Zurich University of Applied Sciences

zh aw School of Engineering CAI Centre for Artificial Intelligence

ZURICH UNIVERSITY OF APPLIED SCIENCES

BACHELOR THESIS

# Hitting the Jackpot: Optimizing Neural Networks with Composite Pruning Strategies

*Authors:*
Urban LUTZ - IT18tb_WIN
Alexandre MANAI - IT19b_WIN

*Supervisors:*
Prof. Dr. Thilo STADELMANN
Dr. Claus HORN

*A Bachelorarbeit submitted in fulfillment of the requirements for the degree of Bachelor of Computer Science*

*at the*

Zurich University of Applied Sciences
School of Engineering
Center for Artificial Intelligence

June 10, 2022

# Declaration of Originality

Bachelor's Thesis at the School of Engineering

By submitting this Bachelor's thesis, the undersigned student confirms that this thesis is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the Bachelor thesis have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

Alexandre Manai

Winterthur, June 10, 2022: [illegible]

Urban Lutz

Weinfelden, June 10, 2022: [illegible]

# *Abstract*

**Hitting the Jackpot: Optimizing Neural Networks with Composite Pruning Strategies**

by Urban LUTZ & Alexandre MANAI

Over the past decade, state-of-the-art Deep Learning models have surpassed important benchmarks across several domains, not least due to the growing numbers of parameters trained. Training these large models requires large amounts of compute and memory, sparking increased interest in the scientific community in the field of neural network optimization, aiming to increase the efficiency of such models. Recent work suggests, that within each large model lies smaller, itself trainable subnetwork, referred to as the "Winning Lottery Ticket" (LT), which achieves similar performance as the full network. The algorithm to find these subnetworks however involves training a network to convergence iteratively many times over, thus further increasing the resource requirements. A number of algorithms have been described since, which do not rely on repeated training, but fail to produce Lottery Tickets with a performance as high as those found during training.
In this thesis, we ask the question of whether multiple state-of-the-art algorithms can be combined to produce better performing Lottery Tickets before training.
We propose two novel methods to combine pruning algorithms: Stacked Scoring, where we apply multiple pruning algorithms in sequence, and SaW, Scores as Weight initializations, where we train the model using the pruning score of a model as weights to train on and another algorithm to prune it.
We show that by combining Stacked Scoring with SaW, we are able to outperform to outperform the best baseline by 2.03% on our model and task at high sparsities, and reach a performance previously reserved for pruning schemes applied during training.
Thereby, our work demonstrates that the limit of finding Lottery Tickets before training has not yet been reached by the existing algorithms, encouraging future research to validate our results and further investigate our proposed methods.

# *Zusammenfassung*

**Hitting the Jackpot: Optimizing Neural Networks with Composite Pruning Strategies**

von Urban LUTZ & Alexandre MANAI

In den letzten zehn Jahren haben moderne Deep-Learning-Modelle wichtige Benchmarks in verschiedenen Bereichen übertroffen, nicht zuletzt aufgrund der wachsenden Zahl der trainierten Parameter. Das Training dieser grossen Modelle erfordert grosse Mengen an Rechenleistung und Speicherplatz, was ein gesteigertes Interesse an der Forschung zur Optimierung neuronaler Netze weckte, um die Effizienz dieser Modelle zu erhöhen. Neue Erkenntnisse deuten darauf hin, dass innerhalb jedes großen Modells kleinere, selbst trainierbare Teilnetze liegen, die eine ähnliche Leistung wie das vollständige Netz erreichen. Der Algorithmus zum Auffinden dieser Teilnetze, die die Autoren als "Lottery Tickets" ("Lotterielose") bezeichnen, erfordert jedoch, dass ein Model iterativ viele Male bis zur Konvergenz trainiert wird. Seither wurde eine Reihe von Algorithmen beschrieben, die nicht auf wiederholtes Training angewiesen sind, aber keine Lotterielose mit einer ebenso hohen Leistung wie beim Training erzeugen.
In dieser Arbeit stellen wir uns die Frage, ob mehrere dieser Algorithmen kombiniert werden können, um leistungsfähigere Lotterielose vor dem Training zu finden.
Wir schlagen zwei neue Methoden zur Kombination von Pruning-Algorithmen vor: Stacked Scoring, bei dem wir mehrere Pruning-Algorithmen nacheinander anwenden, und SaW, Scores as Weight initializations, bei dem wir das Modell trainieren, indem wir den Pruning-Score eines Modells als Gewichte zum Trainieren und einen anderen Algorithmus zum Pruning verwenden.
Wir zeigen, dass wir durch die Kombination von Stacked Scoring mit SaW in der Lage sind, die beste Baseline bei hoher Sparsity um 2,03% zu übertreffen und eine Leistung zu erreichen, die bisher iterativen Pruning-Schemata vorbehalten war.
Damit zeigt unsere Arbeit, dass die Performancelimits des Auffindens von Lottery Tickets vor dem Training von den bestehenden Algorithmen noch nicht erreicht wurde. Dies ermutigt zukünftige Forschung, unsere Ergebnisse zu validieren und unsere vorgeschlagenen Methoden weiter zu untersuchen.

# *Acknowledgements*

We thank our supervisors Prof. Dr. Thilo Stadelmann and Dr. Claus Horn for their support and their guidance on this thesis. We learned a lot about the thoroughness of scientific procedures thanks to their advice.

Further, we want express our gratitude for the computing hardware generously provided by Paul Kort. By foregoing any profits made on cryptocurrency mining and donating a nice GPU to science instead, he made a large impact on our ambitious project.

Thank you!

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

Over the last decade, the field of Deep Learning (Schmidhuber, 2015) has achieved significant breakthroughs in fields such as computer vision (Dosovitskiy et al., 2020), game play (Silver et al., 2017) or natural language processing (Brown et al., 2020). New technologies like Transformers (Vaswani et al., 2017) have been able to raise the bar on a number of benchmarks (Lin et al., 2021) across different tasks (Devlin et al., 2018; Ruan and Jin, 2022), not least due to their ability to scale massively (Jumper et al., 2021). For example, GPT-3, a state-of-the-art language model, contains 175 billion trainable parameters, leading to enormous compute consumption (Brown et al., 2020). According to OpenAI, 2018, the parameter count of the largest models trained has grown exponentially with a 3.4 Month doubling period since 2012.

But, in the context of Deep Learning, not all parameters have a significant impact on the performance of the neural network (Allen-Zhu et al., 2018). Once the network is trained, it can be shown that a large portion of weights can be removed without significantly affecting the performance of the model (LeCun, Denker, et al., 1989), but at the same time, empirical evidence suggests that it is beneficial for any network to have more parameters available at training time (Li et al., 2020). For instance, Brutzkus et al., 2017 prove that over-parameterized shallow neural networks get trained optimally and reach good generalization, meaning how well the network can classify or forecast unseen data, with Stochastic gradient descent (SGD).

This trade-off between needing superfluous weights for training and the increased compute requirements coupled with the rapid growth of models over the last decade has led to increased scientific interest in neural network sparsification (Hoefler et al., 2021).

Sparse neural networks, where parameters have been removed, not only require fewer floating point operations (Hoefler et al., 2021), but can also be represented efficiently in memory, improving hardware interactions and Cache-/Data movements (Elsen et al., 2019; J. Yu et al., 2017;Han et al., 2017). The interest in neural network optimization has led to numerous advancements in this area, which traditionally is part of the field of scientific computing rather than artificial intelligence (Hoefler et al., 2021).

These advancements have a significant impact on multiple aspects of society. Being able to have state-of-the-art neural networks that are cheaper to run and deploy will give the opportunity for each industry to develop Deep Learning based solutions, thus democratizing it. Furthermore, today, the emerging question of data privacy is central (Gruschka et al., 2018), running compressed models, for instance MobileNets (Howard et al., 2017) directly inside the users' phones enables the user to keep its personal data private.
Strubell et al., 2019 report that training a common Natural Language Processing (NLP) model on GPU with fine-tuning and experimentation steps leads to an estimated $CO_2$ consumption

of over 35 tons compared to average $CO_2$ consumption of a Human in a year which is around 5 tons. Further, reduced power consumption leads to less heat dissipation as well, as shown by Zhu et al., 2017 who propose a SparseNN that achieves a 10% - 70% compute improvement while reducing a power by 50%. Thus, optimizing neural networks has a direct impact on the environment.
Finally, optimizing neural networks reduces the complexity thereof and helps reach a goal of Explainable AI making solutions understandable and traceable for humans (Linardatos et al., 2020).

## 1.2 Background

A key development in the field of neural network optimization was the proposal of an algorithm capable of producing highly sparse yet trainable subnetworks(Frankle and Carbin, 2018), seemingly overcoming the necessity of overparameterization during training (Uber et al., 2020), leading the authors to hypothesize that such subnetworks, exist for any fully connected network. These subnetworks, in the terms of the metaphor proposed by the authors, have "won" the "initialisation lottery", and are commonly referred to as "winning lottery tickets". Keeping with this metaphor, the hypothesis about the existence of these Lottery Tickets (LTs) is referred to as the "Lottery Ticket Hypothesis" (LTH). Frankle and Carbin, 2018 empirically prove the existence of sparse subnetworks that can be trained from initialisation through the proposed Iterative Magnitude Pruning (IMP) algorithm, requiring to train the model to convergence many times over. Finding these winning tickets is thus impractical for large datasets and models, for example Imagenet (Russakovsky et al., 2014) with higher training times.

Following this publication, the Lottery Ticket Hypothesis has been the subject of many follow-up publications from large research organisations in the field ranging from company-sponsored efforts at Meta AI (formerly Facebook AI) (Paganini and Forde, 2020) or Uber AI (Uber et al., 2020) to universities like Stanford University (Tanaka et al., 2020) and ETH Zürich/IST Austria (Hoefler et al., 2021).
From this research, several algorithms emerged, promising to produce trainable subnetworks at initialisation without the need for repeated training (Lee et al., 2018; Wang et al., 2020), and, in some cases, without looking at any data at all (Tanaka et al., 2020). Examples of recently proposed algorithms include: Single-Shot Network Pruning (SNIP) (Lee et al., 2018), Gradient Signal Preservation (GraSP) (Wang et al., 2020) and Synaptic Flow Pruning (SynFlow) (Tan and Le, 2019), among others (Dettmers and Zettlemoyer, 2019).

To this end, different approaches can be applied.
SNIP uses a specific saliency criterion (Mozer and Smolensky, 1989; LeCun, Denker, et al., 1989; Han, Pool, et al., 2015) which describes connections in the network that are important to the given task based on the data before training. It is able to prune an non-trained initialized Network in a single iteration to a low ratio of weights remaining while keeping compelling testing results. Secondly, the GraSP approach (Wang et al., 2020) follows the intuition that an effective Neural Network preserves a certain Gradient Flow (Hochreiter et al., 2005), optimizing to keep the same gradient norm before and after pruning. This method demonstrates results that are competitive with pruning algorithms which prune after training, making it interesting to study more in depth. Finally, SynFlow is designed to mitigate the issue called layer collapse, which describes a pruning algorithm deleting all the parameters of one layer, disconnecting the model and rendering it untrainable.
While all of the above algorithms are able to generate sparse trainable subnetworks, therefore Lottery Tickets, and do not require repeated training, the resulting networks fail to perform as well as those found by IMP at high sparsities (Tanaka et al., 2020).

## 1.3 Problem Definition

In this thesis, we investigate ways to combine several different state-of-the-art pruning algorithms and empirically benchmark the resulting composite methods on a basic network designed to perform a standardized task. The goal is to find a combination of pruning algorithms, which can find sparse sub-networks before training that perform better than the current state-of-the-art. By implementing several distinct ways to combine ideas from different publications, we aim to answer the question: Can state-of-the-art pruning algorithms be combined to obtain sparse sub-networks before training exhibiting a higher performance compared to a sub network found by any algorithm by itself?

For reference, the full task proposition can be found in Appendix E.

## 1.4 Contributions and Limitations

In this thesis, we contribute a comparative study on four state-of-the-art pruning algorithms on a single model and task, confirming findings from various sources regarding subnetwork performance and weight selection overlap. Further, we conduct an ablation study on the impact of different pruning schedules to the resulting sub network, again confirming findings made previously and additionally showing, that not every algorithm benefits equally from a computationally more expensive pruning schedule. We demonstrate that the sub networks found by the original algorithm proposed in the Lottery Ticket Hypothesis can be outperformed using state-of-the-art scoring criteria instead of a magnitude based criterion.
Based on the four selected algorithms, we construct a meta learning model (Maclin and Opitz, 1999) and show that it is possible to learn to score weights for pruning, given the scores of the four state-of-the-art algorithms.
With Stacked Pruning, we propose a novel way of combining two arbitrary pruning criteria and demonstrate its ability to produce Lottery Tickets before training.
Further, we introduce the novel idea of using scores obtained by pruning algorithms as weight initialisations and show that initializing the network this way can have a positive impact on the performance of the subnetworks obtained from it.
Lastly, we show that the combination of both Stacked Scoring and using scores as weight initialisations can lead to highly performant Lottery Tickets, outperforming the best algorithm available by 2.03% on our model and task at a non-trivial sparsity of 0.5% weights remaining and reaching levels of performance previously only achieved by algorithms pruning during training.

In the context of this thesis, we are restricted by several limitations. Having a fixed formal time schedule limited to 16 weeks and encountering resource limitations on the compute cluster, we decided to focus on one specific model, LeNet (LeCun, Boser, et al., 1989) with a single dataset, MNIST (LeCun, Cortes, et al., 1999) and evaluate the trained models with three repeated runs whilst keeping the model Hyperparameters stable across experiments.
With these decisions, the results of our work remain comparable to current research without the need for large scale computing.

## 1.5 Outline

The following chapters in this thesis give an answer to our research question. To start, we introduce in "Foundations and Related Work" the most important concepts in the field of neural network pruning and give an overview on the state-of-the-art pruning methods available today. In the chapter "Methods", we propose three approaches to combine different algorithms:

A meta learning model, a sequential method we term Stacked Scoring and a novel approach of using pruning scores as weights. In "Experimental Setup", we describe the environment and conditions in which we implement the three approaches described. In "Results", we give an overview on the performance of each of our approaches. Lastly, we summarize our findings in "Conclusions", contextualize the results and give an outlook on where we see potential future research.

# Chapter 2

# Foundations and Related Work

The field of Compression and Optimization dates back to 1989 (LeCun, Denker, et al., 1989) at the start of the second AI Winter and gradually picked it up in popularity due to the inflation in size of Neural Networks (Neyshabur et al., 2019). Thus, different approaches aiming at reducing the compute and memory overhead generated by these Neural Networks have been devised.
It is essential to place our thesis in this wide landscape and present, in this chapter, topics and techniques particularly related to our thesis.

For a complete and in depth overview of the field of Compression and Optimization of Neural Networks, (Hoefler et al., 2021) provides a comprehensive overview on the foundations as well as the state-of-the-art.

## 2.1 Neural Network Compression and Optimization

Deep learning proved itself to be a central Machine Learning technique that yields unparalleled results in various domains, as outlined in section 1.1. Although they are resource inefficient and bigger networks expensive to train and memory intensive, it is common practice to build over-parameterized models. Similar to Data compression where information is encoded in a smaller representation (e.g. using fewer bits), the field of Deep Learning applies the same idea to Neural Networks.
The Idea of compressing a neural network for optimization essentially aims to find the most efficient representation of the network in terms compute and memory but also in performance (e.g. model accuracy). This can be implemented in very different ways, but it can generally be categorized in a Constructive and Destructive approach. (Paganini and Forde, 2020)

### 2.1.1 Approaches to Compression and Optimization

The field of Compression and Optimization incorporates many different techniques optimizing all aspects of the model, from the architecture and design itself (Elsken et al., 2018; X. He et al., 2019) down to the bit representation of the weights in memory (Nagel et al., 2021). In this chapter, we will focus on a small subset of these techniques which depict well the wide range of possibilities to compress and optimize Neural Networks.

#### Constructive vs Destructive Compression

On one hand, constructive methods (Paganini and Forde, 2020) deal with building of models from the ground-up towards a network architecture which is optimized for a specific task and can be efficiently designed from the beginning. On the other hand, destructive approaches (Paganini and Forde, 2020) work down from a full Network to a compressed Network by removing structures from the network. Human Hand-designed and Automated Neural Network Design techniques are typical examples of constructive approaches, while Quantization and Pruning are examples of destructive approaches.

### Manual and Automated Neural Network Design

As explained in 2.1.1, constructive approaches build efficient model architectures completely which can be achieved through Manual Design by selecting criteria known to be efficient for a certain class of tasks. For example by following architectural design strategies, Iandola et al., 2016 proposers hand-designed "SqueezeNets" which achieve AlexNet-level (Krizhevsky et al., 2012) accuracy on ImageNet with 50x fewer parameters. Exhaustively choosing each parameter and transformation is, nevertheless, time-consuming. Therefore, automated approaches are developed to substantially speed up the development of deep learning models. (Santos et al., 2019; X. He et al., 2019)
The broad domain of AutoML (X. He et al., 2019) encompasses ideas such Feature Engineering, Hyperparameter tuning and others to elimiate the need for manual design. From these, automating the design of network architectures with Neural Architecture Search (NAS) (Elsken et al., 2018), has the highest impact on the efficiency of the model itself (Hoefler et al., 2021).

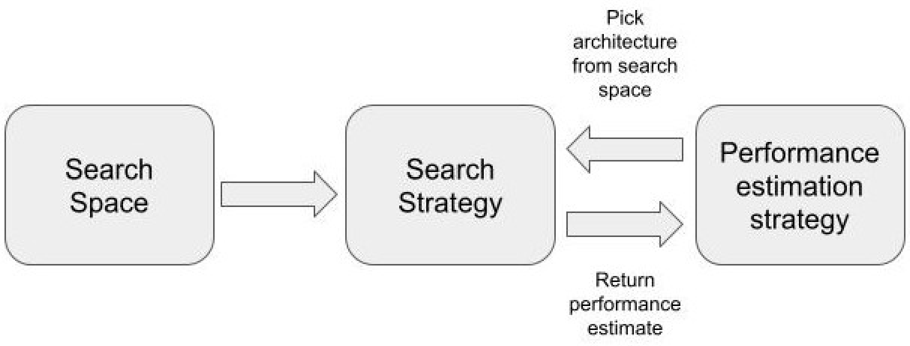

Pick architecture from search space
Search Space
Search Strategy
Performance estimation strategy
Return performance estimate

FIGURE 2.1: Neural Architecture Search. Depiction of a NAS iteration where a candidate architecture gets selected from a Search Space thanks to a Search Strategy then tested the against a Performance estimation strategy which returns a performance estimate.

Inside a predefined Search Space of Neural Network Architectures (see Figure 2.1), NAS applies a Search Strategy to select architectures and testing them against a performance estimation strategy.
NAS designs networks which are on par or outperform hand-designed ones. (Zoph and Le, 2016) For instance, it is used in conjunction with advanced transformation techniques like model scaling to create, for example, EfficentNets (Tan and Le, 2019). EfficentNets achieve state-of-the-art accuracy on ImageNet while being 8.4x smaller and 6.1x faster (Tan and Le, 2019).
Through its iterative procedure, NAS is, thus, able to construct Neural Networks capable of improving memory consumption as well as inference time.

### Distillation

The Distillation process consists of transferring the knowledge contained in a large, pre-trained model to a smaller network. One such example is a Teacher/Student setup, where during the training of the Student Network, the Teacher Network makes predictions from the training

data. The predictions generated by the Student and Teacher models are then averaged together leading a combined loss. Through this process, the Student Network learns from the pre-trained knowledge of the Teacher model.
By transferring the knowledge from the large network to a small network, Hinton et al., 2015 obtain a model demonstrating a 4.4% increase in accuracy compared to the same size model trained without a teacher. Therefore, Distillation can be understood as a form of constructive neural network optimization, as it obtains higher performance with the same number of parameters.

#### Quantization

The efficiency issues which Quantization tries to tackle in a destructive manner are memory requirement and inference time. If one wants to integrate their neural network model into their edge devices (e.g. phones ), the high computational cost of neural networks needs to be reduced (Nagel et al., 2021). Quantization is one of the most effective ways of achieving savings for these issues (Nagel et al., 2021). By reducing the precision of the weights, activation's and biases, the network consumes less memory. For example, instead of using a 32-bit float to represent a parameter in the model a 8-bit integer is taken. But one drawback occurring due to Quantization is accuracy degradation (Mishra et al., 2017).

#### Pruning

As a last general strategy, we introduce Pruning which is the topic we will focus on in our thesis. Pruning (also called Sparsification) is the process of deleting unnecessary individual parameters or groups of parameters from a neural network. It aims to reduce memory and compute cost of the network making it more efficient while keeping the models accuracy as high as possible (Hoefler et al., 2021). Sparsification aims to optimize two functions of Neural Networks: Inference (or forward pass in training) and Training. Both goals of Sparsification entail various techniques and schedules which we will introduce in the following chapters.

## 2.2 Sparsification

In this chapter, we will further elaborate the definition given in 2.1.1 with in depth explanations of concepts and techniques applied in this domain.

