UTRECHT UNIVERSITY

THESIS REPORT

How can the knowledge of lottery tickets be used to adapt SynFlow to early pruning after pre-training?

Author:
Cas Bolwerk
6231489

First supervisor:
Dr. M. (Thijs) van Ommen
Second supervisor:
Dr. K. (Krisztina) Szilagyi

Thesis report for 7.5EC for the Artificial Intelligence bachelor

July 2, 2021

Universiteit Utrecht

Abstract

This research focuses on adapting an early-pruning algorithm called SynFlow to prune after pre-training. Adapting in this way was inspired by work around the Lottery Ticket Hypothesis by Frankle et al. We show that the instability analysis is a way to determine the stability of a network at an iteration k, after which post-training a subnetwork will result in it *matching* a full network's accuracy.

While this still reduces complexity and training time during training as the original SynFlow, it also improves accuracy significantly. In the current training setup, a full ResNet-20 network is able to achieve accuracies as high as 90.14%. SynFlow on average gets up to a 80% accuracy, sometimes reaching the low 80 percentages. This research presents AdSynFlow, where we consistently achieve higher accuracies than SynFlow, with a high of 85.27%. Though not yet matching the full network, AdSynFlow promises a bright future for early pruning methods.

Keywords: Pruning, lottery tickets, pre-training, SynFlow

Contents

List of Abbreviations

AI	Artificial intelligence
CNN	Convolutional neural network
GPT-3	Generative Pre-trained Transformer 3
SynFlow	Synaptic Flow
SGD	Stochastic gradient descent
IMP	Iterative magnitude pruning
MNIST	Modified National Institute of Standards and Technology database
CIFAR	Canadian Institute For Advanced Research
GPU	Graphical Processing Unit

Chapter 1

Introduction

Since the neural networks boom in 2012, artificial intelligence has become almost synonymous with neural networks and deep learning. A mindset that went hand in hand with that, is that bigger is better. Larger networks allegedly leverage more ability to gain knowledge of their own. Which, interestingly, also seems to be the case in recent innovations such as the largest neural network trained yet, GPT-3 [1]. But, the majority of people do not have devices at the ready capable of computing to the level of complexity needed for these large networks. Just storing GPT-3 in memory requires 350GB. To actually take these results and bring them to the masses, there is another path that needs to be followed. Reducing complexity of neural networks has seen an increasing amount of research as a topic within deep learning, focusing mostly on reducing complexity of networks after they have already been trained. The rise of the Lottery Ticket Hypothesis [11], which claims that the large network at the start is just there to allow the smaller, specialized network to be found, is an example of this. Now, researchers are longing to find these smaller subnetworks or lottery tickets through the process of cutting away weights that bear no significance to prediction – pruning.

To be able to compress and prune neural networks with virtually no accuracy deduction after training seemed and still seems to be the way to go. But other ideas definitely are on the horizon, such as pruning before training to find these lottery tickets before ever having trained the network. SynFlow [25] is one such algorithm, managing to prune without ever having seen any data. These idealized approaches have lead to some important realizations and breakthroughs, but never seemed to come to the same level of accuracy as pruning after training. Frankle et al. (from Lottery Ticket Hypothesis fame) show that pruning at initialization is not the faultless approach it seemed to be in their paper on initialization pruning algorithms [13]. This paper showed that indifference at initialization to supposed setbacks of some kind, such as weight reinitialization or pruning mask shuffling (explained in section 2.2.2), tends to be a precedent to worse results later in training.

Frankle et al. then connect this to their Linear Mode Connectivity paper [12], as they now state that once a model is stable to noise it loses its indifference to these supposed setbacks and that this leads to better final accuracy as a result. Through these findings, it seems obvious to want to adapt one of these initialization pruning algorithms to prune after some training time and stability. In adapting SynFlow, this research aims to combat the shortcomings of initialization pruning, while still keeping some of the optimistic ideas intact. In this paper, we present Adapted Synaptic Flow (AdSynFlow). AdSynFlow pre-trains until stabilization of the network, afterwards pruning with some further insights thanks to the already trained state of the network. This leads us to the research question, which reads

How can the knowledge of lottery tickets be used to adapt SynFlow to early pruning after pre-training?

This research question, though ambitious, is limited in scope through limitations on the model, the dataset and the pruning algorithm. Doing so will enable work to bear fruit while not going overboard through comparison of all possible variables.

