $\beta_i^{(n,m)}$.

Since interspace representations are also used at training time, the backpropagation formulas for (5.7) need to be determined as well. It holds [194]

$$\nabla_{\varphi^{(n,m)}} \mathscr{L} = \nabla_{\mathbf{W}_{n,:,:}} \mathscr{L} \star \mathbf{V}_{m,:,:} \quad (5.8)$$

$$\nabla_{\beta_i^{(n,m)}} \mathscr{L} = \left\langle \mathbf{F}^{(i)}, \nabla_{\varphi^{(n,m)}} \mathscr{L} \right\rangle = \left\langle \nabla_{\mathbf{W}_{n,:,:}} \mathscr{L}, \mathbf{F}^{(i)} \star \mathbf{V}_{m,:,:} \right\rangle \quad (5.9)$$

$$\nabla_{\mathbf{F}^{(i)}} \mathscr{L} = \sum_{n=1}^{c_{\text{out}}} \sum_{m=1}^{c_{\text{in}}} \beta_i^{(n,m)} \cdot \nabla_{\varphi^{(n,m)}} \mathscr{L} = \sum_{n=1}^{c_{\text{out}}} \sum_{m=1}^{c_{\text{in}}} \beta_i^{(n,m)} \cdot \left(\nabla_{\mathbf{W}_{n,:,:}} \mathscr{L} \star \mathbf{V}_{m,:,:}\right) \,. \quad (5.10)$$

As can be seen, (5.10) has computational costs in the same order as $\nabla_{\varphi^{(n,m)}} \mathscr{L}$. More-

over, (5.9) and (5.8) have equal costs since $\mathbf{F}^{(i)} \star \mathbf{V}_{m,:,:}$ is known for all i, m from the computations in the forward pass.

5.4.3 Sharing and Initializing Filter Bases

After the interspace representations and the derived 2D convolutions have been introduced in Sections 5.4.1 and 5.4.2, respectively, it is important to see how FBs can be initialized. Furthermore, the *sharing* of a FB for a set of $K \times K$ filters is discussed.

5.4.3.1 Sharing Filter Bases

A FB $\mathcal{F} = \{\mathbf{F}^{(1)}, \ldots, \mathbf{F}^{(K^2)}\} \subset \mathbb{R}^{K \times K}$ requires K^4 parameters to be stored. If each convolutional filter $\varphi^{(n,m)} \in \mathbb{R}^{K \times K}$ had its own FB, the parameter count in the CNN would explode. Therefore, FBs must be shared for more than one $K \times K$ filter. Since a primary goal of pruning is to reduce the number of parameters, the number $n_{\mathcal{F}}$ of shared FBs should be quite small, optimally $n_{\mathcal{F}} \cdot K^4 \ll d$. The three proposed sharing schemes in Wimmer et al. [194] are called `coarse`, `medium` and `fine`. They differ in the number of total FBs for a CNN. The `coarse` scheme uses one global FB which is shared for all $K \times K$ filters in the network. The `fine` sharing on the other hand uses one FB for each layer, *i.e.* $n_{\mathcal{F}} = L$. In between these methods is the `medium` sharing which uses 5 FBs in total. For ResNets [64], `medium` has one FB for each of the 5 convolutional blocks. For VGG-16 [168], a FB is shared for the convolutional layers $(1,2)$, $(3,4)$, $(5,6,7)$, $(8,9,10)$ and $(11,12,13)$, respectively. Since CNNs usually have a parameter count in the millions, $n_{\mathcal{F}} \cdot K^4 \ll d$ even holds for high compression rates. To be precise, ${}^{n_{\mathcal{F}} \cdot K^4}/_{D} \leq 10^{-4}$ for sharing FBs with `fine`, `medium` and `coarse` for the VGG-16 and ResNets tested in Section 5.5. Consequently, using these FB sharing schemes does not increase the parameter size of the network significantly.

The space spanned by the FBs does not enlarge with increasing $n_{\mathcal{F}}$. However, if the FBs are shared more fine-grained, the corresponding interspaces can adapt better to the represented filters. This increases the capacity of the model. On the other hand, if more filters share the same FB, they are becoming entangled by relying on the same FB. The backpropagation formula (5.10) shows that all filters represented by a FB have an effect on the gradient of the FB. Different filters want the FB to move in different directions. Therefore, sharing FBs over more than one $K \times K$ filter regularizes training.

In summary, using more FBs increases the capacity while using less FBs yields a stronger regularization on the training procedure. Therefore, `fine` sharing is assumed to lead to better results for high compression rates where the model lacks of capacity. For lower compression rates on the other hand, using only one FB via `coarse` sharing should improve the model's generalization ability by regularizing the training more strongly.

5.4.3.2 Initializing Filter Bases

Contrarily to most of the other sparse dictionary learning methods, IP learns FBs and FB coefficients jointly. Therefore, the FBs are optimized during training and can be initialized with simple heuristics. Three different methods for initializing FBs are proposed in Algorithm 5.1. All initializations are chosen such that the spatial representation of the filters follows a Kaiming-normal initialization. To simplify notations, the Algorithm is restricted to initialize a single layer. Note, for sharing a FB over more than one layer, the FB is initialized only once and not for each layer again.

The `standard` initialization in Algorithm 5.1 initializes each FB as the standard basis $\mathcal{B}$. Therefore, the FB coefficients are the spatial coefficients at the beginning of training. However, the FBs are able to adapt during training and the FB will change into a more optimal one. The `standard` initialization is used in most experimental comparisons in Section 5.5 because then IP and SP start at exactly the same position.

Algorithm 5.1 also proposes the `ONB` initialization. In contrast to the `standard` initialization, a randomly chosen *orthonormal basis* (ONB) defines the FB at the beginning of training. The FB coefficients are obtained via inverting (5.6) by transforming the basis representation of a Kaiming-normal initialized spatial weight tensor.

Finally, both already presented initializations require $\{\mathbf{F}^{(1)}, \ldots, \mathbf{F}^{(K^2)}\}$ to form a basis. However, the interspace can also be spanned by an undercomplete- or overcomplete dictionary. The most intuitive method to initialize an arbitrary sized set of filters is by initializing them randomly. This is shown in Algorithm 5.1 with the `random` initialization which can be applied to form dictionaries with an arbitrary number of filters $n_{\#}$. Note, the rescaling of the filters $\mathbf{F}^{(i)}$ in line 19 guarantees that the spatial coefficients follow a Kaiming-normal initialization if the FB coefficients β are initialized with a Kaiming-normal initialization. More details about the rescaling can be found in Section G in Wimmer et al. [194]

Algorithm 5.1: Initializations for FBs and FB coefficients

Input: Filter size $K \times K$,
Number of output channels c_{out},
Number of input channels c_{in},
Initialization function $h_{\text{init}} \in \{\texttt{standard}, \texttt{ONB}, \texttt{random}\}$,
Number of filters spanning interspace $n_\# \leftarrow$ default: K^2 to set $\mathcal{F}$ as a base

Output: FB $\mathcal{F} = \{\mathbf{F}^{(1)}, \dots, \mathbf{F}^{(n_\#)}\} \subset \mathbb{R}^{K \times K}$,
FB coefficients $\beta = (\beta_i^{(n,m)})_{n,m,i} \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}} \times n_\#}$

Global parameters: $\mu \leftarrow 0$ mean of spatial coefficients,
$\sigma^2 \leftarrow \frac{2}{c_{\text{in}} \cdot K^2}$ variance of spatial coefficients

function `main` (h_{init}, K, c_{out}, c_{in}, $n_\#$) **:**
  if $h_{\text{init}} =$ *`random`* **then**
    return $h_{\text{init}}(K, c_{\text{out}}, c_{\text{in}}, n_\#)$
  else
    return $h_{\text{init}}(K, c_{\text{out}}, c_{\text{in}})$

function `standard` (K, c_{out}, c_{in}) **:**
  Initialize $\mathbf{F}^{(1)}, \dots, \mathbf{F}^{(K^2)} \in \mathbb{R}^{K \times K}$ via $\mathbf{F}^{(k)} = \mathbf{E}^{(i_k, j_k)}$ $\forall k \in \{1, \dots, K^2\}$
  I.i.d. $\beta_i^{(n,m)} \sim \mathcal{N}(\mu, \sigma^2)$ $\forall n \in \{1, \dots, c_{\text{out}}\}, m \in \{1, \dots, c_{\text{in}}\}, i \in \{1, \dots, K^2\}$
  return $\{\mathbf{F}^{(1)}, \dots, \mathbf{F}^{(K^2)}\} \subset \mathbb{R}^{K \times K}$, $(\beta_i^{(n,m)})_{n,m,i} \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}} \times K^2}$

function `ONB` (K, c_{out}, c_{in}) **:**
  Initialize $\tilde{\mathbf{F}}^{(1)}, \dots, \tilde{\mathbf{F}}^{(K^2)} \in \mathbb{R}^{K \times K}$ with $\tilde{\mathbf{F}}_{i,j}^{(k)} \sim \mathcal{N}(0,1)$ i.i.d. $\forall k \in \{1, \dots, K^2\}$
  Apply Gram-Schmidt on $\{\tilde{\mathbf{F}}^{(1)}, \dots, \tilde{\mathbf{F}}^{(K^2)}\}$ to obtain ONB $\{\mathbf{F}^{(1)}, \dots, \mathbf{F}^{(K^2)}\}$
  Initialize flattened spatial coefficients $\bar{\varphi}_i^{(n,m)} \sim \mathcal{N}(\mu, \sigma^2)$ i.i.d.
  $\forall n \in \{1, \dots, c_{\text{out}}\}, m \in \{1, \dots, c_{\text{in}}\}, i \in \{1, \dots, K^2\}$
  Transform $\beta^{(n,m)} = (\mathbf{A}^{(\mathcal{B},\mathcal{F})})^T \cdot \bar{\varphi}^{(n,m)}$ for all n, m by inverting (5.6)
  return $\{\mathbf{F}^{(1)}, \dots, \mathbf{F}^{(K^2)}\} \subset \mathbb{R}^{K \times K}$, $(\beta_i^{(n,m)})_{n,m,i} \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}} \times K^2}$

function `random` (K, c_{out}, c_{in}, $n_\#$) **:**
  Initialize $\tilde{\mathbf{F}}^{(1)}, \dots, \tilde{\mathbf{F}}^{(n_\#)} \in \mathbb{R}^{K \times K}$ with $\tilde{\mathbf{F}}_{i,j}^{(k)} \sim \mathcal{N}(0,1)$ i.i.d. $\forall k \in \{1, \dots, n_\#\}$
  Compute $\bar{\mu}_{i,j} = \frac{1}{n_\#} \sum_{k=1}^{n_\#} \tilde{\mathbf{F}}_{i,j}^{(k)}$, $\bar{\sigma}_{i,j}^2 = \frac{1}{n_\#} \sum_{k=1}^{n_\#} (\tilde{\mathbf{F}}_{i,j}^{(k)} - \bar{\mu}_{i,j})^2$ $\forall i, j \in \{0, \dots, K\}$
  Rescale $\mathbf{F}_{i,j}^{(k)} = \sqrt{\frac{1}{n_\#} - \frac{1}{n_\#^2}} \cdot \frac{\tilde{\mathbf{F}}_{i,j}^{(k)} - \bar{\mu}_{i,j}}{\bar{\sigma}_{i,j}^2} + \frac{1}{n_\#}$ $\forall i, j \in \{0, \dots, K\}, k \in \{0, \dots, n_\#\}$
  I.i.d. $\beta_i^{(n,m)} \sim \mathcal{N}(\mu, \sigma^2)$ $\forall n \in \{1, \dots, c_{\text{out}}\}, m \in \{1, \dots, c_{\text{in}}\}, i \in \{1, \dots, n_\#\}$
  return $\{\mathbf{F}^{(1)}, \dots, \mathbf{F}^{(n_\#)}\} \subset \mathbb{R}^{K \times K}$, $(\beta_i^{(n,m)})_{n,m,i} \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}} \times n_\#}$

Algorithm 5.2: 2D FB Convolution with IP

Input: Input feature map $\mathbf{V}$
Output: Output feature map $\mathbf{W}$
Instance variables: Filter basis $\{\mathbf{F}^{(1)},\dots,\mathbf{F}^{(K^2)}\} \subset \mathbb{R}^{K\times K}$
FB coefficients: $(\beta_i^{(n,m)})_{n,m,i} \in \mathbb{R}^{c_{\text{out}}\times c_{\text{in}}\times K^2}$,
Pruning mask: $(\mathbf{m}_i^{(n,m)})_{n,m,i} \in \{0,1\}^{c_{\text{out}}\times c_{\text{in}}\times K^2}$,
`conv_args`, e.g. stride, padding, groups, ...

function `forward(V)`**:**
  for $m \in \{1,\dots,c_{in}\}, i \in \{1,\dots,K^2\}$ **do**
    $Z_i^{(m)} = \texttt{Conv2D}(\mathbf{F}^{(i)}, \mathbf{V}_{m,:,:}, \texttt{conv\_args})$ `// Standard Conv2D ⋆`
  for $n \in \{1,\dots,c_{out}\}$ **do**
    $\mathbf{W}_{n,:,:} = \sum_{m=1}^{c_{\text{in}}} \sum_{i\in \operatorname{supp}\mathbf{m}^{(n,m)}} \beta_i^{(n,m)} \cdot Z_i^{(m)}$
  return $\mathbf{W} = (\mathbf{W}_{n,:,:})_{n=1}^{c_{\text{out}}}$

5.4.4 Interspace Pruning

The interspace representations (5.5) of $K \times K$ filters can be used to prune FB coefficients instead of spatial ones. Furthermore, (5.7) can be exploited to *reduce* the number of computations for a model which is sparsely represented in the interspace.

For spatial weights, a pruning mask for the model's parameter vector is defined as $\mathbf{m} \in \{0,1\}^D$. Then, the pruned weights are given by $\mathbf{m} \odot \Theta$. Each $K \times K$ filter $\varphi^{(n,m)}$ therefore has a corresponding pruning mask $\mathbf{m}^{(n,m)} \in \{0,1\}^{K\times K}$ where the pruned filter is defined by $\mathbf{m}^{(n,m)} \odot \varphi^{(n,m)}$. Equivalently, a pruning mask for the interspace representation of $\varphi^{(n,m)}$ is defined by $\mathbf{m}^{(n,m)} \in \{0,1\}^{K^2}$ with the pruned FB coefficients given by $\mathbf{m}^{(n,m)} \odot \beta^{(n,m)}$. Therefore, the IP filter represented by $\mathbf{m}^{(n,m)} \odot \beta^{(n,m)}$ is given by

$$\varphi^{(n,m)} = \sum_{i=1}^{K^2} (\mathbf{m}^{(n,m)} \odot \beta^{(n,m)})_i \cdot \mathbf{F}^{(i)} = \sum_{i\in\operatorname{supp}\mathbf{m}^{(n,m)}} \beta_i^{(n,m)} \cdot \mathbf{F}^{(i)} \,. \tag{5.11}$$

By using the derived formula for convolutions with interspace representations, (5.7), sparse filters in the interspace yield sparse computations via

$$\mathbf{W}_{n,:,:} = \sum_{m=1}^{c_{\text{in}}} (\mathbf{m}^{(n,m)} \odot \varphi^{(n,m)}) \star \mathbf{V}_{m,:,:} = \sum_{m=1}^{c_{\text{in}}} \sum_{i\in\operatorname{supp}\mathbf{m}^{(n,m)}} \beta_i^{(n,m)} \cdot \left(\mathbf{F}^{(i)} \star \mathbf{V}_{m,:,:}\right) \,. \tag{5.12}$$

By looking at (5.12), evaluating sparse FB convolutions only requires to sum up all $\beta_i^{(n,m)} \cdot \left(\mathbf{F}^{(i)} \star \mathbf{V}_{m,:,:}\right)$ with corresponding $\mathbf{m}_i^{(n,m)} = 1$. Similar formulas can also be shown in the backward pass [194]. Algorithm 5.2 shows the forward pass of a 2D convolution with sparse interspace coefficients which is called 2D *FB convolution*.

5.4.4.1 Computational Cost Comparison Between IP and SP

As discussed in Section 5.4.3, SP and IP have similar parameter costs if d parameters are trained due to sharing a relatively small number of FBs. However, pruning can also be used to reduce the computational cost of a CNN.

A 2D convolution of an input feature map $\mathbf{V} \in \mathbb{R}^{c_{\text{in}} \times d_1 \times d_2}$ with a filter $\varphi \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}} \times K \times K}$ in spatial representation, pruned with the pruning mask $\mathbf{m}^{(n,m)} \in \{0,1\}^{c_{\text{out}} \times c_{\text{in}} \times K \times K}$ is given for each output channel n by

$$\mathbf{W}_{n,:,:} = \sum_{m=1}^{c_{\text{in}}} (\mathbf{m}^{(n,m)} \odot \varphi^{(n,m)}) \star \mathbf{V}_{m,:,:} \tag{5.13}$$

$$= \left(\sum_{m=1}^{c_{\text{in}}} \sum_{i,j \in \operatorname{supp} \mathbf{m}^{(n,m)}}^{K} \varphi_{i,j}^{(n,m)} \cdot \mathbf{V}_{m,i+a,j+b} \right)_{a,b} \in \mathbb{R}^{d_1-K+1 \times d_2-K+1} . \tag{5.14}$$

Let $\tilde{d_1} = d_1 - K + 1$ and $\tilde{d_2} = d_2 - K + 1$ denote the output size of the feature maps. Counting each individual addition and multiplication as one `FLOP`, evaluating a convolutional layer with sparse weight tensor $\varphi \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}} \times K \times K}$ and input feature map $\mathbf{V} \in \mathbb{R}^{c_{\text{in}} \times \tilde{d_1} \times \tilde{d_2}}$ requires $2 \cdot (1-q) \cdot c_{\text{out}} \cdot c_{\text{in}} \cdot K^2 \cdot \tilde{d_1} \cdot \tilde{d_2}$ `FLOPs` in spatial representation. Note, this is only a lower bound for the actually required `FLOPs` since additional work has to be done for finding the indices of the non-zero parameters in real life applications.

Evaluating a 2D convolution with interspace representation first requires the computation of all $\mathbf{F}^{(i)} \star \mathbf{V}_{m,:,:}$, $i \in \{1,\dots,K^2\}$, $m \in \{1,\dots,c_{\text{in}}\}$, regardless of the compression rate q. This generates a computational overhead of $2 \cdot c_{\text{in}} \cdot K^4 \cdot \tilde{d_1} \cdot \tilde{d_2}$ `FLOPs` compared to evaluating convolutions in the spatial domain. After the $\mathbf{F}^{(i)} \star \mathbf{V}_{m,:,:}$ are computed, using (5.12) shows that the remaining `FLOPs` are the same as for computing the convolution in spatial representation, *i.e.* $2 \cdot (1-q) \cdot c_{\text{out}} \cdot c_{\text{in}} \cdot K^2 \cdot \tilde{d_1} \cdot \tilde{d_2}$ `FLOPs`. In total, computing a sparse 2D convolution in the interspace representations by (5.12) requires $2 \cdot c_{\text{in}} \cdot K^2 \cdot \tilde{d_1} \cdot \tilde{d_2} \cdot (c_{\text{out}} \cdot (1-q) + K^2)$ `FLOPs` – which again should be seen as a lower

bound for FLOPs in real life applications.

The relative increase in computational costs for sparse FB convolutions compared to sparse standard convolutions is therefore given by

$$\frac{2 \cdot c_{\text{in}} \cdot K^2 \cdot \tilde{d}_1 \cdot \tilde{d}_2 \cdot (c_{\text{out}} \cdot (1-q) + K^2)}{2 \cdot (1-q) \cdot c_{\text{out}} \cdot c_{\text{in}} \cdot K^2 \cdot \tilde{d}_1 \cdot \tilde{d}_2} = 1 + \frac{K^2}{c_{\text{out}} \cdot (1-q)} \,. \tag{5.15}$$

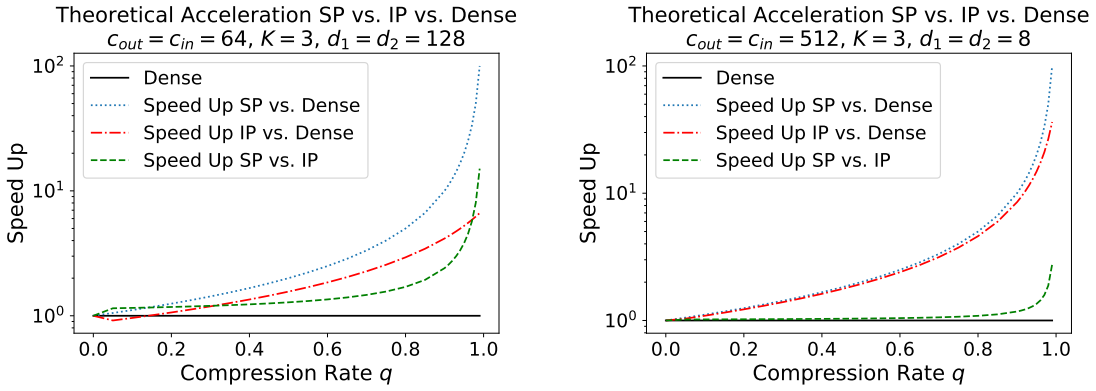
Figure 5.4 shows this relative increase in computational costs together with the relative increase compared to dense convolutions. As shown, IP is always a little bit slower than SP. Especially for high compression rates q, the spatial convolutions are significantly faster than those evaluated with interspace representations. Also, for layers where the feature maps have a high resolution, like the first layer of a network, IP is slower than SP. In both cases, the constant overhead corresponding to the computation of all $\mathbf{F}^{(i)} \star \mathbf{V}_{m,:,:}$ is relatively big compared to the computational costs of the spatial convolution. It should be noticed that IP can even increase the computational costs of the dense layer for small compression rates.