#### Classical Methods

Classical methods that shaped the modern idea of pruning introduced the measure of importance of weights or structures by analyzing their impact on the loss function of the network. They observed the change of the loss when a weight or structure is pruned away and giving a higher importance to those which when pruned led to the least change in the loss function.
Several methods utilizing this idea are described in literature:

**Skeletonization.** In Mozer and Smolensky, 1989 each weight $w$ is given an attention strength value $\alpha \in [0,1]$. If $\alpha = 0$, the weight associated to that alpha value becomes irrelevant or unnecessary and $\alpha = 1$ signifies a conventional weight. To determine the importance $\rho$ of the weight $i$, they calculate the derivative of the loss with respect to $\alpha_i$:

$$\hat{\rho}_i = \frac{\partial E}{\alpha_i}\bigg|_{\alpha_i=1} \tag{2.1}$$

*with* $E$ the loss function

The idea of studying the impact of the presence or lack of it of certain weights on the loss function is central to multitude of state-of-the-art pruning algorithms, for example SNIP which we will introduce in section 2.2.4.

**Optimal Brain Damage.** The central idea of LeCun, Denker, et al., 1989 is to estimate the importance of a weight with a Taylor series approximation of the loss function. Multiple assumptions for example that off-diagonal elements of the Hessian matrix $H$, representing a square matrix of second order derivatives of the loss function with respect to the weights, with elements $h_{i,j}$ are equal to 0 lead to the adapted loss function:

$$\partial E = \frac{1}{2}\sum_i h_{ii}\delta w_i^2 \tag{2.2}$$

*with* $E$ the loss function, perturbed weights denoted with $\delta$

After deriving to the 2nd degree the loss function, we get the importance of the weight (also called saliency) $s_i$ with:

$$s_i = \frac{h_{ii}w_i^2}{2} \tag{2.3}$$

The weights with the lowest $s_i$ get pruned.

In this paper, LeCun, Denker, et al., 1989 develop a pruning algorithm on the base of the effect of weight $w_i$ on the approximated loss function. Additionally, a saliency score is given each weight which is then used to prune accordingly. These ideas are central to current pruning algorithms like GraSP, as described in section 2.2.4.

### Sparsity relevance

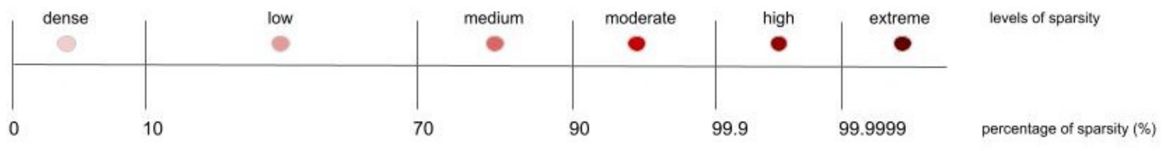


FIGURE 2.2: Visualization of different sparsity levels by name and their corresponding percentage. The upper row of the visualization represents the name of the different sparsity levels and the lower row the according sparsity percentage.

The sparsity percentage of a Neural Network can be split up in different levels (see Figure 2.2). In the range of a dense model up to a low 30% sparsity, pruning procedures aren't worth it, because no gain in speed will be acquired due to overheads in storing sparse structures and controlling sparse computations.(Hoefler et al., 2021) Nevertheless, today's state-of-the-art resides between sparsities of 40% and 99.99% with sparsities between 40% and 95% where models keep an equivalent accuracy to their dense representation and sparsities between 95% and 99.99%, on the other hand, producing models which present some accuracy loss. Above the threshold of 99.99% sparsity is a domain that is tackled by Scientific Computing and isn't reached by Deep learning models. (Lin et al., 2020; Hoefler et al., 2021)

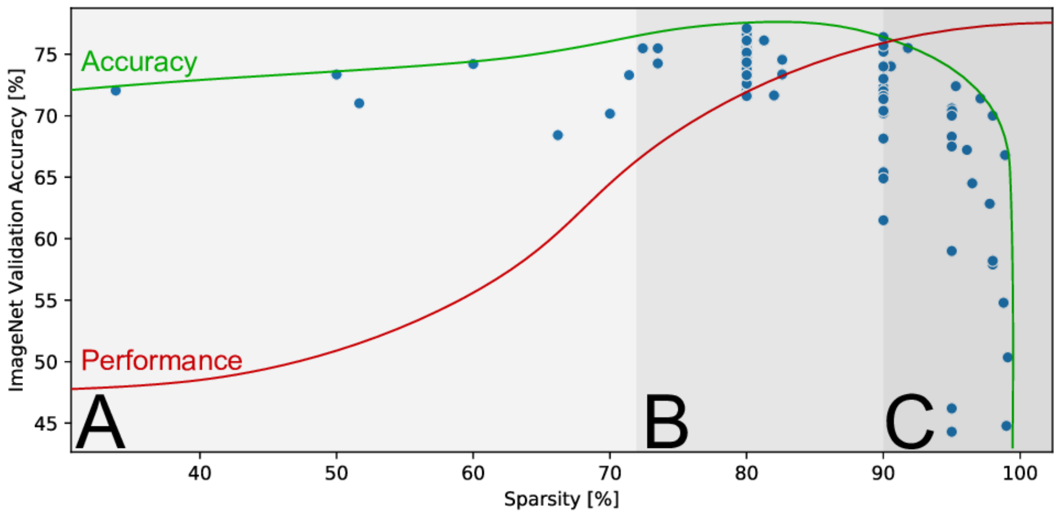


FIGURE 2.3: Typical test error vs. sparsity showing Occam's hill (network: ResNet-50 on Top-1 ImageNet) (source: Hoefler et al., 2021)

To that end, Sparsification follows Occam's hill (see green line in Figure 2.3) (Rasmussen and Ghahramani, 2000). As stated before, between 0% and 80% sparsity, models keep a comparable accuracy to their dense representation. Occam's hill, interestingly, depicts an increase in accuracy in that range which could be explained by the Sparsification "trimming" the model of its obviously unnecessary weights certainly representing learned noise, an effect comparable to dropout (Srivastava et al., 2014a) in the domain of ephemeral pruning as described below. Accuracies stay stable in the range of 80% to 95%, but afterwards the accuracy rapidly worsens.

It is important to keep in mind that memory and computation efficiency depend strongly on the implementation of the sparse data types. Being out of the scope of our thesis, please refer to (Pooch and Nieder, 1973) for more information on that subject.
Additionally, the sparsity level a model needs to achieve to see a signficant efficiency increase depends largely on the neural networks initial architecture. Certain architectures such as MobileNet V1 (Howard et al., 2017) or EfficientNet-B0 (Tan and Le, 2019) are already built in a efficient way pruning from such architecture would automatically mean pruning necessary weights. Meanwhile, architecture such as AlexNet (Krizhevsky et al., 2012) present a low parameter efficiency (Bianco et al., 2017) and would gain from being pruned to a higher sparsity level. (Hoefler et al., 2021) defines the hardness-normalized parameter efficiency metric; a modified parameter efficiency (Bianco et al., 2017) which incorporates the difficulty of the task into the parameter.

### 2.2.1 What to prune?

In this part, we explain which elements of a neural network can be pruned away. There are two overarching principles of pruning: Persistent and Ephemeral Sparsification (Hoefler et al., 2021).

**Structured vs Unstructured Sparsification.** We need to make a distinction between structured and unstructured Sparsification (Neill, 2020; Wang et al., 2019; Zeng and Urtasun, 2018). Unstructured pruning also called fine-grained looks at single weights (see Figure 2.4 on the right) and not structures such as neurons (see Figure 2.4 on the left), groups of weights or filters in convolutional layers which are taken into consideration in unstructured Sparsification.
The motivation of this categorical distinction is the storage impact each has (Neill, 2020). The

flexibility of unstructured pruning comes with a overhead on memory due to the necessary storage of each pruned weights index (Bianco et al., 2017). But the flexibility permits a wider range of pruning actions which enables better results to the expense of slower computations. (Prechelt, 1996)
Meanwhile, the structure of structured pruning (e.g. weight patterns or neurons) can be represented in an efficient way in memory reducing the index storage overhead. It, therefore, enables faster computation at the cost of less flexibility. (Hoefler et al., 2021)

### Persistent vs Ephemeral Sparsification

With persistent Sparsification the structure of the network gets changed once and permanently. Accordingly, it can be labelled as a generalization of Neural Architecture Search as described in section 2.1.1.
The two main elements that get pruned away by this procedure are Weights and Neurons. Pruning away a weight (see Figure 2.4 on the right) in a Neural Network entails cutting the link between two Neurons of subsequent Layers, on the other hand , pruning a whole Neuron (see Figure 2.4 on the left) means deleting all incoming and outgoing Weights from that Neuron. Additionally special elements such as filters in convolutional layers or heads in attention layers can be removed as well.

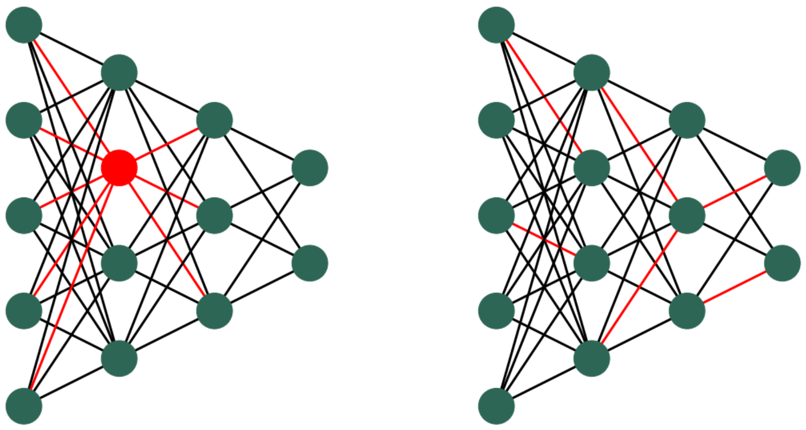


FIGURE 2.4: Visualization of Weight Sparsification (on the right) and Neuron Sparsification (on the left). Elements marked in red are the ones being pruned

Ephemeral Sparsification, on the other hand, is applied during training and inference calculations of single data point independently from the rest.
Elements of Neural Networks which do ephemeral Sparsification are, for example, Activation functions such as ReLU that set values to zero which are below a certain threshold. Additionally, dropout methods can also be considered as such (Srivastava et al., 2014b).

Both of these types of sparsification can be used independently or simultaneously. Persistent Sparsification with structured or unstructured pruning affects the forward pass during training and inference whereas Ephemeral Sparsification can be used during training with for example Dropout techniques or as well during the forward pass and inference with a specific choice of activation functions.
The complex nature of persistent pruning compared to ephemeral (Hoefler et al., 2021) leads to the development of schedules which we will elaborate in the next section.

### 2.2.2 When to prune?

Previously we answered the question as to what to prune specifically which elements in the neural network will be taken away. In this section, we study which schedule these pruning steps follow.

**Schedule steps.** Following Figures 2.5, 2.6, 2.7 depict the procedure in which we see when the pruning happens. The "Initialization" block defines the initialization of the neural network that will be pruned. In this block any initialization schemes can be utilized (e.g. Random, Kaiming (K. He et al., 2015b). Secondly, the "Prune" block is where the model gets "scored" meaning that the weights will be given a score according to a scoring method (see later in 2.2.3). These scores determine if the weight will be pruned. Finally, the "Train" block simply symbolizes that the model will get trained at this stage of the schedule.

**Layer-wise and global Sparsification.** Firstly, the aspect of layer-wise and global Sparsification needs to be introduced. During the "Prune" block of the procedure explained above, the weights of the network get "scored" based on their importance inside of the model according to a specific scoring method. Based on the scores, the pruning algorithm determines which weights to prune. Two ways of comparing the scores of the weights present themselves: layer-wise or global (Dong et al., 2017). In the layer-wise comparison, scores only get compared to other scores inside the same layer of the neural network compared to the global approach which compares all scores in the network at once (Ding et al., 2019).
One drawback which occurs in layer-wise sparsification is an Independency issue meaning that scores appearing as minima in the layer might not be global minima and, thus, taking a sub-optimal decision (Hong and Han, 2021). Additionally, in layer-wise sparsification smaller layers get pruned at the same rate as larger layers do which might, intuitively, create bottlenecks where one layer only consists of a couple of neurons. Similarly, global sparsification presents the tendency to prune later layers more than earlier layers which can lead to the same bottleneck issue. (Lee et al., 2019) Furthermore, an issue which could arise with global sparsification is "layer-collapse" where all weights in a layer get pruned away making the model untrainable (seen later in 2.2.4).

The pruning schedules can be divided in three categories: Pruning after training, Pruning During Training and Pruning before Training.

#### Pruning after training

This commonly used method consists of training a network to convergence and then pruning it. Pruning in such a schedule enables comparisons with the fully trained dense network as a baseline. Furthermore, (Janowsky, 1989) introduced the concept of "Fine tuning" (see Figure 2.5) where the pruned model gets retrained to reach a significantly better accuracy.

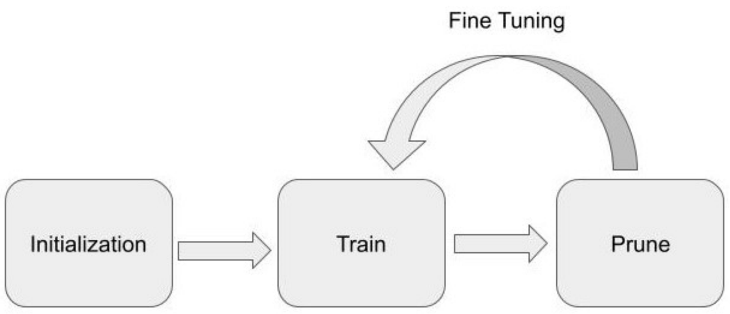


FIGURE 2.5: Fine tuning step. Visualization depicts the fine-tuning iteration represented by the arrow in a after training pruning scheme.

Fine tuning re-trains the pruned model based on the weights achieved during the previous training, Rewinding, on the other hand, (see Figure 2.6) resets the weights to the values of a previous training iteration $i$. And then fine-tunes the model with the same Hyperparameters than the previous iteration. (Renda et al., 2020) establishes that rewinding techniques outperform fine-tuning ones.

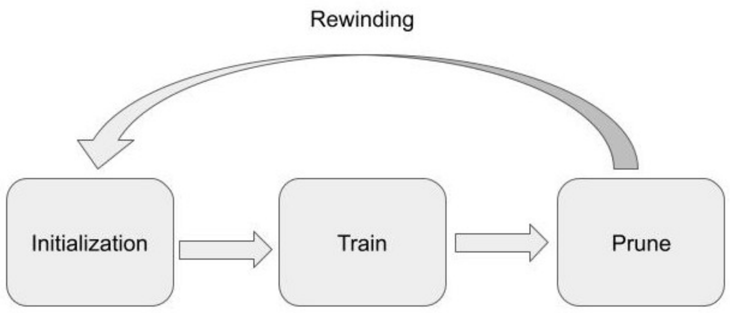


FIGURE 2.6: Rewinding step. Visualization depicts the rewinding iteration represented by the arrow in a after training pruning scheme.

Additionally, this schedule provides the fully trained dense neural network as a baseline that can be used for testing purposes compared to the following pruning schedules.

### Pruning during training

In this approach also called Iterative pruning, a network gets repeatedly trained, pruned and reset over a certain amount of iterations. As an example the algorithm introduced in the lottery ticket hypothesis paper (Frankle and Carbin, 2018) named Iterative magnitude pruning utilizes such an iterative rewinding procedure.
To that end, iterative procedures with fixed pruning methods see an improvement in their results when pruned during training (Finnoff et al., 1993). Additionally, training a dense model

fully before hand might lead to overfitting which is not correctable solely with pruning (Hoefler et al., 2021).

### Pruning before training

Finally, we'll look at the upfront approach that prune the network before training right after initialization (Lee et al., 2018; Wang et al., 2020) (see Figure 2.7). By pruning so early, this procedure avoids the expensive train-prune iterations present in the other procedures. Moreover, Liu et al., 2018 demonstrates that fine-tuning the pruned model with inherited weights is not better than training it from scratch, thus the resulting pruned architectures are what brings the benefit. Recent works, additionally, demonstrate that randomly initialized networks can be pruned before training with close to no loss in test accuracy (Liu et al., 2018).

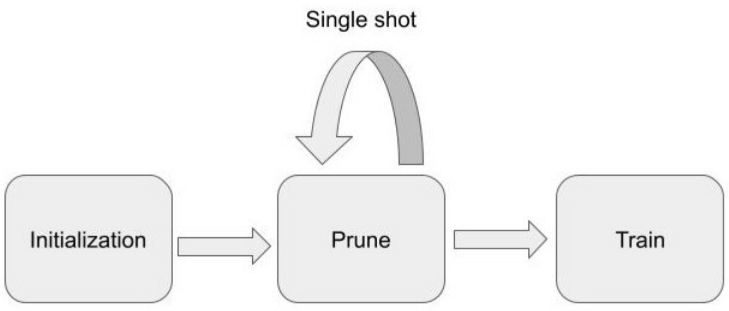


FIGURE 2.7: Up-front Single Shot step. Visualization depicts the single shot step represented by the arrow in a before training pruning scheme.

**Single-shot vs Multi-shot.** Previously explained procedure steps can be ran in a single-shot manner meaning once or in a multi-shot manner. (Lee et al., 2019; Wang et al., 2020) demonstrate a drawback of SNIP (Lee et al., 2018), a single-shot procedure we are analyzing in section 2.2.4, that stops the information flow (also called the gradient flow). Verdenius et al., 2020 propose a multi-shot approach applying the SNIP saliency criterion iteratively. It follows the intuition that a initially irrelevant parameter becomes more important after the last pruning step. Therefore, by pruning in multiple consecutive steps, this approach gives parameters "another chance". On the other hand, Janowsky, 1989 says that "There is no a priori reason why their initial values should remain optimal after the pruning process". In fact, it has been shown that re-training (finetuning) is essential for well-performing pruning schedules (Mehta, 2019).

### 2.2.3 How to prune?

In this section, we explain how weights get pruned. Testing each combination of pruned weights isn't feasible because of the exponential number of combinations to try, as well as because training only one Neural Network fully can cost up to 12 millions dollar (Brown et al., 2020) and, even for smaller networks, the amount of pruned possibilities emerging from the count of parameters is immense. Therefore, optimized pruning selection criteria have been developed.

Firstly, the simplest and least computationally expensive regime is called Data-free. With this approach, weights get pruned away solely based on a selection criterion without influence of the data observed or the training specifics. One selection criteria is called "neuron- / weight-similarity". The scoring mechanism follows the selection criteria and distributes lower scores to similar weights/neurons and, therefore, they get reduced to a single weight/neuron effectively keeping only one of both. Srinivas and Babu, 2015 demonstrate that with this method they can remove up to 85% of the total parameters for an MNIST network and about 35% for AlexNet while keeping comparable results, but prunes less efficiently for bigger networks. Another approach consists of pruning away weights which have low magnitude values $\|\mathbf{v}\|$ meaning they have a very low impact on the decision of the neural network.

Secondly, we'll look into a class of pruning methods which is called "Data-aware" approaches. In these approaches, we observe the sensitivity of the network towards the input data. One method has been to let the all data run through the model and if one neuron keeps its value close to zero it'll be pruned away. This is a simple approach to prune away trivial neurons in the network (Hoefler et al., 2021). Furthermore, the "input sensitivities" method defines the variation of two neurons values depending on different input samples. The neurons which are strongly positively or negatively correlated to certain inputs don't help the classifcation and thus are collapsed to one neuron only (Castellano et al., 1997). These methods are more expressive than data-free approaches. We'll observe in detail an approach in the next chapter which uses such an idea.

Finally, the most computationally expensive regime is called "training-aware" because the weights importance is defined by the impact it has on the loss function of a fully trained network. This idea has been used in classical methods such as LeCun, Denker, et al., 1989, where they look at which parameter to prune resulting in the least increase in the second order Taylor approximation of the loss function. This work can be considered as an "optimization" approach to pruning.

### 2.2.4 State-of-the-art Network Pruning

In this part, we present four pruning algorithms in depth that are essential to our thesis. We took the Lottery Ticket Hypothesis paper (Frankle and Carbin, 2018) as base for our research and decided to find solutions implementing upfront methods, contrary to Frankle and Carbin, 2018, with the optic to avoid the computationally expensive training-pruning loops. To that end, we found SNIP (Lee et al., 2018), GraSP (Wang et al., 2020) and Synflow (Tanaka et al., 2020); three promising state-of-the-art pruning algorithms with especially interesting techniques.

#### Neural Network Pruning

Neural network pruning can be theoretically describe as entailing:

- a dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$
- a goal sparsity level $\kappa$ which defines the number of non-zero weights
- a loss function $l(\cdot)$ for example cross-entropy loss
- a set of parameters $\theta$
- a set of weights $w \in \mathbb{R}^{|\theta|}$
- a $L_0$ norm $||\cdot||_0$ describing the number of non-zero elements in a vector

It can, therefore, be written as an optimization problem:

$$\min_{w} L(w\,;\,D) = \min_{w} \frac{1}{n} \sum_{i=1}^{n} l(w\,;\,(x_i\,,\,y_i))\,, \qquad (2.4)$$

$$with\;||w||_0\;\leq \kappa$$

Recent approaches use a saliency methodology which treats the above described optimization problem as removing unnecessary weights in a neural network. Therefore, adequate selection criteria need to be determined to find these unnecessary weights.