As already stated, neural networks are an integral part of artificial intelligence and deep learning in particular. Thus, the training and pruning of these networks must also be relevant to the topics of this bachelor. In introducing a new, or adapted, version of an algorithm, this research stays close to artificial intelligence and comes to some exciting conclusions along the way.

Chapter 2

Theoretical background

2.1 Convolutional neural networks

Convolutional neural networks are the subset that will be used in this research. They are most notable for their use in image recognition, which is the task that most pruning is tested on first. Being a type of neural network, convolutional neural networks (CNN's) have at least one convolutional layer. The Latin word *convolutus* already explains their working, as it means to fold together. A convolution is an operation that applies a matrix with weights – a kernel – to a group of input data, combining their values into a less dimensional output according to these weights. The idea is that the output is smaller in dimensions, but still carries information of all earlier data points. A convolutional layer is a layer of nodes that use this operation to reduce the number of nodes of a previous layer, acting as regularization.

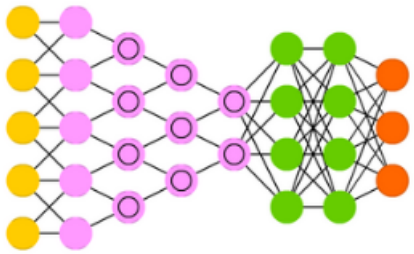

FIGURE 2.1: CNN.
Convolutional nodes are displayed as pink nodes with circles inside.

These networks have proven to be especially useful in computer vision. The pixels in images that act as input data to these CNN's have the property of being more related to other pixels close to them compared to pixels farther away. This property is capitalized on by CNN's, capturing neighborhood relations and reducing dimensions of layers as a result [18]. Nodes extract visual features such as edges and corners from these neighborhood relations, resulting in a set of features called a feature map. If we think of these nodes as feature detectors, their workings are equal across an image. Keeping that thought, we can imitate the kernel mentioned earlier, if all the nodes use the same weights it is as if it is the kernel moving across the image. Having equal weights greatly reduces the number of parameters and the resulting model is more general and introduces an invariance to transformations on images [2].

2.2 Pruning

2.2.1 History of pruning

Pruning has been around for a long time ([20], [19], [6]), but hasn't been very popular up until more recently. Since neural networks as an idea were mostly inspired by the brain, other events the brain goes through have also been applied to neural networks. Pruning was an obvious one, since the brain also goes through a stage of synaptic pruning when coming of age. Around the age of 2 or 3, humans tend to have the largest amount of synapses per neuron. From ages 5 to 10, a fast decline sets in that becomes more stable but continues until the late 20s. During this time, up to 50% of the synapses are pruned [7].

2.2.2 The workings of pruning

Pruning as a concept might be simple, but the heuristics driving the pruning choices are more sophisticated. Since pruning was thought up as a way to reduce model size and complexity while keeping accuracy, most pruning methods prune after the full network has first trained to convergence [3]. An iterative cycle of pruning and retraining sets in to let the subnetwork get accommodated as to not overprune. Most methods do not prune destructively, as destroying connections or weights produces sparse networks that the GPU's used to train neural networks are not optimized for. Instead, a binary *pruning mask* is generated that determines which weights or neurons go to a value of zero - essentially disconnecting them from the rest of the network.

For determining which weights are more important than others, the most commonly used and simple method is magnitude pruning. This method prunes based on the values of weights, where lower is worse as lower values technically contribute less to the forward propagation. Though simple, this method has often proven more powerful and generalizable than more sophisticated ones. Iterative magnitude pruning is also the method used in the Lottery Ticket Hypothesis [11].

Pruning has been around for about as long as neural networks themselves, and especially with the rise of deep neural networks has pruning also seen a popularity boom. One of the most insightful breakthroughs here has been the Lottery Ticket Hypothesis and it has spawned a whole family of papers influenced by it. Next, we will touch on the hypothesis itself and some follow-up papers that will be relevant for this research.