However, for almost all compression rates, IP shows a significant speed up compared to the dense model. Also, as shown in Section 5.5, IP and SP have quite similar runtimes in practice since the computed lower bounds for required FLOPs is not reached in reality. This is due to the computational overhead induced by indexing the non-zero coefficients which is similar for IP and SP. Experimental results in Section 5.5 further show that IP achieves better results than SP for equal runtimes.

A comparison for the computational costs in the backward pass can be found in Section D.2 in Wimmer et al. [194] and shows resembling results as discussed here for the forward pass.

5.5 Experimental Results and Discussion for Interspace Pruning

The pruning methods which are used to compare IP and SP are first described in Section 5.5.1. Then, IP and SP are evaluated for numerous SOTA pruning methods applied before, during or after training in Section 5.5.2. The comparison is done w.r.t. similar compression rates and runtimes. Afterwards, different FB sharing and interspace initialization schemes are evaluated in Section 5.5.3. Section 5.5.4 analyzes different sizes of the spanning system $\mathcal{F} \subset \mathbb{R}^{K \times K}$ of the interspace. The improved trainability and



(a) (b)

Figure 5.4: Theoretical speed ups compared for IP and SP with the dense baseline for two scenarios. Scenario (a) corresponds to a a layer close to the input with fewer channels but higher resolution whereas (b) shows a layer close to the output with lower resolution but more channels.

generalization ability by using interspace representations is discussed in Section 5.5.5. Finally, the interspace representation is also tested for freezing weights via FreezeNet [192] in Section 5.5.6.

IP and SP are compared for a VGG-16 [168] on CIFAR-10 [85], as well as for ResNet18 and ResNet50 [64] on ImageNet ILSVRC2012 [28]. The models are trained with the cross entropy loss function. Results on CIFAR-10 are reported as the mean $\pm$ one standard deviation over 5 runs whereas results for ImageNet are generated by 3 runs. For each run, a different random seed is chosen. Note, ResNets have 1×1 convolutions. For these 1×1 convolutions, no interspace representation is used for IP, *i.e.* the FBs are fixed as the standard 1×1 basis $\mathcal{F} = \mathcal{B}$. Moreover, the used ResNets have 7×7 convolutions in the first layer. For the small ResNet18, again, the FBs for the 7×7 filters are fixed as the standard basis. However, the ResNet50 has a trainable FB for the 7×7 convolutions. All remaining convolutional layers are 3×3 convolutions which have a trainable FB for the interspace setting. The results for IP are obtained by `fine` sharing for the ResNet50, `medium` sharing for the CIFAR-10 experiments and `coarse` sharing for ResNet18. Of course, standard models are used for the SP results and the dense baseline. If not mentioned otherwise, models for SP and IP are initialized with the `standard` initialization from Algorithm 5.1. The used hyperparameters are

provided in Table A.11 in the Appendix, Section A.5. Like before for FreezeNet in Section 3 and COPS in Section 4, results for IP are obtained by using the training setup from the compared SP methods. In particular, FBs are trained without specialized hyperparameters. Therefore, the learning rate for FBs and FB coefficients is chosen the same as for standard training. As common in the literature, ImageNet results are reported on the validation set

5.5.1 Conducted Pruning Methods

Since IP is a method that can be applied on top of arbitrary unstructured pruning methods, different pruning methods are compared for SP and IP. Even though all tested pruning criteria were developed for pruning spatial weights, *i.e.* for SP, they are applied *unchanged* to the interspace setting. As shown in Section 5.5.2, using IP still improves SP significantly on these underlying pruning methods. The conducted pruning criteria are given by:

Pruning at Initialization. PaI trains a sparse network with a fixed topology and is introduced in Section 2.3.1. The tested PaI methods are SNIP [94], GraSP [184] and SynFlow [174] together with random PaI.

Dynamic Sparse Training. DST is also summarized in Section 2.3.1. DST starts with a randomly initialized pruning transformation $\psi^{(0)}$. During training, $\psi^{(t)}$ is updated after each t_p iterations. IP and SP are compared on two DST methods, RigL by Evci et al. [40] and SET by Mocanu et al. [122]. After t_p steps, both methods use magnitude pruning to prune insignificant weights. For SET, randomly chosen weights are regrown. RigL on the other hand uses the magnitude of the gradient in each layer to measure the significance of previously zeroed weights. Those weights with high gradient magnitude are then regrown. Hereby, the number of regrown weights is equal to the number of newly pruned weights for each layer. The update frequency is set as $t_p = 1,500$ for SET and $4,000$ for RigL. Furthermore, a cosine schedule is used to reduce the number of pruned/regrown weights to further stabilize training [40].

Lottery Ticket Hypothesis. An introduction to LTH is given in Section 2.3.1. LTH experiments are conducted by rewinding the weights not to their initialization, but pre-trained weights at training step $t_0 = 500$ as suggested by Frankle et al. [44]. After that, the network is trained to convergence, 20% of the parameters are pruned and the remaining ones are reset to their values at step t_0. This procedure is applied iteratively until the desired compression rate q is reached. Then, the sparse model with sparsity q is trained a final time.

Gradual Magnitude Pruning. Gradual magnitude pruning (GMP) [49] increases the sparsity in the network step by step. Training is started with a dense network. The compression rate is updated each t_p iterations after pre-training the dense network for t_0 epochs. The compression rate at iteration $k \cdot t_p$ is given by

$$\begin{cases} 0, & k \cdot t_p < t_0 \\ q \cdot \left(1 - \left(1 - \frac{k \cdot t_p - t_0}{t_1 - t_0}\right)^3\right), & k \cdot t_p \in [t_0, t_1] \\ q, & k \cdot t_p > t_1 \end{cases}, \tag{5.16}$$

where q is the desired, final compression rate. To guarantee convergence after training, pruning is ended after t_1 iterations. The weights are pruned based on their magnitude. Therefore, if a weight is pruned once, it will never regrow. Note, this is not a DRT method since the dimensionality of the submanifold spanned by $\psi^{(t)}$ is not constantly d, but bigger during most parts of the training. The choices for t_p, t_0, t_1 follow the suggestions in Gale et al. [49] adapted to the used batch size. This results in $t_p = 8,000$, $t_0 = 160,000$ and $t_1 = 400,000$ for the compression rate $q = 0.8$. For $q = 0.9$, the values are chosen as $t_p = 8,000$, $t_0 = 160,000$ and $t_1 = 304,000$.

Fine-Tuning. Fine-tuning (FT) prunes a pre-trained model [153]. The weights are pruned one-shot based on their magnitudes. After pruning, the non-zero coefficient – and the FBs in the case of IP – are fine-tuned with the learning rate schedule which was previously used for the dense training. As shown in Table A.11, FT uses the same training setup as RigL.

5.5.2 Experimental Results for Interspace Pruning

In this Section, IP is compared with SP for different pruning methods applied before, during and after training on the image classification tasks CIFAR-10 and ImageNet. For the comparison, a VGG-16 is trained for CIFAR-10 while a ResNet18 and a ResNet50 are used for the ImageNet results.

5.5.2.1 Pruning at Initialization

Since FBs are trained jointly with their coefficients, IP is especially helpful for dimensionality reduced training. Therefore, IP and SP are first of all compared for the three PaI methods SNIP [94], GraSP [184] and SynFlow [174] together with random PaI. Figure 5.5 shows the results for a VGG-16 trained on CIFAR-10 and a ResNet18 trained on ImageNet.

The experiments show that IP outperforms SP on all tested PaI methods significantly for both, CIFAR-10 and ImageNet. The improvements are bigger for higher compression rates. Compared to CIFAR-10, IP shows more improvements for ImageNet. Therefore, IP seems to improve SP for PaI methods especially in challenging situations, *i.e.* pruning to high sparsity or pruning for complicated tasks like ImageNet. Note, for ImageNet, all three SOTA methods SNIP, GraSP and SynFlow are outperformed by random PaI. Consequently, PaI is able to achieve good results for more simple tasks like CIFAR-10 while failing for complicated ones like ImageNet. However, applying IP to these pruning criteria still improves performance in all situations significantly as discussed before.

IP trains FBs and FB coefficients jointly. As shown in Figure 5.6(a), the joint training does not induce instability in the training procedure. Therefore, the FBs $\mathcal{F}$ and the coefficients β converge stably while the convergence behavior of β and the spatial coefficients φ is similar.

Figure 5.6(b) even shows that the adaptivity of FBs helps to overcome weak information flow in the model. As can be seen, SP-SNIP has zero gradient flow during the whole training process for $q = 0.995$. IP-SNIP starts at the exact same position. Due to the adaptivity of the FBs, the interspace is adapted such that the gradient flows in the sparse architecture again. Therefore, IP leads to better trainable models, especially for high compression rates. This also explains the much better results of IP for high compression rates q where information flow is a bottleneck in the sparse architecture.

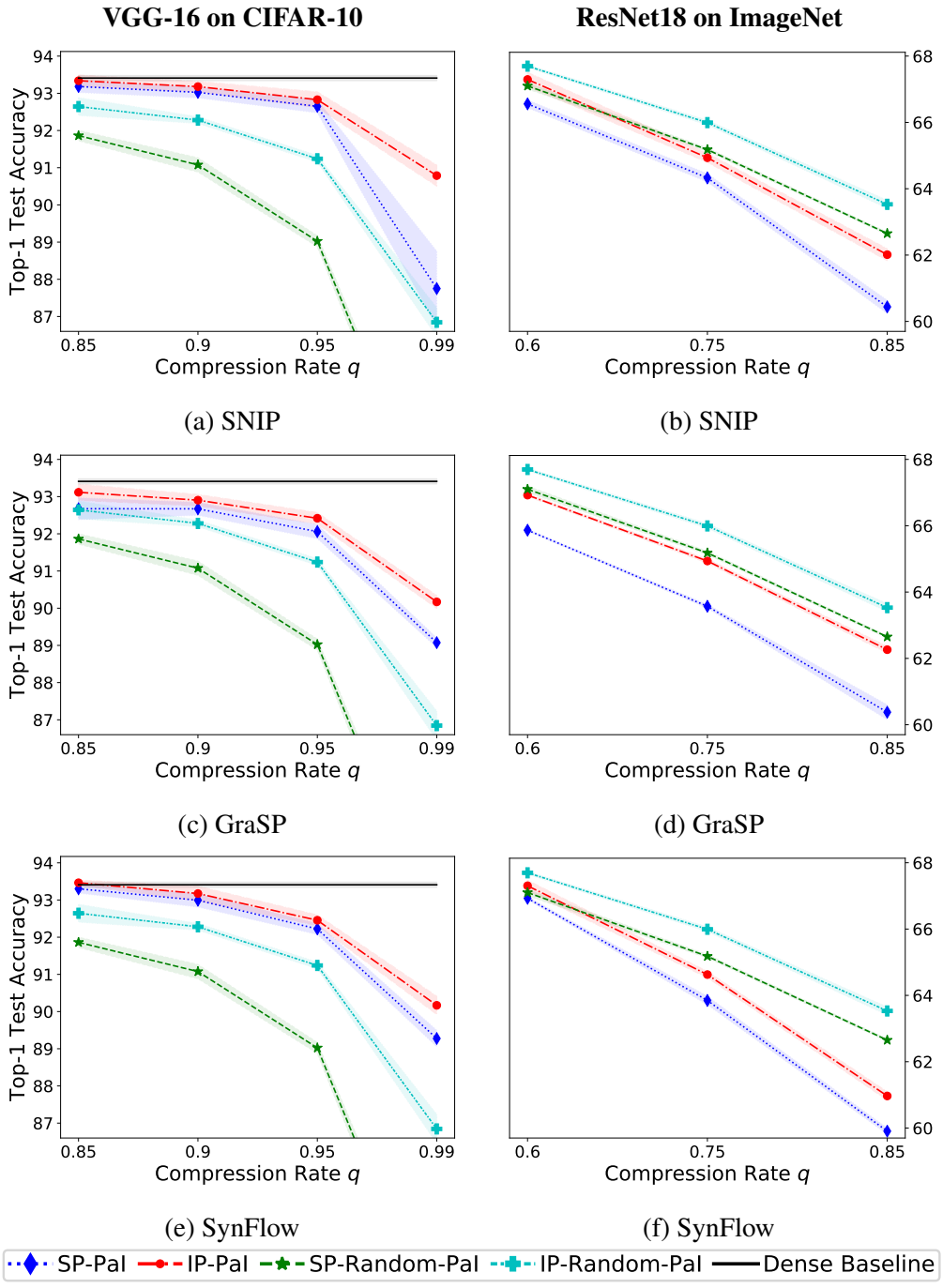
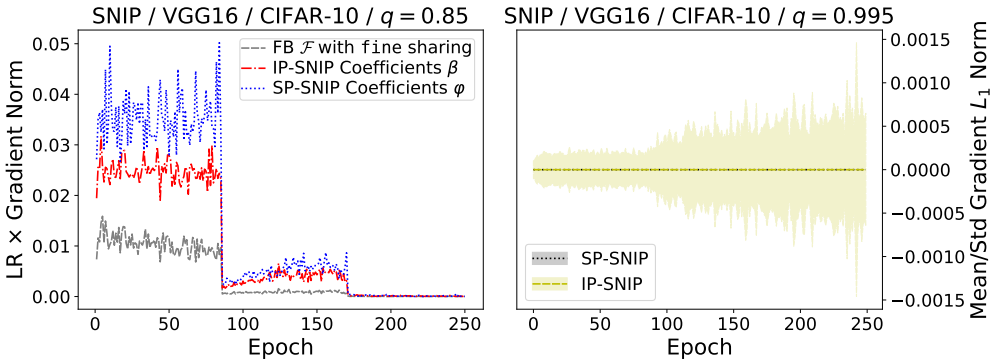


Figure 5.5: (Figure 3 in Wimmer et al. [194]) Comparing SP and IP for PaI methods SNIP, GraSP and SynFlow together with random PaI for CIFAR-10 and ImageNet.



(a) (b)

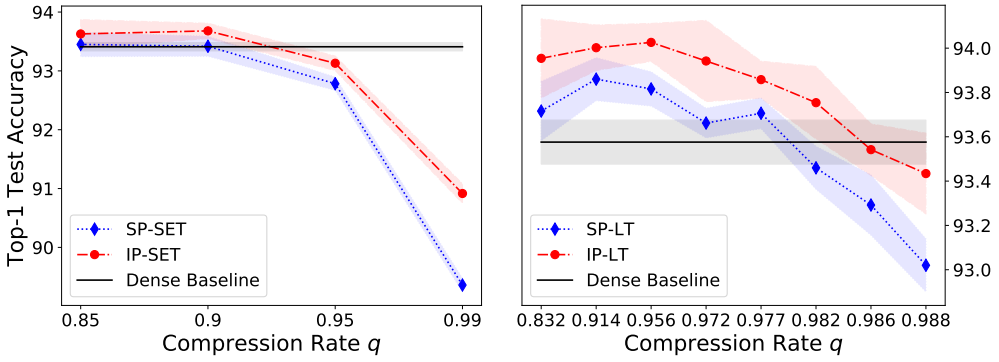
Figure 5.6: Both plots show results for SNIP with a VGG-16 on CIFAR-10. (a)(Figure 5(b) in Wimmer et al. [194]) Shows Gradient L_2 norm $\times$ the learning rate (LR) for spatial coefficients φ, FB coefficients β and FBs $\mathcal{F}$ for `fine` sharing. (b)(Figure A6(a) in Wimmer et al. [194]) Shows mean and one standard deviation of the L_1 norm of the gradient of the coefficients for IP- and SP-SNIP for $q = 0.995$.

5.5.2.2 Dynamic Sparse Training and Lottery Ticket Hypothesis

As shown by the results in Figure 5.5, PaI methods are not best suited to prune models for difficult problems. Therefore, IP and SP are now also compared on more sophisticated DRT methods, namely on dynamic sparse training and the Lottery Ticket Hypothesis. These methods have shown improved performance compared to the SOTA PaI methods in the SP setting [40, 44, 104]. Note, the VGG-16-LTH used for LTH is slightly smaller than the VGG-16 for PaI and DST since the VGG-16-LTH only has one instead of three fully connected layers. For comparing IP and SP, the DST method SET and the LTH with rewinding weights to the 500$^{\text{th}}$ epoch are used. Results can be seen in Figure 5.7.

Again, IP improves results significantly compared to SP for both, SET and LTH. For LTH, IP needs to train 3.7 times less parameters ($q = 0.977$) than SP to achieve SP's best result for $q = 0.914$. Moreover, IP matches the dense baseline while training only 1.4% of all parameters in the network. For lower compression rates, IP even improves results compared to the dense baseline.

Similar results also hold for SET, where IP is able to outperform the dense baseline for $q \leq 0.9$. However, SET shows worse results than LTH for high compression rates $q \geq 0.95$. Here, carefully choosing the sparse architecture via iterative – but expensive – train-prune-reset cycles shows its strength.



(a) SET for VGG-16 on CIFAR-10 (b) LTH for VGG-16-LTH on CIFAR-10

Figure 5.7: (Figure 4 in Wimmer et al. [194]) Comparison between SP and IP for SET and LTH with rewinding weights to iteration 500, both trained on CIFAR-10.

Table 5.1: (Table 1 in Wimmer et al. [194]) SOTA pruning methods for ResNet50 trained on ImageNet for 100 epochs.

	Top-1 Accuracy for ResNet50 on ImageNet		
Method	$p = 0.0$	$p = 0.8$	$p = 0.9$
SP-FT	77.15 ± 0.04	77.02 ± 0.03	75.67 ± 0.09
IP-FT	$\mathbf{77.30 \pm 0.04}$	$\mathbf{77.18 \pm 0.01}$	$\mathbf{75.89 \pm 0.09}$
SP-GMP	76.64 ± 0.06	75.37 ± 0.01	73.57 ± 0.06
IP-GMP	$\underline{77.16 \pm 0.04}$	$\underline{75.71 \pm 0.05}$	$\underline{74.20 \pm 0.07}$
SP-RigL	77.15 ± 0.04	75.75 ± 0.10	73.88 ± 0.06
IP-RigL	$\mathbf{77.30 \pm 0.04}$	$\underline{76.03 \pm 0.08}$	$\underline{74.32 \pm 0.11}$

In summary, spending more effort in choosing the sparse architecture via DST or LTH shows improved performance compared to pruning a model directly at initialization.

5.5.2.3 SOTA Pruning Methods on a ResNet50

In Table 5.1, SOTA pruning methods are compared for a ResNet50 trained on ImageNet. For the comparison, the DST method RigL [40] with improved training setup provided by Liu et al. [104], gradual magnitude pruning during training [49] and FT [153] which prunes a pre-trained model and fine-tunes the sparse model afterwards are compared for SP and IP.

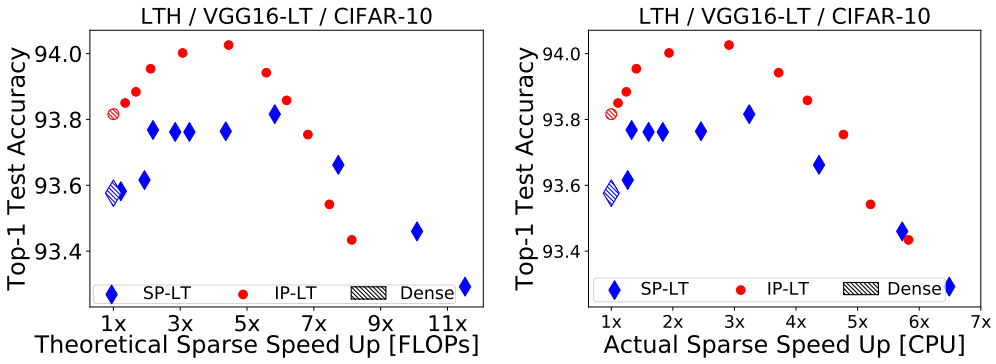
As can be seen, IP outperforms SP on all underlying pruning methods. Notably, the results are improved significantly despite the fact that more than 50% of the parameters in a ResNet50 correspond to 1×1 convolutions which are trained equivalently for IP and SP. Moreover, FT outperforms the other methods for all q where IP-FT shows similar results than dense training for a compression rate $q = 0.8$. This demonstrates that SOTA DRT methods like RigL are not able to match the performance of pre-training a dense model, prune it and fine-tune it afterwards – which is of course more expensive due to the need of a dense pre-training. Finally, using the interspace representation does not only improve sparse models, but also dense training. This is confirmed by comparing the dense baselines, *i.e.* $q = 0$, for IP and SP. In Section 5.5.5, the improved generalization ability obtained by using interspace representations will be discussed in more detail.

5.5.2.4 Computational Costs

So far, IP and SP have only been compared in terms of their performance at different compression rates, *i.e.* memory costs. The results have revealed that IP improves SP with similar memory requirements, or equivalently, IP can reduce memory costs compared to SP while having a similar generalization ability. However, pruning is not only used for lowering the memory costs, but also to reduce the computational costs for the models. As discussed in Section 5.4.4 and shown in Figure 5.4, IP and SP both are able to reduce the `FLOP` count compared to the dense baseline. However, IP has a higher `FLOP` cost than SP for a similar memory footprint.

Figure 5.8 shows the test accuracy over speed up for IP- and SP-LTH compared to the dense baseline. Figure 5.8(a) provides the *theoretical* acceleration, calculated according to Section 5.4.4, while Figure 5.8(b) shows the *actual* acceleration. The actual acceleration is obtained by using the `sparse` Python package provided by `scipy`. The sparse coefficient tensors in a layer are converted to `scipy.sparse.csr_matrix` objects.[7] The actual speed up is measured on one core of an Intel XEON E5-2680 v4 2.4 GHz CPU with batch size 1 and the mean runtime of 25 runs. A full description for the measurements of the actual speed up is provided in Section D.4 in Wimmer et al. [194]. For results in Figure 5.8, the acceleration is computed/measured for the model at inference time.