#### IMP: Iterative Magnitude Pruning

This algorithm has been introduced in the paper by Frankle and Carbin, 2018 building on the "Lottery Ticket Hypothesis" which states that each randomly initialized dense neural network has a subnetwork that when trained in isolation, for at least as many iterations as the original network, matches the test accuracy of the full-network. This trainable subnetwork is, then, also called the "Lottery Ticket".
Iterative Magnitude Pruning finds such "Lottery Ticket" in a data-free iterative manner applying a smallest-magnitude criterion to decide which weights get pruned away. More specifically these are steps that the algorithm takes:

1. Randomly initializes the neural network $f(x, \theta)$, initial parameters $\theta = \theta_0$
2. Trains the network with Stochastic Gradient Descent (SGD) for $i$ iterations, arriving at parameters $\theta_i$
3. Prunes in an unstructured way $p\%$ of the parameters in $\theta_i$ based on the smallest-magnitude criterion $|w| \in \theta_i$, creating a mask $m = \{0, 1\}^{|\theta|}$
4. Resets the remaining parameters to their initial values $\theta_0$

IMP iterates through steps 2,3 and 4 for $n$ rounds each round pruning $p^{1/n}$% of the weights. At the end of the $n$ rounds, a Lottery Ticket $f(x; m \odot \theta_0)$ is found.

Eventhough, IMP generates smaller and more accurate LT and its iterative nature makes it avoid layer collapse (Tanaka et al., 2020), it is computationally expensive because of its training-pruning iterations.

**SNIP: Single-Shot Network Pruning**

SNIP (Lee et al., 2018) is a data-aware upfront pruning algorithm deleting redundant connections in a network before training thanks to a "connection sensitivity" criterion.

Recent works have used the magnitude of the weights (see section 2.2.4) $|w|$ as a saliency criterion which presents a dependency on the scale of the weights and, further, needs significant pre-training.

SNIP introduces data-aware upfront approach which prunes before training at initialization avoiding the expensive and time-consuming training-pruning iterations. It is able to prune without the influence of the weights thus the equation of [2.1] can be adjusted to:

$$\min_{c,w} L(c \odot w\,;\, D) = \min_{c,w} \frac{1}{n} \sum_{i=1}^{n} l(c \odot w\,;\, (x_i\,,\, y_i))\,, \tag{2.5}$$

$$with\, c \in \{0,1\}^{|\theta|},\ ||c||_0\ \leq \kappa$$

The introduction of $c$ permits to determine the importance of a certain weight in relation to the loss function. $c_j$ describes if the connection $j$ is active ($c_j = 1$) or pruned ($c_j = 0$). (see section 2.2.4) studies the difference in loss when $c_j$ is active or pruned while keeping everything else constant. This difference can be defined by:

$$\Delta L_j(w\,;\, D) = L(1 \odot w; D) - L((1 - e_j) \odot w; D), \tag{2.6}$$

$$with\, e_j \text{ the unity vector of } j$$

This way of representation is problematic because $\Delta_j$ needs to be compute for every $j \in \{1, ..., |\theta|\}$. Thus the binary constraint on $c$ gets relaxed becoming differentiable. It is now represented as $g_j$ that is equivalent to the rate of change of $L$ by $\delta$ steps and can be computed in one pass for all j at once.

We can now formulate the saliency criterion "connection sensitivity" by taking the magnitude of $g_j$ written as:

$$s_j = \frac{|g_j(w; D)|}{\sum_{k=1}^{|\theta|} |g_k(w; D)|} \tag{2.7}$$

To update $c_j$, we test if it is contained in the top-$\kappa$ weights.

We can put all these elements together to formulate the SNIP pruning algorithm:

1. Initializes the weights $w$ with a Variance Scaling Initialization (Glorot and Bengio, 2010)
2. Samples a mini-batch of the training data $D^b$
3. Computes the connection sensitivity $s_j$ of each weight $j$ (equation 2.7)
4. $\bar{s}$ = SortDescending(s)

5. Updates the $c_j$ for each weights depending if it belongs to the top-$\kappa$ or not
6. Follows with a regular training and masking step yielding:

$$w^f = c \odot w^* \tag{2.8}$$

$$\textit{with}\ w^* \text{ the trained weights}$$

**GraSP: Gradient Signal Preservation**

GraSP works upon the idea of SNIP but proposes another saliency criterion "Gradient Signal Preservation". "Connection Sensitivity" is deemed by Wang et al., 2020 sub-optimal because the gradient of each weight could change drastically after the pruning step, hypothetically, due to interactions between weights. Because SNIP observes weights one by one, it might prune away a weight that is important to the "flow" of information in the network. For that reason, GraSP proposes a saliency criterion which keeps the gradient flow, essentially pruning the weights which when taken away lead to the least decrease in the gradient norm of the network.

Wang et al., 2020 can mathematically define the influence of a gradient norm on the loss function with the following directional derivative:

$$\Delta L(\theta) = \lim_{\epsilon \to 0} \frac{L(\theta + \epsilon \nabla L(\theta)) - L(\theta)}{\epsilon} = \nabla L(\theta)^T \nabla L(\theta) \tag{2.9}$$

Equation 2.9 describes that a large gradient norm indicates that each gradient update creates a greater loss reduction.

To analyze the change in gradient flow in reaction to a perturbation $\delta$ to the weights, emulating the pruning of those weights, after pruning, Wang et al., 2020 utilize:

$$S(\delta) = \Delta L(\theta_0 + \delta) - \Delta L(\theta_0) = 2\delta^T H_g + O(||\delta||_2^2) \tag{2.10}$$

The Hessian matrix $H_g$ depicting the correlation between weights in the network with the influence of the perturbation $\delta$ explains how the pruning of $\theta_0$ affects all other weights in the network.

The GraSP pruning algorithm works in the following way:

1. Samples a mini-batch of the training data $D^b$
2. Computes the Hessian and gradient product as $H_g$
3. Computes the importance of each weight:

$$S(-\theta_0) = -\theta_0 \odot H_g \tag{2.11}$$

4. Computes $\kappa$th percentile of $S(-\theta_0)$ as $\tau$
5. Removes weights larger than threshold $\tau$
6. Trains the network with weights $m \odot \theta$

**SynFlow: Iterative Synaptic Flow Pruning**

In Tanaka et al., 2020 IMP, GraSP and SNIP get studied and directly compared to the algorithm called "Iterative Synaptic Flow Pruning" developed in this paper. The core idea of this paper is that gradient-based approaches such as SNIP and GraSP inevitably fall short to "layer collapse"

which happens when all the weights of a layer get pruned making the network untrainable. Therefore, Tanaka et al., 2020 elaborate an iterative algorithm with positive saliency scoring respecting a "Maximal Critical Compression" axiom called Synflow avoiding layer collapse while reaching state-of-the-art results.
Saliency scores are defined as such:

$$S = \frac{\partial R_{SF}}{\partial \theta} \odot \theta \tag{2.12}$$

They develop a new loss function:

$$R_{SF} = 1^T (\prod_{l=1}^{L} |\theta^{|l|}|) 1 \tag{2.13}$$

$$with\, 1 \text{ the all ones vector, } L \text{ the amount of layers in the network}$$

Equation 2.12 yields positive synaptic saliency scores called Synaptic Flow.

The full algorithm goes as follows:

Initialized mask $\mu = 1$ and Compression ratio $\rho$

Repeat for $k \in \{1, ..., n\}$ iterations:

1. Mask parameters:
$$\theta_\mu = \mu \odot \theta_0 \tag{2.14}$$
2. Evaluate and Compute the Synflow score [Equation 2.12] with $\theta = \theta_\mu$
3. Calculate the theshold:
$$\tau = (1 - \rho^{-k/n}) \text{ percentile of} S \tag{2.15}$$
4. Update the mask:
$$\mu \leftarrow (\tau < S) \tag{2.16}$$

## 2.3 Further Concepts relevant to our Thesis

Early research on decision trees helped shape the ideas we have today. Popular decision tree were for example Decision Trees (Quinlan, 1986) and Random Forest Trees (Breiman, 2001).

**Decision Trees.** They are formed, like the name suggests, in a tree like structure formed of nodes which represent a condition on an attribute of the data which needs to be tested. On the leaf nodes of the tree is the classification decision of the model.

The induction of decision trees is one of the oldest and most popular techniques for discriminatory classification models. which has been developed in the machine learning (Utgoff, 1989) and statistical (Kass, 1980) domains.
Nevertheless, decision trees present disadvantages. They are unstable to small changes in the data that lead to large changes in the structure of the decision tree. Additionally, data including multiple leveled categorical data show bias towards those (H. Deng et al., 2013). Finally, they are often inaccurate and overfit to the data that's why we utilize another approach called Random Forest.

Ensemble learning methods, which include Random Forest, use different learning algorithms and their predictions to gain a better overall predictions. (Valentini and Masulli, 2002)

**Random Forest.** Being a ensemble learning method, it can be applied to classification and regression tasks. In our thesis, we'll focus on classification therefore the output of the random forest will be the class chosen by most trees.

Random Forest work on the basis of multiple other trees by collecting their decisions and using the decision that has been taken the most as the final one.
Random Forests correct the overfitting habit of decision trees. (Breiman, 2001) Additionally, Random Forests generally outperform Decision trees. (Ali et al., 2012)

# Chapter 3

# Methods

The task of obtaining better performing Lottery Tickets from combinations of existing algorithms can be approached in various ways. One strategy would the the analysis of the mathematical concepts upon which the algorithms are based and proposing an algorithm which reuses elements from previous work. Indeed, many sources (Frankle and Carbin, 2018; Lee et al., 2018; Wang et al., 2020; Tanaka et al., 2020; Sanh et al., 2020) use a theory based approach successfully. Our research is conducted in the context of applied science and therefore, we have decided to instead investigate the topic with a focus on empirical experiments. As a result, we are looking to combine different pruning algorithms in such ways that the original implementations can be reused.

Collecting empirical evidence on the performance of pruning algorithms is not trivial, as it involves pruning, training and evaluation of a signification number of networks and therefore requires a lot of compute and time. As outlined in chapter 1, in the context of this thesis, both of which are limited. With the restrictions in place, we select to conduct experiments that are computationally feasible in our setting.
With pruning being mature research field going back decades (LeCun, Denker, et al., 1989), many things have already been described previously (Hoefler et al., 2021). To still be able to contribute to the state-of-the-art, we also select experiments which have not been described in the sources we are aware of.

To further mitigate the issue of resource scarcity, we decided to focus our efforts on pruning a single model trained to perform a single task. However, results of pruning algorithms have been shown previously to have difficulty to transfer to a different task or model (Frankle et al., 2020). With this in mind, we are aware that any findings we observe might not fully generalize to other models or tasks, yet we believe that such experiments may yield interesting findings nonetheless. In this sense, we follow the approach taken by Frankle and Carbin, 2018.

## 3.1 Hypothesis

Research building on the original discoveries made by the authors of the Lottery Ticket Hypothesis (Frankle and Carbin, 2018) found, that different strategies of obtaining a Lottery Ticket tend to produce partially overlapping subnetworks (Paganini and Forde, 2020). Firstly, this means that there are many different sparse trainable sub networks and the Iterative Magnitude Pruning (Frankle and Carbin, 2018) algorithm represents only one of the ways to find one. Secondly, while the Lottery Tickets are different, there exists a subset of weights present in all of the algorithms they looked at (Paganini and Forde, 2020). For this subset, there is consensus between all algorithms about the importance of these weights. This subset of the Lottery Ticket, to which the authors refer to as the core part of the LT, therefore is the source of any differences in performance between the different algorithms.

We hypothesize that an algorithm leveraging all information that led to the differences in the Lottery Tickets is able to produce Lottery Tickets closer to an ideally performing LT.

If this hypothesis holds, we are be able to measure an increase in performance of the resulting Lottery Ticket obtained by the combined algorithm on the task compared to a Lottery Ticket at the same sparsity obtained by any of the baseline algorithms alone.

We conduct a series of experiments to validate this hypothesis, which we describe in the following sections. Every experiment implements a different way of combining multiple algorithms. The first approach is a meta learning model, which learns to predict approximately optimal score for a weight, given the scores from the four existing algorithms. In a second experiment, we build an algorithm where one scoring criterion is used to score the weights and then a second scoring criterion is applied on the output of the first one, allowing us to sequentially use two different algorithms in a setup we call Stacked Scoring. Lastly, we build on this experiment and raise the question whether using scores as weight initialisation results in trainable networks and how this impacts the performance of the Lottery Ticket after pruning.
In the following sections, we give a detailed overview on the motivation and setup of these experiments.

## 3.2 Meta Learning Scores

The first experiment we conduct is based on the idea, that the ideal scores producing an ideally performing Lottery Ticket could be learnable, based on the observation of many such ideal subnetworks. In this sense, we are implementing an ensemble learning model (Maclin and Opitz, 1999) which combines the output of several models, in our case pruning algorithms. However, as outlined in chapter 2, finding the ideal Lottery Ticket for a given sparsity is not trivial and therefore, an approximation of the ideal ticket has to be used instead. For this purpose, we propose to use the best combination of scoring criterion and pruning regime to generate a ticket in the most informed way possible.
With the target variable defined, several options remain what data the model should operate on. The two main options are either full networks or individual weights, both of which we are going to elaborate on in the next sections.

### 3.2.1 Per-Network Prediction

On one end of the scale, the model takes scores for a full network as an input and generates all scores to prune the whole network with as an output. The benefits of such model is the vast amount of information available to the model. By seeing all layers at once, it is theoretically able to optimize for producing highly connected sub-structures across layers. Further, such model would not even necessarily need to have the existing pruning scores available, as it may learn its own and improved scoring based on the weights alone.
One way to implement this model could be the adaption of Hypernetworks (Ha et al., 2016), which observe many examples of a network training and are, in their original form, able to directly generate the weights themselves (Zhmoginov et al., 2022). While generating the weights directly does more than just finding Lottery Tickets as it affects all weights in the network, we hypothesize, that when observing many Lottery Tickets where weights are masked out and therefore zero, this approach would lead a Hypernetwork to also produce sparse networks where a large portion of the weights are zero. It remains an open question how one would control the ratio of weights to remain in this scenario.
Also a promising approach could be to represent the data as a sequence of layers as done in previous work (Zhmoginov et al., 2022), for it to be able to cope with different model dimensions.

In the context of pruning, a potential strategy could be to let a Hypernetwork observe Lottery Tickets train as they are found by an algorithm producing close to ideal Lottery Tickets, for example IMP.
However, this approach also comes with significant drawbacks. Since a full network and a corresponding Lottery Ticket represent a single data point, many LTs need to be found first to produce a dataset large enough to train a model (Sun et al., 2017). In the context of IMP, this training is particularly compute expensive as many iterations of the network need to be trained. The compute required is, in the context of this thesis, not available and no dataset mapping networks to the respective lottery tickets exists to our knowledge.

### 3.2.2 Per-Weight Prediction

At the other end of the spectrum, it is also possible to have the model learn to predict scores on a per weight basis. For each weight, the model predicts its likely score based on the scores the existing algorithms, which run in an upfront schedule, approximating the ideal LT. Implicitly, we hypothesize, some the context of the weight in the network is encoded in the existing scores themselves by virtue of the ways they are calculated and therefore available to the model to learn from. In the case of data-aware pruning schemes, information about the task is also encoded in the score.

The benefit of this version is, that training a single network yields as many data points as it has parameters, generating large amounts data to train a model with.
Also, the pipeline to apply the model once trained is straightforward:

1. Initialize the model
2. Run each algorithm to score the model
3. For each weight: predict the score
4. Apply top k selection on the initial weights

In figure 3.1, we visualize these steps on a single example layer.

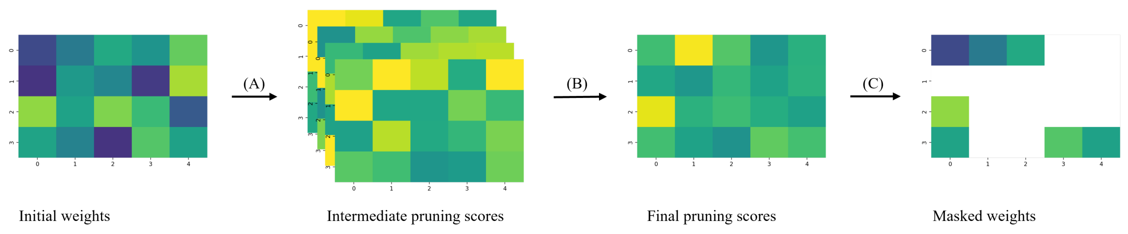


FIGURE 3.1: Illustrative visualisation of the meta-learning model. From a weight matrix, four matrices with intermediate scores are produced (A). These four scores are the input to our model, which generates a single matrix of final scores (B). These final scores are the selection criteria that get applied to the weight matrix (C). Of these operations, only (B) runs on a per-weight basis, all others remain a layer-wise operation.

Due to the advantages in both the volume of data available and the simplicity of the implementation as described above, we decide to implement this option in this thesis. We measure the success of this model by whether it outperforms the individual algorithms on the same set of initial weights. The maximum performance we expect such model to achieve is the performance of the algorithm we used as an approximation of the ideal LT for training. While we do not expect our model to outperform the existing the baseline, it would be possible to do so without iterative training, saving compute at training time.
A risk with this approach lies in the fact that the approximately ideal LT was found with a pruning method involving training on a given dataset. Therefore, it can be argued that the model might learn to produce data and task specific LTs. While we believe this to be a valid risk and hypothesize that, as the model only has a the context of the scores for an individual weight, it is also possible that generally applicable patterns are found. If the model performs well on the same task it was trained on, the model has to be validated by applying it to another model and task. Therefore, we argue this to still be a valid experiment to pursue.

In the case the model successfully learns to produce Lottery Tickets trainable to a higher accuracy compared to other baselines before training, we would be able to demonstrate that it is indeed possible to obtain higher performing Lottery Tickets with combinations of algorithms, and thus prove our hypothesis.

## 3.3 Stacked Scoring

While a meta model is a traditional way of learning to score weights for pruning, we also propose alternative options. In this section, we introduce the idea of what we term Stacked Scoring, where two pruning algorithms are applied sequentially as opposed the parallel evaluation of scores in the context of the meta model.

### 3.3.1 Motivation

In the context of weight magnitude based pruning, we observe a close relationship between the value of the weight and its score. We hypothesize that intuitively, the magnitude of a weight can be understood to define its contribution to the final result, while the scores obtained from a pruning algorithm indicate the importance of the weight. When a network is pruned to a low ratio of weights remaining using these scores of importance, the remaining weights with a high score have a higher contribution to the result of the pruned network. Therefore, we hypothesize that while the weight value is the actual contribution, the pruning scores can be seen the expected contribution of the weight once pruned.

When pruning a network, there are two transformations involved (LeCun, Denker, et al., 1989). The first one transforms the weight matrix to scores and by applying what we refer to as the scoring criterion. The second operation transforms the scores to a Boolean mask. It works by looking at the magnitude of the scores and selecting the $k$ largest values to be part of remaining pruned network. In this way, we draw comparisons between the construction of scores and the selection thereof. When framed in this way, the traditional magnitude pruning scheme can be restated as a combination of two operations on the magnitude criterion: First scoring the weights by magnitude and then selecting the scores by their magnitude.

### 3.3.2 Proposed Algorithm

Based on this observation outlined above, we propose a novel approach of combining two scoring mechanisms by substituting the second, previously always magnitude based operation, with a different state-of-the-art scoring criterion.

By doing this, we hope to be able to leverage insights from the information available to both algorithms and produce a better performing lottery ticket as a result of this.

**Algorithm 1** Stacked Scoring

```
Require: network $f(x;\theta_0)$, compression ratio $\rho$, scoring function $a$, scoring function $b$
  $m \leftarrow 0$                                  ▷ Initialize mask
  $S_a \leftarrow a(\theta_0)$                       ▷ Obtain intermediate scores from $a$ on $\theta_0$
  $S_b \leftarrow b(S_a)$                            ▷ Obtain intermediate scores from $b$ on $S_a$
  $\tau \leftarrow (1-\rho)$ percentile of $S_b$     ▷ Find threshold
  $\mu \leftarrow \tau < S_b$                        ▷ Update mask
  $f(x;\mu \odot \theta_0)$                          ▷ Apply mask
```

At the core of the algorithm proposed lies the sequential application of two scoring algorithms as outlined above. The first one, Step 1 obtains the intermediate scores from the scoring function A applied on the initial weights of the model. The second one applies scoring function B on the intermediate scores. For the scoring functions, we test the same selection of state-of-the-art algorithms as in the preceding section, SNIP, GraSP and SynFlow, and magnitude based pruning as a baseline. In the case of SNIP and GraSP, the scoring functions require more information than just the value of the weight. For the backward passes performed, the scoring function needs to have the data loader and model available. Therefore, the simplest implementation is to temporarily replace the model weights with the intermediate scores, resetting them to their original initialisation after the the computation of the mask.

We evaluate this algorithm on the accuracy of the Lottery Ticket at different values of sparsity and compare it to the existing algorithms.

With this experiment, we again hope to prove that combining pruning algorithms leads to higher performing sub networks.

## 3.4 SaW: Scores as Weights

Building on the idea outlined in the previous section, we raise the question whether the scores themselves can be used as initial values to prune on and also begin training with.
As with previous the experiment, this experiment is also motivated by the relationship between pruning scores and weight magnitudes as both represent a metric of weight importance. Further, to apply Stacked Scoring as described in the preceding section, we already implement a mechanism to apply the loss function to pruning scores to be able to use data-aware pruning schemes as the second phase scoring criterion.

In the implementation of Stacked Scoring, weights are replaced by the intermediate score and later reset to their initial value. The two main differences to Stacked Scoring therefore are, that we only calculate the intermediate scores and we we don't reset the weights to their initial values.