2.2.3 Lottery ticket hypothesis

The lottery ticket hypothesis [11] is one of the more popular papers on pruning. Its significant achievement is a hypothesis that might offer an explanation of the inner workings of neural networks, which is useful because neural networks are largely black boxes that are able to output results although we don't really understand how they are able to get to these results. The lottery ticket hypothesis aims to explain why it is easier to train large networks. It mainly does this by stating a hypothesis, that has not been disproven up until today. The hypothesis reads: *A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.* What it basically says, is that, once a lottery ticket has

been found, the subnetwork does not need the larger network anymore. This subnetwork, when trained from scratch, can match or even improve on the accuracy of the full network. The power of the larger network lies in offering so many weights that a lottery ticket is almost guaranteed to be found.

Pruning had been found to be successful earlier, but Frankle et al. decided to take a unique approach that supported their final hypothesis. What had already been established through the modern advances in pruning, was that a prune-retrain cycle after training was an effective way to reduce the parameters while retaining the initial accuracy of the full network [14]. More significantly, Frankle et al. went on to prune to a subnetwork after training and *reset* the weights to the weights before training. When retrained from scratch, this subnetwork is still able to get the same or even better accuracy up to a certain sparsity. When a subnetwork can be trained to the same accuracy as a full network, the subnetwork is *matching*. This finding led them to their hypothesis, that the large size of the network is mostly there to enable the network to find a subnetwork that is able to do most of the heavy lifting.

More recently, it has been found that instead of reinitialization, the rewinding of the learning rate without reinitializing the weights gives even better results [22].

2.2.4 Synaptic Flow

Building on the achievements of the lottery ticket hypothesis, several papers followed that also tried to find these lottery tickets. The establishment of lottery tickets was a big breakthrough, but only finding them after training and several stages of retraining meant that the reduction of computation during inference required a huge increase in training time. As a result, one of the more popular follow-up topics was trying to start off with lottery tickets without ever having trained the network.

One of the three established initialization pruning algorithms is Synaptic Flow (SynFlow) [25]. The approach by Tanaka et al. does not even look at the data of the model, pruning the network solely on a priori knowledge. It also solves a big problem that most pruning algorithms have to deal with, layer collapse. Layer collapse occurs when all the weights in a layer are pruned, removing the connection between a layer and its neighbor. To explain the final SynFlow algorithm, the theorems motivating the algorithm need to be explained first.

Pruning methods often prune based on a saliency score, a score resembling the importance of weights according to, for example, their training loss. An important thing to realize, is that the saliency score of a weight often increases when the number of weights in their layer is lower. Tanaka et al. then connected this saliency score to an observation they made, which is that iterative magnitude pruning (IMP) does not suffer from the layer collapsing problems that one-shot pruning methods suffer from. According to Tanaka et al., this is directly related to saliency scoring – where iterative pruning methods recalculate saliency scores before pruning again and one-shot pruning methods do not. As saliency scores increase when the number of weights in their layer get smaller, recalculating them as the number of weights decreases before pruning again directly combats the layer collapse phenomenon. This forms the most important theorem inspiring the SynFlow algorithm, which reads *iterative, positive, conservative scoring achieves Maximal Critical Compression.* Maximal critical compression means that a pruning algorithm can achieve the maximal compression of a network, leaving just layers with one weight each.

The SynFlow algorithm uses iterative score evaluation to combat layer collapse, feeding a data-independent image into an efficient scoring procedure. Since saliency score has a bias towards high, positive numbers, the need for positivity and layer conservation motivate a loss function that yields positive synaptic saliency scores. They combine these two insights into a loss function that works by feeding a simple data-independent pseudo image through the network and summing the network outputs (all positive) up so as to end up with a pseudo loss function output. The output of this loss function is then backpropagated through the network and the saliency score for each weight uses this loss function output in their calculation. This adds a unique synaptic flow to the magnitude score of the weights, taking the interactions of weights between layers into account. So the SynFlow algorithm prunes after calculating saliency scores that discourage layer collapse and can work without ever looking at the data.

2.2.5 Linear mode connectivity

Frankle et al. continued research surrounding the lottery ticket hypothesis, of which the linear mode connectivity paper [12] is the most relevant to this research topic. In the Lottery Ticket Hypothesis, Frankle et al. found that larger, more complex models did not end up with subnetworks that matched full networks when reinitialized to the weights from initialization. What they did find however, was that the subnetworks worked when pre-trained for several epochs and then reinitialized to the weights from after pre-training. The linear mode connectivity is Frankle et al.'s explanation for this phenomenon.