[7] Documentation: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html`

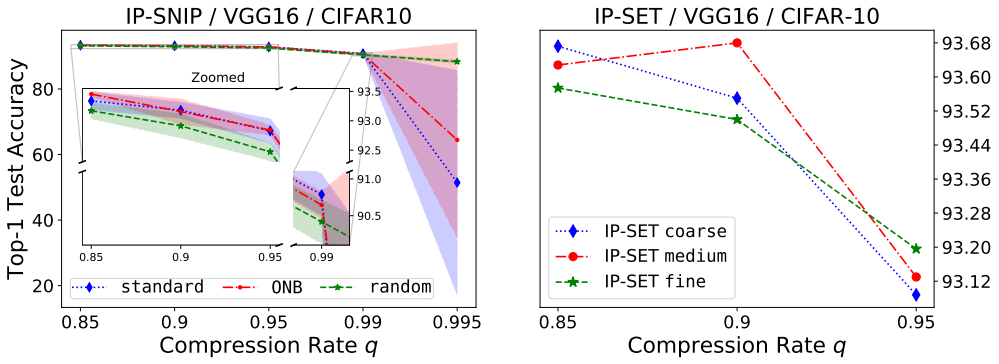


(a) (b)

Figure 5.8: (Figure 5(a) in Wimmer et al. [194]) Theoretical (a) and actual (b) sparse speed up for IP- and SP-LTH measured via `FLOPs` and runtime on a CPU, respectively. Results are obtained by a VGG-16-LTH trained on CIFAR-10.

In theory, IP has a constant computational overhead compared to SP which would make SP much faster for high compression rates. This can also be seen in Figure 5.8(a) where SP has better results than IP for a speed up > 7 times. However, for smaller accelerations, IP again greatly exceeds SP. Moreover, the actual computational overhead of IP is not as pronounced as in theory, see Figure 5.8(b). The theoretical speed up is of course faster than the practical one. This is because of a computational overhead of the used CSR representation and derived sparse computations [177]. The additional overhead is due to handling the indexing of non-zero coefficients together with indexing the corresponding entries in the input feature maps (for SP) or corresponding $\mathbf{F}^{(i)} \star \mathbf{V}_{m,:,:}$ (for IP). In practice, this overhead seems to be more dominant than the computational overhead induced by the interspace representation.

In summary, IP shows better results than SP for a similar runtime measured on a CPU. Moreover, IP reaches equal results than the dense network while speeding up computations up to 5.2 times. In theory, however, SP achieves better performance than IP for high speed ups due to the constant computational overhead induced by the interspace representation.



(a) (b)

Figure 5.9: (a) (Figure A1(a) in Wimmer et al. [194]) `Standard`, `ONB` and `random` initializations for IP-SNIP for a VGG-16 trained on CIFAR-10. (b) (Figure 2(b) in Wimmer et al. [194]) Shows `coarse`, `medium` and `fine` sharing of FBs for IP-SET with a VGG-16 trained on CIFAR-10.

5.5.3 Initializing and Sharing Filter Bases

As discussed in Section 5.4.3, different initialization and FB sharing schemes can be used for IP. Now it will be analyzed which sharing and initialization scheme works best in a given situation.

5.5.3.1 Initializing the Interspace

Figure 5.9(a) compares the three different initializations provided in Algorithm 5.1. As can be seen, `standard` initialization and `ONB` initialization show similar performance. Therefore, the standard basis is not a special ONB but only one of many. For the highest tested compression rate, the `ONB` initialization even shows better results since it starts with dense FBs while using the `standard` initialization yields a sparse basis at the beginning of training. Since information flow is a problem for such high compression rates, using a randomly initialized and therefore dense ONB improves results here.

Moreover, a model with a `random` initialization shows significantly better results for the highest compression rate $q = 0.995$. However for lower rates, `random` initialization is worse than initializing FBs as an ONB. A randomly initialized basis has elements which are more similar to another than basis elements of an ONB. Therefore, a randomly initialized basis has more redundancy than an ONB. This redundancy can be seen as an

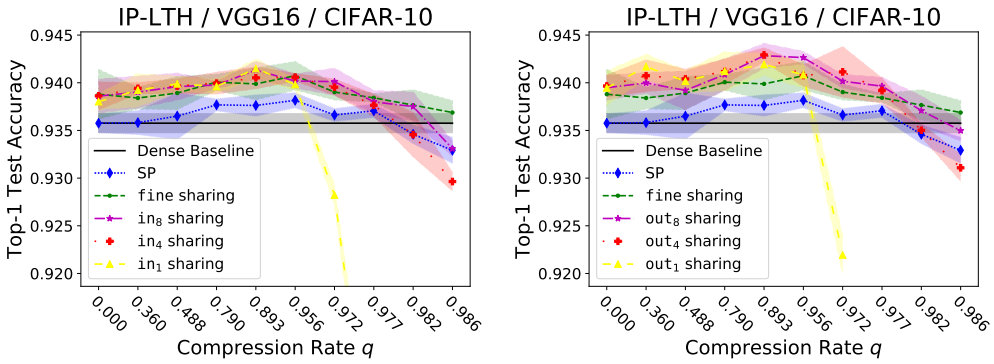
Table 5.2: (Table 3 in Wimmer et al. [194]) Test accuracies for SP and IP with varying FB sharing schemes for lower pruning rates q. Note, for all q, SP- and IP-SNIP with all sharing schemes achieve $\approx 100\%$ training accuracy.

	Compression rate q			
	0.0	0.35	0.6	0.85
	SNIP			
SP	93.4 ± 0.1	93.4 ± 0.1	93.3 ± 0.2	93.2 ± 0.2
IP-`coarse`	$\mathbf{93.9 \pm 0.2}$	$\mathbf{93.7 \pm 0.2}$	$\mathbf{93.8 \pm 0.1}$	$\mathbf{93.5 \pm 0.0}$
IP-`medium`	$\mathbf{93.9 \pm 0.2}$	93.6 ± 0.2	93.7 ± 0.2	93.3 ± 0.2
IP-`fine`	93.7 ± 0.1	93.3 ± 0.2	93.3 ± 0.2	93.2 ± 0.1
	SET			
SP	93.4 ± 0.1	93.5 ± 0.2	93.3 ± 0.2	93.5 ± 0.2
IP-`coarse`	$\mathbf{93.9 \pm 0.1}$	$\mathbf{93.9 \pm 0.2}$	$\mathbf{93.8 \pm 0.1}$	$\mathbf{93.7 \pm 0.2}$
IP-`medium`	$\mathbf{93.9 \pm 0.1}$	93.7 ± 0.2	$\mathbf{93.8 \pm 0.1}$	93.6 ± 0.2
IP-`fine`	93.7 ± 0.1	93.6 ± 0.2	93.6 ± 0.2	93.6 ± 0.2

implicit dimensionality reduction of the space spanned by the FB. For a low compression rate, only a few coefficients are non-zero. With only a few coefficients, it is easier to train in a low dimensional interspace than in a higher one as shown and discussed in Section 5.5.4. However, if more coefficients are available, results in a higher dimensional space are better since more filters can be expressed here.

5.5.3.2 Sharing Filter Bases

In the experimental evaluation in Section 5.5.2, different FB sharing schemes are used for varying dataset and model combinations. The intuition for choosing a FB sharing method was given in Section 5.4.3 by "`coarse` sharing works best for lower compression rates while `fine` sharing should be used for high q". Figure 5.9(b) and Table 5.2 show that the intuition derived in Section 5.4.3 is indeed true. As can be seen, `coarse` sharing has the best results for smaller q and `fine` sharing works best for the highest compression rate. In between, `medium` sharing shows the best performance.



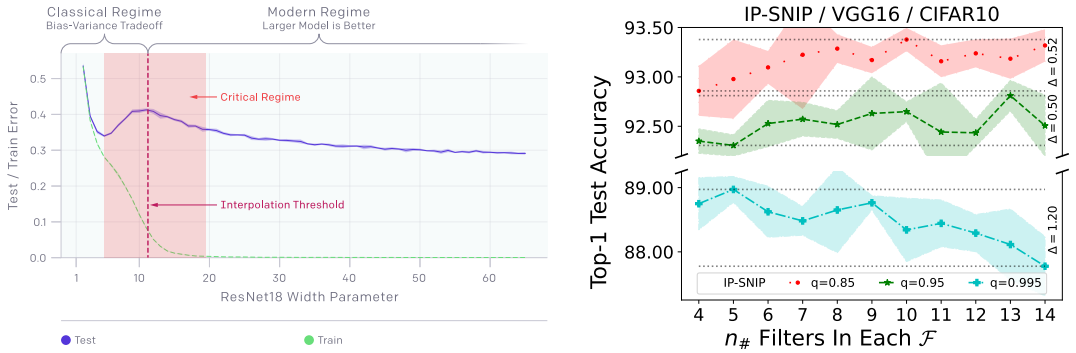
(a) (b)

Figure 5.10: Compares SP and IP with `fine` sharing for LTH with (a) $\texttt{in}_{n_c}$ sharing and (b) $\texttt{out}_{n_c}$ sharing for a VGG-16-LTH trained on CIFAR-10.

5.5.3.3 Sharing More Filter Bases

Up to now, a small number of FBs was used to keep the number of trained parameters in the model as low as possible. However, it might be beneficial to use more than one FB per layer to strengthen the model's expressibility. Figure 5.10 shows results for sharing a FB for each n_c input channels in (a) or output channels in (b). Note, if $n_c \geq c_{\text{in}}$ (or $n_c \geq c_{\text{out}}$), then one FB is shared for the corresponding layer. Therefore, the proposed FB sharing scheme is equal to `fine` sharing if n_c is greater or equal to the maximum number of channels in a model. Sharing a FB for n_c input channels is called $\texttt{in}_{n_c}$ sharing while sharing a FB for n_c output channels is called $\texttt{out}_{n_c}$ sharing. By looking at the computational costs for FB convolutions in Section 5.4.4 it becomes clear that the computation overhead of IP compared to SP remains the same for $\texttt{in}_{n_c}$ sharing, while it is multiplied by $\lceil \frac{c_{\text{out}}}{n_c} \rceil$ if the FBs are shared between output channels. This increase in `FLOPs` for sharing filters between output channels should be kept in mind.

Using $\texttt{out}_1$ and $\texttt{in}_1$ sharing for a VGG-16 yields up to 512 FBs per layer. Of course, this induces a massive amount of extra parameters. To obtain fair comparisons, the compression rates for Figure 5.10 take all FB parameters into account, *i.e.* all methods have the same number of non-zero parameters (including FBs) for a similar compression rate q. As a result, using $n_c = 1$ massively drops performance around $q = 0.972$ because the number of trained FB coefficients is too small to achieve good results. For $\texttt{out}_{n_c}$ and $\texttt{in}_{n_c}$, FBs are always shared between neighboring channels, starting with channel



(a) (b)

Figure 5.11: (a) (From the OpenAI blog post [129]) Shows the deep double decent phenomenon for a ResNet18 with increasing width. (Figure A5(a) in Wimmer et al. [194]) Different numbers $n_\#$ of filters in $\mathcal{F}$ spanning the interspace for IP-SNIP with a VGG-16 trained on CIFAR-10.

index 1.

As can be seen, using the $\texttt{in}_{n_c}$ sharing method does not significantly improve results. Therefore, increasing the number of FBs does not automatically lead to better results. However, by looking at $\texttt{out}_{n_c}$, it can be seen that increasing the number of FB can improve results significantly, *e.g.* for $q = 0.893$. Nevertheless, the required number of FLOPs is increased by using $\texttt{out}_{n_c}$ sharing of FBs.

Notably, using $\texttt{out}_{n_c}$ even improves results for low compression rates $q \leq 0.5$ which can be explained by the *deep double decent phenomenon* [130]. The double descent phenomenon states that deep learning methods are able to overcome the degrading generalization ability in the bias-variance trade off induced by high variance *if* the model complexity is chosen big enough, see Figure 5.11(a). Up to some point, the increased variance term in the bias-variance trade off still decreases the generalization ability, but if this critical point is exceeded, the classical bias-variance trade off is not valid anymore and the generalization ability increases again. In the end, this leads to a model that generalizes better than the one found at the sweet spot of the classical bias-variance trade off – at the cost of a huge number of parameters and FLOPs.

5.5.4 Generalizing Filter Bases to Dictionaries

The interspace is a linear space spanned by $\mathcal{F} \subset \mathbb{R}^{K\times K}$. Up to now, $\mathcal{F}$ was restricted to form a basis of the K^2 dimensional space. However, choosing a $\mathcal{F}$ with $n_\# = \#\mathcal{F} > K^2$ or $\#\mathcal{F} < K^2$ is possible. The former defines an overcomplete system while the latter defines an undercomplete system. Using $\mathcal{F}$ as such systems is compared for a VGG-16 in Figure 5.11(b) for $n_\# \in \{4,5,\dots,14\}$. Note $n_\# = 9$ corresponds to $\mathcal{F}$ forming a basis, since convolutions in a VGG-16 have 3×3 kernels. The compression rates in Figure 5.11(b) correspond to the compression rates for $n_\# = 9$. Other choices of $n_\#$ have the same number of non-zero parameters for this q. Therefore, $n_\# < 9$ leads to less sparse architectures while $n_\# > 9$ leads to higher sparsity. The $\mathcal{F}$ and its coefficients are initialized with a `random` initialization. The reported results are those with the best validation accuracy among `coarse`, `medium` and `fine` sharing.

Results show that using more than 9 filters in a $\mathcal{F}$ improves results if the number of trained coefficients is not too low, *e.g.* $n_\# = 10$ for $q = 0.85$ or $n_\# = 13$ for $q = 0.95$. Since the initial $\mathcal{F}$ are chosen by random, results might be improved by using more sophisticated methods to initialize them. However, if the number of trained parameters is low, using overcomplete $\mathcal{F}$ degrades performance. The coefficients w.r.t. interspaces spanned by overcomplete systems are sparser than w.r.t. interspaces with $n_\# \leq 9$. Therefore, the network can not train well anymore if the number of trained parameters is low and $n_\# > 9$.

Consequently, using less than 9 filters even improves results for a low number of trained coefficients since the coefficients in this interspace are more dense, like $n_\# = 5$ for $q = 0.995$. Notably, SP-SNIP is not able to train for such a compression rate. However, undercomplete $\mathcal{F}$ show worse results for low compression rates. The expressibility of these models is too small since the corresponding interspaces are low dimensional subspaces. Therefore, not all convolutional filters can be expressed in such a space.

In summary, generalizing filter bases to filter dictionaries might be a good option, especially for high compression rates where undercomplete $\mathcal{F}$ show better or equal results than using filter bases with the same number of non-zero coefficients. By again looking at the computational costs for IP derived in Section 5.4.4, the computational overhead of IP compared to SP is scaled by $n_\#/K^2$ if the number of filters in a FB is equal to $n_\#$ which is another reason to use undercomplete $\mathcal{F}$.

Table 5.3: (Table 2 in Wimmer et al. [194]) Generalization gaps for various pruning methods.

VGG-16 on CIFAR-10				
	$q = 0.85$		$q = 0.99$	
Method	Train	Test	Train	Test
SP-SET	99.85	93.45	94.20	89.36
IP-SET	99.89	**93.63**	**96.89**	**90.92**
SP-SNIP	99.94	93.18	93.96	87.75
IP-SNIP	99.96	**93.34**	**98.38**	**90.79**

ResNet50 on ImageNet				
	$q = 0.8$		$q = 0.9$	
Method	Train	Test	Train	Test
SP-RigL	74.64	75.75	71.30	73.88
IP-RigL	**75.39**	**76.03**	**72.08**	**74.32**

5.5.5 Generalization Ability of Interspace Representations

So far, it was experimentally shown that IP improves results compared to SP for various underlying SOTA pruning methods, CNN architectures and classification tasks. Now, it is time to have a closer look why IP improves results compared to SP for low and high compression rates. In Section 2.1.2, the *generalization* ability of a network is defined as the ability to perform well on unseen data. In general, it is not possible to measure the performance of a model on the whole set of unseen data, since the whole dataset is usually unknown, often uncountable and therefore not available in its entirety. Consequently, the generalization ability of a network is measured on the relationship between the model's performance on the training and the test dataset. Ideally, the performance on the train set should be optimal and a strong indicator for the performance on the test set. A metric to determine the relationship between train and test set is given by the *generalization gap*, the difference between performance on the train and test data.

Table 5.3 shows the correspondence between train and test accuracy for different tested pruning methods for IP and SP, namely SET [122], SNIP [94] and RigL [40]. Results for SET and SNIP are obtained by a VGG-16 trained on CIFAR-10 while RigL is trained on ImageNet with a ResNet50. The training scheme used for ImageNet is more regularized than the one for CIFAR-10, see Table A.11. Therefore, the performance for SP- and IP-RigL is even better on the test set than on the training set.

For ImageNet and $q = 0.99$ for CIFAR-10, IP has a bigger generalization gap than SP. This is due to a significantly higher training performance. Therefore, IP yields better trainable models for complicated situations like high compression rates or training on

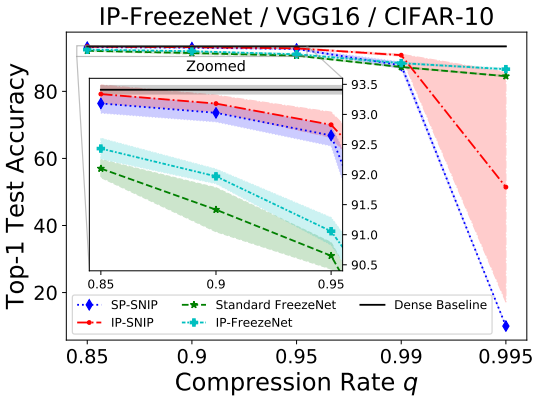


Figure 5.12: (Figure A5(b) in Wimmer et al. [194]) Comparing FreezeNet with SP- and IP-SNIP as well as freezing FB coefficients – IP-FreezeNet

ImageNet. Despite a bigger generalization gap, IP achieves better results than SP on the test set.

Moreover, IP shows a lower generalization gap in situations where the model overfits on the training data, like $q = 0.85$ for CIFAR in Table 5.3. This finding is confirmed by the results in Table 5.2. A compression rate $q = 0$ corresponds to dense training while SP with $q = 0$ is the dense baseline. The significantly improved performance of IP compared to SP for dense training can not be explained by IP's superior expressibility, Theorem 5.3.1, as both representations are able to represent exactly the same. However, by having a closer look on the results in Table 5.2, it can be seen that `coarse` FB sharing achieves the best results for all $q \leq 0.85$. An explanation for the superior results with `coarse` sharing is the fact that one global FB correlates *all* $K \times K$ filters in a network by using and updating the same FB. The opposing effects on the FB's gradient (5.10) through influences of different filters φ regularizes training which, in the end, leads to a better generalizing model.

As can be seen in Table 5.1, interspace representations even help to improve dense training of a ResNet50 on ImageNet. After training a dense model in the interspace, the representation can be converted to a standard one to circumvent the additional computational costs of FB convolutions at inference time.

5.5.6 Combining Interspace Representations and FreezeNet

Up to now, interspace representations were only applied to improve pruning methods. However, the enhanced trainability and generalization ability of models in the interspace are not restricted to the pruning case. As an example, interspace representations can also be used to improve models with frozen coefficients. This is called *IP-FreezeNet* as it combines the idea of FreezeNet and IP.

Figure 5.12 shows results for a VGG-16 trained on CIFAR-10. Here, SP- and IP-SNIP are compared with the standard FreezeNet, as presented in Section 3, and IP-FreezeNet. Consistent with the analysis in Section 3, SP-SNIP outperforms FreezeNet for low compression rates while FreezeNet has superior results than SP-SNIP for high compression rates. Moreover, IP-FreezeNet consistently improves the standard FreezeNet for all compression rates q. This shows that interspace representation also help to improve other DRT methods than pruning.

5.6 Conclusions for Interspace Pruning

Pruning convolutional filters in the interspace significantly improves results compared to pruning the spatial coefficients. The interspace is a linear space spanned by a filter basis $\mathcal{F} \subset \mathbb{R}^{K \times K}$. Convolutional filters are represented by basis coefficients w.r.t. $\mathcal{F}$. During training, the FB coefficients are trained jointly with the filter bases. As a result, the interspace dynamically adapts to span the best possible space for the represented filters. This adaptivity is especially useful if convolutions are represented through a sparse set of coefficients, *i.e.* if they are pruned.

Theorem 5.3.1 shows that IP yields sparse models that better approximate the underlying dense model than SP. Moreover, IP's enhanced performance is demonstrated on several image classification tasks, CNN architectures and SOTA pruning methods applied before, during and after training. The improved results achieved by using interspace representations for $K \times K$ filters have two reasons. First, representing filters in a *dynamic* space adds additional capacity to a model which improves results especially for high compression rates. Second, sharing FBs across a bunch of $K \times K$ filters regularizes training and therefore improves the generalization ability of the model – also for low compression rates and even for dense training. Further, increasing the number of FBs to

more than one FB per layer shows a double descent phenomenon: after a critical point, increasing capacity by adding FBs even improves the generalization ability of the CNN again.

For comparing the performance of IP and SP, two metrics are used: the number of trained parameters and the computational cost. IP and SP both have similar memory requirements if the number of FBs is not chosen too big. For similar memory requirements, IP always shows better results than SP. However, IP has a higher computational cost due to a constant overhead induced by (5.12). In theory, SP therefore has better performance than IP for high accelerations of the dense baseline while IP outperforms SP for lower speed ups. For real measurements however, the computational overhead of IP is not as significant anymore and IP shows better or equal results than SP for all speed ups.

The interspace representation is not restricted to improve pruning methods but can also be used for other DRT methods like FreezeNet. Also, reducing the spanning system $\mathcal{F}$ to less than K^2 elements might help to further speed up sparse models in the interspace while keeping a high performance. By looking at the results in Theorem 5.3.1, the idea of IP is generally applicable to any parameter tensor represented in a linear space. Therefore, the interspace representations can also be used for other DNN architectures, like fully connected layers or attention blocks.

Chapter 6
Summary

In this thesis, three different methods for improving the training of sparse DNNs have been proposed. Hereby, sparse training is defined as training a model on a low dimensional submanifold – the so called *dimensionality reduced training*.