The impact of weight initialisation schemes has been discussed in literature Lee et al., 2019, finding that the random initialisation for a network has a large impact on the effectiveness of pruning methods applied. Effective initialisation of neural networks however is an entire field of study (Narkhede et al., 2022) separate from pruning and not the focus for this thesis. It has been shown that there are initialisation schemes that perform better than a uniformly random distribution (Glorot and Bengio, 2010; K. He et al., 2015b) when adhering to the stochastic attributes as outlined in the schemes described. Pruning scores are not designed to be effective initial values. However, their magnitude is related to the importance of the weight and thus also related to the contribution the weight should make to the final output. Therefore, we hypothesize that pruning scores are suitable to be used as weights.

We evaluate this experiment on the basis of whether an unpruned network is able to train successfully at all when using pruning scores as weights. Further, we apply all pruning algorithms in scope on weights generated from all pruning algorithms and test the performance in term of accuracy of the Lottery Ticket on the task.

# Chapter 4

# Experimental Setup

The experiments described in chapter 3 are implemented on common code base providing a shared environment. Based on the publicly available repository from the team behind the SynFlow algorithm Tanaka et al., 2020, we implemented an environment where we can conduct and analyze the experiments described. In this chapter, we aim to give a brief overview on how the experiment environment works and how we implement the concepts described in chapter 3.

## 4.1 Code base

We selected the SynFlow code base as the foundation, despite the availability of OpenLTH Jonathan Frankle, 2020, a framework to conduct lottery ticket experiments provided by the author of the original Lottery Ticket Hypothesis Frankle and Carbin, 2018. OpenLTH implements all experiments found in Frankle and Carbin, 2018; Frankle et al., 2019; Frankle et al., 2020, along with a highly configurable structure to run custom experiments with the Lottery Tickets. Experiments can be conducted from the included command line interface, without changing any source code.
For our purposes, we found OpenLTH to be less suitable, as it is not easily extendable and does not include any other pruning strategy than iterative magnitude pruning. In comparison, the SynFlow code base not only includes implementations of the different pruning algorithms in scope of this paper, it also is easier to understand and extend. Therefore, we decided to base our implementation on it instead of the more obvious choice of OpenLTH. Significant changes to the code base however were necessary to be able to specify, parallelize, monitor, persist and analyze the experiments in a way suitable in our context. All code is written in Python (Van Rossum and Drake, 2009) and uses PyTorch (Paszke et al., 2019) as the machine learning library and is available on GitHub[1].
A full overview on the software and hardware used can be found in Appendix A.

In total, our final results build on over 7000 individual runs, with those models taking over 120GB of storage to persist. All of this data is available on the ZHAW SharePoint[2].

## 4.2 Model and Task

In the literature, several standard models and tasks are used to compare the performance of the different pruning algorithms. Among the widely used models are LeNet (LeNet_300_100, LeNet-5, Lecun et al., 1998), VGG (-11, -16, -19, Simonyan and Zisserman, 2014) , ResNet (-16,

[1]GitHub: https://github.zhaw.ch/lutzurb1/BAIT

[2]Sharepoint: https://zhaw-my.sharepoint.com/:f:/g/personal/lutzurb1$_s tudents_z haw_c h/Env - l4iTtWlNlpovR4F_x 7sBwSKsC2ksfapyHkl8jenK2g?e = UNS5nk$

-18, -20, K. He et al., 2015a) and the WideResNet variants thereof (Zagoruyko and Komodakis, 2016). Between these models, there is a significant difference in size, ranging from 266’200 parameters for a LeNet_300_100 to over 140 million for a VGG-16. For us, the ability to iterate the experiments and get results quickly is crucial. Therefore, only smaller models are feasible for us.

Based on these observations, we decide to implement a LeNet_300_100 model, for several reasons. Firstly, it is well established and used across multiple publications (Frankle and Carbin, 2018; Lee et al., 2018; Uber et al., 2020) relevant in our context. Secondly, it has a small number of parameters that allow us to train it in a reasonable time on our setup. Lastly, its simple architecture allows us to easily reason about it. Despite its name sounding similar to the more popular LeNet-5 (Lecun et al., 1998), the LeNet_300_100 architecture is a fully connected Multi Layer Perceptron (Hornik et al., 1989) with two hidden layers and contains no convolutional layers (LeCun, Haffner, et al., 1999) like a LeNet-5 would have (Lecun et al., 1998). This leads to a model that is comparably fast to train on the hardware available to us in the context of this thesis. The Model is trained on the task of recognizing hand written digits from images, the dataset used for this task is MNIST (L. Deng, 2012), as used in Frankle and Carbin, 2018 and Lee et al., 2018.

Many different architectures have been suggested in the literature for the task of recognizing hand written digits, often with Top-1 accuracy [3] scores much higher than what we achieve with our LeNet_300_100 model (An et al., 2020; Hirata and Takahashi, 2020). From this, we conclude that a simple architecture such as a LeNet_300_100 is not the ideal choice for this task when aiming for maximum performance (Gupta, 2020). For the purposes of this thesis however, absolute performance is not the priority. Instead, we aim to observe the change in accuracy under pruning. Therefore, we take the Hyperparameters found in Frankle and Carbin, 2018 and keep them static and unoptimized across the experiments conducted, to ensure comparability between results.

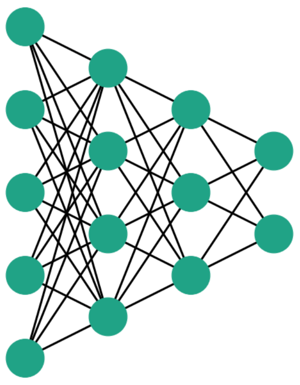


FIGURE 4.1: Illustrative example of a fully connected neural network with two hidden layers. In the case of a LeNet_300_100 model to be used on MNIST, the input layer (leftmost layer on illustration) has a dimension of 784 neurons, corresponding to each pixel in the input image with dimensions 28 by 28. The hidden layers are 300 and 100 neurons in size according to the definition of the architecture. The right most layer represents the output classes, for MNIST this results in 10 neurons needed.

[3]Top-1 Accuracy is one of the standard performance metrics used in our context (Frankle and Carbin, 2018; Lee et al., 2018; Wang et al., 2020; Tanaka et al., 2020)

### 4.2.1 Hyperparameters

To better understand the effects of the changes to variables, we choose to keep the basic Hyperparameters stable across all of our experiments. For our model and training process, we adapt the Hyperparameters used in the original Lottery Ticket Hypothesis. A full reference of all hyper parameters used can be found in appendix D.

We focus on a small number of variables to specify our experiments. All of the following parameters are changed depending on the experiment:

- Pruning Algorithm
- Desired Sparsity
- Initialisation Strategy
- Pruning Schedule: Upfront or Iterative
- Pruning Schedule: Single- or Multi-Shot

With changes to any of these variables, we are able to specify all the experiments in this thesis.

## 4.3 Statistical Significance

To test results for statistical significance, we apply a well established p-test criterion, measuring whether or not a given result compared to a baseline is the product of randomness. As was done by Tanaka et al., 2020 and is common practice (Bzdok et al., 2018), we choose $\alpha$ to be 0.05 as the threshold.
For more information on statistical significance tests in the context of machine learning, Horel and Giesecke, 2019 provide an overview.

# Chapter 5

# Results

In this chapter, we give a detailed overview on the results of the experiments described in chapter 3, as executed in the environment described in chapter 4.

## 5.1 Baseline Performance

To establish a baseline to compare our changes to the algorithms in our experiments, we run each pruning algorithm in its original form on our model and task. In addition to the four state-of-the-art pruning algorithms we selected, we run each experiment on a randomly pruned baseline (Su et al., 2020). This allows us to better contextualize the results produced by more sophisticated algorithms.

Without pruning, our networks achieves an accuracy of 98.24, which is in the expected range of such basic architecture (Gupta, 2020).

The out of the box, the unaltered versions of the algorithms on our model and dataset mirrors findings from previous work (Tanaka et al., 2020) for SNIP and SynFlow. Surprisingly, in our context, we observe better results from SNIP compared to GraSP, whereas in the original paper for GraSP, a higher accuracy of the LTs produced by GraSP is reported (Wang et al., 2020). We theorize that the this difference could be due to our model architecture which is a simple multi layer perceptron, whereas Wang et al., 2020 only benchmark their algorithm on VGG and ResNet architectures. Figure 5.1 illustrates the drop in Top-1 Accuracy of a LeNet_300_100 model compared to the unpruned model across sparsity levels of 50% to 0.4% of weights remaining on a logarithmic scale.

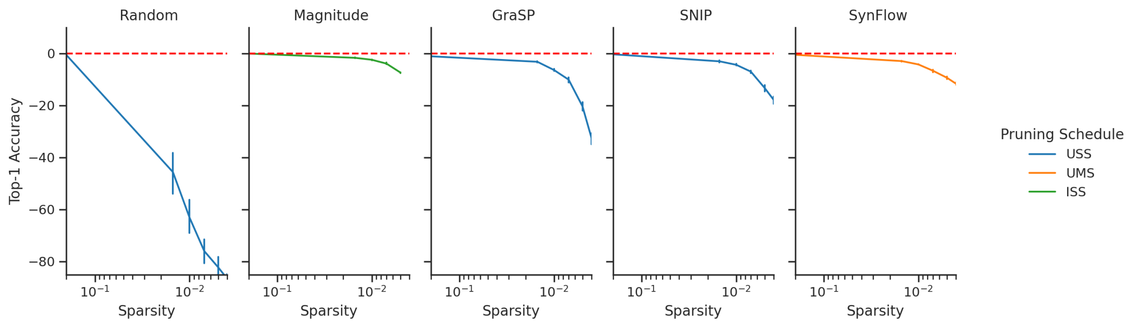


FIGURE 5.1: Baseline Performance of pruned networks compared to the unpruned model (red line) with the model sparsity on the x-axis with a logarithmic scale and Top-1 Accuracy on the y axis.

Similar to previous findings (Frankle and Carbin, 2018; Tanaka et al., 2020) and in line with intuitive expectations, the accuracy of the resulting lottery ticket generally decreases as more weights are removed, as fewer weights limit the models ability to learn (Hoefler et al., 2021). In this case, as we use a model with 266'200 parameters in total, only 1'064 parameters remain at the lowest measured sparsity of 0.4%. Interestingly, random pruning is able to perform decently until a sparsity of 2%, after which accuracy drops significantly. This again is the effect of networks being easy to prune in the trivial range, which others have described previously (Su et al., 2020; Hoefler et al., 2021).

When investigating the magnitude of weights that have been selected by the pruning algorithms, we observe a pattern across all pruning algorithms. All pruning algorithms appear to be selecting weights from a bi-modal distribution of initial values. For magnitude pruning, which is applied iteratively here as per IMP, SNIP and GraSP, this bi-modal pattern is also present in the distribution of final weights after training, where as for SynFlow, this is not the case.

We hypothesize that despite the information advanced pruning algorithms have available, the magnitude of a weight still is one of the most important factors in whether or not to prune a weight. In the case of data aware algorithms like SNIP and GraSP, Su et al., 2020 argue that this might indicate the failure of the pruning algorithms to use the information about the data available to them and instead to a large extent use the magnitude criterion. For SynFlow, we hypothesize that this is an indication on its reliance on the magnitude criterion when scoring weights.

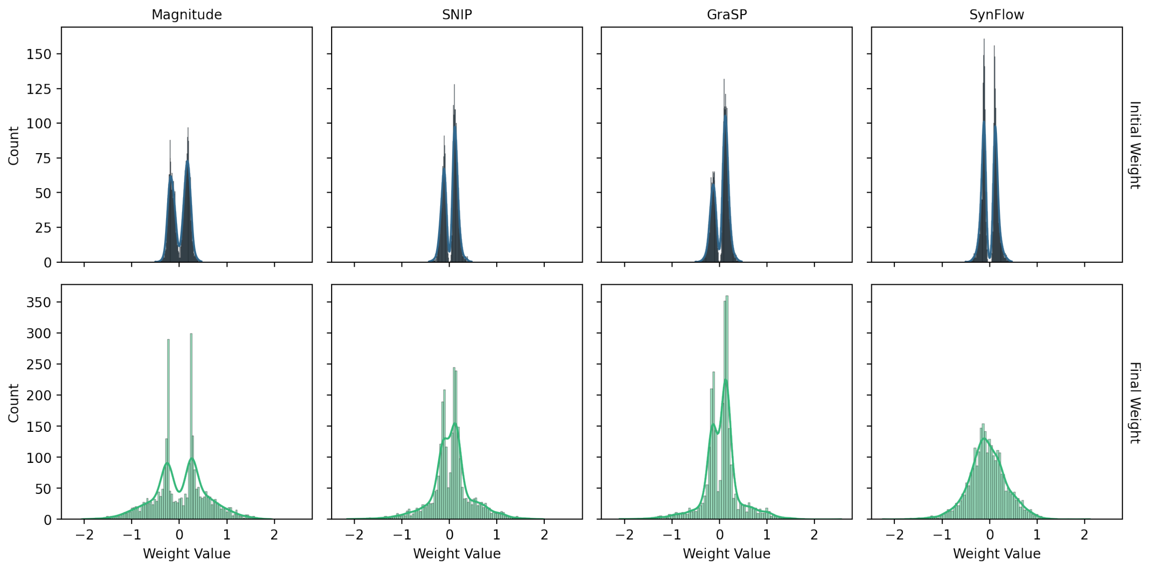


FIGURE 5.2: Distribution of weights chosen by different algorithms chosen at 0.5% sparsity for a single set of initial weights.

With the distribution of weights being similar across different algorithms as shown in figure 5.2, it might appear as if the different algorithms find similar Lottery Tickets. As shown in figure 5.3 as we measured with the Jaccard index (Costa, 2021), the lottery tickets do share a common core. Thus we are able to confirm the findings from Paganini and Forde, 2020.

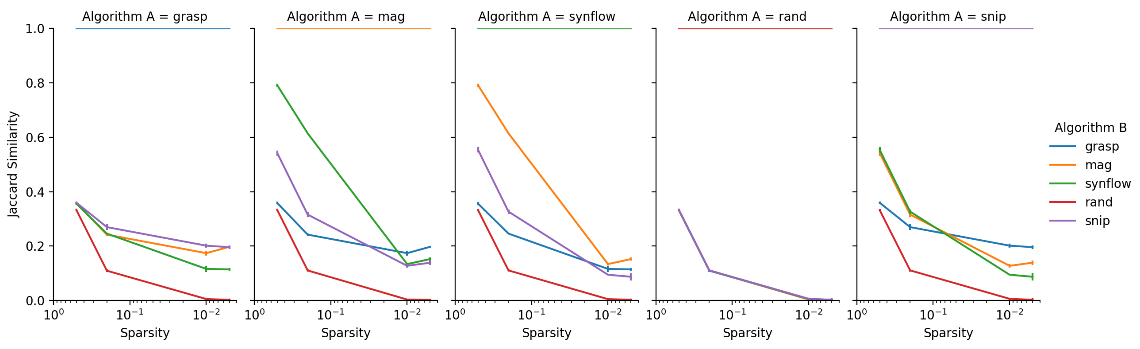


FIGURE 5.3: Jaccard similarity between two Lottery Tickets found by algorithm A and algorithm B. Even at low sparsity of 0.5%, a common set of weights is selected. On the y-axis, higher Jaccard similarity implies a larger set of overlapping weights.

In summary, by confirming findings described in previous work as outlined above, we are able to verify that our implementation of pruning scores works and we are able to build our experiments on it.

### 5.1.1 Ablation Study: Influence of Pruning Schemes on Lottery Ticket Performance

As detailed in Chapter 3, a substantial part of any pruning algorithm other than the scoring mechanism itself is the way it is applied. For example, the algorithm proposed in the original Lottery Ticket Hypothesis, Iterative Magnitude Pruning, uses a well-known and basic scoring criterion but applied it in an iterative pruning schedule during training and additionally applied weight rewind after each pruning iteration. Others, like SNIP, are proposed to be used in a single-shot pruning schedule before training. In this section, we investigate ways to compare the performance of different pruning criteria independently of the way it is applied. We empirically test every combination of pruning criteria and schedule and measure the accuracy of the resulting Lottery Ticket.

We implement the following pruning schedules as described in literature:

- Iterative Single Shot (ISS): The network is trained $k$ times until an early stopping criterion is met, after each of which $\frac{n}{p \cdot k}$ weights are removed, for $n$ is the total number of weights and $p$ is the ratio of weights remaining at the end of the process. After pruning, the weights are reset to the value they were before training. This schedule, in combination with the magnitude scoring, is the basis for the original lottery ticket research (Frankle and Carbin, 2018).
- Upfront Single Shot Pruning (USS): After initialization, before training, the pruning scores are calculated and applied. Both GraSP and SNIP proposed this schedule (Wang et al., 2020; Tanaka et al., 2020).
- Upfront Multi Shot Pruning (UMS): Similar to Single Shot Pruning, the network is scored and pruned before training. However, in a multi shot schedule, there are multiple epochs of scoring and pruning, leading to the desired prune ratio. This schedule was proposed by Han, Mao, et al., 2015 and again in the context of SynFlow (Tanaka et al., 2020).
- Iterative Multi Shot (IMS): Same as ISS, but instead of applying a Single Shot schedule at each pruning stage, pruning as done in an Multi Shot way. This schedule was not used in the original publication of any of the four selected algorithms.

While the recent work we base our experiments on proposed these different schedules, they in fact predate these publications (Han, Mao, et al., 2015). As outlined in chapter 3, other schedules exist (Hoefler et al., 2021). We chose this selection of schedules specifically due to their appearance in the context of the algorithms we are investigating.

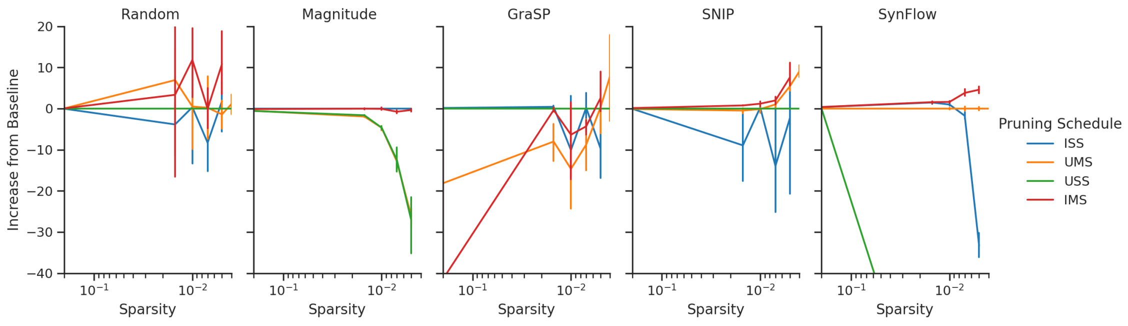
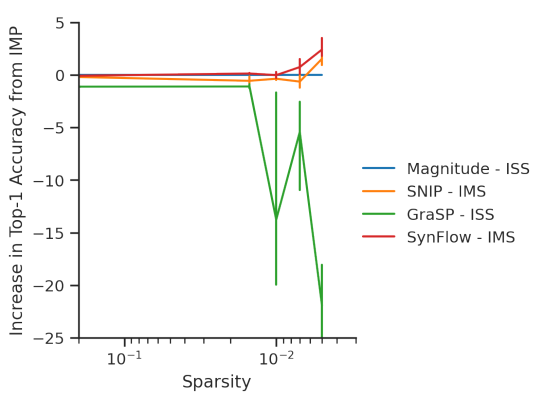


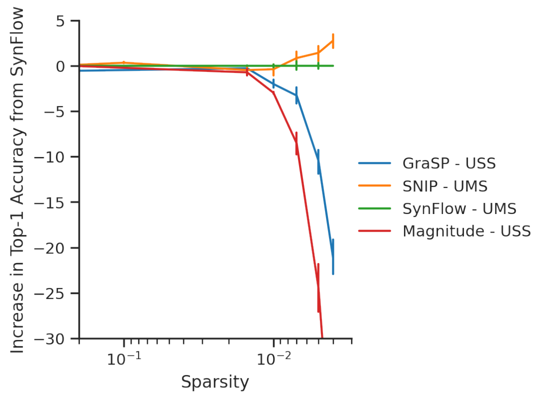
FIGURE 5.4: Change in accuracy for a given algorithm across different pruning schedules, compared to the originally proposed version of the algorithm.

The idea to see if altering the way a pruning algorithm is applied improves performance is not new. Tanaka et al., 2020 apply the same multi-shot approach they apply to their own algorithm to SNIP and found an improvement in performance as well. We are able to confirm this observation and hypothesize that the reason for this be due to more information being available to the algorithm, as with each epoch in the multi-shot pruning regime, data is loaded and gradients calculated. The same is not true for magnitude pruning, where it both empirically and theoretically (LeCun, Denker, et al., 1989) does not make a difference regardless how many pruning epochs are applied, as the magnitude of the weights remains the same, regardless of whether or not other values have been pruned in the meantime. When running SynFlow with less pruning epochs than originally proposed, performance drops significantly, as described by Tanaka et al., 2020.

The performance of GraSP under different pruning schedules seems to improve at lower ratios of weights remaining but is worse at trivial sparsity. The large variation in results across the three repeats, we hypothesize, potentially indicates an issue related to the specific implementation in the way the PyTorch loss is used and later re-used, but did not find a solution to this issue in the time available. From the theory (Wang et al., 2020), we did not find any reason why it should not improve when applied iterative during training. While we always find improvements when switching to an iterative schedule instead of an upfront one, there is no fixed value by which the accuracy of the resulting LT improve.