Linear mode connectivity as a term has two essential parts to understanding the term itself. The paper revolves around the gradient descent optimization landscape. In training a network, we apply an optimizer function with noise to optimize the network. For their purposes, Frankle et al. use stochastic gradient descent (SGD) noise. This noise alters the path through the optimization landscape, so that every path is different and so that it is possible to escape local minima. Frankle et al. found through experimentation that the subnetworks can be reinitialized with good results after the network has become stable to SGD noise. Stability here means that the network will find the same, linearly connected minimum in the optimization landscape regardless of SGD noise.

To test the (in)stability of a network, Frankle et al. introduce the instability analysis. When rewinding to a point k after training, the pruning mask that was found can be applied and the analysis can be started. The analysis is a comparison between two subnetworks trained from this point k. In comparing these fully-trained subnetworks through a certain function and comparing the outcome of this function to a certain threshold, we can determine the (in)stability of the original network at point k. Frankle et al. use linear interpolation as their function of choice, but also suggest other possible functions such as the L2 distance. The function that is ultimately used outputs a value that represents the similarity between the two subnetworks, where a high similarity implies a stability of the network.

Linear interpolation

Linear interpolation is a technique to create new data points within a linear range of other data points that have already been introduced. In the case of neural networks, you create a new neural network based on the weights of other neural networks.

Though this may sound like a large undertaking, what Frankle et al. end up doing is quite simple. They introduce a formula with a factor alpha that determines how much the new model takes away from the second of two given models. Then the error between two models' weights $E(W_1, W_2)$ for a certain alpha is as follows.

$$\text{Let } E_\alpha(W_1, W_2) = E(\alpha W_1 + (1\alpha)W_2) \text{ for } \alpha \in [0, 1]$$

Then they create an undisclosed number of models with different alphas and take the max error E_{sup} of those alpha networks and the mean error between the two given models E_{mean} to calculate the error barrier height, which is used as the output number for the instability analysis.

$$\text{Error barrier height} = E_{sup}(W_1, W_2) - E_{mean}(W_1, W_2)$$

If this number is smaller than a certain threshold, Frankle et al. use ~ 0.2, the model at the point that the two given models were trained from is considered stable to SGD noise.

Chapter 3

Methodology

AdSynFlow's foundation comes in the form of the instability analysis. To be able to determine (in)stability during training and point to that turning point where a subnetwork matches a full network, is where we can say that AdSynFlow really overtakes the original SynFlow. This turning point is attainable through an instability threshold. After the threshold is established, a final experiment needs to be carried out to determine if there is a significant difference between the two SynFlow's.

In explaining the experiments that will make up AdSynFlow, we also explain the methods that will be used to answer the research question. To reiterate, the research question is *How can the knowledge of lottery tickets be used to adapt SynFlow to early pruning after pre-training?* Through experimentation and later through the analysis of the results, we expand on methods derived from the knowledge of lottery tickets. AdSynFlow uses these methods to improve on SynFlow. First, we establish a threshold for instability analysis. This threshold will be used to determine the turning point of stability during the training of the final AdSynFlow subnetwork. At that point, the subnetwork should be matching the full network and can be called a lottery ticket. To confirm this, the stable AdSynFlow network will be compared to the full network and a SynFlow network.

3.1 Adapting SynFlow

Now that all the ideas needed for the adaptation have been introduced, we are ready to dive into it. SynFlow was originally designed to prune before training. After conclusions from [13] and [12], it has become clear that this is not a viable strategy with the currently known approaches. That is why this research aims to adapt SynFlow to prune after pre-training. The adaptation will be based around the instability analysis introduced in Linear Mode Connectivity, but will also aim to make further changes now that SynFlow is no longer restricted to prune based on a priori knowledge. Experiments have been done to figure out the best adaptation, which led to the following changes:

1. Prune after stable to SGD noise instead of at initialization.
2. Change the input into the pruning mechanism of SynFlow from an image with pixel values being all 1's to either a single image from the dataset (1) or an average image of the dataset (2).
3. SynFlow was meant to prune before ever having seen the training data. Now that there has already been some training done, certain restrictions of the SynFlow algorithm can be left out.