6.1 Conclusions

Training a model in a low dimensional space reduces its capacity. Since modern DNNs are highly overparameterized, this can improve the model's generalization ability due to the bias-variance trade off. Moreover, memory and runtime can be reduced for training and inference by training a DNN on a low dimensional submanifold.

The most prominent DRT category is pruning parts of a model to zero which reduces the memory and computational requirements of a DNN. However, it has been shown that SOTA pruning methods applied before training have weaknesses. First, models tend to have a poor information flow if they are compressed to a high level. A weak information flow leads to insufficient training and therefore bad performance after training. Second, pruning methods which are specially designed to guarantee a faithful information flow for high compression rates tend to perform bad for lower compression rates.

The first proposed method in Chapter 3, FreezeNet, is a DRT method that *freezes* untrained weights at their random initial position. Weights are frozen if their saliency score – the magnitude of the weight times the gradient – is below a given threshold which is determined by the desired compression rate. Therefore, only weights which have a significant influence in changing the loss function are trained. By initializing DNNs with the help of pseudo-random number generators, the frozen weights do not need to be stored but can be recovered with a single 32bit random seed. Therefore, FreezeNet has the same storage requirement as a pruned model. In contrast to pruning,

dense tensors are used throughout the whole training. During training, the computational cost for FreezeNet is lower than for dense models due to reduced gradient computations while FreezeNet has the same computational requirement as a dense model for inference. Pruned models can reduce computational costs for training and inference. However, FreezeNet uses the dense tensors filled with frozen weights to its advantage. For high compression rates, FreezeNet does not have the problem of a weak information flow. Therefore, results are improved significantly compared to pruned models. Freezing *all* weights while training only the bias terms in the network leads to 56.2% test accuracy on MNIST. In situations where the information flow is not a bottleneck for training, using frozen weights still improves results. This was shown by FreezeNet-WD which smoothly shrinks the frozen weights to zero by updating them only via weight decay. Ablations show that FreezeNet is robust against the choice of the random initialization and the frozen weights can even be re-initialized before training begins.

FreezeNet shows really good results for high compression rates but at the cost of having an equal runtime as a dense model. The second proposed method in Chapter 4, COPS, is a pruning method applied before training begins. It tackles the problem of SOTA pruning criteria applied before training which perform either well for high or for low compression rates. The main motivation for COPS is that information flow in the model is a necessary condition for successful training, but not a sufficient one. Therefore, the COPS pruning mask is found by solving a constrained combinatorial optimization problem. The constraint can be chosen arbitrarily, but is inspired by constraining the gradient flow of the sparse model to guarantee a faithful information flow after pruning. However, the COPS mask primarily optimizes a target criterion, like the saliency score, such that the COPS pruned model also performs well for lower compression rates. The tightness of the constraint on the COPS mask is given by the balance factor $\alpha_\kappa \in [0,1)$ which is tight for $\alpha_\kappa \to 1$ and loose for $\alpha_\kappa \to 0$. The balance factor is an additional hyperparameter which needs to be set for training. The corresponding optimization problem is a discrete one and therefore relaxed to form a LP. The dual problem of the LP is then solved with an efficient bisection approach which shows faster convergence than the fastest general LP solver by Jiang et al. [79]. Using COPS improves results compared to both underlying pruning criteria for lower and higher compression rates. Although COPS was developed for optimizing the saliency score while controlling the sparse model's gradient flow, it can also be used for combining other pruning methods, like

magnitude pruning or unsupervised methods that improve the robustness of the COPS pruned model. Moreover, the COPS mask is robust w.r.t. changing the hyperparameter α_K. It therefore suffices to find a balance factor close to the optimal one to obtain a well-trainable sparse model.

The third proposed method in Chapter 5 is called interspace pruning. IP improves general pruning methods, including pruning applied before training starts, but also other DRT methods like FreezeNet. Interspace pruning is motivated by sparse dictionary learning which jointly optimizes a dictionary and corresponding coefficients such that they represent a dense matrix best by using only a sparse part of the coefficients. This idea can be lifted to $K \times K$ filters in CNNs. These filters can be represented in the K^2 dimensional vector space $\mathbb{R}^{K \times K}$ by an arbitrary basis – the filter basis. The filter basis together with the corresponding coefficients form the interspace representation of a filter. For interspace pruning, the coefficients of the interspace representation are set to zero. In contrast, standard pruning zeroes the spatial weights of the spatial, standard representation. During training, the FBs and the non-zero FB coefficients can be trained jointly which makes IP especially useful for pruning methods applied before training or at an early training stage. To fulfill one of the main purposes of pruning, *i.e.* to reduce the number of parameters, only a few FBs are shared for the whole CNN. Consequently, IP and SP have the same memory requirements. IP can also lower the runtime of a CNN. Nevertheless, IP adds a small, constant computational overhead compared to SP. In practical tests, however, this overhead has proven to be small. The experimental evaluations show that IP outperforms SP for all compression rates and pruning methods, but also for other DRT methods like FreezeNet. Additionally, IP has better performance than SP for similar runtime. The improvements obtained by using the interspace representations can be explained by:

- Allowing the basis to adapt increases expressibility which improves the training results, especially for high compression rates. The improved expressibility of sparse models in the interspace is proven in Theorem 5.3.1.
- Sharing FBs for a set of filters regularizes training and therefore improves the generalization ability of the models.

6.2 Outlook

The experimental evaluation in this work focuses on image classification tasks. For future work, the proposed methods can be tested on other important deep learning tasks like image segmentation, multi-task learning or object detection with various sensor types. Moreover, the tested network structures are MLPs or CNNs. However, many SOTA results are nowadays attained by using Vision Transformers [35]. Of course, pruning and freezing can also be applied to these models but the proposed methods might require some adjustments. Moreover, interspace representations are *not* limited on $K \times K$ filters, but can be applied to any element of a vector space. For attention modules, blocks of the matrices could be represented by an adaptive basis. Note, using a basis of whole matrices would most likely require too much memory to store.

The three discussed DRT methods are *unstructured* pruning/freezing methods. Therefore, they help to drastically reduce the memory requirements for the models, but specialized soft- and hardware is required to also reduce the computational costs for training the model, and also for inference if the model is pruned. In the future, it will be interesting to see how these methods can be generalized to structured pruning and freezing. For structured freezing of a layer, the computation of gradients for a whole layer might be skipped. Especially if the first layers of a model are frozen, this massively reduces the computational and memory costs required for backpropagation.

DRT was defined as training on a low dimensional submanifold. However, all proposed methods train on a low dimensional subspace whereas submanifolds can represent much more than linear spaces. For example, the interspace representation could be generalized to express filters in a non-linear manner, as Schwarz et al. [164] do for single weights to enforce sparsity in a DNN. Also for interspace pruning, investigating the impact of using under- and overcomplete systems for representing the interspace is a promising direction for future work. Moreover, tailoring a pruning method to the setting of interspace representations, *e.g.* by using sophisticated sparse dictionary learning techniques, instead of applying pruning techniques developed for spatial representations should help to boost IP further. FreezeNet uses the saliency score to select the frozen weights. This criterion however is motivated for setting parameters to zero, not freezing them. Therefore, finding a more specialized freezing criterion might help to further improve the training of a partly frozen DNN.

It still remains a challenge for pruning methods applied before training to achieve similar results as pruning a densely pre-trained model [195]. Up to now, it is not answered *if* it is possible to train a model on a low dimensional submanifold to the same performance than pruning and fine-tuning a densely pre-trained DNN. For example, in Section 5.5.2 it is shown, that the SOTA DRT methods still are not able to exactly match the more expensive pre-train, prune and fine-tune strategy. Nevertheless, this thesis proposes three methods which shrink this gap, especially for high compression rates.

Appendix A
Experimental Setups

This Chapter provides the experimental setups for the results in Chapters 3 - 5. First, the image classification datasets used in this work are presented in Section A.1. Then, the tested DNN architectures are described in Section A.2. Finally, Sections A.3 - A.5 describe the experimental setups used for the results in Chapters 3 - 5, respectively.

A.1 Datasets

The proposed methods in this thesis are analyzed on several image classification tasks, namely MNIST [92], CIFAR-10 and CIFAR-100 [85], Tiny ImageNet [89] and ImageNet ILSVRC2012 [28].

MNIST. MNIST [92] is a dataset consisting of 28×28 grayscale images for handwritten digit recognition. Hereby, a digit corresponds to a class $c \in \{0, 1, \dots, 9\}$. The dataset is divided into a training set with $60,000$ samples and a test set with $10,000$ samples. For the experiments in this thesis, the training dataset is randomly split into a part only used for validation while the remaining images are used for training. The split is $9/1$ between training and validation data.

CIFAR. The CIFAR dataset [85] consists of $60,000$ 32×32 RGB images. These images are divided into $50,000$ training images and $10,000$ test images. Again, the training images are randomly split $9/1$ into two parts, one used for optimizing the DNN while the other is only used for validation. For CIFAR-10, the dataset has 10 classes with $6,000$ images per class. Each class contains 10 sub-classes with 600 images, each. Combined, this yields 100 sub-classes which are used for the CIFAR-100 image classification task.

Tiny ImageNet. The Tiny ImageNet dataset [89] consists of $100,000$ RGB images. The images have a resolution of 64×64 and are divided in 200 different classes. Those classes have 500 training images, 50 validation images and 50 test images, each. Tiny ImageNet can be seen as a *miniature* of the ImageNet dataset as it has fewer classes with lower resolution and also less images per class.

ImageNet. ImageNet ILSVRC2012 [28] is a large scale image classification dataset with approximately 1.2 million RGB images for training and $150,000$ RGB images for validation. These images are divided into $1,000$ classes where not all classes have the same number of images. The test dataset for ImageNet is publicly not available. In literature and also in this work, results on ImageNet are therefore reported on the validation dataset.

A.2 Network Architectures

In this Section the used network architectures are provided:

- Table A.1 shows a LeNet-300-100 which is used for experimental evaluations on the MNIST classification task in Chapter 3.
- LeNet-5-Caffe is specified in Table A.2. This architecture is used for the MNIST classification task in Chapters 3 and 4.
- VGG-16 and VGG-16-LTH are used for CIFAR-10 and CIFAR-100 classification for evaluations in Chapters 3, 4 and 5. The architecture is provided in Table A.3.
- ResNet18 is used for three different image classification tasks in this thesis, CIFAR-100, Tiny ImageNet and ImageNet. The architecture for CIFAR-100 and Tiny ImageNet – which is evaluated in Chapter 4 – is given in Table A.4. Table A.5 yields the ResNet18 for ImageNet, tested in Chapter 5.
- Chapter 3 contains experiments with a ResNet34 on Tiny ImageNet. The architecture of a ResNet34 is given in Table A.6.
- Interspace pruning, Chapter 5, is also evaluated for a ResNet50 on ImageNet. Table A.7 shows the network architecture of a ResNet50.

Table A.1: LeNet-300-100 for MNIST. The input grayscale 28×28 image is reshaped to a vector of size $28^2 = 784$.

Module	Output Size	c_{in}	c_{out}	Bias	BatchNorm	ReLU
`Linear`	300	784	300	✓	✗	✓
`Linear`	100	300	100	✓	✗	✓
`Linear`	10	100	10	✓	✗	✗

- For COPS in Chapter 4, results are also compared on a Wide-ResNet18-2. The corresponding network is specified in Table A.8.

A graphical description of the residual block and the bottleneck block used for ResNets, is shown in Figures A.1 (a) and (b), respectively. Furthermore, all CNN architectures are used in their standard, spatial form for results in Chapter 3 and 4. They are also used in their standard form for baseline and SP results in Chapter 5. However, interspace representations use FB convolutions. Therefore, (FB) always indicates, that a FB version of the filter is used for the FB-CNN. Additionally, the FB sharing schemes are indicated in Tables A.3, A.5 and A.7.

Strides, paddings and convolutional kernel sizes of (FB) 2D convolutions are all quadratic. All tensor dimensions of standard 2D convolutional filters are given as $c_{\text{out}} \times c_{\text{in}} \times K \times K$, where c_{in} equals the number of input channels, c_{out} the number of output channels and $K \times K$ the size of each convolutional kernel. FB convolutions have coefficients represented by a tensor of size $c_{\text{out}} \times c_{\text{in}} \times K^2$. For pooling layers, $K \times K$ denotes the tiling size. In the case of linear layers, the tensor size is given as $c_{\text{out}} \times c_{\text{in}}$, where c_{in} is the number of incoming neurons and c_{out} the number of outgoing neurons. The networks are trained with the cross entropy loss function.

A.3 Experimental Setup for FreezeNet in Chapter 3

Experiments are run on a single Nvidia GeForce 1080ti GPU by using the deep learning framework PyTorch1.4.0 [138]. SNIP [94] results are obtained by using the PyTorch adaptation of SNIP.[8] Models are optimized on the cross entropy loss function by using SGD with momentum [172]. The labeled training datasets for MNIST and CIFAR are

[8] Based on the official implementation `https://github.com/namhoonlee/snip-public`.

Table A.2: LeNet-5-Caffe for MNIST. Before the first linear layer, the feature map with size $50 \times 4 \times 4$ is reshaped to a vector of size $50 \cdot 4 \cdot 4 = 800$.

Module	Output Size	c_{in}	c_{out}	K	Stride	Padding	Bias	BatchNorm	ReLU
`Conv2D`	24×24	1	20	5	1	0	✓	✗	✓
`MaxPool2D`	12×12	20	20	2	2	0	✗	✗	✗
`Conv2D`	8×8	20	50	5	1	0	✓	✗	✓
`MaxPool2D`	4×4	50	50	2	2	0	✗	✗	✗
`Linear`	500	800	500	—	—	—	✓	✗	✓
`Linear`	10	500	10	—	—	—	✓	✗	✗

randomly split $9/1$ into training and validation data to find the best performing model via early stopping. For training on the CIFAR and Tiny ImageNet datasets, the training data is augmented by random horizontal flipping and random translations by up to 4 pixels. All used hyperparameters are listed in Table A.9.

A.4 Experimental Setup for COPS in Chapter 4

Experimental results are generated with the deep learning framework PyTorch1.5 [138]. The experiments were run on a single Nvidia GeForce 1080ti GPU. Data augmentations for the experiments are the same ones as used for the FreezeNet results, see Section A.3.

Pruning with a single score is done by solving (4.7), *i.e.* by pruning the weights with the $D-d$ biggest scores. The COPS masks are computed via Algorithm 4.1. The numerical precision is chosen as $\varepsilon_n = 10^{-6}$. To obtain pruning and control scores in the same range, both scores are normalized to have $\|\mathbf{s}\|_1 = 1$.

A.5 Experimental Setup for Interspace Pruning in Chapter 5

Experimental results are generated with the deep learning framework PyTorch1.9 [138]. The experiments were run on $n \in \{1,2,4,8\}$ Nvidia GeForce 1080 ti GPU. The number of used GPUs together with the experimental setup for each conducted experiment is given in Table A.11. Also, the used hyperparameters are provided in Table A.11.

Table A.3: VGG-16 and VGG-16-LTH for CIFAR-10 and CIFAR-100. For interspace pruning, the `coarse`, `medium` and `fine` FB sharing schemes are specified in the last three columns. Note, for VGG-16-LTH, the first and second `Linear` layers are removed. The number of classes C corresponds to the number of nodes in the output layer. It holds $C = 10$ for CIFAR-10 and $C = 100$ for CIFAR-100.

Module	Output Size	c_{in}	c_{out}	K	Repeat	Stride	Padding	Bias	BatchNorm	ReLU	coarse	medium	fine
(FB) Conv2D	32×32	3	64	3	$\times 1$	1	1	✓	✓	✓			
(FB) Conv2D	32×32	64	64	3	$\times 1$	1	1	✓	✓	✓			
MaxPool2D	16×16	64	64	2	$\times 1$	2	0	✗	✗	✗			
(FB) Conv2D	16×16	64	128	3	$\times 1$	1	1	✓	✓	✓			
(FB) Conv2D	16×16	128	128	3	$\times 1$	1	1	✓	✓	✓			
MaxPool2D	8×8	128	128	2	$\times 1$	2	0	✗	✗	✗			
(FB) Conv2D	8×8	128	256	3	$\times 1$	1	1	✓	✓	✓			
(FB) Conv2D	8×8	256	256	3	$\times 2$	1	1	✓	✓	✓			$\times 2$
MaxPool2D	4×4	256	256	2	$\times 1$	2	0	✗	✗	✗			
(FB) Conv2D	4×4	256	512	3	$\times 1$	1	1	✓	✓	✓			
(FB) Conv2D	4×4	512	512	3	$\times 2$	1	1	✓	✓	✓			$\times 2$
MaxPool2D	2×2	512	512	2	$\times 1$	2	0	✗	✗	✗			
(FB) Conv2D	2×2	512	512	3	$\times 3$	1	1	✓	✓	✓			$\times 3$
MaxPool2D	1×1	512	512	2	$\times 1$	2	0	✗	✗	✗			
Linear	512	512	512	—	$\times 2$	—	—	✓	✓	✓	**Removed for VGG16-LT**		
Linear	C	512	C	—	$\times 1$	—	—	✓	✗	✗			

Table A.4: ResNet18 for CIFAR-100 ($C = 100$) and Tiny ImageNet ($C = 200$). `ResBlock` is shown in Figure A.1 (a). For this architecture, no FBs are used since it is not tested on interspace representations. The stride in the first `(FB) ResBlock` equals $s = 1$ for CIFAR-100 and $s = 2$ for Tiny ImageNet.

Module	Output Size	c_{in}	c_{out}	n_b	K	Stride	Padding	Bias	BatchNorm	ReLU
Conv2D	$32 \cdot s \times 32 \cdot s$	3	64	—	3	1	1	✗	✓	✓
ResBlock	32×32	64	64	2	3	s	1	✗	✓	✓
ResBlock	16×16	64	128	2	3	2	1	✗	✓	✓
ResBlock	8×8	128	256	2	3	2	1	✗	✓	✓
ResBlock	4×4	256	512	2	3	2	1	✗	✓	✓
AvgPool2D	1×1	512	512	—	4	4	0	✗	✗	✗
Linear	C	512	C	—	—	—	—	✓	✗	✗

Table A.5: ResNet18 for ImageNet with `(FB) ResBlock`, shown in Figure A.1 (a). If interspace representations are used, the last column declares the `coarse` FB sharing scheme.

Module	Output Size	c_{in}	c_{out}	n_b	K	Stride	Padding	Bias	BatchNorm	ReLU	`coarse`
`Conv2D`	112×112	3	64	—	7	2	3	✗	✓	✓	
`MaxPool2D`	56×56	64	64	—	3	2	1	✗	✗	✗	
`(FB) ResBlock`	56×56	64	64	2	3	1	1	✗	✓	✓	▮
`(FB) ResBlock`	28×28	64	128	2	3	2	1	✗	✓	✓	▮
`(FB) ResBlock`	14×14	128	256	2	3	2	1	✗	✓	✓	▮
`(FB) ResBlock`	7×7	256	512	2	3	2	1	✗	✓	✓	▮
`AvgPool2D`	1×1	512	512	—	7	7	0	✗	✗	✗	
`Linear`	1,000	512	1,000	—	—	—	—	✓	✗	✗	

Table A.6: ResNet34 for Tiny ImageNet with $C = 200$ classes, where `ResBlock` is shown in Figure A.1 (a). For ResNet34, FBs are not used since no interspace representations are tested for this architecture.

Module	Output Size	c_{in}	c_{out}	n_b	K	Stride	Padding	Bias	BatchNorm	ReLU
`Conv2D`	$32 \cdot s \times 32 \cdot s$	3	64	—	3	1	1	✗	✓	✓
`ResBlock`	32×32	64	64	3	3	2	1	✗	✓	✓
`ResBlock`	16×16	64	128	4	3	2	1	✗	✓	✓
`ResBlock`	8×8	128	256	6	3	2	1	✗	✓	✓
`ResBlock`	4×4	256	512	3	3	2	1	✗	✓	✓
`AvgPool2D`	1×1	512	512	—	4	4	0	✗	✗	✗
`Linear`	200	512	200	—	—	—	—	✓	✗	✗

Table A.7: ResNet50 for ImageNet with `(FB) Bottleneck`, shown in Figure A.1 (b). If interspace representations are used, the last column declares `fine` FB sharing scheme.

Module	Output Size	c_{in}	c_{mid}	c_{out}	n_b	K	Stride	Padding	Bias	BN	ReLU	`fine`
`(FB) Conv2D`	112×112	3	—	64	—	7	2	3	✗	✓	✓	▮×1
`MaxPool2D`	56×56	64	—	64	—	3	2	1	✗	✗	✗	
`(FB) Bottleneck`	56×56	64	64	256	3	3	1	1	✗	✓	✓	▮×3
`(FB) Bottleneck`	28×28	256	128	512	4	3	2	1	✗	✓	✓	▮×4
`(FB) Bottleneck`	14×14	512	256	1,024	6	3	2	1	✗	✓	✓	▮×6
`(FB) Bottleneck`	7×7	1,024	512	2,048	3	3	2	1	✗	✓	✓	▮×3
`AvgPool2D`	1×1	512	—	512	—	7	0	0	✗	✗	✗	
`Linear`	1,000	512	—	1,000	—	—	—	—	✓	✗	✗	

Table A.8: Wide-ResNet18-2 used for Tiny ImageNet classification. The `ResBlock` is shown in Figure A.1 (a), but applied without FBs since no interspace representation is tested for this architecture.

Module	Output Size	c_{in}	c_{out}	n_b	K	Stride	Padding	Bias	BatchNorm	ReLU
`Conv2D`	64×64	3	128	—	3	1	1	✗	✓	✓
`ResBlock`	32×32	128	128	2	3	2	1	✗	✓	✓
`ResBlock`	16×16	128	256	2	3	2	1	✗	✓	✓
`ResBlock`	8×8	256	512	2	3	2	1	✗	✓	✓
`ResBlock`	4×4	512	1024	2	3	2	1	✗	✓	✓
`AvgPool2D`	1×1	1024	1024	—	4	4	0	✗	✗	✗
`Linear`	200	1024	200	—	—	—	—	✓	✗	✗

Table A.9: Experimental setup for conducted FreezeNet experiments – with references to used training and hyperparameter setup given in the third row.