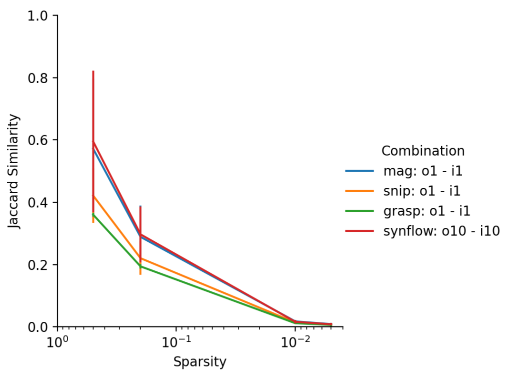
(A) The performance of the different prunign algorithms when applied iteratively compared to IMP.



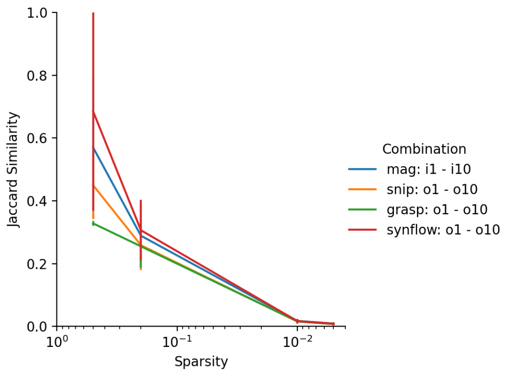
(B) The performance of the different pruning algorithms when applied before training compared to SynFlow.

FIGURE 5.5: Algorithms applied in all schedules compared to the previous best version for a given upfront or iterative schedule.

When comparing the performance of each algorithm on the same pruning schedule, improvements over IMP can be found. Figure 5.5a illustrates the relative improvement of each algorithm over IMP, showing that both SynFlow and SNIP outperform IMP at high values of sparsity. Further, when applying the multi-shot approach described in the context of SynFlow, SNIP outperforms SynFlow at high sparsity but not otherwise. This is a surprising finding given that SynFlow's main focus was to improve performance in this area. More research is required to investigate why this is the case.



(A) Comparison between LTs found by the same algorithm in iterative and before training schedules.



(B) Comparison between LTs found by the same algorithm in one-shot and mutishot-shot schedules.

FIGURE 5.6: Comparison of Jaccard similarity between lottery tickets found by the same algorithm, once applied iteratively and once upfront. At high values of sparsity, a more informed pruning schedule leads to a different Lottery Ticket compared to an upfront version of the same algorithm.

Building on the findings from Paganini and Forde, 2020 detailed in chapter 3, we investigate whether the core set of weights present in all of the algorithms when pruning with the original schedule remains when changing the pruning schedule. As visualized in figure 5.7, we find this not to be the case. Both switching from an upfront to an iterative schedule as well as applying multi-shot instead of single-shot way creates highly distinct Lottery Tickets at high sparsity. We interpret this finding to signify that the set of weights selected by the algorithm is

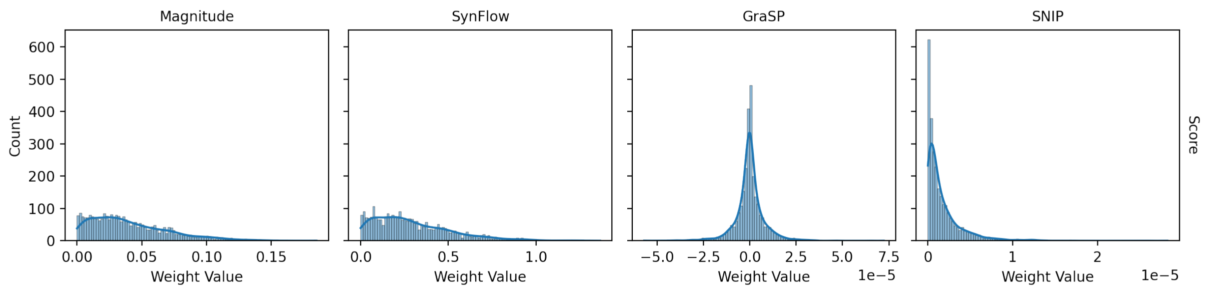
highly sensitive to the way it is applied.
Also, we conclude from these results, that the core subset of weights found by all of the algorithms may not be as important as thought. When comparing upfront LTs with the corresponding iterative LT, no weights are shared, thus the original core subset of weights is lost. Yet, there is a performance increase.
Therefore, when searching for higher performing Lottery Tickets, we will not focus on the core LT or try to optimize for it. We suspect that the overlap between the Lottery Tickets might stem from the fact, that all algorithms are reliant on the magnitude criterion to a certain degree as shown in figure 5.7a. To validate this assumption, more research is required.

## 5.2 Meta Learning Scores

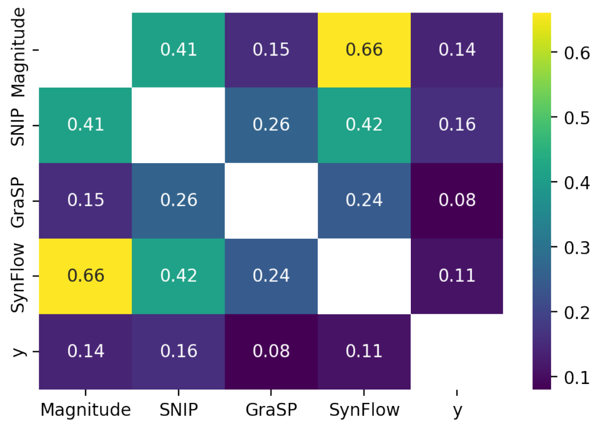
In this section, we describe the results of the meta learning approach. Firstly, we give an overview on how the data used was obtained and transformed. Then, we compare the training metrics of the selected models on our data. Lastly, we use each model to prune a LeNet_300_100 to values of sparsity up to 0.5% and measure its Top-1 Accuracy on MNIST and describe the results.

### 5.2.1 Data Preparation

As shown in the previous section, of all algorithms, the best Lottery Ticket can be found with SynFlow with an iterative multi-shot (IMS) schedule. Therefore, we are using this Lottery Ticket as our target variable. Training data comes from the scores of the four state-of-the-art algorithms, each calculated on the exact same weight initialization as the target variable. We use three different runs with all five pruning strategies necessary to a sparsity of 0.5% applied. We decided to prune to this ratio of weights remaining for our training data, as we observe the largest differences in this sparse environment. In total, we have 798'600 individual data points.



(A) Histogram of the scores of the algorithms



(B) Correlation between the features and the target.

FIGURE 5.7: The data used to train the model

As shown in figure 5.7a, the different algorithms do not produce scores in the same range, therefore we scale each feature to a range of $[0, 1]$. Of this data, we found no single feature to correlate well with the target variable, as shown in figure 5.7b.

A promising idea is to rank all of the values for a feature and use this rank instead of the value of the score output, as this scales the scores part of a Lottery Ticket at a given sparsity into to the same range, meaning all weights that should be part of the LT at the same sparsity have scores in the same range.

However, in empirical tests we could not confirm any benefit of this method, leading us to hypothesize, that it is more important for the model whether or not a score is close to zero instead of the exact position in the ranking.

In appendix B, we explore options to improve the performance of the model by providing it with more features to learn on, but are not able to demonstrate an improvement by this approach.

### 5.2.2 Training

We train multiple models on the data described above and evaluate their performance on the test dataset. In this chapter, we describe which models we selected to implement and how well they were able to learn on our dataset.

We compare the performance of the different models with both the mean squared error metric (Das et al., 2004) as well as the $R^2$ (Draper and Smith, 1998) score. Both of these are well established metrics in the domain of linear regression (Redell, 2019) and simple to calculate.

#### Linear Regression

To evaluate if there is a simple linear relationship between the four scores and the approximation of the ideal scores to produce a lottery ticket, we implement two linear regression (Kuchibhotla et al., 2019) models. The first one is a plain least squares regression (Kuchibhotla et al., 2019), while the second is based on stochastic gradient descent or SGD (Robbins and Monro, 1951). The selected models are standard options for regression problems (Kuchibhotla et al., 2019). For both, we use implementations available in Scikit-learn Pedregosa et al., 2011 in the form of the LinearRegressor and SGDRegressor classes. All Hyperparameters were left at the documented default.

| Model | $R^2$ (train) | $R^2$ (test) | MSE (train) | MSE (test) |
|---|---|---|---|---|
| LinearRegression | 0.033137 | 0.032998 | 0.000053 | 0.000057 |
| SGDRegressor | 0.022484 | 0.021587 | 0.000054 | 0.000057 |

TABLE 5.1: Training metrics of the linear regression models.

For both models, table 5.1 contains the evaluation metrics $R^2$ and Mean Squared Error (MSE). As for $R^2$, a value closer to 1 implies that the model has predicted everything correctly (Draper and Smith, 1998). With scores below 0.05, both models have failed to learn meaningful representations of the data. In regards to mean squared error, both models are of similar performance. While it is close to zero, the absolute value in it self does not convey much information. We will later use it to compare the different models.

#### Tree based Regression

We select two models as examples of regression based on trees and evaluate, whether this class of regression models is suitable for our task. Specifically, we select both a Decision Tree (Quinlan, 1986) as well as a Random Forest (Maclin and Opitz, 1999).

| Model | $R^2$ (train) | $R^2$ (test) | MSE (train) | MSE (test) |
|---|---|---|---|---|
| DecisionTreeRegressor | 1.000000 | -0.865044 | 1.074603e-19 | 0.000109 |
| RandomForestRegressor | 0.859253 | 0.010510 | 7.765114e-06 | 0.000058 |

TABLE 5.2: Training metrics of tree based models.

As can be seen in table 5.2, the Decision Tree achieved an $R^2$ score of 1.0, while on the test set, the same metric is negative. In combination with the similar pattern for MSE, we interpret the model to overfit on the data. For the Random Forest, it is a slightly better picture but still a very high $R^2$ on train while it being low on test, with MSE indicating the same overfitting behaviour. The RandomForestRegressor with its default parameters trained on the data described takes 2GB to store, which by far is the largest model of the ones tested and has led to challenges when applying the model in our parallelized setup.

### Multi Layer Perceptron

In addition to the basic models from Scikit-Learn, we implement a fully connected Multi Layer Perceptron (Hornik et al., 1989) in PyTorch and train it on the data described above. Based on observations described in Lecun et al., 1998, we choose a simple architecture with two hidden layers, one of size 64, the latter of size 32. The input layer corresponds to the four features available and as an output, a single neuron is configured. Between the layers, we chose a basic ReLU activation function (Agarap, 2018) with a Sigmoid activation (Nwankpa et al., 2018 bringing the values in the desired range at the very end. For the loss function, a Mean Squared Error loss is used, as we use this metric to evaluate the performance later. Again keeping with well established options, we use an SGD optimizer with a learning rate of 0.001.

With the limitations in place in the context of this thesis, the Hyperparameters of this network were not systematically optimized as could be done with mechanisms described in the literature (T. Yu and Zhu, 2020). Instead, we used what we recognize as default parameters based on the documentation ("PyTorch documentation", n.d.).

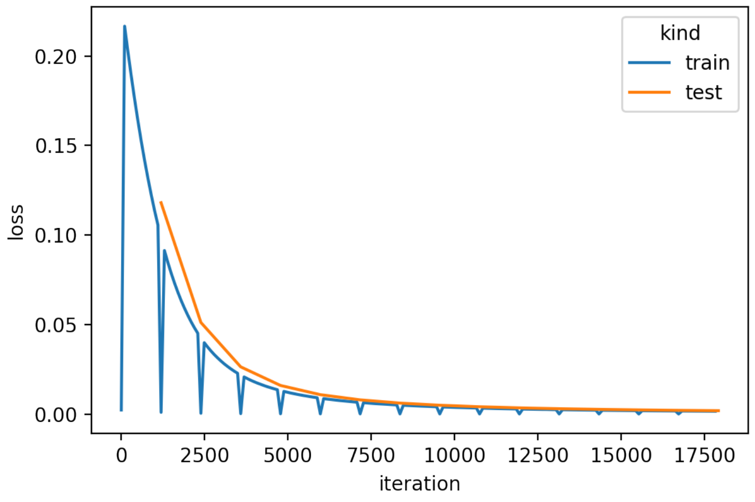


FIGURE 5.8: Learning curve of the Multi Layer Perceptron implemented in PyTorch

While training loss of the MLP model decreases over the course of the first iterations, it fails to achieve a loss as low as the tree based or even linear models.

| Model | $R^2$ (train) | $R^2$ (test) | MSE (train) | MSE (test) |
|---|---|---|---|---|
| MLP | -42.310778 | -45.687378 | 0.002472 | 0.002467 |

TABLE 5.3: Training results of the MLP implemented in Pytorch.

Despite being a comparatively complex model, the MLP as it is implemented has the lowest $R^2$ and the highest mean squared error of all the models tried. We suspect that this might be an indication that the information to successfully learn to predict Lottery Tickets based on LTs found iteratively might not be present in the data.

### 5.2.3 Evaluation

Overall, Lottery Tickets obtained by all models achieve a Top-1 Accuracy higher than those from random pruning but fail to outperform any of the preexisting algorithms. However, of all the models implemented, the MLP is one of the highest performing models of the selection. This is unexpected considering the metrics obtained after training, which were orders of magnitude better for other models when training and testing on the dataset used. In the context of these results, the training metrics observed for the MLP may not signify bad performance but good generalization. Also relatively well performing is the linear model based on SGD, implying the presence of linear relationships in some form. However, we found the results of the SGD model not to be statistically significant compared to the magnitude criterion we use as a baseline in figure 5.9

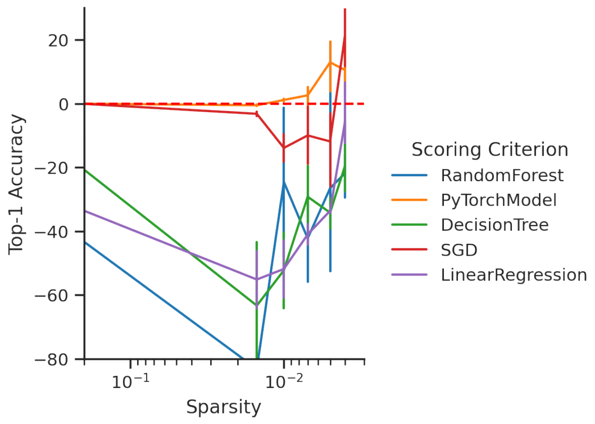


FIGURE 5.9: Top-1 Accruary of Lottery Tickets obtained by different models relative the performance of a Lottery Ticket obtained by the magnitude criterion. While both SGD and the PyTorchModel on average outperform the baseline of magnitude pruning, only for the PyTorchModel we are able to measure statistical significance.

One of the key areas to further improve our approach we suspect is the fact that all models only consider the context of the scores for a single weight at the time. While each score on its own tries to produce a connected network, our model chooses weights from all of these different, in itself connected, sub network. This, we suspect, could be limiting the effectiveness of a model trained in this way. As discussed in chapter 3, alternative approaches exist. If a model would be trained on full networks instead, a mechanism to optimize the pruning scores to produce connected networks could be introduced, mitigating the suspected issue. More research comparing a metric of connectedness of the different LTs is required to validate this hypothesis.

Further, we raise the question whether training a model to produce scores based on other, separate scores is a promising idea to reach new levels of performance in pruning in the first place. With our setup, the model itself is not aware what the desired sparsity of the Lottery Ticket will be. Instead, it produces scores which are then selected by their magnitude by an operation after the prediction itself. However, we question the assumption, that the weights selected to be part of the LT at 0.5% sparsity are also part of the best LT at 5%, as we hypothesize that the importance of substructures could be dependent on the number of weights allowed in the final result. From the state-of-the-art algorithms we selected in the context of this thesis, none is sparsity-aware. We suspect, that interesting questions could be investigated in this area in the future.

Overall, more research is required to validate whether our findings and assumptions above generally hold.

## 5.3 Stacked Scoring

To assess the performance of the lottery tickets obtained by a stacked scoring algorithm, we implement all possible combinations of SNIP, SynFlow, GraSP and Magnitude pruning. We then apply them to find LTs on the model and task selected in chapter 4. The LTs found are evaluated on the performance and the Top-1 accuracy is measured. To contextualize the algorithm, we compare the results to the unpruned network, the best approximation of the ideal LT found by iterative, 10x multi-shot SNIP and to the individual algorithms used in a given stack.

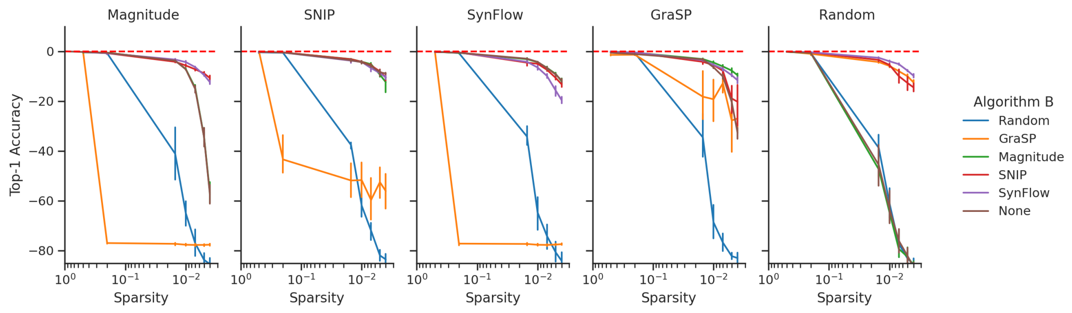
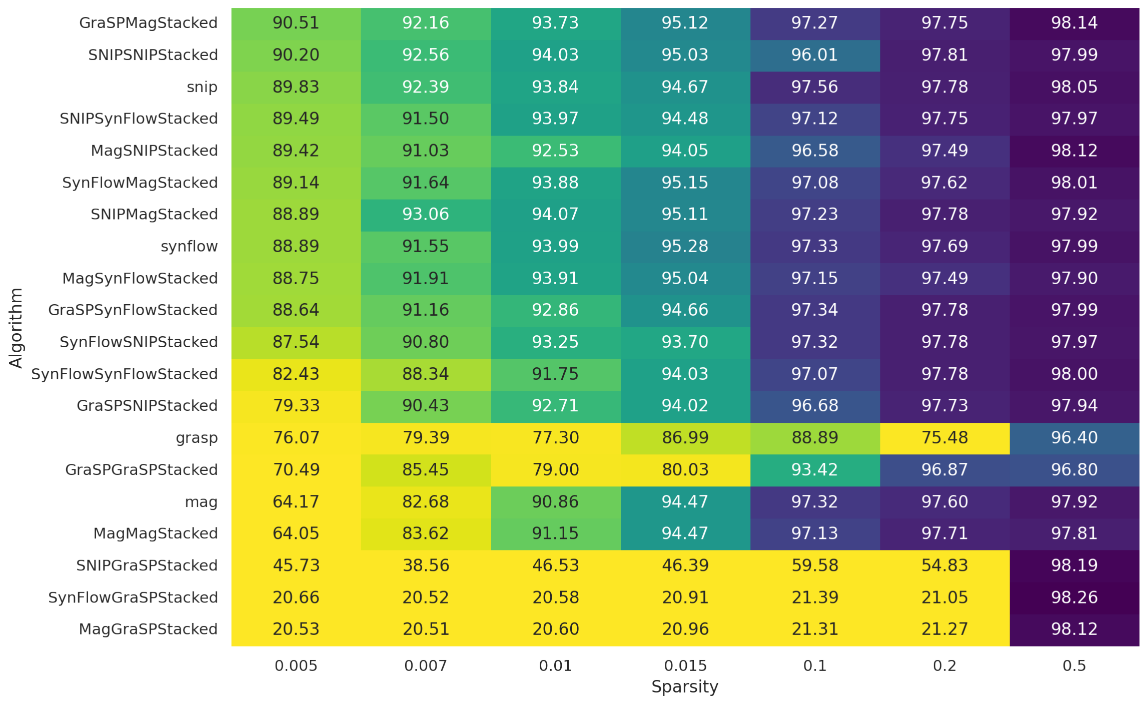


FIGURE 5.10: Stacked pruning performance relative to the unpruned network. Stacks sharing the same algorithm A are visualized in the same diagram.