3.1.1 Instability analysis

Pruning after stabilization to SGD noise is an important step in adapting SynFlow to more effectively prune during training. To determine stability, the instability analysis introduced in Linear Mode Connectivity is used. Frankle et al. use linear interpolation as their function, where the *error barrier height* is their metric for instability analysis. This error barrier height then has to be compared to a certain threshold of instability, which Frankle et al. establish as being around 0.2. Since we use a different pruner than IMP, it means that the threshold has to be determined through the means of our own experimentation.

The threshold was ultimately determined by training a network to iteration k, then prune and afterwards train two subnetworks with different SGD noise from iteration k until completion (total epochs – k iterations). These subnetworks can then be used to perform instability analysis and to find the point k at which the error of the subnetwork at completion dropped enough for the subnetwork to be matching. Frankle et al. do not provide a way to exactly determine the threshold either, so it is assumed that the threshold is picked by eye from the graphs that depict instability at different iterations k.

3.2 Training setup

For the experiments and the training process, there was need for a training environment. This would be a framework that we could train our models from.

3.2.1 Previous repositories

Luckily, academics in the artificial intelligence domain realize that open-sourcing their work will be an advantage to others that want to continue on the path of previous research. The main papers that influenced our training process both have open source Github repositories. Jonathan Frankle has prepared an entire training framework for the lottery ticket family of papers, open_lth [10]. The SynFlow researchers have their own repository, Synaptic-Flow [24]. Since the Synaptic-Flow repository is a bit more easily modulated, it is the repository that is used most for this research.

3.2.2 Model

The model used is ResNet20, also used in the Lottery Ticket Hypothesis. This model is implemented on PyTorch and trained using a GPU.

Our work can be found here [4].

Network	Dataset	Training Epochs	Batch	Optimizer	Accuracy
ResNet-20	CIFAR-10	160	128	SGD	70.98
ResNet-20	CIFAR-10	160	128	Momentum	85.27

3.2.3 Hyperparameters

Hyperparameters were unchanged from the original SynFlow ResNet-18 ones [25]. These parameters were the same for every model trained.

Learning rate	Learning rate drops	Drop factor	Weight decay
0.01	60, 120	0.2	5×10^{-4}

3.2.4 Dataset

Since we are training a neural network, we need something to train it on. Previous research has mostly focused on images, so that is the path we will follow. To illustrate the adaptability and general applicability of their algorithms, previous research also trained models for multiple datasets of different scales. MNIST [17], CIFAR-10 [15] and ImageNet [16] are the main three datasets that an algorithm's accuracy would be derived from. Since this is a small thesis with a limited time period, we will keep it at one dataset and a simple one at that. MNIST is a popular machine learning dataset of handwritten digits, which would be ideal to use here. But it turns out that models trained for MNIST are already stable to SGD noise at initialization most of the time. Since the instability requires a more complicated dataset, we will be using CIFAR-10. CIFAR can be considered a precedent to the more popular ImageNet, with the same objective to introduce more image classes with more differences between them as a challenge for the models trained on them. ImageNet is currently seen as the quintessential dataset to compare your models to, but due to time constraints we will just focus on the smaller CIFAR-10.

Chapter 4

Results

4.1 Threshold

With the repository and dataset in place, we can continue setting up our training framework. As was stated before, the instability analysis was not part of the open_lth repository. Much of the development time of this thesis was spent programming the instability analysis and experimentally determining a threshold to compare the outcome of this instability analysis to.

Linear mode connectivity used the iterative magnitude pruning (IMP) method to go back and determine instability after training. So the original workflow of finding the threshold for subnetworks would be to first train the full network, then iteratively rewinding and pruning to find a pruning mask that can then be applied to the rewound network at an early iteration k from where instability analysis can be performed. Since SynFlow does not rely on rewinding to prune, in this research the pruning mask is found after pre-training for k iterations instead.

To find a threshold, a full network was pre-trained to iteration k, pruned and then instability analysis was performed. For linear interpolation, we used 20 evenly-spaced values for alpha. The first of the two lines in figure 4.1 shows instability analysis performed on full networks, while the other is for subnetworks. Instability analysis on full networks means that there was no pruning being done between pre- and post-training.