Dataset	MNIST	CIFAR-10	CIFAR-100	Tiny ImageNet
Model	LeNet-300-100 & LeNet-5-Caffe	VGG-16	VGG-16	ResNet34
Setup from	[94]	[94]	[94]	[174]
#Epochs	250	250	250	100
Batch Size	100	128	128	128
Optimizer: SGD-	Momentum 0.9	Momentum 0.9	Momentum 0.9	Momentum 0.9
Learning Rate	0.1	0.1	0.01	0.1
LR Decay	$\times 0.1$	$\times 0.1$	$\times 0.1$	$\times 0.1$
	every 25k iterations	every 30k iterations	every 30k iterations	epochs 30/60/80
Weight Decay	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$
Initialization	Xavier-normal	Xavier-normal	Kaiming-uniform	Kaiming-uniform

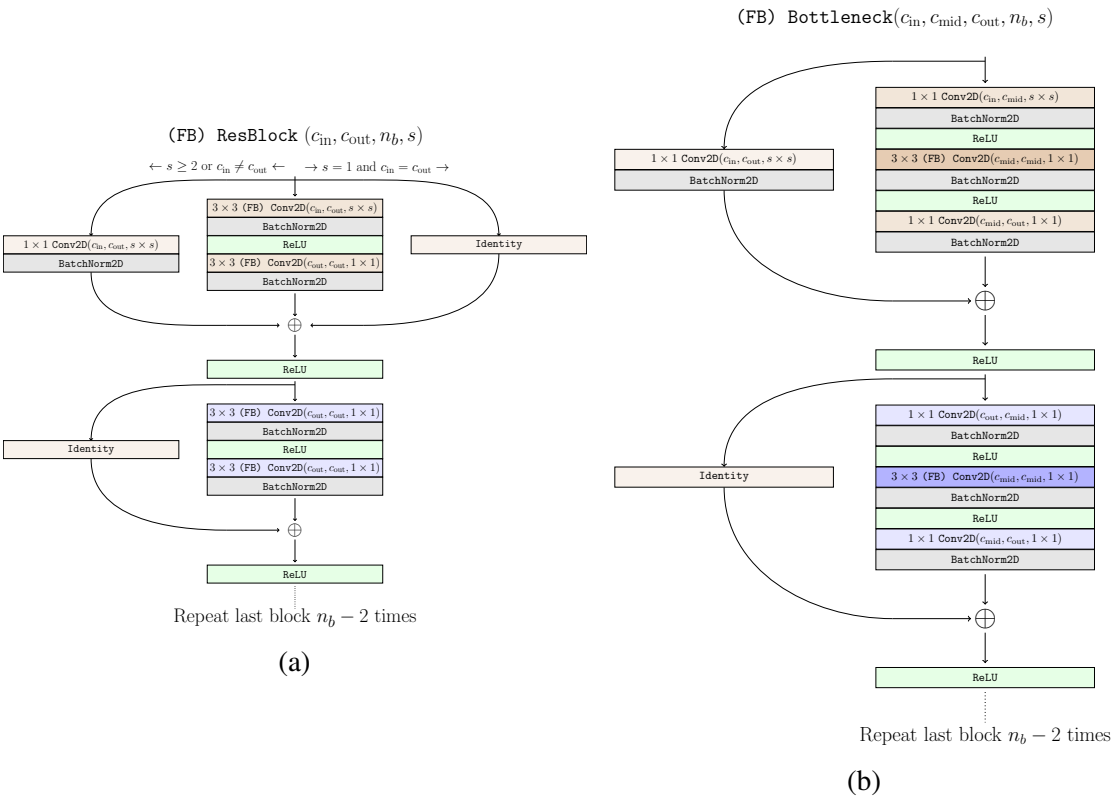


Figure A.1: (a) Architecture of a `(FB) ResBlock` with c_{in} input channels, c_{out} output channels and stride s for the first 2D (FB) convolution. The first residual connection is is a standard 1×1 2D convolution followed by a `BatchNorm2D` layer if $s > 1$ or $c_{\text{in}} \neq c_{\text{out}}$. However, if $c_{\text{in}} = c_{\text{out}}$ and $s = 1$, the first residual connection is simply an identity mapping. The following $n_b - 1$ blocks always have an identity mapping as residual connection. (b) Architecture of a `(FB) Bottleneck` with c_{in} input channels, c_{mid} middle channels and c_{out} output channels and stride s for the first 1×1 convolution. The number of small blocks forming the `(FB) Bottleneck` is given by n_b. The first residual connection is a standard 1×1 `2D Convolution` followed by a `BatchNorm2D` layer. The following residual connections are always identity mappings. All `(FB) Conv2D` layers do not have biases.

Table A.10: (Table II in Wimmer et al. [193]) Experimental setups – with references to used training and hyperparameter setup given in the third row.

Dataset	MNIST	CIFAR-10	CIFAR-100	Tiny ImageNet	Tiny ImageNet
Model	LeNet-5-Caffe	VGG-16	ResNet18	ResNet18	Wide-ResNet18-2
Setup from	[94]	[94]	[174]	[174]	[174]
#Epochs	250	250	160	100	100
Batch Size	100	128	128	128	128
Optimizer: SGD-	Momentum 0.9	Momentum 0.9	Momentum 0.9	Momentum 0.9	Momentum 0.9
Learning Rate	0.1	0.1	0.01	0.1	0.1
LR Decay	$\times 0.1$	$\times 0.1$	$\times 0.2$	$\times 0.1$	$\times 0.1$
	every 25k iterations	every 30k iterations	epochs $60/120$	epochs $30/60/80$	epochs $30/60/80$
Weight Decay	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	10^{-4}	10^{-4}
Initialization	Xavier-normal	Kaiming-uniform	Kaiming-uniform	Kaiming-uniform	Kaiming-uniform

Table A.11: (Table A6 in Wimmer et al. [194]) Setups for experiments for the comparison between SP and IP with references to used training and hyperparameter setup in the fourth row. The number of used GPUs for each experiment is provided in this Table.

Dataset	CIFAR-10	CIFAR-10	ImageNet	ImageNet	ImageNet
Methods	PaI, SET, FreezeNet	LTH	PaI	RigL, FT	GMP
Model	VGG-16	VGG-16-LTH	ResNet18	ResNet50	ResNet50
Setup from	[94]	[44]	[184]	[104]	[49]
# Trainings	5	5	3	3	3
# Epochs	250	160	90	100	100
Batch Size	128	128	512	128	256
# GPUs	1	1	8	2	4
Optimizer: SGD-	Momentum	Momentum	Momentum	Momentum	Momentum
Momentum	0.9	0.9	0.9	0.9	0.9
Learning Rate	0.1	0.1	0.1	0.1	0.1
LR Decay	$\times 0.1$	$\times 0.1$	$\times 0.1$	$\times 0.1$	$\times 0.1$
	every 30k iterations	epochs $80/120$	epochs $30/60$	epochs $30/60/90$	epochs $30/60/80$
LR Warm-up	✗	✗	✗	5 epochs, linear	5 epochs, linear
Weight Decay	$5 \cdot 10^{-4}$	10^{-4}	10^{-4}	10^{-4}	10^{-4}
Label smoothing	✗	✗	✗	$\varepsilon = 0.1$	$\varepsilon = 0.1$

Appendix B
Mathematical Proofs

The mathematical results in this thesis are proven in this Chapter. Section B.1 contains the proofs for the result in Chapter 4. The proof of Theorem 5.3.1 in Section 5 is given in Section B.2.

B.1 Proofs for Results in Chapter 4

In this Section, the results from Chapter 4 are proven.

B.1.1 Proof of Lemma 4.3.1

Proof. The Lagrangian can be rewritten as

$$\Lambda(\mathbf{m},\lambda) = -\lambda\cdot\kappa + \sum_{i=1}^{D} \mathbf{m}_i \cdot (\mathbf{s}_i^{(0)} + \lambda\cdot\mathbf{s}_i^{(1)}) \,. \tag{B.1}$$

Since $-\lambda\cdot\kappa$ is constant for a given λ, the dual function for $\lambda \geq 0$ equals

$$g(\lambda) = -\lambda\cdot\kappa + \inf_{\mathbf{m}\in\mathcal{V}_{1,d}} \sum_{i=1}^{D} \mathbf{m}_i \cdot (\mathbf{s}_i^{(0)} + \lambda\cdot\mathbf{s}_i^{(1)}) \,. \tag{B.2}$$

Since $\mathcal{V}_{1,d} \subset \mathbb{R}^D$ is a non-empty compact set and $\left\langle \cdot, \mathbf{s}^{(0)} + \lambda\cdot\mathbf{s}^{(1)} \right\rangle$ is a continuous function, the minimum is actually attained by a $\tilde{\mathbf{m}} \in \mathcal{V}_{1,d}$. For each $i \in \{1,\ldots,D\}$, $\mathbf{m}_i \cdot (\mathbf{s}^{(0)} + \lambda\cdot\mathbf{s}^{(1)})$ is a monotonic function in $\mathbf{m}_i$. Therefore,

$$\min_{\mathbf{m}_i\in[0,1]} \mathbf{m}_i \cdot (\mathbf{s}_i^{(0)} + \lambda\cdot\mathbf{s}_i^{(1)}) = \begin{cases} 0, & \mathbf{s}^{(0)} + \lambda\cdot\mathbf{s}^{(1)} \geq 0 \\ \mathbf{s}^{(0)} + \lambda\cdot\mathbf{s}^{(1)}, & \mathbf{s}^{(0)} + \lambda\cdot\mathbf{s}^{(1)} < 0 \end{cases} , \tag{B.3}$$

is achieved by setting $\mathbf{m}_i = 0$ for $\mathbf{s}^{(0)} + \lambda \cdot \mathbf{s}^{(1)} \geq 0$ and $\mathbf{m}_i = 1$ for $\mathbf{s}^{(0)} + \lambda \cdot \mathbf{s}^{(1)} < 0$. As a consequence,

$$g(\lambda) = -\lambda \cdot \kappa + \min_{\mathbf{m} \in \mathcal{V}_{1,d}} \sum_{i=1}^{D} \mathbf{m}_i \cdot (\mathbf{s}_i^{(0)} + \lambda \cdot \mathbf{s}_i^{(1)}) \tag{B.4}$$

$$= -\lambda \cdot \kappa \tag{B.5}$$

$$+ \min\{ \sum_{i \in \mathcal{J}} \mathbf{s}_i^{(0)} + \lambda \cdot \mathbf{s}_i^{(1)} : \mathcal{J} \subset \{1, \dots D\}, \#\mathcal{J} \leq d, \mathbf{s}_i^{(0)} + \lambda \cdot \mathbf{s}_i^{(1)} < 0 \, \forall i \in \mathcal{J} \} \tag{B.6}$$

$$= \sum_{i \in \mathcal{I}_\lambda} \mathbf{s}_i^{(0)} + \lambda \cdot \mathbf{s}_i^{(1)} - \lambda \kappa \, . \tag{B.7}$$

■

B.1.2 Proof of Lemma 4.3.2

Proof. As point wise infimum over affine linear functions, the dual function g is concave [13]. Since the dual function is a piecewise affine linear function $g(\lambda) = \sum_{i \in \mathcal{I}_\lambda} \mathbf{s}_i^{(0)} + \lambda \cdot \mathbf{s}_i^{(1)} - \lambda \kappa$, the derivative of $\sum_{i \in \mathcal{I}_\lambda} \mathbf{s}_i^{(0)} + \lambda \cdot \mathbf{s}_i^{(1)} - \lambda \kappa$ w.r.t. λ defines a superderivative of g at each $\lambda \geq 0$. Therefore,

$$g'(\lambda) = \sum_{i \in \mathcal{I}_\lambda} \mathbf{s}_i^{(1)} - \kappa \, . \tag{B.8}$$

■

B.1.3 Proof of Corollary 4.3.1

Proof. For concave functions, $0 \in \hat{\partial} g(\lambda_0)$ is a necessary and sufficient condition to have a maximum in λ_0 [13]. For $\xi = 0$ the claim is therefore fulfilled. Now, assume the existence of a λ^* that maximizes g, otherwise the claim is fulfilled trivially. Moreover, the superderivatives ξ of g at λ are characterized by [13]

$$\xi_+(\lambda) := \lim_{t \searrow 0} \frac{g(\lambda + t) - g(\lambda)}{t} \leq \xi \leq \lim_{t \nearrow 0} \frac{g(\lambda + t) - g(\lambda)}{t} =: \xi_-(\lambda) \, , \tag{B.9}$$

where both, the left and right derivative $\xi_-(\lambda)$ and $\xi_+(\lambda)$, respectively, are monotonically decreasing functions in λ. Consequently, all superderivatives fulfill

$$\max\{\tilde{\xi} : \tilde{\xi} \in \hat{\partial} g(\lambda), \lambda > \lambda_0\} \leq \xi \leq \min\{\tilde{\xi} : \tilde{\xi} \in \hat{\partial} g(\lambda), \lambda < \lambda_0\} \tag{B.10}$$

where $\xi \in \hat{\partial} g(\lambda_0)$. For $\xi \in \hat{\partial} g(\lambda_0)$ with $\xi > 0$ it holds

$$\min\{\tilde{\xi} : \tilde{\xi} \in \hat{\partial} g(\lambda), \lambda < \lambda_0\} \geq \xi > 0 \tag{B.11}$$

for all $\lambda < \lambda_0$. Consequently, $\lambda^* \geq \lambda_0$. Equivalently it can be shown that $\lambda^* \leq \lambda_0$ if $\xi \in \hat{\partial} g(\lambda_0)$ with $\xi < 0$. ■

B.1.4 Proof of Lemma 4.3.3

Proof. Define the set of all feasible points of the primal problem (4.13) as

$$\mathcal{Q}_{1,d} := \mathcal{V}_{1,d} \cap \{\mathbf{m} \in \mathbb{R}^D : S_1(\mathbf{m}) \leq \kappa\} . \tag{B.12}$$

The set $\mathcal{Q}_{1,d}$ is closed as an intersection of two closed sets and non-empty since $\mathbf{m}^\circ \in \mathcal{Q}_{1,d}$. Moreover, $\mathcal{Q}_{1,d}$ is bounded and therefore a compact set. The primal problem (4.13) can be now rewritten as

$$\min_{\mathbf{m} \in \mathcal{Q}_{1,d}} S_0(\mathbf{m}) . \tag{B.13}$$

Since S_0 is a continuous function, the primal problem attains a minimum in $\mathcal{Q}_{1,d}$. Since Slater's condition is fulfilled, strong duality holds. By strong duality, also the dual problem has an optimum $d^* = p^*$.

First assume the optimum of g to be attained for a $\lambda^* > 0$. The dual function g is concave and piecewise linear. A necessary condition for a maximum of g on the domain $\lambda > 0$ in λ^* is given by $0 \in [\xi_+(\lambda^*), \xi_-(\lambda^*)]$, see Corollary 4.3.1. Now, the left and right derivative of g in λ^* is given by

$$\xi_+(\lambda) = S_1(\mathbf{m}^+) - \kappa \, , \; \xi_-(\lambda) = S_1(\mathbf{m}^-) - \kappa \, . \tag{B.14}$$

where both, $\operatorname{supp} \mathbf{m}^+$ and $\operatorname{supp} \mathbf{m}^-$ are elements of $d - \operatorname{idx} \min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\}$. This

holds true since by (4.18),

$$g(\lambda^*) = S_0(\mathbf{m}) + \lambda^* \cdot (S_1(\mathbf{m}) - \kappa) \tag{B.15}$$

for any $\mathbf{m} \in d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\}$ while g is also piecewise linear. Note, it might be possible that $\mathbf{m}^+ = \mathbf{m}^-$ which yields that both, left and right derivative of g are zero. Now set $\mathbf{m}(\gamma) := \gamma \cdot \mathbf{m}^- + (1-\gamma) \cdot \mathbf{m}^+$. Since $\mathcal{V}_{1,d}$ is a convex set, $\mathbf{m}(\gamma) \in \mathcal{V}_{1,d}$. By linearity of S_0 and S_1 it holds for all $\gamma \in [0,1]$

$$\begin{aligned} g(\lambda^*) &= S_0(\mathbf{m}^+) + \lambda^* \cdot (S_1(\mathbf{m}^+) - \kappa) && \text{(B.16)} \\ &= S_0(\mathbf{m}^-) + \lambda^* \cdot (S_1(\mathbf{m}^-) - \kappa) && \text{(B.17)} \\ &= S_0(\mathbf{m}(\gamma)) + \lambda^* \cdot (S_1(\mathbf{m}(\gamma)) - \kappa)\,. && \text{(B.18)} \end{aligned}$$

By the necessary criterion for λ^* to maximize g, $S_1(\mathbf{m}^+) - \kappa \leq 0 \leq S_1(\mathbf{m}^-) - \kappa$ is true. Therefore, there exists a $\bar{\gamma} \in [0,1]$ with $\bar{\mathbf{m}} := \mathbf{m}(\bar{\gamma}) \in \mathcal{V}_{1,d}$ satisfying $S_1(\bar{\mathbf{m}}) - \kappa = 0$. Moreover, it holds by strong duality and (B.18)

$$p^* = d^* = g(\lambda^*) = S_0(\bar{\mathbf{m}}) + \lambda^* \cdot \underbrace{(S_1(\bar{\mathbf{m}}) - \kappa)}_{=0} = S_0(\bar{\mathbf{m}})\,. \tag{B.19}$$

Therefore, $\bar{\mathbf{m}}$ solves the primal relaxed problem (4.13). This shows the claim for $\lambda^* > 0$.

For $\lambda^* = 0$, it holds $p^* = d^* = g(\lambda^*) = S_0(\mathbf{m})$ for any $\mathbf{m} \in d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\}$ which shows the claim for $\lambda^* = 0$. ■

B.1.5 Proof of Theorem 4.3.1

Proof. Since $\mathcal{V}_{0,d} \subset \mathcal{V}_{1,d}$, it holds by assumption

$$\kappa > \min_{\mathbf{m} \in \mathcal{V}_{0,d}} S_1(\mathbf{m}) \geq \min_{\mathbf{m} \in \mathcal{V}_{1,d}} S_1(\mathbf{m})\,. \tag{B.20}$$

Therefore, there exists a $\mathbf{m} \in \mathcal{V}_{1,d}$ with $S_1(\mathbf{m}) < \kappa$. By continuity of S_1 and $\|\cdot\|_1$, there exists a strictly feasible point $\mathbf{m}^\circ \in (0,1)^D$ with $\|\mathbf{m}^\circ\|_1 < d$ and $S_1(\mathbf{m}^\circ) < \kappa$. As a consequence, Lemma 4.3.3 can now be applied. Therefore, the dual problem (4.17) is solved by $\lambda^* \geq 0$ with a corresponding $\mathbf{m}^* \in \mathcal{V}_{1,d}$ that solves the primal, relaxed problem

(4.13). Moreover, the un-relaxed problem is a combinatorial problem with at least one feasible point which is minimized over finitely many points. Therefore, (4.11) is solved by a $\bar{\mathbf{m}} \in \mathcal{V}_{0,d}$.

By looking at the proof of Lemma 4.3.3, all $\mathbf{m} \in d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\}$ solve the relaxed primal problem (4.13) if $\lambda^* = 0$. Since $\mathbf{m} \in \mathcal{V}_{0,d} \subset \mathcal{V}_{1,d}$, these $\mathbf{m}$ also solve the un-relaxed discrete problem.

Now, assume $\lambda^* > 0$. Applying Lemma 4.3.3 yields $\mathbf{m}^-, \mathbf{m}^+ \in \mathcal{V}_{0,d}$. The proof of Lemma 4.3.3 shows that $S_1(\mathbf{m}^+) \leq \kappa \leq S_1(\mathbf{m}^-)$. By Lemma 4.3.3, $\mathbf{m}^* = \gamma \cdot \mathbf{m}^- + (1 - \gamma) \cdot \mathbf{m}^+ \in \mathcal{V}_{1,d}$ solves the primal problem (4.13) for some $\gamma \in [0,1]$. Since $\mathcal{V}_{0,d} \subset \mathcal{V}_{1,d}$, it holds

$$S_0(\mathbf{m}^*) \leq S_0(\bar{\mathbf{m}}) \,. \tag{B.21}$$

Moreover, $\mathbf{m}^+$ is feasible for (4.11), therefore $S_0(\bar{\mathbf{m}}) \leq S_0(\mathbf{m}^+)$. The proof of Lemma 4.3.3 further shows that $S_0(\mathbf{m}^-) \leq S_0(\mathbf{m}^*)$ holds. Altogether, the inequality

$$S_0(\mathbf{m}^-) \leq S_0(\mathbf{m}^*) \leq S_0(\bar{\mathbf{m}}) \leq S_0(\mathbf{m}^+) \tag{B.22}$$

is true.

Finally, it remains to show that $\mathbf{m}^-$ and $\mathbf{m}^+$ exists which differ in at most two element. All $\mathbf{m} \in d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\}$ fulfill $g(\lambda^*) = S_0(\mathbf{m}) + \lambda^* \cdot (S_1(\mathbf{m}) - \kappa)$. If $\#d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\} = 1$, $\mathbf{m}^* \in d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\} \subset \mathcal{V}_{0,d}$ which shows the claim in this case. Therefore, assume $\#d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\} > 1$. Let $i_1, \ldots, i_D$ be an ordering of $\{1, \ldots, D\}$ such that for all $j \in \{1, \ldots, D-1\}$, $\mathbf{s}_{i_j}^{(0)} + \lambda^* \cdot \mathbf{s}_{i_j}^{(1)} \leq \mathbf{s}_{i_{j+1}}^{(0)} + \lambda^* \cdot \mathbf{s}_{i_{j+1}}^{(1)}$ and if $\mathbf{s}_{i_j}^{(0)} + \lambda^* \cdot \mathbf{s}_{i_j}^{(1)} = \mathbf{s}_{i_{j+1}}^{(0)} + \lambda^* \cdot \mathbf{s}_{i_{j+1}}^{(1)}$ it holds $\mathbf{s}_{i_j}^{(1)} \leq \mathbf{s}_{i_{j+1}}^{(1)}$.