As visualized in figure 5.10, both LTs from stacks based on SNIP and SynFlow demonstrate performance comparable with the individual baselines. As shown in figure 5.10, multiple stacks are capable of finding trainable sub networks.



| Algorithm | 0.005 | 0.007 | 0.01 | 0.015 | 0.1 | 0.2 | 0.5 |
|---|---|---|---|---|---|---|---|
| GraSPMagStacked | 90.51 | 92.16 | 93.73 | 95.12 | 97.27 | 97.75 | 98.14 |
| SNIPSNIPStacked | 90.20 | 92.56 | 94.03 | 95.03 | 96.01 | 97.81 | 97.99 |
| snip | 89.83 | 92.39 | 93.84 | 94.67 | 97.56 | 97.78 | 98.05 |
| SNIPSynFlowStacked | 89.49 | 91.50 | 93.97 | 94.48 | 97.12 | 97.75 | 97.97 |
| MagSNIPStacked | 89.42 | 91.03 | 92.53 | 94.05 | 96.58 | 97.49 | 98.12 |
| SynFlowMagStacked | 89.14 | 91.64 | 93.88 | 95.15 | 97.08 | 97.62 | 98.01 |
| SNIPMagStacked | 88.89 | 93.06 | 94.07 | 95.11 | 97.23 | 97.78 | 97.92 |
| synflow | 88.89 | 91.55 | 93.99 | 95.28 | 97.33 | 97.69 | 97.99 |
| MagSynFlowStacked | 88.75 | 91.91 | 93.91 | 95.04 | 97.15 | 97.49 | 97.90 |
| GraSPSynFlowStacked | 88.64 | 91.16 | 92.86 | 94.66 | 97.34 | 97.78 | 97.99 |
| SynFlowSNIPStacked | 87.54 | 90.80 | 93.25 | 93.70 | 97.32 | 97.78 | 97.97 |
| SynFlowSynFlowStacked | 82.43 | 88.34 | 91.75 | 94.03 | 97.07 | 97.78 | 98.00 |
| GraSPSNIPStacked | 79.33 | 90.43 | 92.71 | 94.02 | 96.68 | 97.73 | 97.94 |
| grasp | 76.07 | 79.39 | 77.30 | 86.99 | 88.89 | 75.48 | 96.40 |
| GraSPGraSPStacked | 70.49 | 85.45 | 79.00 | 80.03 | 93.42 | 96.87 | 96.80 |
| mag | 64.17 | 82.68 | 90.86 | 94.47 | 97.32 | 97.60 | 97.92 |
| MagMagStacked | 64.05 | 83.62 | 91.15 | 94.47 | 97.13 | 97.71 | 97.81 |
| SNIPGraSPStacked | 45.73 | 38.56 | 46.53 | 46.39 | 59.58 | 54.83 | 98.19 |
| SynFlowGraSPStacked | 20.66 | 20.52 | 20.58 | 20.91 | 21.39 | 21.05 | 98.26 |
| MagGraSPStacked | 20.53 | 20.51 | 20.60 | 20.96 | 21.31 | 21.27 | 98.12 |

FIGURE 5.11: Absolute Top-1 Accuracy for the top 20 best performing algorithms, sorted by their performance at 0.5% sparsity. For the two Algorithms outperforming SNIP, not statistical significance is observed compared to SNIP.

For the combination of GraSP in phase A and Magnitude pruning in phase B and by using SNIP as both A and B phase of the stack, we observe higher accuracy compared to the next best baseline. To inspect these results further, we conducted P-Tests for the two best performing algorithms and the baseline SNIP results, finding no statistical significance apart from SNIP/S-NIP at the trivial sparsity of 20%. Therefore, we find that no stacked algorithm outperforms all baselines reliably on our model and task.

Other promising combinations include SNIP/SynFlow, Magnitude/SNIP, SynFlow/Magnitude and SNIP/Magnitude, all of which are able to outperfom the Lottery Ticket found by the vanilla SynFlow algorithm on average, but not to a statistically significant degree over three runs.

All of these stacks mentioned so far besides SNIP/SNIP and SNIP/SynFlow have Magnitude pruning as either the first or second phase algorithm. For SynFlow, SNIP and Magnitude pruning itself, applying the magnitude criterion in the second phase is expected to not influence the performance, as the scores produced are all positive. Indeed, neither the results of SynFlow/Magnitude nor SNIP/Magnitude are statistically significant at any level of sparsity. The same does not apply to GraSP/Magnitude. As visualized on figure 5.7a, we observe GraSP to produce positive as well as negative scores.
As shown in table 5.11, our work on stacked pruning reveals that taking the absolute value of the score found by GraSP improves the Top-1 Accuracy of the LT at high values of sparsity drastically, from 77% in its original form, to 90.5% at 0.5% sparsity.
With such high improvement with such small adjustment and in the light of previously mentioned suspicious observations mentioned in the baseline comparison as well as the ablation study earlier in this thesis, we raise concerns that this might be due to an implementation error in the code base we used. Further research is required to confirm that this result holds true for alternative implementations of GraSP.

In summary, we conclude that it is possible to sequentially apply two pruning algorithms and get a trainable Lottery Ticket with Score Stacking. From the fact that pruning scores themselves can successfully be the basis of another scoring criterion reveals interesting attributes about the scores and the similarities between the initial weights and the scores, as for the second phase, the weights can be substituted by the scores while retaining the ability to produce Lottery Tickets.

## 5.4 SaW: Scores as weights

Building on the findings of stacked scoring, we implement a mechanism where the intermediate scores are kept as weights and not reset to their original value. Again, we implement all combinations of the state-of-the-art pruning algorithm selected and evaluate the Top-1 Accuracy of the LTs produced on our model and task. Based on the observation made earlier in this chapter that the most informed algorithms tend to select weights from a bi-modal distribution, also run an experiment where we initialize the network in such distribution. As most algorithms only produce positive scores, we transfer the sign of the original weight to the score to achieve a balance between positive and negative weights. Also, we scale all scores to fall in the range $[-1, 1]$.

| Initialisation Strategy | Average Top-1 Accuracy | Average Training Iterations |
|---|---|---|
| Standard (Kaiming Normal) | 98.235000 | 12819.333333 |
| SNIP | 97.863333 | 12037.666667 |
| Bi-Modal | 97.810000 | 12506.666667 |
| SynFlow | 96.596667 | 10787.000000 |
| GraSP | 80.833333 | 22512.000000 |

TABLE 5.4: Results of a fully connected Lenet_300_100 on MNIST using pruning scores obtained on the kaiming initialized weights as weight initialisation to begin training with.

With these results, we show that the scoring criterion can also successfully be used as a weight initialization and train to a high accuracy. From the histogram of the scores presented in figure 5.7a, there is a qualitative similarity between the scores from SynFlow and the magnitude of the weights, as expressed by the scores of the magnitude pruning criterion. Quantitatively, there is a measurable correlation between scores from SynFlow and the weight magnitude, as shown figure 5.7b. Therefore, it comes as no surprise that the SynFlow scores work as initial values as well. However, SNIP scores perform even better, despite the difference in the distribution of the values as well as the lower but still present correlation effect shown in figure 5.7a. Of all the initializations observed, scores obtained by GraSP result in the least performant network with an over 15% difference to the next best initialisation. GraSP is also less correlated to the weight magnitude.

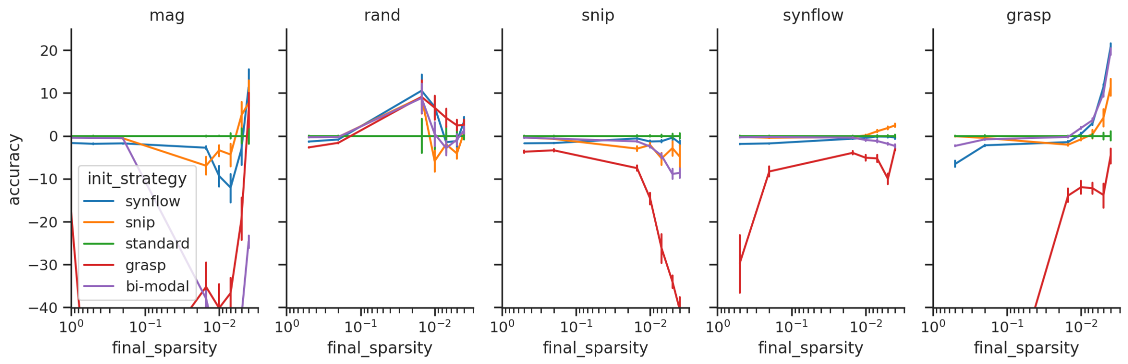
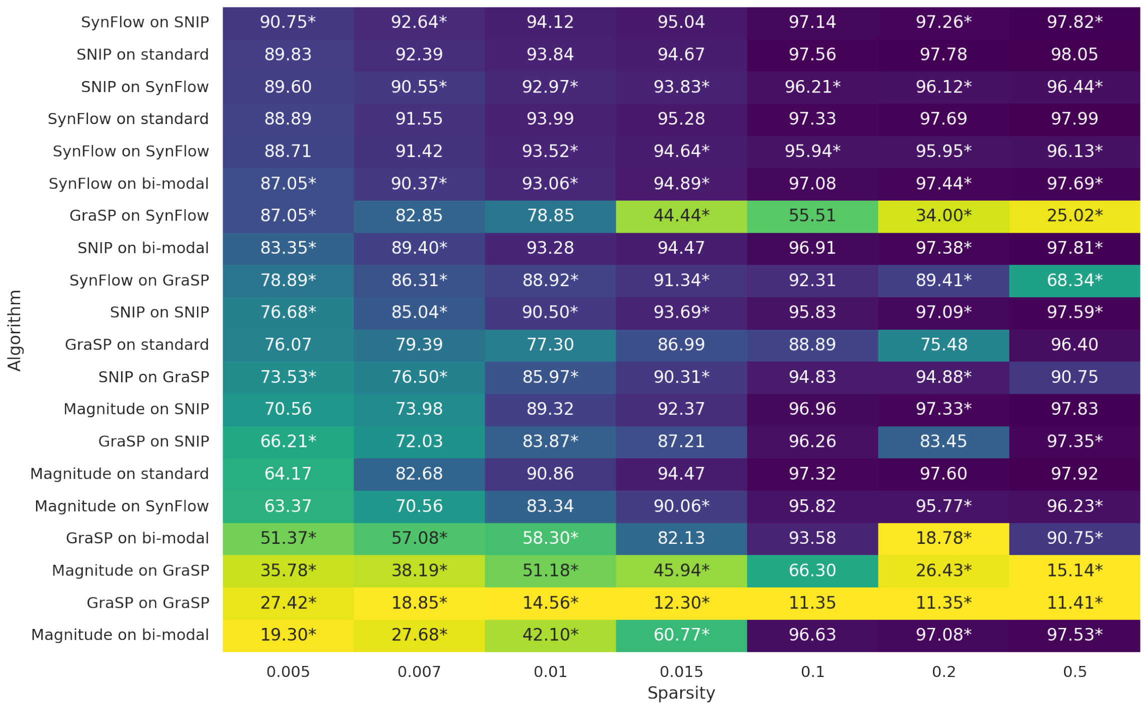


FIGURE 5.12: Performance of Lottery Tickets obtained by different pruning algorithms on the different initial weights relative to the performance on weights initialized with kaiming.

When looking at the performance of the Lottery Tickets after pruning across different initialisation schemes, we observe large performance improvements at high sparsity. While GraSP has performed worse than SNIP and SynFlow at high values of sparsity, it benefits the most from an alternative initialisation scheme. Both SynFlow and SNIP scores increase the performance of the Lottery Ticket drastically when used as initial values when pruning with GraSP. Interestingly, the random values from a bi-modal distribution also perform well, leaving us to hypothesize that the performance increase is not due to more information being available but related to general dynamics present with different initialisation schemes. Also a significant impact was observed at high values of sparsity when pruning with magnitude pruning. However, while the improvement compared to magnitude pruning on standard initializations is large, it is still well below the Top-1 accuracy LTs produced by both SNIP and SynFlow.



| Algorithm \ Sparsity | 0.005 | 0.007 | 0.01 | 0.015 | 0.1 | 0.2 | 0.5 |
|---|---|---|---|---|---|---|---|
| SynFlow on SNIP | 90.75* | 92.64* | 94.12 | 95.04 | 97.14 | 97.26* | 97.82* |
| SNIP on standard | 89.83 | 92.39 | 93.84 | 94.67 | 97.56 | 97.78 | 98.05 |
| SNIP on SynFlow | 89.60 | 90.55* | 92.97* | 93.83* | 96.21* | 96.12* | 96.44* |
| SynFlow on standard | 88.89 | 91.55 | 93.99 | 95.28 | 97.33 | 97.69 | 97.99 |
| SynFlow on SynFlow | 88.71 | 91.42 | 93.52* | 94.64* | 95.94* | 95.95* | 96.13* |
| SynFlow on bi-modal | 87.05* | 90.37* | 93.06* | 94.89* | 97.08 | 97.44* | 97.69* |
| GraSP on SynFlow | 87.05* | 82.85 | 78.85 | 44.44* | 55.51 | 34.00* | 25.02* |
| SNIP on bi-modal | 83.35* | 89.40* | 93.28 | 94.47 | 96.91 | 97.38* | 97.81* |
| SynFlow on GraSP | 78.89* | 86.31* | 88.92* | 91.34* | 92.31 | 89.41* | 68.34* |
| SNIP on SNIP | 76.68* | 85.04* | 90.50* | 93.69* | 95.83 | 97.09* | 97.59* |
| GraSP on standard | 76.07 | 79.39 | 77.30 | 86.99 | 88.89 | 75.48 | 96.40 |
| SNIP on GraSP | 73.53* | 76.50* | 85.97* | 90.31* | 94.83 | 94.88* | 90.75 |
| Magnitude on SNIP | 70.56 | 73.98 | 89.32 | 92.37 | 96.96 | 97.33* | 97.83 |
| GraSP on SNIP | 66.21* | 72.03 | 83.87* | 87.21 | 96.26 | 83.45 | 97.35* |
| Magnitude on standard | 64.17 | 82.68 | 90.86 | 94.47 | 97.32 | 97.60 | 97.92 |
| Magnitude on SynFlow | 63.37 | 70.56 | 83.34 | 90.06* | 95.82 | 95.77* | 96.23* |
| GraSP on bi-modal | 51.37* | 57.08* | 58.30* | 82.13 | 93.58 | 18.78* | 90.75* |
| Magnitude on GraSP | 35.78* | 38.19* | 51.18* | 45.94* | 66.30 | 26.43* | 15.14* |
| GraSP on GraSP | 27.42* | 18.85* | 14.56* | 12.30* | 11.35 | 11.35* | 11.41* |
| Magnitude on bi-modal | 19.30* | 27.68* | 42.10* | 60.77* | 96.63 | 97.08* | 97.53* |

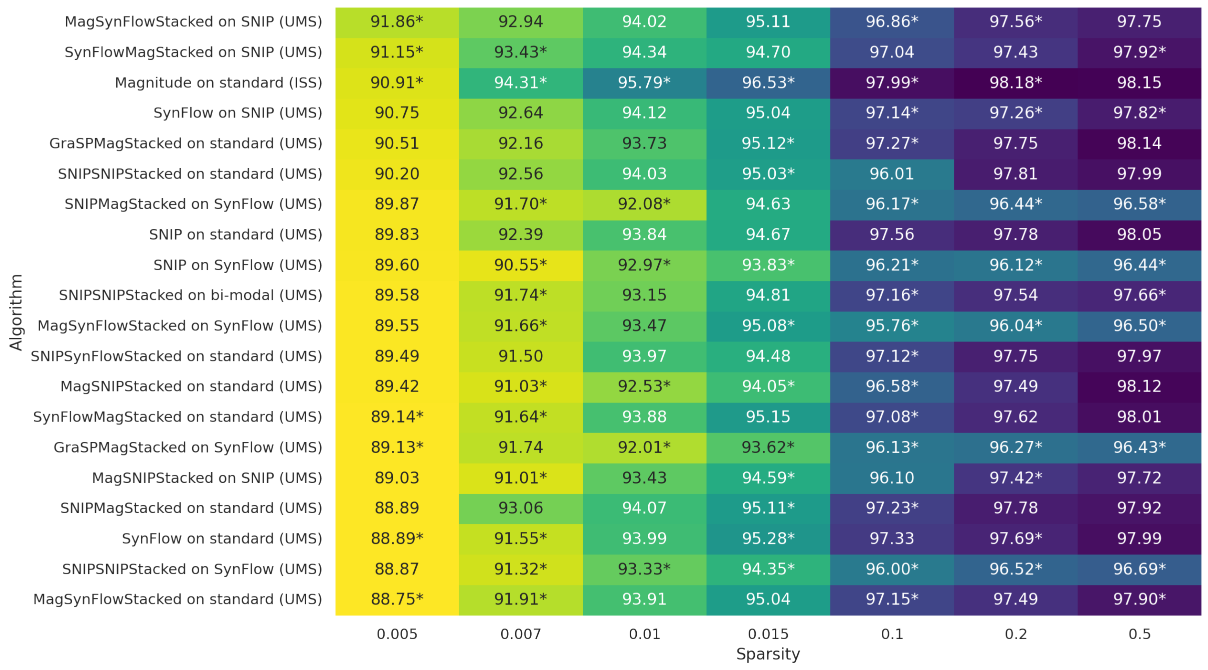
FIGURE 5.13: Top 20 best performing baselines on different Initialization Strategies. On weights initialized with SNIP scores, SynFlow outperforms itself compared to standard weights. (* denotes a statistically significant difference compared to the same algorithm on standard initializations at the same sparsity across three runs).

While for SNIP, we could not measure any improvement on any initialization, we observe an improvement to the LT obtained with SynFlow when initialized with SNIP scores. Figure 5.13 shows the absolute Top-1 Accuracy for the baseline algorithms applied on weights initialized with different scores. While the improvement from SynFlow on SNIP compared to SynFlow on standard initializations is statistically relevant, the same is not the case when comparing it to the baseline SNIP on standard inits. Therefore, with this experiment, the SNIP baseline continues to produce the best performing Lottery Tickets at low sparsity.

With this result, we show that weight initializations can have a large effect on the performance of a given pruning algorithm and are thus able to demonstrate, that it is possible to produce Lottery Tickets by combining two pruning algorithms in this way.

### 5.4.1 Combining all approaches

As a last experiment, we run the meta models trained as well as the stacked variants on all different initialization strategies and compare the results.



| Algorithm | 0.005 | 0.007 | 0.01 | 0.015 | 0.1 | 0.2 | 0.5 |
|---|---|---|---|---|---|---|---|
| MagSynFlowStacked on SNIP (UMS) | 91.86* | 92.94 | 94.02 | 95.11 | 96.86* | 97.56* | 97.75 |
| SynFlowMagStacked on SNIP (UMS) | 91.15* | 93.43* | 94.34 | 94.70 | 97.04 | 97.43 | 97.92* |
| Magnitude on standard (ISS) | 90.91* | 94.31* | 95.79* | 96.53* | 97.99* | 98.18* | 98.15 |
| SynFlow on SNIP (UMS) | 90.75 | 92.64 | 94.12 | 95.04 | 97.14* | 97.26* | 97.82* |
| GraSPMagStacked on standard (UMS) | 90.51 | 92.16 | 93.73 | 95.12* | 97.27* | 97.75 | 98.14 |
| SNIPSNIPStacked on standard (UMS) | 90.20 | 92.56 | 94.03 | 95.03* | 96.01 | 97.81 | 97.99 |
| SNIPMagStacked on SynFlow (UMS) | 89.87 | 91.70* | 92.08* | 94.63 | 96.17* | 96.44* | 96.58* |
| SNIP on standard (UMS) | 89.83 | 92.39 | 93.84 | 94.67 | 97.56 | 97.78 | 98.05 |
| SNIP on SynFlow (UMS) | 89.60 | 90.55* | 92.97* | 93.83* | 96.21* | 96.12* | 96.44* |
| SNIPSNIPStacked on bi-modal (UMS) | 89.58 | 91.74* | 93.15 | 94.81 | 97.16* | 97.54 | 97.66* |
| MagSynFlowStacked on SynFlow (UMS) | 89.55 | 91.66* | 93.47 | 95.08* | 95.76* | 96.04* | 96.50* |
| SNIPSynFlowStacked on standard (UMS) | 89.49 | 91.50 | 93.97 | 94.48 | 97.12* | 97.75 | 97.97 |
| MagSNIPStacked on standard (UMS) | 89.42 | 91.03* | 92.53* | 94.05* | 96.58* | 97.49 | 98.12 |
| SynFlowMagStacked on standard (UMS) | 89.14* | 91.64* | 93.88 | 95.15 | 97.08* | 97.62 | 98.01 |
| GraSPMagStacked on SynFlow (UMS) | 89.13* | 91.74 | 92.01* | 93.62* | 96.13* | 96.27* | 96.43* |
| MagSNIPStacked on SNIP (UMS) | 89.03 | 91.01* | 93.43 | 94.59* | 96.10 | 97.42* | 97.72 |
| SNIPMagStacked on standard (UMS) | 88.89 | 93.06 | 94.07 | 95.11* | 97.23* | 97.78 | 97.92 |
| SynFlow on standard (UMS) | 88.89* | 91.55* | 93.99 | 95.28* | 97.33 | 97.69* | 97.99 |
| SNIPSNIPStacked on SynFlow (UMS) | 88.87 | 91.32* | 93.33* | 94.35* | 96.00* | 96.52* | 96.69* |
| MagSynFlowStacked on standard (UMS) | 88.75* | 91.91* | 93.91 | 95.04 | 97.15* | 97.49 | 97.90* |

FIGURE 5.14: Comparison of the top 20 algorithm and initialisation combinations across all algorithms described. "Magnitude on standard (ISS)" is IMP from the original Lottery Ticket Hypothesis. A * denotes a statistically significant difference compared to SNIP on weights initialized with a standard procedure on the same sparsity.

As shown in figure 5.14, we are able to outperform the baseline SNIP algorithm when using both stacked variants of SynFlow and Magnitude scoring on SNIP inits. For the non-trivial sparsity of 0.5%, the results are statistically significant over three runs. Further figure 5.14 also shows performance of the original Lottery Ticket Hypothesis algorithm (Magnitude on standard (ISS)), where the Lottery Ticket is found by training the network repeatedly. As shown, our best algorithms even outperform this Lottery Ticket on average at 0.5% of weights remaining, despite pruning the network before initialisation instead of during training, but the result is not statistically significant in this regard.

Yet, these results close the gap in performance between LTs obtained before training and LTs obtained with training. With this experiment, we demonstrate that it is possible to obtain a Lottery Ticket of equal performance to one found with traditional IMP, without the need for any training.

However, despite these encouraging findings, we only observe statistical significance for outperforming baseline SNIP at 0.5%. When testing a Magnitude/SynFlow Stack on SNIP for 0.4%, we still observe a better performance on average, but it is no longer statistically significant. We observe the same when comparing our results with IMP, additionally, IMP always outperforms our best algorithm when more than 0.5% of weights remain.
With the finding being this sensitive to the specific sparsity, we suspect that running more runs is required to validate our results.