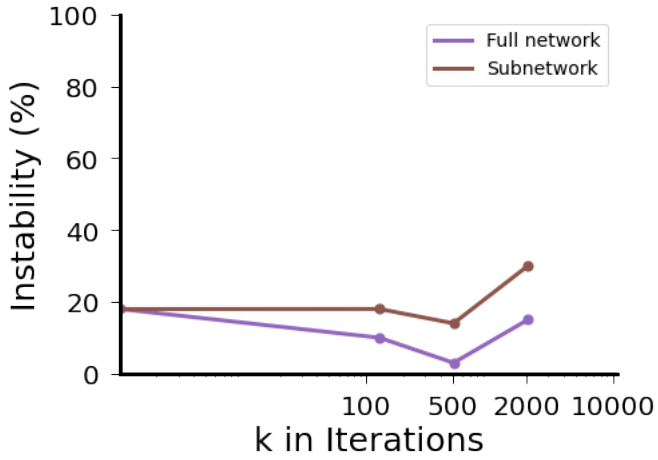

FIGURE 4.1: Instability analysis.
Instability analysis on full and subnetworks of ResNet20, trained on CIFAR-10. The analyses were performed after pre-training up to different iterations k. These models were all trained once based on a single initialization.

In interpolating the two temporary models, we expect unstable models to cause a growth in error through the different values for alpha. Frankle et al. document this change as being very large, a difference of up to 80% in error between the worst interpolated model and the best temporary model. Interestingly, throughout this research the size of this difference could not be replicated once using SGD as an optimizer. When seeing this divide between our recreation and the instability analysis in linear mode connectivity, and aiming to exclude all possible errors in this recreation, the only conclusion that remained was to ascribe the divide to the differences in training setup.

If we take two temporary, fully-trained yet unstable subnetworks as independent, local or global optima, the surrounding modes or areas of these two networks will *decrease* in accuracy when *increasing* in diversity from these optima [9]. This phenomenon is described well by the linear interpolation we perform. After a certain amount of training, the interpolated models between the two subnetworks must decrease in accuracy - given that the subnetworks are unstable. Interestingly, the diversity and independence between the two temporary subnetworks still allows for a similarity in accuracy. This is because they train to independent modes or areas in the loss landscape. Independence here stands for a disagreement and diversity in predictions, even though the two subnetworks can have similar accuracies.

Now, the reason for initial confusion was that the subnetworks trained with SynFlow ended up in flatter modes, where the difference when interpolating was smaller compared to Frankle et al.'s results using IMP. Through experimentation, it was found that this can be attributed to the optimizer used in our earlier research. We used SGD as opposed to the momentum optimizer used in *Linear mode connectivity*. SGD has been known and recently proven [5] to regularize not (only) to a minimum, but a mode or area where the surrounding curvature is low. Even more so, the SynFlow algorithm is inspired by path-SGD [21] to prune based on a more generalized weight magnitude, that considers the network's parameters before and after it. Training using both SynFlow as a pruner and SGD as an optimizer leads to a very generalized or regularized loss landscape, with networks that as a result are more stable at all stages of training (see 4.2).

To conform to the instability analysis of Frankle et al., the optimizer was changed to a momentum optimizer. Momentum is SGD that uses the gradients of multiple steps, as opposed to the default single most recent step, in its consideration of the next step. This leads to less meandering and SGD cycles, which on their part lead to a more severe instability *if* the network is unstable [23]. The difference can be found in figure 4.2. Since instability using this optimizer resembled Linear mode connectivity, experiments were continued using this optimizer. The outcome of these experiments can be found in 4.3, which comes short due to a time constraint. Given more time and the shape of the graph, a stable subnetwork is expected to be found.

Since experimentation had to be stopped, a final comparison was done using the model at the lowest available instability value found. The training traject and its accuracy is shown in figure 4.4.

With momentum, the results seem to reflect the *Linear mode connectivity* paper better. If we establish stability as being lower than 2%, the threshold does occur much later in training using SynFlow. Such a significant divide, from reaching the threshold at a number of iterations of around 2000 to a number of epochs consisting of 50,000 iterations each, could very well be related to the inferior pruning mask that SynFlow ends up using.