Since, $\#d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\} > 1$, there exists $r, R \geq 1$ with $\mathbf{s}_{i_{d-r}}^{(0)} + \lambda^* \cdot \mathbf{s}_{i_{d-r}}^{(1)} < \mathbf{s}_{i_{d-r+1}}^{(0)} + \lambda^* \cdot \mathbf{s}_{i_{d-r+1}}^{(1)}$ and $\mathbf{s}_{i_{d-r+1}}^{(0)} + \lambda^* \cdot \mathbf{s}_{i_{d-r+1}}^{(1)} = \ldots = \mathbf{s}_{i_{d+R}}^{(0)} + \lambda^* \cdot \mathbf{s}_{i_{d+R}}^{(1)}$. Define $\mathcal{J}_0 = \{i_{d-r+1}, \ldots, i_{d+R}\} \subset \{1, \ldots, D\}$. It holds for all $\mathcal{J} \in d - \operatorname{idx}\min\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\}$

$$\mathcal{J} = \{i_1, \ldots, i_{d-r}\} \cup \mathcal{J}_r \,, \tag{B.23}$$

for some $\mathcal{J}_r \subset \mathcal{J}_0$ with $\#\mathcal{J}_r = r$. Define the partial sum

$$\zeta_l := \underbrace{\sum_{j=1}^{d-r} \mathbf{s}_{i_j}^{(1)}}_{const} + \sum_{j=0}^{r-1} \mathbf{s}_{i_{d-r+l+j}}^{(1)} . \tag{B.24}$$

By the definition of $i_1, \ldots, i_D$ it holds $\zeta_l \leq \zeta_{l+1}$ for all $l \in \{1, \ldots, R\}$ and

$$\zeta_1 \leq \sum_{j \in \mathcal{J}_r} \mathbf{s}_j^{(1)} \leq \zeta_{R+1} \tag{B.25}$$

for all $\mathcal{J}_r \subset \mathcal{J}_0$ with $\#\mathcal{J}_r = r$. By the proof of Lemma 4.3.3, there exists $\mathbf{m}^-, \mathbf{m}^+ \in d - \operatorname{idx\,min}\{\mathbf{s}_i^{(0)} + \lambda^* \cdot \mathbf{s}_i^{(1)} < 0\}$ with $S_1(\mathbf{m}^+) \leq \kappa \leq S_1(\mathbf{m}^-)$ which implies $\zeta_1 \leq \kappa \leq \zeta_{R+1}$. Especially, there exists a l_0 with $\zeta_{l_0} \leq \kappa \leq \zeta_{l_0+1}$. Setting the new $\mathbf{m}^-$ and $\mathbf{m}^+$ as $\tilde{\mathbf{m}}^-, \tilde{\mathbf{m}}^+ \in \mathcal{V}_{0,d}$ with $\operatorname{supp} \tilde{\mathbf{m}}^- = \{i_1, \ldots, i_{d-r}\} \cup \{i_{d-r+l_0+1}, \ldots, i_{d+l_0}\}$ and $\operatorname{supp} \tilde{\mathbf{m}}^+ = \{i_1, \ldots, i_{d-r}\} \cup \{i_{d-r+l_0}, \ldots, i_{d+l_0-1}\}$ shows the claim. ■

B.2 Proof for Result in Chapter 5

Theorem 5.3.1 can be easily generalized to $\mathbf{T}$ forming an overcomplete system, see Theorem A.1 in Wimmer et al. [194]. The proof of Theorem 5.3.1 is split in several parts.

Lemma B.2.1. *The optimization problem* (5.1) *always has a solution* $\bar{\mathbf{S}}^{(\mathrm{id})} \in \mathbb{R}^{D_1 \times D_2}$ *given by*

$$\bar{\mathbf{S}}_{i,j}^{(\mathrm{id})} = \begin{cases} \mathbf{R}_{i,j} , & (i,j) \in \mathcal{J} \\ 0 , & (i,j) \notin \mathcal{J} \end{cases} , \tag{B.26}$$

where $\mathcal{J} \in d - \operatorname{idx\,max}\{|\mathbf{R}_{i_0,j_0}| : (i_0, j_0) \in \{1, \ldots, D_1\} \times \{1, \ldots, D_2\}\}$ *and* $k - \operatorname{idx\,max}\{\alpha_i : i \in \mathcal{I}\} := k - \operatorname{idx\,min}\{-\alpha_i : i \in \mathcal{I}\}$ *for a index set* $\mathcal{I}$ *and* $\alpha_i \in \mathbb{R}$ *for* $i \in \mathcal{I}$.

Proof of Lemma B.2.1. To solve (5.1), the optimization problem can be rewritten into its equivalent, squared form

$$\inf_{\mathbf{S}^{(\mathrm{id})} \in \mathbb{R}^{D_1 \times D_2}} \left\| \mathbf{R} - \mathbf{S}^{(\mathrm{id})} \right\|_F^2 \text{ such that } \left\| \mathbf{S}^{(\mathrm{id})} \right\|_0 \leq d . \tag{B.27}$$

Now, this can be written as

$$\inf_{\mathbf{S}^{(\mathrm{id})}\in\mathbb{R}^{D_1\times D_2},\mathbf{M}\in\{0,1\}^{D_1\times D_2},\|\mathbf{M}\|_0\leq d} \left\|\mathbf{R}-\mathbf{M}\odot\mathbf{S}^{(\mathrm{id})}\right\|_F^2 \tag{B.28}$$

which is equal to

$$\inf_{\mathbf{S}^{(\mathrm{id})}\in\mathbb{R}^{D_1\times D_2},\mathbf{M}\in\{0,1\}^{D_1\times D_2},\|\mathbf{M}\|_0\leq d} \sum_{(i,j)\in\operatorname{supp}\mathbf{M}} (\mathbf{R}-\mathbf{S}^{(\mathrm{id})})_{i,j}^2 + \sum_{(i,j)\notin\operatorname{supp}\mathbf{M}} \mathbf{R}_{i,j}^2 \tag{B.29}$$

Now for an arbitrary $\mathbf{M}\in\{0,1\}^{D_1\times D_2}$ with $\|\mathbf{M}\|_0\leq d$ it holds

$$\inf_{\mathbf{S}^{(\mathrm{id})}\in\mathbb{R}^{D_1\times D_2}} \sum_{(i,j)\in\operatorname{supp}\mathbf{M}} (\mathbf{R}-\mathbf{S}^{(\mathrm{id})})_{i,j}^2 = 0 \tag{B.30}$$

by setting $\mathbf{S}_{i,j}^{(\mathrm{id})} = \mathbf{R}_{i,j}$ for all $(i,j)\in\operatorname{supp}\mathbf{M}$. Therefore, (B.29) is equivalent to

$$\inf_{\mathbf{M}\in\{0,1\}^{D_1\times D_2},\|\mathbf{M}\|_0\leq d} \sum_{(i,j)\notin\operatorname{supp}\mathbf{M}} \mathbf{R}_{i,j}^2 \tag{B.31}$$

which is solved by setting $\operatorname{supp}\bar{\mathbf{M}}\in d-\operatorname{idxmax}\{|\mathbf{R}_{i_0,j_0}| : (i_0,j_0)\in\{1,\dots,D_1\}\times\{1,\dots,D_2\}\}$. Together, $\bar{\mathbf{M}}\odot\mathbf{R}$ solve (5.1) which shows the claim. ■

Lemma B.2.2. *For each $\mathbf{S}^{(\mathrm{id})}\in\mathbb{R}^{D_1\times D_2}$ there exists a $\mathbf{T}\in\mathbb{R}^{D_1\times D_1}$ and a $\mathbf{S}^{(\mathbf{T})}\in\mathbb{R}^{D_1\times D_2}$ with $\left\|\mathbf{S}^{(\mathbf{T})}\right\|_0 = \left\|\mathbf{S}^{(\mathrm{id})}\right\|_0$ and $\mathbf{T}\cdot\mathbf{S}^{(\mathbf{T})} = \mathbf{S}^{(\mathrm{id})}$.*

Proof of Lemma B.2.2. Let $\mathbf{S}^{(\mathrm{id})}\in\mathbb{R}^{D_1\times D_2}$ be given. Now, set $\mathbf{S}^{(\mathbf{T})} := \mathbf{S}^{(\mathrm{id})}\in\mathbb{R}^{D_1\times D_2}$ and $\mathbf{T} := \mathrm{id}\in\mathbb{R}^{D_1\times D_1}$. By construction of $\mathbf{S}^{(\mathbf{T})}$ and $\mathbf{T}$, it holds $\mathbf{S}^{(\mathrm{id})} = \mathbf{T}\cdot\mathbf{S}^{(\mathbf{T})}$ and $\left\|\mathbf{S}^{(\mathbf{T})}\right\|_0 = \left\|\mathbf{S}^{(\mathrm{id})}\right\|_0$. ■

Corollary B.2.1. *Let $\rho^{(\mathbf{T})}$ be the infimum of* (5.2) *and $\rho^{(\mathrm{id})}$ be the minimum of* (5.1)*, respectively. Then it holds $\rho^{(\mathbf{T})}\leq\rho^{(\mathrm{id})}$.*

Proof of Corollary B.2.1. By Lemma B.2.1, (5.1) is always minimized by a $\bar{\mathbf{S}}^{(\mathrm{id})}\in\mathbb{R}^{D_1\times D_2}$. By using Lemma B.2.2, there exists $\hat{\mathbf{T}}\in\mathbb{R}^{D_1\times D_1}$ and $\hat{\mathbf{S}}^{(\hat{\mathbf{T}})}\in\mathbb{R}^{D_1\times D_2}$ with $\bar{\mathbf{S}}^{(\mathrm{id})} = \hat{\mathbf{T}}\cdot\hat{\mathbf{S}}^{(\hat{\mathbf{T}})}$ and $\left\|\bar{\mathbf{S}}^{(\mathrm{id})}\right\|_0 = \left\|\hat{\mathbf{S}}^{(\hat{\mathbf{T}})}\right\|_0 = d$. Thus, $(\hat{\mathbf{T}},\hat{\mathbf{S}}^{(\hat{\mathbf{T}})})$ is feasible for (5.2) and consequently $\rho^{(\mathbf{T})}\leq\rho^{(\mathrm{id})}$. ■

Proof of Theorem 5.3.1. By Corollary B.2.1, $\rho^{(\mathbf{T})} \leq \rho^{(\mathrm{id})}$ always holds. If $\rho^{(\mathbf{T})} < \rho^{(\mathrm{id})}$, the proof would be finished. Therefore, assume w.l.o.g. $\rho^{(\mathbf{T})} = \rho^{(\mathrm{id})}$.

First part of proof with $\operatorname{supp}\mathbf{S}^{(\mathbf{T})} = \operatorname{supp}\bar{\mathbf{S}}^{(\mathrm{id})}$, *i.e.* point 2 in Theorem 5.3.1.

Let $\bar{\mathbf{S}}^{(\mathrm{id})} \in \mathbb{R}^{D_1 \times D_2}$ solve (5.1). Then, there exists an equivalent feasible point $(\hat{\mathbf{T}}, \hat{\mathbf{S}}^{(\hat{\mathbf{T}})}) = (\mathrm{id}, \bar{\mathbf{S}}^{(\mathrm{id})}) \in \mathbb{R}^{D_1 \times D_1} \times \mathbb{R}^{D_1 \times D_2}$ for (5.2), see proof of Lemma B.2.2. By the assumption $\rho^{(\mathbf{T})} = \rho^{(\mathrm{id})}$, $(\hat{\mathbf{T}}, \hat{\mathbf{S}}^{(\hat{\mathbf{T}})})$ also solves (5.2). Especially, $\hat{\mathbf{T}}$ defines a *global* minimum of the smooth, convex function

$$H : \mathbb{R}^{D_1 \times D_1} \to \mathbb{R}, \mathbf{T} \mapsto \left\| \mathbf{R} - \mathbf{T} \cdot \hat{\mathbf{S}}^{(\hat{\mathbf{T}})} \right\|_F^2 . \tag{B.32}$$

A necessary, and by convexity of H even sufficient, condition for $\hat{\mathbf{T}}$ to minimize H is given by

$$\left. \frac{\partial H}{\partial \mathbf{T}} \right|_{\mathbf{T}=\hat{\mathbf{T}}} = 0 \in \mathbb{R}^{D_1 \times D_1} . \tag{B.33}$$

It holds

$$\frac{\partial H}{\partial \mathbf{T}} = \frac{\partial}{\partial \mathbf{T}} \left\| \mathbf{R} - \mathbf{T} \cdot \hat{\mathbf{S}}^{(\hat{\mathbf{T}})} \right\|_F^2 = 2 \cdot (\mathbf{T} \cdot \hat{\mathbf{S}}^{(\hat{\mathbf{T}})} - \mathbf{R}) \cdot (\hat{\mathbf{S}}^{(\hat{\mathbf{T}})})^T . \tag{B.34}$$

Combining (B.33) and (B.34) defines a necessary and sufficient condition for $\mathbf{T}$ yielding a minimum for H given by

$$(\mathbf{R} - \mathbf{T} \cdot \hat{\mathbf{S}}^{(\hat{\mathbf{T}})}) \cdot (\hat{\mathbf{S}}^{(\hat{\mathbf{T}})})^T = 0 . \tag{B.35}$$

Consequently, (B.35) is a necessary condition for $(\hat{\mathbf{T}}, \hat{\mathbf{S}}^{(\hat{\mathbf{T}})})$ to define the minimum for (5.2). By construction, $\hat{\mathbf{T}} \cdot \hat{\mathbf{S}}^{(\hat{\mathbf{T}})} = \bar{\mathbf{S}}^{(\mathrm{id})}$, $\hat{\mathbf{T}} = \mathrm{id}$ and $\hat{\mathbf{S}}^{(\hat{\mathbf{T}})} = \bar{\mathbf{S}}^{(\mathrm{id})}$. Choose a $\bar{\mathcal{J}} \in d - \operatorname{idxmax}\{|\mathbf{R}_{i_0,j_0}| : (i_0, j_0) \in \{1, \ldots, D_1\} \times \{1, \ldots, D_2\}\}$ which is by the assumption of $\mathbf{R}_{i,j}$ i.i.d. $\mathcal{N}(0,1)$ with probability $\mathbb{P} = 1$ unique. By Lemma B.2.1, $\bar{\mathbf{S}}^{(\mathrm{id})}$ is given by $\bar{\mathbf{S}}_{i,j}^{(\mathrm{id})} = \chi_{(i,j)}(\bar{\mathcal{J}}) \cdot \mathbf{R}_{i,j}$. Combining this with (B.35) leads to the necessary condition

$$(\mathbf{R} \odot \mathbf{M}) \cdot (\mathbf{R} \odot (\mathbf{1} - \mathbf{M}))^T = 0 \tag{B.36}$$

with $\mathbf{1} \in \mathbb{R}^{D_1 \times D_2}$ and

$$\mathbf{M}_{i,j} = \begin{cases} 1, & \text{if } (i,j) \notin \bar{\mathcal{J}} \\ 0, & \text{if } (i,j) \in \bar{\mathcal{J}} \end{cases} . \tag{B.37}$$

In the following an upper bound ε for the probability $\mathbb{P}((\mathbf{R} \odot \mathbf{M}) \cdot (\mathbf{R} \odot (\mathbf{1} - \mathbf{M}))^T = 0)$

will be computed. By using the fact that $(\mathbf{R} \odot \mathbf{M}) \cdot (\mathbf{R} \odot (\mathbf{1} - \mathbf{M}))^T = 0$ is a necessary condition for $(\hat{\mathbf{T}}, \hat{\mathbf{S}}^{(\hat{\mathbf{T}})})$ being a minimizer to (5.2), which is equivalent to $\rho^{(\mathbf{T})} = \rho^{(\mathrm{id})}$, it finally holds

$$\begin{aligned} \mathbb{P}(\rho^{(\mathbf{T})} < \rho^{(\mathrm{id})}) &= 1 - \mathbb{P}(\rho^{(\mathbf{T})} \geq \rho^{(\mathrm{id})}) && \text{(B.38)} \\ &= 1 - \mathbb{P}(\rho^{(\mathbf{T})} = \rho^{(\mathrm{id})}) && \text{(B.39)} \\ &\geq 1 - \mathbb{P}((\mathbf{R} \odot \mathbf{M}) \cdot (\mathbf{R} \odot (\mathbf{1} - \mathbf{M}))^T = 0)\,. && \text{(B.40)} \end{aligned}$$

Thus, the last step is to find an upper bound $\varepsilon \geq \mathbb{P}((\mathbf{R} \odot \mathbf{M}) \cdot (\mathbf{R} \odot (\mathbf{1} - \mathbf{M}))^T = 0)$. It holds

$$((\mathbf{R} \odot \mathbf{M}) \cdot (\mathbf{R} \odot (\mathbf{1} - \mathbf{M}))^T)_{i,j} = \begin{cases} \sum_{k \in \bar{\mathcal{J}}_{i,j}} \mathbf{R}_{i,k} \mathbf{R}_{j,k}, & \text{if } \bar{\mathcal{J}}_{i,j} \neq \emptyset \\ 0, & \text{else} \end{cases}, \tag{B.41}$$

where for each $(i, j) \in \{1, \ldots, D_1\}^2$,

$$\bar{\mathcal{J}}_{i,j} := \{k \in \{1, \ldots, D_2\} : (i,k) \notin \bar{\mathcal{J}}\ ,\ (j,k) \in \bar{\mathcal{J}}\}\,. \tag{B.42}$$

Now assume $\mathcal{J}_\times \subset (\{1, \ldots, D_1\} \times \{1, \ldots, D_2\})^2$ with $\mathcal{J}_\times \neq \emptyset$ to be given, then

$$\mathbb{P}\Big(\sum_{(i_1,j_1),(i_2,j_2) \in \mathcal{J}_\times} \mathbf{R}_{i_1,j_1} \cdot \mathbf{R}_{i_2,j_2} = 0\Big) = 0\,. \tag{B.43}$$

This equality holds since for each $\mathcal{J}_\times \subset (\{1, \ldots, D_1\} \times \{1, \ldots, D_2\})^2$ with $\mathcal{J}_\times \neq \emptyset$, $\sum_{(i_1,j_1),(i_2,j_2) \in \mathcal{J}_\times} \mathbf{R}_{i_1,j_1} \cdot \mathbf{R}_{i_2,j_2}$ follows a continuous probability distribution.

Consequently,

$$\mathbb{P}((\mathbf{R}\odot\mathbf{M})\cdot(\mathbf{R}\odot(\mathbf{1}-\mathbf{M}))^T=0) \tag{B.44}$$

$$=\mathbb{P}(\forall(i,j):\bar{\mathcal{J}}_{i,j}=\emptyset\ \vee(\bar{\mathcal{J}}_{i,j}\neq\emptyset\wedge\sum_{k\in\bar{\mathcal{J}}_{i,j}}\mathbf{R}_{i,k}\cdot\mathbf{R}_{j,k}=0)) \tag{B.45}$$

$$\leq\mathbb{P}((\forall(i,j):\bar{\mathcal{J}}_{i,j}=\emptyset)\vee(\exists\mathcal{J}_\times\neq\emptyset:\sum_{(i_1,j_1),(i_2,j_2)\in\mathcal{J}_\times}\mathbf{R}_{i_1,j_1}\cdot\mathbf{R}_{i_2,j_2}=0)) \tag{B.46}$$

$$\leq\mathbb{P}(\forall(i,j):\bar{\mathcal{J}}_{i,j}=\emptyset)+\sum_{\substack{\mathcal{J}_\times\subset(\{1,\dots,D_1\}\times\{1,\dots,D_2\})^2\\ \mathcal{J}_\times\neq\emptyset}}\mathbb{P}(\sum_{(i_1,j_1),(i_2,j_2)\in\mathcal{J}_\times}\mathbf{R}_{i_1,j_1}\cdot\mathbf{R}_{i_2,j_2}=0) \tag{B.47}$$

$$=\mathbb{P}(\forall(i,j):\bar{\mathcal{J}}_{i,j}=\emptyset)\ , \tag{B.48}$$

where the inequality (B.47) uses the subadditivity of probability measures and the final equality (B.48) is achieved by using (B.43). By looking at the definition of $\bar{\mathcal{J}}_{i,j}$, it can be seen that $\forall(i,j):\bar{\mathcal{J}}_{i,j}=\emptyset$ only happens if for each $k\in\{1,\dots,D_2\}$ either

$$\forall i\in\{1,\dots,D_1\}:(i,k)\in\bar{\mathcal{J}} \tag{B.49}$$

or

$$\forall i\in\{1,\dots,D_1\}:(i,k)\notin\bar{\mathcal{J}} \tag{B.50}$$

holds true. This shows, that $\rho^{(\mathbf{T})}=\rho^{(\mathrm{id})}$ is only possible in the trivial case, where each of the D_2 filters (with D_1 coefficients) is either completely pruned or not pruned at all.

Therefore, the probability

$$\mathbb{P}(\forall k:(\forall i:(i,k)\in\bar{\mathcal{J}})\vee(\forall i:(i,k)\notin\bar{\mathcal{J}})) \tag{B.51}$$

needs to be computed. Due to the i.i.d. assumption of the $\mathbf{R}_{i,k}$, all (i,k) have the same probability of being in $\bar{\mathcal{J}}$. Thus, deciding $(i,k)\in\bar{\mathcal{J}}$ or $(i,k)\notin\bar{\mathcal{J}}$ for all $(i,k)\in\{1,\dots,D_1\}\times\{1,\dots,D_2\}$ together can equivalently be modeled with choosing a subset of size d from a set of size $D_1\cdot D_2$, where each subset has the same probability of being sampled, *i.e.* with probability $\frac{1}{\binom{D_1\cdot D_2}{d}}$ since $\#\bar{\mathcal{J}}=d$.