# Chapter 6

# Conclusion

## 6.1 Summary

In this thesis, we have raised the question whether several pruning algorithms can be combined to obtain higher performing Lottery Tickets. We proposed three distinct ways of combining algorithms: A meta learning model, Score Stacking and Scores as Weight initialisation (SaW). For combinations of the four state-of-the-art pruning algorithms SynFlow, SNIP, GraSP and IMP we selected, we implemented the three different approaches in a common environment on a single model and task, building on an existing code base created by the team behind SynFlow.

By comparing all algorithms on the same model and task, we established baseline performance measures on the Top-1 Accuracy of the Lottery Ticket produced at different sparsities. We show that not every algorithm works as well on our model as outlined in the original publication, especially GraSP falls short of its promised performance. To isolate the scoring criterion from the schedule it is applied, we run an ablation study comparing every scoring criterion on every pruning schedule described in the context of the four algorithms selected.

We show that the best baseline on a before-training schedule, SynFlow, can be outperformed by a SNIP when applied in a multi-shot way. Further, we are able to demonstrate, that the original algorithm to find Lottery Tickets can be outperformed when using the iterative schedule from IMP and the scoring criterion from SynFlow. LTs found with this algorithm outperform the accuracy of the baseline by 2.45 percent.

Therefore, we use iterative SynFlow as our closest approximation to the ideal Lottery Ticket to train a meta model with the task to predict pruning scores based on scores provided by the four state-of-the-art algorithms. We show that learning to predict scores to produce sparse trainable sub networks is possible and demonstrate a simple Multi Layer Perceptron which is able to produce Lottery Tickets which outperform those found by the basic Magnitude criterion.

With Stacked Scoring, we propose a novel way of applying different algorithms sequentially and demonstrate its ability to produce Lottery Tickets even at high sparsity, outperforming our proposed meta model. Lastly, we propose a novel way of using the scores obtained from the pruning criteria as weight initialisations and measure the performance of a network initialized in such way both unpruned and under pruning with different algorithms. We show that the performance of GraSP can be much improved this way. Further, with stacked Mag/SynFlow on SNIP initialisations, we demonstrate an upfront pruning scheme capable of outperforming all baselines at 0.5% of weights remaining by 2.03% as well as equalling the performance of LTs found by IMP during training.

From these results we conclude that combinations of algorithms can indeed be used to find better performing Lottery Tickets before training. To conclude on our starting hypothesis, we therefore confirm that it is indeed possible to use combinations of pruning algorithms to find better performing lottery tickets.

Overall, by investigating ideas that have not been tried before, we contribute to the state-of-the-art and provide a starting point to base further research off.

## 6.2 Limitations

However, as per the limitations outlined in chapter 3, the general applicability of our results have to be verified. A major consideration is the fact, that we only used a single model and task to base our research on. While this restriction was necessary for us, it is advisable to validate our results on larger models and more complex tasks, especially since with previous publications in the field of pruning, promising algorithms on small architectures did not scale to larger ones (Frankle et al., 2020).
Further, for many findings, the statistical relevance criterion was not met at all sparsities. This instability leads us to believe that further research conducting more repeats is necessary to confirm any of our findings.

Practially, one has to consider whether a highly sparse Lottery Ticket is the right choice for in real-world applications. Even in the case of the algorithms we propose, training a lottery ticket instead of the unpruned network does come at the cost of a drop in accuracy. When a large model is needed, but the cost of training it is too high, other options like using a particularly parameter efficient model like EfficientNet (Tan and Le, 2019) can lead to a higher performing model at lower cost.

Still, from a scientific point of view, our results show, that the limits of upfront pruning are not yet reached by the existing algorithms.

## 6.3 Future Work

In the context of this thesis, several open questions remain.

With our experiments around meta learning models, further research is required to investigate, how much better the network can get at predicting scores when tuning the architecture, hyper-parameters and increasing the size of the data. Further, it might be worthwhile to investigate ways to implement a model learning on the level of layers or full networks. For layers, it could be interesting to find patterns in the weight matrices, for example with techniques known from image processing. On the full network layer, it remains an open question whether Hypernetworks can be adapted for this task.
Further, we also propose to train a model with more features than just the outputs of the existing algorithms. For example, information about the layer a given weigh is in, such as matrix norms, row and column wise averages could help to build a better performing model.

To understand more about what a leads to a well performing Lottery Ticket, a measurement of connectedness of the Lottery Ticket would be interesting to analyze. If such metrics are found that correlate well with the final performance of the Lottery Ticket it would be interesting to build a model which optimizes for that.

So far in this thesis, we have not analyzed the number of iterations needed until the early stopping criterion is reached. This idea however is integral to the Lottery Ticket Hypothesis in its original form. Therefore, we suspect it might be worthwhile to analyze the impact our proposed algorithms have on this aspect.

With score stacking, follow up research is needed to validate our findings. It could also be interesting to merge the concept of meta learning the scores and stacking, i.e either train on stacked versions of the algorithms, or stack another scoring algorithm on the output of the meta learner. In this way, even more information would be considered for the scoring mechanism. The research around using scores as initial weights opens the door to relate findings from weight initialization to the pruning scores. Any method by which the initial weights can be adapted may be also tried in conjunction with the pruning scores.

Overall, with our demonstration that LTs can be found before training which perform just as well as those found with training raises interesting questions on the influence of the task on the selection of the Lottery Ticket or whether highly peformant Lottery Tickets exhibit patterns that are generally applicable. With our algorithm, a dataset of such Tickets can be built up much more efficiently, as no training is involved, enabling future research to use this strategy.

# Bibliography

Agarap, A. F. (2018). Deep Learning using Rectified Linear Units (ReLU). http://arxiv.org/abs/1803.08375

Ali, J., Khan, R., Ahmad, N., & Maqsood, I. (2012). *Random Forests and Decision Trees Automatic Interior Room Color Designing View project Image based fault prediction in power lines View project Random Forests and Decision Trees* (tech. rep.). www.IJCSI.org

Allen-Zhu, Z., Li, Y., & Song, Z. (2018). A Convergence Theory for Deep Learning via Over-Parameterization. https://doi.org/10.48550/arxiv.1811.03962

An, S., Lee, M., Park, S., Yang, H., & So, J. (2020). An Ensemble of Simple Convolutional Neural Network Models for MNIST Digit Recognition.

Bianco, S., Cadene, R., Celona, L., & Napoletano, P. (2017). Benchmark Analysis of Representative Deep Neural Network Architectures. https://doi.org/10.1109/ACCESS.2017.DOI

Breiman, L. (2001). Random Forests.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language Models are Few-Shot Learners. http://arxiv.org/abs/2005.14165

Brutzkus, A., Globerson, A., Malach, E., & Shalev-Shwartz, S. (2017). SGD Learns Over-parameterized Networks that Provably Generalize on Linearly Separable Data. https://doi.org/10.48550/arxiv.1710.10174

Bzdok, D., Altman, N., & Krzywinski, M. (2018). Points of Significance: Statistics versus machine learning. https://doi.org/10.1038/nmeth.4642

Castellano, G., Fanelli, A. M., & Pelillo, M. (1997). An iterative pruning algorithm for feedforward neural networks. *IEEE Transactions on Neural Networks*, *8*(3), 519–531. https://doi.org/10.1109/72.572092

Costa, L. d. F. (2021). Further Generalizations of the Jaccard Index. http://arxiv.org/abs/2110.09619

Das, K., Jiang, J., & Rao, J. N. K. (2004). Mean squared error of empirical predictor. *Annals of Statistics*, *32*(2), 818–840. https://doi.org/10.1214/009053604000000201

Deng, H., Runger, G., Tuv, E., & Vladimir, M. (2013). A Time Series Forest for Classification and Feature Extraction. http://arxiv.org/abs/1302.2277

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, *29*(6), 141–142.

Dettmers, T., & Zettlemoyer, L. (2019). Sparse Networks from Scratch: Faster Training without Losing Performance. http://arxiv.org/abs/1907.04840

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://doi.org/10.48550/arxiv.1810.04805

Ding, X., Ding, G., Zhou, X., Guo, Y., Han, J., & Liu, J. (2019). Global Sparse Momentum SGD for Pruning Very Deep Neural Networks. http://arxiv.org/abs/1909.12778

Dong, X., Chen, S., & Pan, S. J. (2017). Learning to Prune Deep Neural Networks via Layer-wise Optimal Brain Surgeon. http://arxiv.org/abs/1705.07565

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. http://arxiv.org/abs/2010.11929

Draper, N. R., & Smith, H. (1998). *Applied Regression Analysis*. Wiley. https://doi.org/10.1002/9781118625590

Elsen, E., Dukhan, M., Gale, T., & Simonyan, K. (2019). Fast Sparse ConvNets. http://arxiv.org/abs/1911.09723

Elsken, T., Metzen, J. H., & Hutter, F. (2018). Neural Architecture Search: A Survey. http://arxiv.org/abs/1808.05377

Finnoff, W., Hergert, F., & Zimmermann, H. G. (1993). Improving model selection by nonconvergent methods. *Neural Networks*, *6*(6), 771–783. https://doi.org/https://doi.org/10.1016/S0893-6080(05)80122-4

Frankle, J., & Carbin, M. (2018). The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. http://arxiv.org/abs/1803.03635

Frankle, J., Dziugaite, G. K., Roy, D. M., & Carbin, M. (2019). Stabilizing the Lottery Ticket Hypothesis. http://arxiv.org/abs/1903.01611

Frankle, J., Dziugaite, G. K., Roy, D. M., & Carbin, M. (2020). Pruning Neural Networks at Initialization: Why are We Missing the Mark? http://arxiv.org/abs/2009.08576

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh & M. Titterington (Eds.), *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256). PMLR. https://proceedings.mlr.press/v9/glorot10a.html

Gruschka, N., Mavroeidis, V., Vishi, K., & Jensen, M. (2018). Privacy Issues and Data Protection in Big Data: A Case Study Analysis under GDPR. http://arxiv.org/abs/1811.08531

Gupta, J. (2020). Going beyond 99% — MNIST Handwritten Digits Recognition | by Jay Gupta | Towards Data Science. https://towardsdatascience.com/going-beyond-99-mnist-handwritten-digits-recognition-cfff96337392

Ha, D., Dai, A., & Le, Q. V. (2016). HyperNetworks.

Han, S., Kang, J., Mao, H., Hu, Y., Li, X., Li, Y., Xie, D., Luo, H., Yao, S., Wang, Y., Yang, H., & Dally, W. J. (2017). ESE: Efficient speech recognition engine with sparse LSTM on FPGA. *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 75–84. https://doi.org/10.1145/3020078.3021745

Han, S., Mao, H., & Dally, W. J. (2015). Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding.

Han, S., Pool, J., Tran, J., & Dally, W. J. (2015). Learning both Weights and Connections for Efficient Neural Networks. http://arxiv.org/abs/1506.02626

He, K., Zhang, X., Ren, S., & Sun, J. (2015a). Deep Residual Learning for Image Recognition.

He, K., Zhang, X., Ren, S., & Sun, J. (2015b). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.

He, X., Zhao, K., & Chu, X. (2019). AutoML: A Survey of the State-of-the-Art. https://doi.org/10.1016/j.knosys.2020.106622

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the Knowledge in a Neural Network. http://arxiv.org/abs/1503.02531

Hirata, D., & Takahashi, N. (2020). Ensemble learning in CNN augmented with fully connected subnetworks.

Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2005). *Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies* (tech. rep.).

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., & Peste, A. (2021). Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. http://arxiv.org/abs/2102.00554

Hong, Y., & Han, P. (2021). LSDDL: Layer-Wise Sparsification for Distributed Deep Learning. *Big Data Res.*, *26*(100). https://doi.org/10.1016/j.bdr.2021.100272

Horel, E., & Giesecke, K. (2019). Significance Tests for Neural Networks.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. https://doi.org/10.48550/arxiv.1704.04861

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. http://arxiv.org/abs/1602.07360

Janowsky, S. A. (1989). Pruning versus clipping in neural networks. *Phys. Rev. A*, *39*(12).

Jonathan Frankle. (2020). OpenLTH.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., . . . Hassabis, D. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, *596*(7873), 583–589. https://doi.org/10.1038/s41586-021-03819-2

Kass, G. V. (1980). An Exploratory Technique for Investigating Large Quantities of Categorical Data. *Journal of The Royal Statistical Society Series C-applied Statistics*, *29*, 119–127.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet Classification with Deep Convolutional Neural Networks* (tech. rep.). http://code.google.com/p/cuda-convnet/

Kuchibhotla, A. K., Brown, L. D., Buja, A., & Cai, J. (2019). All of Linear Regression. http://arxiv.org/abs/1910.06386

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

LeCun, Y., Boser, B., & Denker, J. (1989). Backpropagation Applied to Handwritten Zip Code Recognition.

LeCun, Y., Cortes, C., & J.C. Burges, C. (1999). *THE MNIST DATABASE of handwritten digits* (tech. rep.). http://yann.lecun.com/exdb/mnist/

LeCun, Y., Denker, J., & Solla, S. (1989). Optimal Brain Damage. In D. Touretzky (Ed.), *Advances in neural information processing systems*. Morgan-Kaufmann. https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf

LeCun, Y., Haffner, P., Bottou, L., & Bengio, Y. (1999). Object Recognition with Gradient-Based Learning.

Lee, N., Ajanthan, T., Gould, S., & Torr, P. H. S. (2019). A Signal Propagation Perspective for Pruning Neural Networks at Initialization.

Lee, N., Ajanthan, T., & Torr, P. H. S. (2018). SNIP: Single-shot Network Pruning based on Connection Sensitivity. http://arxiv.org/abs/1810.02340

Li, Z., Wallace, E., Shen, S., Lin, K., Keutzer, K., Klein, D., & Gonzalez, J. E. (2020). Train Large, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers. https://doi.org/10.48550/arxiv.2002.11794

Lin, T., Stich, S. U., Barba, L., Dmitriev, D., & Jaggi, M. (2020). Dynamic Model Pruning with Feedback. http://arxiv.org/abs/2006.07253

Lin, T., Wang, Y., Liu, X., & Qiu, X. (2021). A Survey of Transformers. https://doi.org/10.48550/arxiv.2106.04554

Linardatos, P., Papastefanopoulos, V., & Kotsiantis, S. (2020). Explainable AI: A Review of Machine Learning Interpretability Methods. *Entropy*, *23*(1), 18. https://doi.org/10.3390/e23010018

Liu, Z., Sun, M., Zhou, T., Huang, G., & Darrell, T. (2018). Rethinking the Value of Network Pruning. http://arxiv.org/abs/1810.05270

Maclin, R., & Opitz, D. (1999). Popular Ensemble Methods: An Empirical Study. https://doi.org/10.1613/jair.614

Mehta, R. (2019). Sparse Transfer Learning via Winning Lottery Tickets. http://arxiv.org/abs/1905.07785

Mishra, A., Nurvitadhi, E., Cook, J. J., & Marr, D. (2017). WRPN: Wide Reduced-Precision Networks. http://arxiv.org/abs/1709.01134

Mozer, M. C., & Smolensky, P. (1989). *SKELETONIZATION: A TECHNIQUE FOR TRIMMING THE FAT FROM A NETWORK VIA RELEVANCE ASSESSMENT* (tech. rep.).

Nagel, M., Fournarakis, M., Amjad, R. A., Bondarenko, Y., van Baalen, M., & Blankevoort, T. (2021). A White Paper on Neural Network Quantization. http://arxiv.org/abs/2106.08295

Narkhede, M. V., Bartakke, P. P., & Sutaone, M. S. (2022). A review on weight initialization strategies for neural networks. *Artificial Intelligence Review*, *55*(1), 291–322. https://doi.org/10.1007/s10462-021-10033-z

Neill, J. O. (2020). An Overview of Neural Network Compression. http://arxiv.org/abs/2006.03669

Neyshabur, B., Li, Z., Bhojanapalli, S., Lecun, Y., & Srebro, N. (2019). *THE ROLE OF OVER-PARAMETRIZATION IN GENERALIZATION OF NEURAL NETWORKS* (tech. rep.).

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation Functions: Comparison of trends in Practice and Research for Deep Learning. http://arxiv.org/abs/1811.03378

OpenAI. (2018). AI and Compute. https://openai.com/blog/ai-and-compute/

Paganini, M., & Forde, J. (2020). On Iterative Neural Network Pruning, Reinitialization, and the Similarity of Masks. http://arxiv.org/abs/2001.05050

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., . . . Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*(85), 2825–2830. http://jmlr.org/papers/v12/pedregosa11a.html

Pooch, U. W., & Nieder, A. L. (1973). *A Survey of Indexing Techniques for Sparse Matrices* (tech. rep.).

Prechelt, L. (1996). *Connection Pruning with Static and Adaptive Pruning Schedules* (tech. rep.).

PyTorch documentation. (n.d.). https://pytorch.org/docs/stable/index.html

Quinlan, J. R. (1986). *Induction of Decision Trees* (tech. rep.).

Rasmussen, C., & Ghahramani, Z. (2000). Occam s Razor. In T. Leen, T. Dietterich, & V. Tresp (Eds.), *Advances in neural information processing systems*. MIT Press. https://proceedings.neurips.cc/paper/2000/file/0950ca92a4dcf426067cfd2246bb5ff3-Paper.pdf

Redell, N. (2019). Shapley Decomposition of R-Squared in Machine Learning Models. http://arxiv.org/abs/1908.09718

Renda, A., Frankle, J., & Carbin, M. (2020). Comparing Rewinding and Fine-tuning in Neural Network Pruning. http://arxiv.org/abs/2003.02389

Robbins, H., & Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, *22*(3), 400–407. https://doi.org/10.1214/aoms/1177729586

Ruan, L., & Jin, Q. (2022). Survey: Transformer based video-language pre-training. *AI Open*, *3*, 1–13. https://doi.org/10.1016/j.aiopen.2022.01.001

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2014). ImageNet Large Scale Visual Recognition Challenge. http://arxiv.org/abs/1409.0575

Sanh, V., Wolf, T., & Rush, A. M. (2020). Movement Pruning: Adaptive Sparsity by Fine-Tuning.

Santos, A., Castelo, S., Felix, C., Ono, J. P., Yu, B., Hong, S., Silva, C. T., Bertini, E., & Freire, J. (2019). Visus: An interactive system for automatic machine learning model building and curation. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. https://doi.org/10.1145/3328519.3329134

Schmidhuber, J. (2015). Deep Learning in neural networks: An overview. https://doi.org/10.1016/j.neunet.2014.09.003

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. http://arxiv.org/abs/1712.01815

Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition.

Srinivas, S., & Babu, R. V. (2015). Data-free parameter pruning for Deep Neural Networks. http://arxiv.org/abs/1507.06149

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014a). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014b). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and Policy Considerations for Deep Learning in NLP. https://doi.org/10.48550/arxiv.1906.02243

Su, J., Chen, Y., Cai, T., Wu, T., Gao, R., Wang, L., & Lee, J. D. (2020). Sanity-Checking Pruning Methods: Random Tickets can Win the Jackpot. http://arxiv.org/abs/2009.11094

Sun, C., Shrivastava, A., Singh, S., & Gupta, A. (2017). Revisiting Unreasonable Effectiveness of Data in Deep Learning Era.

Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. http://arxiv.org/abs/1905.11946

Tanaka, H., Kunin, D., Yamins, D. L. K., & Ganguli, S. (2020). Pruning neural networks without any data by iteratively conserving synaptic flow. http://arxiv.org/abs/2006.05467

Uber, H. Z., Lan, J., Ai, U., Liu, R., & Yosinski, J. (2020). *Deconstructing Lottery Tickets: Zeros, Signs, and the Supermask* (tech. rep.). https://github.com/uber-research/deconstructing-lottery-tickets.

Utgoff, P. E. (1989). IMPROVED TRAINING VIA INCREMENTAL LEARNING. In A. M. Segre (Ed.), *Proceedings of the sixth international workshop on machine learning* (pp. 362–365). Morgan Kaufmann. https://doi.org/https://doi.org/10.1016/B978-1-55860-036-2.50092-8

Valentini, G., & Masulli, F. (2002). Ensembles of learning machines. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *2486 LNCS*, 3–20. https://doi.org/10.1007/3-540-45808-5{_}1

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. https://doi.org/10.48550/arxiv.1706.03762

Verdenius, S., Stol, M., & Forré, P. (2020). Pruning via Iterative Ranking of Sensitivity Statistics. http://arxiv.org/abs/2006.00896

Wang, C., Grosse, R., Fidler, S., & Zhang, G. (2019). EigenDamage: Structured Pruning in the Kronecker-Factored Eigenbasis. http://arxiv.org/abs/1905.05934

Wang, C., Zhang, G., & Grosse, R. (2020). Picking Winning Tickets Before Training by Preserving Gradient Flow. http://arxiv.org/abs/2002.07376

Yu, J., Lukefahr, A., Palframan, D., Dasika, G., Das, R., & Mahlke, S. (2017). Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism. *ACM SIGARCH Computer Architecture News*, *45*, 548–560. https://doi.org/10.1145/3140659.3080215

Yu, T., & Zhu, H. (2020). Hyper-Parameter Optimization: A Review of Algorithms and Applications. http://arxiv.org/abs/2003.05689

Zagoruyko, S., & Komodakis, N. (2016). Wide Residual Networks.

Zeng, W., & Urtasun, R. (2018). *MLPRUNE: MULTI-LAYER PRUNING FOR AUTO-MATED NEURAL NETWORK COMPRESSION* (tech. rep.).