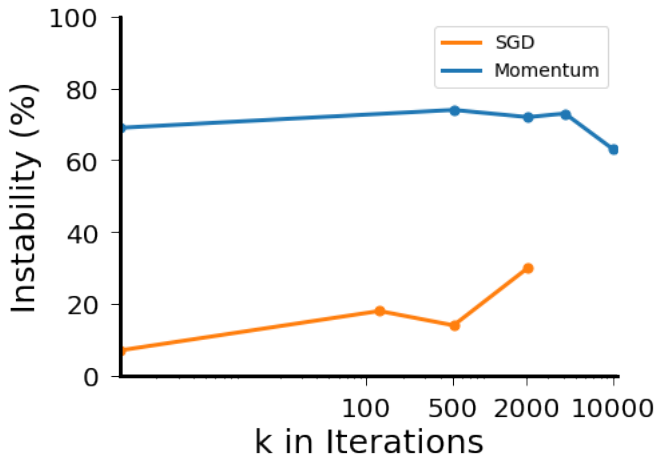

FIGURE 4.2: Optimizers.
Instability analysis on subnetworks with the use of different optimizers. ResNet20 models trained on CIFAR-10. The analyses were performed after pre-training up to different iterations k and then pruning. These models were trained once based on two separate initializations. Momentum was extended for more iterations, as the value did not converge yet.

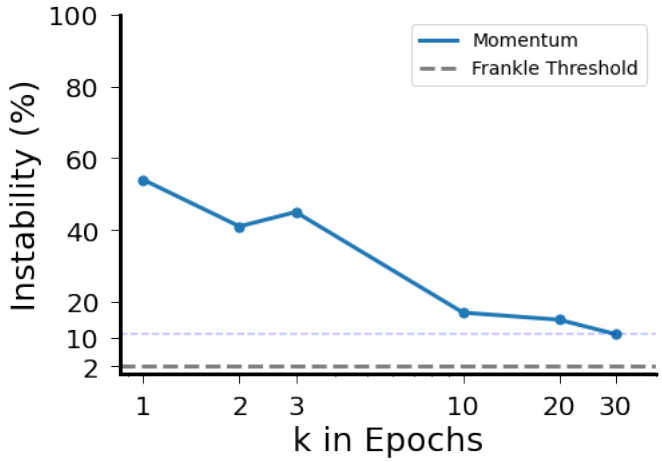

FIGURE 4.3: Momentum.
Instability analysis on subnetworks using the momentum optimizer. As can be seen, momentum is trending downwards in the same manner as expected from Linear mode connectivity. In their paper, Frankle et al. settle on a threshold of 2%. Sadly a time shortage crept in and the threshold was not reached, ending at a very late 11.3% at epoch 30.

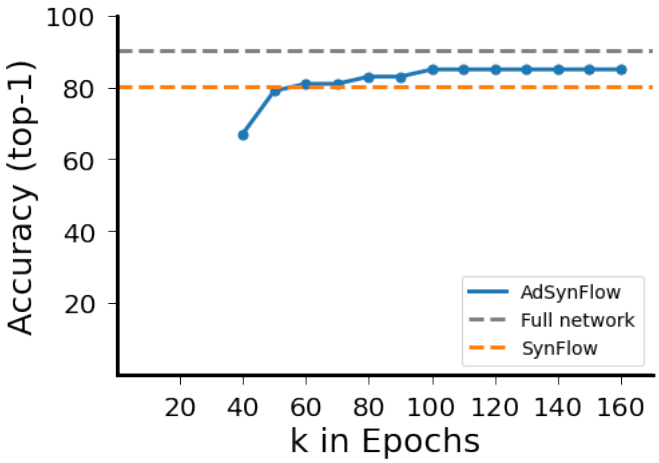

FIGURE 4.4: AdSynFlow comparison.
AdSynFlow accuracy tracked throughout training. This network was pruned at epoch 30, giving the lowest instability found as seen in figure 4.3. The model accuracy was not captured during pre-training, so that data was lost. Though SynFlow was overcome early in training, the network stabilizes and does not match the full network yet at the end of training.

Chapter 5

Discussion

Our final results leave a lot of room for discussion. Sadly, a lot of these results have not been confirmed through multiple training runs or slight changes in parameterization or training setup. The lack of time really crept into the final results because of multiple setbacks.

The first of these setbacks was of course the missing instability analysis implementation. Because of this, the first weeks were mostly spent trying to recreate and reproduce the instability analysis results of Frankle et al. in [12]. After getting that to work, most of the remaining time was spent establishing an instability threshold through experimentation. Given that the average training time for a model was around 3-4 hours, this took a tremendous amount of time. Especially when considering the next setback.