Furthermore, (B.51) is only possible if $d=m\cdot D_1$ for some $m\in\mathbb{N}$. Otherwise, there needs to exists at least one k,i,j such that $(i,k)\notin\bar{\mathcal{J}}$ and $(j,k)\in\bar{\mathcal{J}}$. Assuming $d=m\cdot D_1$

for some $m \in \mathbb{N}$, there exist exactly $\binom{D_2}{m}$ different choices to find m many k that satisfy $\forall i : (i,k) \in \bar{\mathcal{J}}$ which, by the discussion above, all have similar probability.

Altogether, the probability (B.51) is given by

$$\varepsilon = \begin{cases} 0 & , \text{if } d \not\equiv 0 \mod D_1 \\ \frac{\binom{D_2}{\frac{d}{D_1}}}{\binom{D_1 \cdot D_2}{d}} & , \text{if } d \equiv 0 \mod D_1 \end{cases} . \tag{B.52}$$

Finally,

$$\begin{aligned} \mathbb{P}(\rho^{(\mathbf{T})} < \rho^{(\mathrm{id})}) \geq & 1 - \mathbb{P}((\mathbf{R} \odot \mathbf{M}) \cdot (\mathbf{R} \odot (\mathbf{1} - \mathbf{M}))^T = 0) && \text{(B.53)} \\ \geq & 1 - \mathbb{P}(\forall (i,j) : \bar{\mathcal{J}}_{i,j} = \emptyset) && \text{(B.54)} \\ = & 1 - \mathbb{P}(\forall k : (\forall i : (i,k) \in \bar{\mathcal{J}}) \vee (\forall i : (i,k) \notin \bar{\mathcal{J}})) && \text{(B.55)} \\ = & 1 - \varepsilon \, , && \text{(B.56)} \end{aligned}$$

which finishes the first part of the proof where $\operatorname{supp} \mathbf{S}^{(\mathbf{T})}$ is fixed to be equal to $\operatorname{supp} \bar{\mathbf{S}}^{(\mathrm{id})}$.

Second part of proof with arbitrary $\operatorname{supp} \mathbf{S}^{(\mathbf{T})}$, *i.e.* point 1 in Theorem 5.3.1.

As shown in the first part of the proof, (B.51) is a necessary condition for $\rho^{(\mathbf{T})} = \rho^{(\mathrm{id})}$. This means that all columns of $\bar{\mathbf{S}}^{(\mathrm{id})}$ are either $\bar{\mathbf{S}}^{(\mathrm{id})}_{:,k} = 0 \in \mathbb{R}^{D_1}$ or $\left\| \bar{\mathbf{S}}^{(\mathrm{id})}_{:,k} \right\|_0 = D_1$ which is therefore assumed from now on.

By assumption, $0 < d$ and therefore there exists a k_0 with $\left\| \bar{\mathbf{S}}^{(\mathrm{id})}_{:,k_0} \right\|_0 = D_1$. Now, set $\tilde{\mathbf{T}}_{:,1} = \bar{\mathbf{S}}^{(\mathrm{id})}_{:,k_0}$ and $\tilde{\mathbf{T}}_{:,k} = (\hat{\mathbf{T}})_{:,k} = \mathrm{id}_{:,k}$ for all other j. Then, with $\mathbb{P} = 1$, $\tilde{\mathbf{T}}$ still forms a basis.

Setting $\tilde{\mathbf{S}}^{(\tilde{\mathbf{T}})}_{i,k_0} = \delta_{i,1}$ for all $i \in \{1, \ldots, D_1\}$ yields $(\tilde{\mathbf{T}} \cdot \tilde{\mathbf{S}}^{(\tilde{\mathbf{T}})})_{:,k_0} = \mathbf{R}_{:,k_0} = \bar{\mathbf{S}}^{(\mathrm{id})}_{:,k_0}$. For all other $k \neq k_0$ with $\left\| \bar{\mathbf{S}}^{(\mathrm{id})}_{:,k} \right\|_0 = D_1$ there exists a $\tilde{\mathbf{S}}^{(\tilde{\mathbf{T}})}_{:,k}$ with $\left\| \tilde{\mathbf{S}}^{(\tilde{\mathbf{T}})}_{:,k} \right\|_0 \leq \left\| \bar{\mathbf{S}}^{(\mathrm{id})}_{:,k} \right\|_0 = D_1$ and $(\tilde{\mathbf{T}} \cdot \tilde{\mathbf{S}}^{(\tilde{\mathbf{T}})})_{:,k} = \bar{\mathbf{S}}^{(\mathrm{id})}_{:,k} = \mathbf{R}_{:,k}$ since $\tilde{\mathbf{T}}$ forms a basis.

Setting the remaining $\tilde{\mathbf{S}}^{(\tilde{\mathbf{T}})}_{:,k} = 0$ leads to $\tilde{\mathbf{T}} \cdot \tilde{\mathbf{S}}^{(\tilde{\mathbf{T}})} = \bar{\mathbf{S}}^{(\mathrm{id})}$ and $\left\| \tilde{\mathbf{S}}^{(\tilde{\mathbf{T}})} \right\|_0 \leq \left\| \bar{\mathbf{S}}^{(\mathrm{id})} \right\|_0 - (D_1 - 1) < \left\| \bar{\mathbf{S}}^{(\mathrm{id})} \right\|_0$. The last inequality holds, since $D_1 > 1$ is assumed.

Finally, one of the, at least, $D_1 - 1 > 0$ remaining coefficients which were not spend up to now, can be used to better approximate one column of $\mathbf{R}$ which is completely zeroed in $\bar{\mathbf{S}}^{(\mathrm{id})}$. Such a column j_0 must fulfill $\mathbf{R}_{:,j_0} \neq 0$ and $\bar{\mathbf{S}}^{(\mathrm{id})}_{:,j_0} = 0$. Since $d < D_1 \cdot D_2$ and $\mathbf{R}_{i,j}$ i.i.d. $\mathcal{N}(0,1)$, such a column j_0 exists with $\mathbb{P} = 1$. Since $\tilde{\mathbf{T}}$ forms a basis, there

exist l_0, α_{l_0} such that

$$\left\|\mathbf{R}_{:,j_0} - \alpha_{l_0}\tilde{\mathbf{T}}_{:,l_0}\right\|_2 < \left\|\mathbf{R}_{:,j_0}\right\|_2 . \tag{B.57}$$

Setting $\tilde{\mathbf{S}}^{(\tilde{\mathbf{T}})}_{l_0,j_0} = \alpha_{l_0}$ leads to

$$\left\|\mathbf{R} - \tilde{\mathbf{T}} \cdot \tilde{\mathbf{S}}^{(\tilde{\mathbf{T}})}\right\|_F^2 - \left\|\mathbf{R} - \bar{\mathbf{S}}^{(\mathrm{id})}\right\|_F^2 = \left\|\mathbf{R}_{:,j_0} - \alpha_{l_0}\tilde{\mathbf{T}}_{:,l_0}\right\|_2^2 - \left\|\mathbf{R}_{:,j_0}\right\|_2^2 < 0 , \tag{B.58}$$

which finishes the proof. ■

Abbreviations and Symbols

Abbreviations

AI Artificial intelligence

Bias Bias term of the learning error (bias-variance trade off)

CNN Convolutional neural network

COPS Combined Pruning Scores (pruning method)

CSR Compressed sparse row (data format)

DL Deep learning

DNN Deep neural network

DRT Dimensionality reduced training

DST Dynamic sparse training

FB Filter basis

FDMI Faster dynamic matrix inverses

FLOP Floating-point operation

FreezeNet DNN with parts of the weights frozen at their random, initial position

FreezeNet-WD A FreezeNet with weight decay applied to the untrained weights

FT Pruning a pre-trained model and fine-tune it afterwards

G-COPS COPS with SNIP and GraSP

GMP Gradual Magnitude Pruning (pruning method)

GraSP Gradient Signal Preservation (pruning method)

GSS Generalized synaptic score

$\texttt{in}_{n_c}$ Sharing one FB for each n_c input channels

IP Interspace pruning

LE Learning error

LP Linear program

LR Learning rate

LTH Lottery Ticket Hypothesis (pruning method)

M-COPS COPS with SNIP and magnitude pruning

MLP Multi layer perceptron

MSE Mean-squared error

ONB Orthonormal basis

$\texttt{out}_{n_c}$ Sharing one FB for each n_c output channels

PaI Pruning at initialization

$\texttt{pm}_{\sigma}$ Initialization with 0.5 chance to be either $+$ or $-$ σ

ReLU Rectified linear unit

RigL Rigging the Lottery Ticket Hypothesis (pruning method)

SE Squared error

SET Sparse Evolutionary Training (pruning method)

SGD Stochastic gradient descent

sgn Sign of a real number

sim Similarity of pruning masks

SNIP Single Shot Network Pruning (pruning method)

SOTA State-of-the-art

SP Standard pruning of spatial weights

span Linear space spanned by a set of vectors

std Standard deviation

supp Support of vector or tensor

SynFlow Synaptic Flow (pruning method)

U-COPS COPS with SNIP and USNIP

USNIP SNIP with random labels (pruning method)

Var Variance term of the learning error (bias-variance trade off)

Symbols

$\mathbf{1}$ One tensor of arbitrary order and size

$\star$ 2D Convolutional operation for two input images

✡ 2D Convolutional operation describing a whole convolutional layer

$\langle \cdot, \cdot \rangle$ Euclidean inner product

$\odot$ Hadamard product

id Identity operator

$k - \operatorname{idx}\max\{\alpha_i : i \in \mathcal{I}\}$ Set of sets where each $\mathcal{J} \in k - \operatorname{idx}\max\{\alpha_i : i \in \mathcal{I}\}$ has $k_0 = \min\{k, \#\mathcal{I}\}$ indices corresponding to the k_0 biggest entries in $\{\alpha_i : i \in \mathcal{I}\}$

$k-\operatorname{idx}\min\{\alpha_i : i \in \mathcal{I}\}$ Equivalent to $k-\operatorname{idx}\max$, defined as $k-\operatorname{idx}\max\{-\alpha_i : i \in \mathcal{I}\}$

Δ Laplace operator, *i.e.* $\langle \nabla(\cdot), \nabla(\cdot) \rangle$

$\searrow$ Monotonically decreasing function

∇ Order 1 differential operator, *i.e.* $\nabla_{\mathbf{a}} h = \frac{\partial h}{\partial \mathbf{a}}$

$\|\cdot\|_p$ Frobenius norm if $p = F$, otherwise L_p norm for $p \in [0, \infty)$

$\otimes$ Outer product

$\hat{\partial}$ Superderivative operator

$\{\cdot\}^{[l]}$ Corresponding element of $\{\cdot\}$ in layer l

$\{\cdot\}^{(t)}$ Corresponding element of $\{\cdot\}$ in training step t

α_κ Balance factor for COPS

β Filter basis coefficient vector

γ Weight between $\mathbf{m}^+, \mathbf{m}^-$ generating the solution of the relaxed COPS problem

δ Kronecker delta

ε Upper bound of the probability for IP to not be strictly better than SP if the same coefficients are trained

ε_n Numerical precision for COPS Algorithm

$\varepsilon_{q,D}$ Threshold for determining frozen weights of a DNN with D parameters and compression rate q

η Learning rate

Θ Parameters of a DNN f_Θ with $\Theta \in \mathbb{R}^D$

ϑ Low dimensional parameters $\vartheta \in \mathbb{R}^d$ of a sparsely trained DNN $f_{\Psi(\vartheta)}$

κ Constraint for COPS problem

$\boldsymbol{\kappa_{\min}}$ Minimal fulfillable constraint for COPS problem

Λ Lagrangian of the relaxed COPS problem

λ Input of the dual function

$\lambda_+, \lambda_-, \lambda_0$ Upper, lower boundary and midpoint for bisection in COPS Algorithm

λ^* Value optimizing the dual of the COPS problem

$\lambda_{\mathscr{R}}$ Strength of the regularization term in the loss function

μ Mean of random initialization for DNNs

ν White noise with variance σ_ν^2

ξ An element of the set of all superderivatives

ξ_-, ξ_+ Left and right derivative

$\rho^{(\mathrm{id})}$ Optimum of the standard pruning problem with a fixed dictionary id

$\rho^{(\mathbf{T})}$ Optimum of the interspace pruning problem using an adaptive dictionary $\mathbf{T}$

σ Standard deviation of a random initialization for a DNN

σ_ν^2 Variance of the the white noise ν

Φ Additional transformation changing the basis of the submanifold coefficients

ϕ Non-linear activation function for a layer in a DNN

φ Convolutional filter $\varphi \in \mathbb{R}^{K \times K}$

$\bar{\varphi}$ Coefficient vector of φ w.r.t. the standard basis $\mathcal{B}$

χ Characteristic function

Ψ Embedding spanning the low dimensional submanifold

Ψ_z Local embedding of the submanifold around $z \in \mathcal{M}$

ψ Pruning transformation

A Accuracy

$\mathbf{A}^{(\mathcal{B},\mathcal{F})}$ Basis transformation matrix between basis $\mathcal{B}$ and $\mathcal{F}$

A_Δ Accuracy difference between two models

A_{f^*} General measure for validating the quality of DNNs

$\mathcal{A}_{\mathbf{in}}$ Input domain $\mathcal{A}_{\text{in}}$ is a superdomain of the data points $\mathcal{X}$

$\mathcal{A}_{\mathbf{out}}$ Target domain $\mathcal{A}_{\text{out}}$ is a superdomain of the target points $\mathcal{Y}$

B Batch size

$\mathcal{B}$ Standard basis

$\mathbf{b}$ Bias vector of a layer in a DNN

C Total number of classes for a classification task

$c_{\mathbf{in}}, c_{\mathbf{out}}$ Number of input and output channels (convolutional layer) / number of input and output nodes (fully connected layer)

D Number of parameters in a DNN

$D_1 \times D_0$ Resolution of dictionary used for sparse dictionary learning

$D_1 \times D_2$ Resolution of target matrix for sparse dictionary learning

d Sparsity $d \leq D$, *i.e.* dimension of effectively trained submanifold

d^* Optimal value of dual problem

$d_1 \times d_2$ Resolution of input image of a DNN/input feature map of a convolutional layer

$\tilde{d_1} \times \tilde{d_2}$ Resolution of output feature map of a convolutional layer

$d_{\mathbf{in}}, d_{\mathbf{out}}$ Input and output dimensions of the DNN, *i.e.* $\mathcal{A}_{\text{in}} \cong \mathbb{R}^{d_{\text{in}}}$ and $\mathcal{A}_{\text{out}} \cong \mathbb{R}^{d_{\text{out}}}$

$\mathbb{E}$ Expected value

$\mathbf{E}^{(i,j)}$ Element of the standard matrix basis $\mathcal{B}$ with $\mathbf{E}^{(i,j)}_{a,b} = \delta_{a,i} \cdot \delta_{b,j}$

$\mathcal{F}$ Filter basis

$\mathbf{F}^{(i)}$ i^{th} basis element of the FB $\mathcal{F}$

f^* Target function, *i.e.* ground truth label function for the data $\mathcal{X}$

f_I^* Ground truth label function for the sampled data

$f_{\text{test}}^*, f_{\text{train}}^*, f_{\text{val}}^*$ Ground truth label functions for test, training and validation data

$\bar{f}$ Solution for a fitted curve

f_Θ DNN with parameters Θ

g Dual function w.r.t. a convex optimziation problem

$\mathbf{g}$ Gradient of the parameters of a DNN, *i.e.* $\mathbf{g} = \nabla_\Theta \mathscr{L}$

H Convex function used in the proof of Theorem 5.3.1

$\mathcal{H}$ Hypothesis class of potential DNNs

$\mathbf{H}$ Hessian of the parameters of a DNN, *i.e.* $\mathbf{H} = \nabla_\Theta^2 \mathscr{L}$

h_{init} Initialization function for models with interspace representation

$\mathcal{I}_\lambda$ Index set of active scores contributing to g at λ

i_+, i_- Indices where $\mathbf{m}^+$ and $\mathbf{m}^-$ are not equal

$\bar{\mathcal{J}}, \bar{\mathcal{J}}_{i,j}, \mathcal{J}_\times$ Index sets used in the proof of Theorem 5.3.1

K Kernel/filter size of a convolutional layer

L Number of layers in a DNN

$\mathscr{L}$ Loss function used for optimizing the DNN

L_g Lipschitz constant for dual function g

$\mathcal{M}$ Submanifold where low dimensional training takes place

$\mathbf{M}$ Pruning mask used in the proofs of Lemma B.2.1 and Theorem 5.3.1

$\bar{\mathbf{M}}$ Optimal pruning mask derived in the proof of Lemma B.2.1

$\mathbf{m}$ Pruning/freezing mask

$\mathbf{m}^{\dagger}$ Optimal pruning mask for pruning with a single score

$\mathbf{m}^{+}, \mathbf{m}^{-}$ Approximate solutions of un-relaxed COPS problem

$\mathbf{m}^{*}$ Optimal point solving the linear relaxation of the COPS problem

$\bar{\mathbf{m}}$ Optimal mask solving the COPS problem

$\mathbf{m}^{\circ}$ Inner point guaranteeing Slater's condition

$\mathbf{m}^{(\alpha_\kappa)}$ Mask obtained by COPS with balance factor $\alpha_\kappa \in [0,1]$

$\mathbf{m}_\lambda$ Pruning mask satisfying $g(\lambda) = S_0(\mathbf{m}_\lambda) + \lambda \cdot (S_1(\mathbf{m}_\lambda) - \kappa)$

$\mathbb{N}$ Set of natural numbers

$n_\#$ Number of elements in a generalized filter dictionary spanning the interspace

n_c Number of channels sharing one FB for $\texttt{out}_{n_c}$ and $\texttt{in}_{n_c}$ FB sharing

$n_\mathcal{F}$ Number of filter bases in a CNN

$O(\cdot)$ For $f, g : \mathbb{R} \to \mathbb{R}$, $f \in O(g) \Leftrightarrow \exists x_0, M \in \mathbb{R}$ such that $|f(x)| \leq M \cdot g(x)$ for all $x \geq x_0$

$\mathbb{P}$ Probability measure

p^* Optimal value of primal problem

$\hat{p}$ Estimated probability density

$p_\mathcal{X}$ Density of the data distribution

$p_{f^*(\mathcal{X})|\mathcal{X}}$ Conditional target density given the underlying data distribution

$\mathcal{Q}_{1,d}$ Feasible points for relaxed COPS problem fulfilling the constraint on control score

q Compression rate d/D

q_β Real compression rate including bias and batch normalization parameters

q_1, q_2 Discrete, conditional distributions

$\mathcal{R}$ Function determinining the synaptic saliency

$\mathbf{R}$ Dense target matrix which is approximated for sparse dictionary learning

$\mathbb{R}$ Set of real numbers

$\mathscr{R}$ Regularization term added to the loss function

$\mathbf{r}$ Disturbance of parameter vector caused by pruning a model

S Pruning score function

S_0, S_1 Target and control score function

$\mathbf{S}^{(\mathrm{id})}, \mathbf{S}^{(\mathbf{T})}$ Coefficients w.r.t. standard basis/dictionary $\mathbf{T}$ used to approximate $\mathbf{R}$ sparsely

$\hat{\mathbf{S}}^{(\mathrm{id})}$ Feasible point for problem (5.1)

$\hat{\mathbf{S}}^{(\hat{\mathbf{T}})}, \hat{\mathbf{T}}$ Feasible point for problem (5.2)

$\tilde{\mathbf{S}}^{(\tilde{\mathbf{T}})}, \tilde{\mathbf{T}}$ Feasible point of (5.2) which improves (5.1)

$\bar{\mathbf{S}}^{(\mathrm{id})}$ Optimal coefficients approximating $\mathbf{R}$ sparsely in the standard basis

$\mathbf{s}$ Pruning/freezing score vector

$\mathbf{s}^{(0)}$ Target score vector

$\mathbf{s}^{(1)}$ Control score vector

T Total number of training steps

$\mathcal{T}$ Subset of all classes of a classification task

$\mathbf{T}$ Dictionary optimized for sparse dictionary learning

$\mathscr{T}_{\mathcal{Z},\mathcal{X}_{\mathrm{train}},f^*_{\mathrm{train}}}$ Training algorithm on training data $(\mathcal{X}_{\mathrm{train}}, f^*_{\mathrm{train}})$ with hyperparameters $\mathcal{Z}$

t_0 First step where pruning is applied for GMP, also rewinding step for LTH

t_1 Last step where pruning is applied for GMP

t_p Frequency of updates for pruning mask for DST and GMP

$\mathbf{V}$ Input feature map, also called pre-activations

$\mathcal{V}_{0,d}$ Masks optimized for the un-relaxed COPS problem

$\mathcal{V}_{1,d}$ Masks optimized for the relaxed COPS problem

$\mathbf{W}$ Output feature map, also called activations

$\mathbf{X}, \mathbf{x}$ Data point $\mathbf{X} \in \mathcal{X}$ (for tensors) or $\mathbf{x} \in \mathcal{X}$ (for vectors) in the dataset

$\mathcal{X}$ Set of all data points of the dataset

$\mathcal{X}_l$ Labeled data

$\mathcal{X}^B_{\text{train}}$ Batch of training data with batch size B

$\mathcal{X}_{\text{test}}, \mathcal{X}_{\text{train}}, \mathcal{X}_{\text{val}}$ Test, training and validation data

$\mathcal{Y}$ Target points $\mathcal{Y} \ni f^*(\mathcal{X})$

$\hat{\mathbf{y}}$ Prediction of a DNN

y_ν Noisy target of regression task for the bias-variance trade off

$\mathcal{Z}$ Set of hyperparameter choices for a DNN training

z Point on a submanifold $\mathcal{M}$

Lists of Algorithms, Figures and Tables

List of Algorithms

List of Figures

List of Tables

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016. URL `http://arxiv.org/abs/1603.04467`. Last accessed: 09/13/2022.

[2] Michal Aharon, Michael Elad, and Alfred M. Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, 2006.