Zhmoginov, A., Sandler, M., & Vladymyrov, M. (2022). HyperTransformer: Model Generation for Supervised and Semi-Supervised Few-Shot Learning.

Zhu, J., Jiang, J., Chen, X., & Tsui, C.-Y. (2017). SparseNN: An Energy-Efficient Neural Network Accelerator Exploiting Input and Output Sparsity. http://arxiv.org/abs/1711.01263

Zoph, B., & Le, Q. V. (2016). Neural Architecture Search with Reinforcement Learning. http://arxiv.org/abs/1611.01578

# List of Figures

# List of Tables

# Appendix A

# Experimental setup

In this chapter, we give a detailed overview on how our experiment environment is setup.

## A.1 Basic Structure

As described in chapter 4, we build upon work published by the authors of SynFlow. The original code is available freely on GitHub[1]. From this implementation, we reuse the pruning mechanism, the train/eval loop as well as the data loading logic. On top of these elements, we completely rebuild the way experiments are specified, executed and persisted. With the new logic in place, we are able to specify an Experiment object with the variables relevant for us, create a State object containing the model, data, loss, optimizer and pruner as well as all relevant Hyperparameters which we then are able to store to disk to analyze later.

A required feature to be able to compare the Lottery Tickets of different pruning algorithms is the ability of running them on the same set of initial weights. For this requirement, we implement functionality to reference a set of initialized weights from the model object which then gets loaded and stored separately and only once across all experiment runs.

From the State object, we construct a dictionary with all relevant Hyperparameters and metrics for a given experiment run for it to easily be analyzed and aggregated. The Result dictionary contains a reference to the persisted State of the experiment. The scores, weights and masks for all layers of the model stored in the state can easily be accessed by a set of functions part of the analysis module, which allow the caller to search for experiments by passing values with desired attributes.

## A.2 Important changes to the original code base

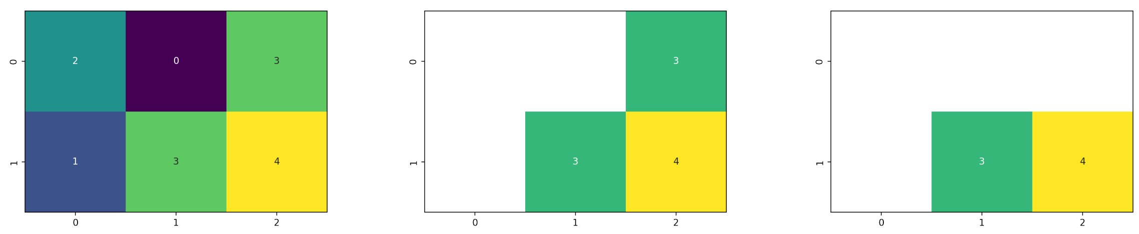
Several features were added to the original implementation to accommodate our experiments.

As a start, while the code base includes many models and datasets out of the box, the selected LeNet_300_100 architecture is not included. We implement this model architecture by adapting the implementation found in OpenLTH. Additionally, we transfer the early stopping mechanism from OpenLTH as well, which was not part of the code base.

Further, we found the mechanism to prune weight based on the scores provided by the pruning algorithms to be sub-optimal for our experiments. The Boolean mask defining which weights are pruned and which ones are available is constructed by the steps described in 3. By default,

[1]https://github.com/ganguli-lab/Synaptic-Flow

the original implementation uses global pruning which is implemented by calculating a threshold value globally across all layers and then constructing Boolean masks for each layer individually. This works well as long as the scoring mechanism produces unique scores for each weight, which, with the scoring algorithms in their original form, is unlikely as two weights would have to be initialized with the exact same floating point number. However, we noticed the issue in our own implementations of scoring criteria, where scores might not be as fine-grained. Figure A.1 illustrates the issue, which implies that k+n weights might be pruned, for k is the number of weights that we expect to be pruned and n is the number of duplicates at the threshold.



(A) Example of a matrix of pruning scores.

(B) The scores selected with k=2, resulting in a threshold of 3.

(C) The scores selected based on a true Top-k mechanism with k=2. From the two duplicate values, the first one in the index will be chosen.

FIGURE A.1: A visualisation of the issue when selecting weights based on a calculated threshold value with duplicate values

As the we use global pruning, implementing such Top-k mechanism is not trivial, as the Top-k calculation must be performed across all layers at once, but the resulting masks have to be layer specific and in the correct dimensions. We solved the issue by constructing a combined score matrix of size 1x266200 for all layers, calculating the mask on this global scores and then mapping the result back to the individual layers. This solution ensures that the correct number of weights are pruned, however it also comes with a drawback. In the case of duplicate values, it will always prune the weights early in the index first. This results in a pruning decision taken entirely based on the position of the weight in the matrix and not informed by the score. While it is useful for testing purposes to be able to deal with duplicate scores, ideally the scoring mechanism would not produce duplicates in the first place.

## A.3 Parallelization

The Experiment execution is entirely containerized. We work with the PyTorch base image pytorch/pytorch:1.11.0-cuda11.3-cudnn8-devel, which already contains all dependencies for running PyTorch with CUDA support. In addition to docker itself, we use docker-compose to orchestrate the three containers used to parallelize the execution of the experiment. We selected to run three workers after a series of experiments we conducted while observing the resource usage. Since much of the workload in our case has to do with data loading, the bottleneck is not the GPU but the CPU and RAM, both of which are primarily occupied in loading the data. When looking at the GPU usage via nvidia-smi, the CUDA Cores where typically at 80% load. Via docker-compose, we bind mount the code into the containers, allowing us to run an updated version of the experiment without rebuilding the image. Via docker resource reservations[2], we enable CUDA support within the containers.

[2]https://docs.docker.com/compose/gpu-support/

## A.4 Monitoring

All training is monitored with Tensorboard, allowing us to follow the progress of the experiments in real time. Apart from train/test loss, we track the Top-1 accuracy, the sparsity as well as the time per epoch. Further, all hyperparameters as well as histograms of the weights at different stages during training are stored. Since all workers write to the same bind mounted directory, we are able to monitor all results across all workers.

## A.5 Hardware

Within the ZHAW School of Engineering, a high-performance computing cluster offering 32GB nVidia V100 GPUs exists[3], however, during the duration of this thesis, this cluster was unavailable. Instead, we were able to use the cluster based on OpenStack, which provided us with a Tesla T4 GPU. For the latter part of the thesis, we also had access to a much faster Nvidia RTX 3070.

## A.6 Development Setup

We use the tools available within VSCode[4] to connect remotely via SSH to the respective machine. From there, a development container specified in a Dockerfile, separate from the actual productive one, is built and started. In this way, we have a highly portable environment we can use on any machine.

## A.7 Analysis Setup

To analyze the results generated, we utilize Jupyter[5] Notebooks, running withing the VSCode Jupyter Extension[6]. From there, we access the data via a set of helper functions part of the analysis module. Further analyis is done using Numpy[7] and Pandas[8] for data wrangling as well as Seaborn[9] for visualisations. This toolset was selected due to prior experience of the authors with this stack.

[3]https://info.cloudlab.zhaw.ch/
[4]https://code.visualstudio.com/
[5]https://jupyter.org/
[6]https://code.visualstudio.com/docs/datascience/jupyter-notebooks
[7]https://numpy.org/
[8]https://pandas.pydata.org/
[9]https://seaborn.pydata.org/

# Appendix B

# Meta Model with additional Features

The meta model producing pruning scores on the basis of a single weight can be extended to use more features with the goal of providing more information about the weight itself and the layer a weight is in to the model.

We implement six additional features:

- The weight itself
- The frobenius norm of the whole layer
- The value of the highest activation of the input neuron
- The value of the highest activation of the output neuron
- The average activation of the input neuron
- The average activation of the output neuron

Thus we bring the total number of features up to 10, including the four existing algorithms.

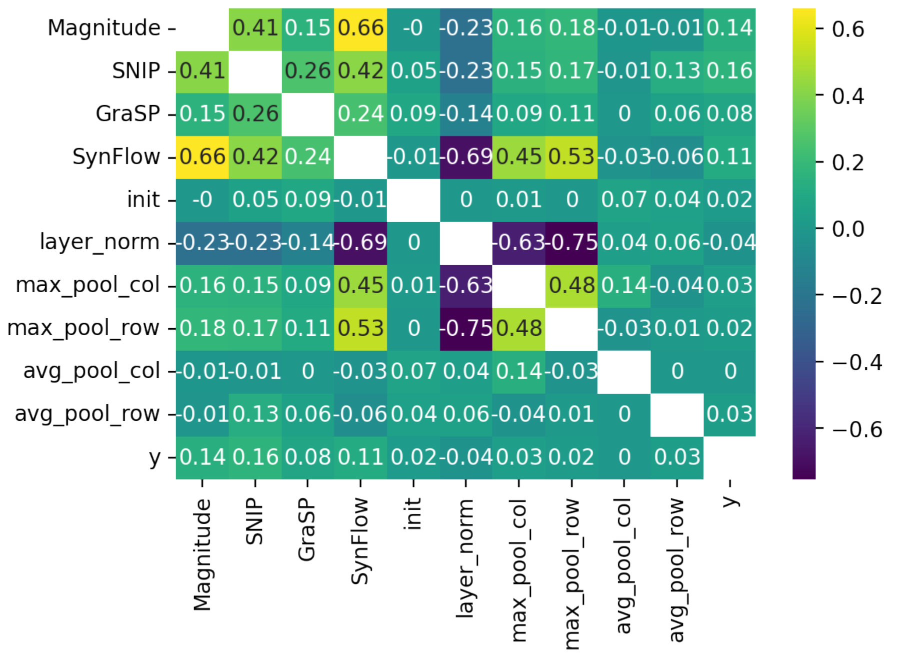


FIGURE B.1: Correlation between all features used in the extended version of the meta model.

We observe interesting correlations between some of the features, for example SynFlow correlating well with both the maximum value of the input and of the output value. As for most other features, the new features are uncorrelated with the scores.

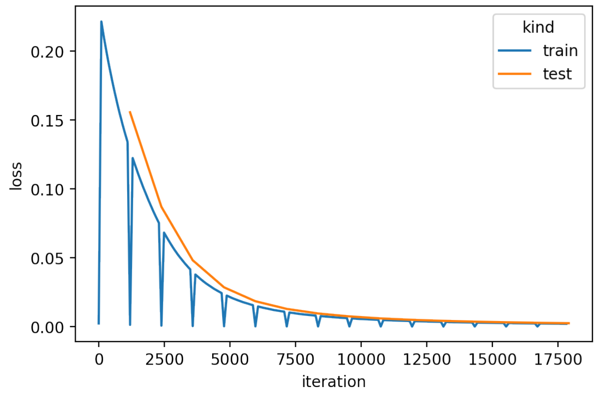


FIGURE B.2: Learning dynamics of the same MLP architecture as use in chapter 5.

As shown in figure B.2, the extended model exhibits a similar training curve as the regular one.

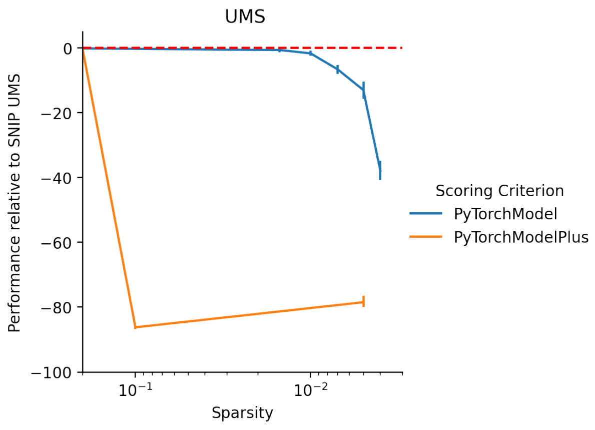


FIGURE B.3: Performance of the extended meta model compared to the original implementation.

Figure B.3 visualizes the performance of the new extended meta model compared to the original implementation described in chapter 3. As is clearly visible, the new model is not able to produce Lottery Tickets which train to a non-trivial accuracy.

To us, worse performance with more data is unexpected. We hypothesize there to be some form of implementation error present and recommend to conduct more research to investigate whether or not the model can be improved.

# Appendix C

# Hyperparameters

The table C.1 lists all configurable Hyperparameters in our environment and the values we used for them.

| Parameter Group | Parameter | Value |
|---|---|---|
| Model Parameters | model_class | lottery |
| | model | lenet_300_100 |
| | lr | 0.1 |
| | lr_drop_rate | 0.0 |
| | lr_drops | [] |
| | weight_decay | 0.0 |
| | dense_classifier | False |
| | pretrained | False |
| | optimizer | sgd |
| | init_strategy | standard |
| Data Parameters | dataset | mnist |
| | train_batch_size | 64 |
| | test_batch_size | 256 |
| | prune_batch_size | 256 |
| | workers | 4 |
| | prune_dataset_ratio | 10 |
| | input_shape | (1, 28, 28) |
| | num_classes | 10 |
| Pruning Parameters | strategy | mag |
| | sparsity | 0.2 |
| | prune_epochs | 1 |
| | prune_bias | False |
| | prune_batchnorm | False |
| | prune_residual | False |
| | compression_schedule | exponential |
| | mask_scope | global |
| | reinitialize | False |
| | prune_train_mode | False |
| | shuffle | False |
| | invert | False |
| Training Parameters | train_epochs | 100.0 |

TABLE C.1: Full set of Hyperparameters used.

# Appendix D

# Meeting protocols

| Date | Attendees | Notes |
|---|---|---|
| 21/02/2022 | Prof. Dr. Thilo Stadelmann, Dr. Claus Horn, Urban Lutz, Alexandre Manai | Kick-off meeting:<br>- organizational matters:<br>– Meeting dates (biweekly - 45min)<br>– Workplaces (At CAI at ZHAW)<br>- First steps:<br>– important papers to read first<br>(lottery ticket hypothesis, Hypernetwork (decided to set focus on that), Variational Autoencoders) |
| 08/03/2022 | Prof. Dr. Thilo Stadelmann, Dr. Claus Horn, Urban Lutz, Alexandre Manai | - Decided t focus on:<br>– VGG and ResNet models<br>– Investigate Transformers and Hypertransformers<br>- Look into:<br>– ADA project<br>– goal: for different tasks train model in constrained enviroment in less than 5 min<br>– adapt challenge for our thesis |
| 22/03/2022 | Prof. Dr. Thilo Stadelmann, Dr. Claus Horn, Urban Lutz, Alexandre Manai | - Emerging questions:<br>– Clustering of the LTs possible?<br>– Do LTs always have same structure?<br>- Goal setting for next time:<br>– Generate Dataset<br>– Preparation of possible models<br>– what can we expect?<br>What do we do with it?<br>What task are we looking at? |
| 01/04/2022 | Prof. Dr. Thilo Stadelmann, Dr. Claus Horn, Urban Lutz, Alexandre Manai | - Emergency meetind due to unfeasible first idea<br>– look into Dr. Claus Horns slides<br>– urgently read more papers |

| Date | Attendees | Notes |
|---|---|---|
| 05/04/2022 | Prof. Dr. Thilo Stadelmann, Dr. Claus Horn, Urban Lutz, Alexandre Manai | - Presented new idea of combining SotA algorithms<br>- Interesting ideas:<br>– weight patterns emerging from different results from different algorithms<br>– Explainable AI angle<br>– simple firstly: compare scores and the existence of overlap |
| 19/04/2022 | Prof. Dr. Thilo Stadelmann, Dr. Claus Horn, Alexandre Manai | - Coding Enviroment set-up with Tensorboard, checkpointing, ...<br>- Used SNIP, GraSP, IMP and Synflow as central algorithms to combine with data free and iterative implementations<br>- 400 ran Experiments with 50/50 split made of data-free and interative pruning each having all possible combinations of algorithms and sparsities between 0.8 and 0.01<br>- Used model/dataset: LeNet on MNIST<br>- Presentation of first results:<br>– observations: up to 90% sparsity not difference between approaches, LTH baseline best as expected,<br>interesting at high sparsities<br>where combinations yield<br>notably different results |
| 03/05/2022 | Prof. Dr. Thilo Stadelmann, Dr. Claus Horn, Urban Lutz, Alexandre Manai | - Updates on experiments:<br>– access to new Harware<br>– 50/50 finished and analysed<br>– 90/10 split finished but not yet analysed<br>– ResNet with Cifar-10 tried but took too long<br>- New ideas:<br>– looking at ensemble and vertical combinations<br>– Detect correlations in clusters of pruned/unprune<br>– best weights for all approaches<br>– try start every algorithm with same init |
| 10/05/2022 | Prof. Dr. Thilo Stadelmann, Dr. Claus Horn, Urban Lutz | - Emergency meeting because not conclusive results<br>- Came to idea of coming up<br>with own pruning algo<br>based on finding of bimodal<br>distribution in good results<br>- Start of writing the thesis:<br>– Abstract, Introduction<br>- Also looking into:<br>– metric of Trainability and Performance<br>– Change in weight over time<br>- Generalization over different tasks |

| Date | Attendees | Notes |
|---|---|---|
| 17/05/2022 | Prof. Dr. Thilo Stadelmann, Dr. Claus Horn, Urban Lutz, Alexandre Manai | - No conclusive results thus:<br>– reframing of work to identify low hanging fruits<br>– try find a combination that is just a tiny bit better<br>– Non conclusive is a conclusion<br>- Structure of written thesis:<br>– fuse related work and theoretical foundations |
| 31/05/2022 | Prof. Dr. Thilo Stadelmann, Urban Lutz, Alexandre Manai | - Structural questions to written thesis:<br>– Difference between Research question and hypothesis (they are the same)<br>– Related Work and Foundations (fuse together)<br>– what is important for appendix and project management? (everything that was tried out but not relevant to the thesis anymore and specifications on algos or test results)<br>– Need a german abstract? (yes)<br>git repo and data where do we hand them in? (send to pascal sager) |

# Appendix E

# Original Task

*See subsequent pages*

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

# Learning to Learn: Learning Successful Weight Patterns for Instant Deep Neural Network Training
# BA22_stdm_01

| | |
|---|---|
| BetreuerInnen: | Thilo Stadelmann, stdm<br>Claus Horn, horc |
| Fachgebiete: | Artificial Intelligence (AI)<br>Bildverarbeitung (BV)<br>Datenanalyse (DA)<br>Machine Learning (ML)<br>Software (SOW) |
| Studiengang: | IT |
| Zuordnung: | Centre for Artificial Intelligence (CAI) |
| Interne Partner: | Departement N (DeptN) |
| Gruppengrösse: | 2 |

**Kurzbeschreibung:**
In the last few years, it has become clear that most of the weights of fully connected layers are not needed, and full performance usually can be achieved with 10% - 20% of the connections (see [1]). I.e., neural networks naturally learn sparse representations. Additionally, recent papers have observed that some of the connectivity patterns learned by neural networks (especially transformer architectures) may represent generally useful inductive biases that can be useful to solve other tasks (see [2], [3], [4]). We therefore, aim to study the connectivity patterns learned by deep neural networks more in detail in order to improve the speed of training convergence ("learning to learn").

## Approach

The project aims to construct a generative deep learning architecture for learning weight patterns of neural networks, then train it on sparse networks and their performances, and evaluate if it can generate the weights for other high-performance networks. To this end, the use of HyperNetworks (see [5], [6]) shall be evaluated to directly generate the weights of winning Lottery Ticket networks (see [1]; alternative approaches include variants of Variational Graph Auto-Encoders [7]).

## Work packages

- Review of related work (scientific literature, public github repos)
- Define prototype scope and training data (e.g., networks performing well on MNIST digit recognition)
- Set up the development environment (locally and on our GPU cluster)
- Build initial prototype, iterate (focus on algorithms, machine learning, functionality)
- Write a scientific report with a focus on motivation, argumentation, methods, evaluation & results (the BA thesis)

**Voraussetzungen:**
This thesis can be conducted in German or English. The students should have basic knowledge in deep learning and initial experience in constructing deep learning models in Python. Strong interest in AI and scientific research as well as scholarly work is mandatory, as well as the ability to independent conceptual work, good general programming skills and a pragmatic approach to experimentation and thorough evaluation of results.

## Contact

The Computer Vision, Perception and Cognition Group at the newly-founded ZHAW Centre for Artificial Intelligence conducts research and teaching in pattern recognition using deep learning methodology. In our thesis proposals together with the Institute of Applied Simulation at the School of Life Sciences and Facility Management in Wädenswil, we want to address high-achieving students with a curiosity for gaining (first) experience with scientific research applied to the problem at hand. The supervisors are very enthusiastic about the topics and have plenty of experience as well as detailed ideas (and offering of support) for the project startup. We are looking for equally enthusiastic and high-achieving students, with the hope (given good results) of a joint scientific publication.

claus.horn@zhaw.ch, thilo.stadelmann@zhaw.ch

## References

[1] The Lottery Ticket Hypothesis, https://arxiv.org/abs/1803.03635

[2] PerceiverIO: A General Architecture for Structured Inputs & Outputs, https://arxiv.org/abs/2107.14795

[3] Pretrained Transformers as Universal Computation Engines, https://arxiv.org/abs/2103.05247

[4] LIME: Learning Inductive Bias for Primitives of Mathematical Reasoning, https://arxiv.org/abs/2101.06223

[5] Hyper Networks, https://arxiv.org/abs/1609.09106v4

[6] Continual learning with hypernetworks, https://arxiv.org/abs/1906.00695

[7] Variational Graph Auto-Encoders, https://arxiv.org/abs/1611.07308

**Weiterführende Informationen:**
https://www.zhaw.ch/en/engineering/institutes-centres/cai/computer-vision-perception-and-cognition-group/