[3] Dario Amodei, Danny Hernandez, Girish Sastry, Jack Clark, Greg Brockman, and Ilya Sutskever. Ai and compute. OpenAI Blog, 2018. URL `https://openai.com/blog/ai-and-compute/`. Last accessed: 09/12/2022.

[4] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems*, 13(3):1–18, 2017.

[5] Maximilian Baader, Matthew Mirman, and Martin Vechev. Universal approximation with certified networks. In *8th International Conference on Learning Representations*, 2020.

[6] Brian Bartoldson, Ari Morcos, Adrian Barbu, and Gordon Erlebacher. The

generalization-stability tradeoff in neural network pruning. In *Advances in Neural Information Processing Systems 33*, 2020.

[7] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In *6th International Conference on Learning Representations*, 2018.

[8] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[9] Christopher M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995. ISBN 978-0-19-853864-6.

[10] Davis W. Blalock, Jose J. G. Ortiz, Jonathan Frankle, and John V. Guttag. What is the state of neural network pruning? In *Proceedings of Machine Learning and Systems 2*, 2020.

[11] Léon Bottou. On-line Learning and Stochastic Approximations. In *On-Line Learning in Neural Networks*, pages 9–42. Cambridge University Press, 1999. ISBN 978-0-521-65263-6.

[12] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of the 19th International Conference on Computational Statistics*, 2010.

[13] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. ISBN 0-521-83378-7.

[14] Siegmund Brandt. *Data Analysis: Statistical and Computational Methods for Scientists and Engineers*. Springer New York, 1998. ISBN 978-0-387-98498-8.

[15] Rebekka Burkholz, Nilanjana Laha, Rajarshi Mukherjee, and Alkis Gotovos. On the existence of universal lottery tickets. In *10th International Conference on Learning Representations*, 2022.

[16] Miguel A. Carreira-Perpinan and Yerlan Idelbayev. ”Learning-compression” algorithms for neural net pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[17] Yves Chauvin. A back-propagation algorithm with optimal use of hidden units. In *Advances in Neural Information Processing Systems 1*, 1989.

[18] Jianda Chen, Shangyu Chen, and Sinno J. Pan. Storage efficient and dynamic flexible runtime channel pruning via deep reinforcement learning. In *Advances in Neural Information Processing Systems 33*, 2020.

[19] Scott S. Chen, David L. Donoho, and Michael A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, 1998.

[20] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[21] Albert Cohen, Wolfgang Dahmen, and Ronald Devore. Compressed sensing and best k -term approximation. *Journal of the American Mathematical Society*, 22 (1):211–231, 2009.

[22] Taco S. Cohen and Max Welling. Group equivariant convolutional networks. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

[23] Benjamin Coors, Alexandru P. Condurache, and Andreas Geiger. Spherenet: Learning spherical representations for detection and classification in omnidirectional images. In *Proceedings of the European Conference on Computer Vision*, 2018.

[24] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems 28*, 2015.

[25] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[26] George B. Dantzig. Origins of the Simplex Method. In *A History of Scientific Computing*, pages 141–151. Association for Computing Machinery, 1990. ISBN 0-201-50814-1.

[27] Pau de Jorge, Amartya Sanyal, Harkirat Behl, Philip Torr, Grégory Rogez, and Puneet K. Dokania. Progressive skeletonization: Trimming more fat from a network at initialization. In *9th International Conference on Learning Representations*, 2021.

[28] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[29] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems 27*, 2014.

[30] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *CoRR*, abs/1907.04840v2, 2019. URL `http://arxiv.org/abs/1907.04840v2`. Last accessed: 03/31/2022.

[31] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[32] James Diffenderfer and Bhavya Kailkhura. Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network. In *9th International Conference on Learning Representations*, 2021.

[33] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

[34] David L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[35] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth

16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations*, 2021.

[36] Jarek Duda, Khalid Tahboub, Neeraj J. Gadgil, and Edward J. Delp. The use of asymmetric numeral systems as an accurate replacement for huffman coding. In *Picture Coding Symposium*, 2015.

[37] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Proceedings of the 29th Conference on Learning Theory*, 2016.

[38] Erich Elsen, Marat Dukhan, Trevor Gale, and Karen Simonyan. Fast sparse convnets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[39] Kjersti Engan, Sven O. Aase, and John H. Husøy. Method of optimal directions for frame design. *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing 5*, 1999.

[40] Utku Evci, Trevor Gale, Jacob Menick, Pablo S. Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[41] Stanley Finger. *Origins of Neuroscience: A History of Explorations Into Brain Function*. Oxford University Press, 1994.

[42] Vincent François-Lavet, Guillaume Rabusseau, Joelle Pineau, Damien Ernst, and Raphael Fonteneau. On overfitting and asymptotic bias in batch reinforcement learning with partial observability. *Journal of Artificial Intelligence Research*, 65: 1–30, 2019.

[43] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *6th International Conference on Learning Representations*, 2018.

[44] Jonathan Frankle, Gintare K. Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[45] Jonathan Frankle, Gintare K. Dziugaite, Daniel Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *9th International Conference on Learning Representations*, 2021.

[46] Jonathan Frankle, David J. Schwab, and Ari S. Morcos. Training batchnorm and only batchnorm: On the expressive power of random features in CNNs. In *9th International Conference on Learning Representations*, 2021.

[47] Ragnar Frisch. Statistical confluence analysis by means of complete regression systems. *The Economic Journal*, 45(180):741–742, 1935.

[48] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[49] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. In *36th International Conference on Machine Learning Joint Workshop on On-Device Machine Learning & Compact Deep Neural Network Representations*, 2019.

[50] Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse gpu kernels for deep learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020.

[51] Rong Ge, Sham M. Kakade, Rahul Kidambi, and Praneeth Netrapalli. The step decay schedule: A near optimal, geometrically decaying learning rate procedure for least squares. In *Advances in Neural Information Processing Systems 32*, 2019.

[52] Thomas Gebhart, Udit Saxena, and Paul Schrater. A unified paths perspective for pruning at initialization. *CoRR*, abs/2101.10552, 2021. URL `https://arxiv.org/abs/2101.10552`. Last accessed: 04/19/2022.

[53] Stuart Geman, Elie Bienenstock, and Rene Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[54] Raja Giryes, Guillermo Sapiro, and Alexander M. Bronstein. Deep neural networks with random gaussian weights: A universal classification strategy? *IEEE Transactions on Signal Processing*, 64(13):3444–3457, 2016.

[55] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 2010.

[56] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. ISBN 978-0-262-03561-3.

[57] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances in Neural Information Processing Systems 29*, 2016.

[58] John L. Gustafson. Moore's Law. In *Encyclopedia of Parallel Computing*, pages 1177–1184. Springer US, 2011. ISBN 978-0-387-09766-4.

[59] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems 28*, 2015.

[60] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. Eie: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

[61] Stephen J. Hanson and Lorien Y. Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in Neural Information Processing Systems 1*, 1989.

[62] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 4*, 1992.

[63] Soufiane Hayou, Jean-Francois Ton, Arnaud Doucet, and Yee W. Teh. Robust pruning at initialization. In *9th International Conference on Learning Representations*, 2021.

[64] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[66] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Proceedings of the European Conference on Computer Vision*, 2016.

[67] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL `http://arxiv.org/abs/1207.0580`. Last accessed: 03/04/2022.

[68] Elad Hoffer, Itay Hubara, and Daniel Soudry. Fix your classifier: the marginal value of training the last weight layer. In *6th International Conference on Learning Representations*, 2018.

[69] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[70] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[71] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *IEEE International Joint Conference on Neural Networks 2*, 2004.

[72] Guang-Bin Huang, Dian H. Wang, and Yuan Lan. Extreme learning machines: a survey. *International journal of machine learning and cybernetics*, 2(2):107–122, 2011.

[73] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European Conference on Computer Vision*, 2018.

[74] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[75] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[76] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In *Advances in Neural Information Processing Systems 28*, 2015.

[77] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer New York, 2014. ISBN 978-1-0716-1418-1.

[78] Steven A. Janowsky. Pruning versus clipping in neural networks. *Physical Review A*, 39(12):6600–6603, 1989.

[79] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. A faster algorithm for solving general lps. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021.

[80] Donggyu Joo, Eojindl Yi, Sunghyun Baek, and Junmo Kim. Linearly replaceable filters for deep network channel pruning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[81] Sangwon Jung, Hongjoon Ahn, Sungmin Cha, and Taesup Moon. Continual learning with node-importance based adaptive group sparse regularization. In *Advances in Neural Information Processing Systems 33*, 2020.

[82] Ehud D. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks*, 1(2):239–242, 1990.

[83] Jack Kiefer and Jacob Wolfowitz. Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics*, 23(3):462–466, 1952.

[84] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015.

[85] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 2012. URL: `http://www.cs.toronto.edu/~kriz/cifar.html`. Last accessed: 05/13/2022.

[86] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *Advances in Neural Information Processing Systems 4*, 1992.

[87] Solomon Kullback and Richard A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951.

[88] Serge Lang. *Fundamentals of Differential Geometry*. Springer New York, 1999. ISBN 978-0-387-98593-0.

[89] Ya Le and Xuan S. Yang. Tiny imagenet visual recognition challenge, 2015. `https://www.kaggle.com/competitions/tiny-imagenet/overview`. Last accessed: 09/13/2022.

[90] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan V. Oseledets, and Victor S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *3rd International Conference on Learning Representations*, 2015.

[91] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems 2*, 1990.

[92] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

[93] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–48. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-35289-8.

[94] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *7th International Conference on Learning Representations*, 2019.

[95] Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip H. S. Torr. A signal propagation perspective for pruning neural networks at initialization. In *8th International Conference on Learning Representations*, 2020.

[96] Moshe Leshno, Vladimir Y. Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993.

[97] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. In *6th International Conference on Learning Representations*, 2018.

[98] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans P. Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations*, 2017.

[99] Rundong Li, Yan Wang, Feng Liang, Hongwei Qin, Junjie Yan, and Rui Fan. Fully quantized network for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[100] Yawei Li, Shuhang Gu, Luc Van Gool, and Radu Timofte. Learning filter basis for convolutional neural network compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019.

[101] Zhengang Li, Geng Yuan, Wei Niu, Pu Zhao, Yanyu Li, Yuxuan Cai, Xuan Shen, Zheng Zhan, Zhenglun Kong, Qing Jin, Zhiyu Chen, Sijia Liu, Kaiyuan Yang, Bin Ren, Yanzhi Wang, and Xue Lin. Npas: A compiler-aware framework of unified network pruning and architecture search for beyond real-time mobile acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[102] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[103] Junjie Liu, Zhe Xu, Runbin Shi, Ray C. C. Cheung, and Hayden K. H. So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In *8th International Conference on Learning Representations*, 2020.

[104] Shiwei Liu, Lu Yin, Decebal C. Mocanu, and Mykola Pechenizkiy. Do we actually need dense over-parameterization? In-time over-parameterization in sparse training. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[105] Xingyu Liu, Jeff Pool, Song Han, and William J. Dally. Efficient sparse-winograd convolutional neural networks. In *6th International Conference on Learning Representations*, 2018.

[106] Zhenhua Liu, Jizheng Xu, Xiulian Peng, and Ruiqin Xiong. Frequency-domain dynamic pruning for convolutional neural networks. In *Advances in Neural Information Processing Systems 31*, 2018.

[107] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *7th International Conference on Learning Representations*, 2019.

[108] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[109] Po-Ling Loh and Martin J. Wainwright. Regularized m-estimators with nonconvexity: Statistical and algorithmic theory for local optima. In *Advances in Neural Information Processing Systems 26*, 2013.

[110] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l0 regularization. In *6th International Conference on Learning Representations*, 2018.

[111] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems 30*, 2017.

[112] Julia Lust and Alexandru P. Condurache. Gran: An efficient gradient-norm based detector for adversarial and misclassified examples. In *28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2020.

[113] Julia Lust and Alexandru P. Condurache. Efficient detection of adversarial, out-of-distribution and other misclassified samples. *Neurocomputing*, 470:335–343, 2022.

[114] Siwei Ma, Xinfeng Zhang, Chuanmin Jia, Zhenghui Zhao, Shiqi Wang, and Shanshe Wang. Image and video compression with neural networks: A review. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(6):1683–1698, 2020.

[115] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European Conference on Computer Vision*, 2018.

[116] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11(2):19–60, 2010.

[117] Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[118] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN 978-0-521-86571-5.

[119] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[120] Shervin Minaee. 20 popular machine learning metrics. part 1: Classification & regression evaluation metrics. Towards Data Science, 2019. URL `https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce`. Last accessed: 09/13/2022.

[121] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., 1967.

[122] Decebal C. Mocanu, Elena Mocanu, Peter Stone, Phuong Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9(1):2383, 2018.

[123] Hosein Mohimani, Massoud Babaie-Zadeh, and Christian Jutten. A fast approach for overcomplete sparse decomposition based on smoothed ℓ^0 norm. *IEEE Transactions on Signal Processing*, 57(1):289–301, 2009.

[124] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *Advances in Neural Information Processing Systems 32*, 2019.

[125] Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[126] Michael C. Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in Neural Information Processing Systems 1*, 1989.

[127] Noboru Murata. A statistical study on on-line learning. In *On-Line Learning in Neural Networks*, pages 63–92. Cambridge University Press, 1998. ISBN 978-0-511-56992-0.

[128] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[129] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent. OpenAI Blog, 2019. URL `https://openai.com/blog/deep-double-descent/`. Last accessed: 09/12/2022.

[130] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *8th International Conference on Learning Representations*, 2020.

[131] Behnam Neyshabur, Ruslan Salakhutdinov, and Nathan Srebro. Path-sgd: Path-normalized optimization in deep neural networks. In *Advances in Neural Information Processing Systems 28*, 2015.

[132] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Proceedings of the 28th Conference on Learning Theory*, 2015.

[133] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P. Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems 28*, 2015.

[134] Steven J. Nowlan and Geoffrey E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992.

[135] Laurent Orseau, Marcus Hutter, and Omar Rivasplata. Logarithmic pruning is all you need. In *Advances in Neural Information Processing Systems 33*, 2020.

[136] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W. Keckler, and William J. Dally. Scnn. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017.

[137] Daniel S. Park, Yu Zhang, Chung-Cheng Chiu, Youzheng Chen, Bo Li, William Chan, Quoc V. Le, and Yonghui Wu. Specaugment on large scale datasets. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2020.

[138] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, 2019.

[139] Shreyas M. Patil and Constantine Dovrolis. PHEW: Constructing sparse networks that learn fast and generalize well without training data. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[140] Ankit Pensia, Shashank Rajput, Alliot Nagle, Harit Vishwakarma, and Dimitris Papailiopoulos. Optimal lottery tickets via subset sum: Logarithmic overparameterization is sufficient. In *Advances in Neural Information Processing Systems 33*, 2020.

[141] Hieu Pham, Zihang Dai, Qizhe Xie, Minh-Thang Luong, and Quoc V. Le. Meta pseudo labels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2021.

[142] Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999.

[143] Boris T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5): 1–17, 1964.

[144] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in Neural Information Processing Systems 29*, 2016.

[145] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

[146] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary neural networks: A survey. *Pattern Recognition*, 105:107281, 2020.

[147] Yuanyuan Qing, Yijie Zeng, Yue Li, and Guang-Bin Huang. Deep and wide feature based extreme learning machine for image classification. *Neurocomputing*, 412:426–436, 2020.

[148] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[149] Vinay V. Ramasesh, Ethan Dyer, and Maithra Raghu. Anatomy of catastrophic forgetting: Hidden representations and task semantics. In *9th International Conference on Learning Representations*, 2021.

[150] Matthias Rath and Alexandru P. Condurache. Invariant integration in deep convolutional feature space. In *28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2020.

[151] Matthias Rath and Alexandru P. Condurache. Improving the sample-complexity of deep classification networks with invariant integration. In *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 2022.

[152] Jie Ren, Peter J. Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. Likelihood ratios for out-of-distribution detection. In *Advances in Neural Information Processing Systems 32*, 2019.

[153] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *8th International Conference on Learning Representations*, 2020.

[154] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[155] Ralph T. Rockafellar and Roger J. B. Wets. *Variational Analysis*. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-62772-2.

[156] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[157] Amir Rosenfeld and John K. Tsotsos. Intriguing properties of randomly weighted networks: Generalizing while learning next to nothing. In *Conference on Computer and Robot Vision*, 2019.

[158] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, pages 318–362. MIT Press, 1987. ISBN 978-0-262-18120-4.

[159] Tara N. Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.

[160] Victor Sanh, Thomas Wolf, and Alexander M. Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems 33*, 2020.

[161] Andrew Saxe, Pang W. Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Ng. On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[162] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *2nd International Conference on Learning Representations*, 2014.

[163] Samuel S. Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. In *5th International Conference on Learning Representations*, 2017.

[164] Jonathan Schwarz, Siddhant Jayakumar, Razvan Pascanu, Peter E. Latham, and Yee W. Teh. Powerpropagation: A sparsity inducing weight reparameterisation. In *Advances in Neural Information Processing Systems 34*, 2021.

[165] Joan Serrà, David lvarez, Vicen Gmez, Olga Slizovskaia, Jos F. Nez, and Jordi Luque. Input complexity and out-of-distribution detection with likelihood-based generative models. In *8th International Conference on Learning Representations*, 2020.

[166] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

[167] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[168] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations*, 2015.

[169] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Conference of the Association for Computational Linguistics*, 2019.

[170] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[171] Yi-Lin Sung, Varun Nair, and Colin Raffel. Training neural networks with fixed sparse masks. In *Advances in Neural Information Processing Systems 34*, 2021.

[172] Ilya Sutskever, James Martens, George Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

[173] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[174] Hidenori Tanaka, Daniel Kunin, Daniel L. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Advances in Neural Information Processing Systems 33*, 2020.

[175] Yehui Tang, Yunhe Wang, Yixing Xu, Yiping Deng, Chao Xu, Dacheng Tao, and Chang Xu. Manifold regularized dynamic network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[176] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[177] William F. Tinney and John W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proceedings of the IEEE*, 55(11):1801–1809, 1967.

[178] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. In *5th International Conference on Learning Representations*, 2017.

[179] Vladimir N. Vapnik. *The nature of statistical learning theory*. Springer New York, 1995. ISBN 978-0-387-94559-0.

[180] Vladimir N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer New York, 2006. ISBN 978-0-387-34239-9.

[181] Stijn Verdenius, Maarten Stol, and Patrick Forré. Pruning via iterative ranking of sensitivity statistics. *CoRR*, abs/2006.00896v2, 2020. URL `https://arxiv.org/abs/2006.00896v2`. Last accessed: 04/19/2022.

[182] Matthew Wallingford, Hao Li, Alessandro Achille, Avinash Ravichandran, Charless Fowlkes, Rahul Bhotika, and Stefano Soatto. Task adaptive parameter sharing for multi-task learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[183] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

[184] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *8th International Conference on Learning Representations*, 2020.

[185] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan M. Liao. Scaled-yolov4: Scaling cross stage partial network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[186] Yulong Wang, Xiaolu Zhang, Xiaolin Hu, Bo Zhang, and Hang Su. Dynamic network pruning with interpretable layerwise channel selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[187] Zi Wang, Chengcheng Li, and Xiangyang Wang. Convolutional neural network pruning with structural redundancy reduction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[188] Ziheng Wang. Sparsert: Accelerating unstructured sparsity on gpus for deep learning inference. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 2020.

[189] Peter Whittle. Tests of fit in time series. *Biometrika*, 39(3/4):309–318, 1952.

[190] N. Wiener and Business Library Johnson Foundation Collection. *Cybernetics Or Control and Communication in the Animal and the Machine*. MIT Press, 1948.

[191] Simon Wiesler and Hermann Ney. A convergence analysis of log-linear training. In *Advances in Neural Information Processing Systems 24*, 2011.

[192] Paul Wimmer, Jens Mehnert, and Alexandru P. Condurache. FreezeNet: Full performance by reduced storage costs. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[193] Paul Wimmer, Jens Mehnert, and Alexandru P. Condurache. COPS: Controlled pruning before training starts. In *International Joint Conference on Neural Networks*, 2021.

[194] Paul Wimmer, Jens Mehnert, and Alexandru P. Condurache. Interspace pruning: Using adaptive filter representations to improve training of sparse cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[195] Paul Wimmer, Jens Mehnert, and Alexandru P. Condurache. Dimensionality reduced training by pruning and freezing parts of a deep neural network, a survey. *Artificial Intelligence Review*, 2023. URL `https://doi.org/10.1007/s10462-023-10489-1`.

[196] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[197] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10, 000-layer vanilla convolutional neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[198] Jian Xue, Jinyu Li, and Yifan Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech*, 2013.

[199] Huanrui Yang, Wei Wen, and Hai Li. DeepHoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. In *8th International Conference on Learning Representations*, 2020.

[200] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32*, 2019.

[201] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

[202] Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Richard G. Baraniuk, Zhangyang Wang, and Yingyan Lin. Drawing early-bird tickets: Toward more efficient training of deep networks. In *8th International Conference on Learning Representations*, 2020.

[203] Jihun Yun, Peng Zheng, Eunho Yang, Aurelie Lozano, and Aleksandr Aravkin. Trimming the ℓ_1 regularizer: Statistical analysis, optimization, and applications to deep learning. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[204] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference*, 2016.

[205] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations*, 2017.

[206] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European Conference on Computer Vision*, 2018.

[207] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision*, 2018.

[208] Zhenyu Zhang, Xuxi Chen, Tianlong Chen, and Zhangyang Wang. Efficient lottery ticket finding: Less data is more. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[209] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. In *5th International Conference on Learning Representations*, 2017.

[210] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems 32*, 2019.

[211] Tao Zhuang, Zhixuan Zhang, Yuheng Huang, Xiaoyi Zeng, Kai Shuang, and Xiang Li. Neuron-level structured pruning using polarization regularizer. In *Advances in Neural Information Processing Systems 33*, 2020.

[212] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jin-Hui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems 31*, 2018.