Training using the SGD optimizer did not give the expected results. Starting off close to stability, training only increased instability. This was the entire opposite of both the Linear mode connectivity paper and intuition. The later in training a model is, the lower chances should be to end up in a different valley. Although switching to momentum did follow a more similar path to Linear mode connectivity, it was more stretched out. This stretching is likely due to SynFlow itself, especially the fact that it prunes during training. A better pruning mask makes a subnetwork more likely to find a valley fast, while SynFlow might be more dependent on a well-trained full network. To be sure of this, future work will be needed to confirm it.

As for further suggestions for future work or just general remarks about the work surrounding this thesis, there is lots of angles to look from. One such point is that a subnetwork that is trained for long enough might always find the local or global minimum that all subnetworks trained from an iteration k are bound to reach. After a certain number of epochs trained, it might be the case that the instability analysis is less of a test of stability and more one of accuracy compared to another network in the same valley. Our hyperparameters resembled the original SynFlow hyperparameters, which were not fumbled with. The assumption that our instability analysis is a test of stability, is thus dependent on the hyperparameters used in the original SynFlow research and an upper limit to the amount of iterations of pre-training.

Now that we are on the point of hyperparameters, it might be worth it to state again that the original SynFlow hyperparameters were not changed. One might wonder if the hyperparameters carry over well to AdSynFlow, if the general assumption of the two networks is not the same. If these adaptations would include changes to the training process now that this process is no longer restricted to pruning before training, maybe the hyperparameters should have been subject to adaptations too.

If the instability analysis is apparently so dependent on a certain type of optimizer, one might wonder if the analysis is a good analysis of stability at all. Subtle changes

to the training setup leading to such large differences in 'stability' might be an indication this is not the case. Should an instability analysis be stable itself?

Finally, the loss landscape seems to be a big black hole for a lot of research, not being focused on while not being completely understood at the moment. The Deep Ensembles paper [9] also went into this, but it might also be interesting to look into it from more of a pruning perspective.

Chapter 6

Conclusion

The results presented here seemingly confirm the *Linear mode connectivity*'s statement that subnetworks can only match their full networks if they are stable to SGD noise. A confirmation of this statement may encourage other research to shift their focus from initialization pruning to pre-training pruning, as the difference in training time is negligible seeing the increase in accuracy.

Our final results might sadly not be all-embracing. Though there was improvement on SynFlow, stability was not reached at the threshold Frankle et al. established. Since the optimizer problem was only found later during the thesis, there was no time to concretely establish the problems and differences between instability analyses and training setups in this research and the research by Frankle et al. This means that all in all AdSynFlow, though an improvement through pre-training, is not a final and satisfying adaptation of SynFlow.

Lottery tickets and their almost guaranteed existence at early points in training enable us to prune more effectively during pre-training. Especially when stable to SGD noise are we able to leverage on the knowledge of their existence. Adapting SynFlow to only prune after stability, allows us to consistently get accuracies that are at least matching that of the full network. So to answer the research question, the knowledge of lottery tickets enables us to prune more effectively once we can say for certain that these tickets are existing within the network.

To establish a certainty that AdSynFlow improves on SynFlow, the research needs to be extended to different datasets, models and optimizers. A minimum would be to expand to MNIST using LeNet and to ImageNet using ResNet-50. The ideas presented here could also be used to adapt other successful initialization pruning algorithms such as SNIP and GraSP. Besides that, designing a dedicated pre-training pruning algorithm is maybe a better option. Frankle et al. already stated in their paper on initialization algorithms [13] that early pruning methods might not even work with signals such as magnitude, pointing to work that works with ever-changing pruning masks that in this way exploit signals from a later training stage [8].

Touching one last time on the relevance of this work to artificial intelligence, this research seems to be in line with other research on the topics presented. Taking from and building on this research surrounding neural networks and other topics such as pruning, this research contributes to these topics in a tangible way. To conclude, this research is in that way connected to artificial intelligence.

Bibliography

[1] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165.

[2] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006, pp. 267–269.

[3] Davis Blalock et al. *What is the State of Neural Network Pruning?* 2020. arXiv: 2003.03033.

[4] Cas Bolwerk. *Adapted-Synaptic-Flow*. https://github.com/casbolwerk/Synaptic-Flow. 2021.

[5] Pratik Chaudhari and Stefano Soatto. *Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks*. 2018. arXiv: 1710.11029.

[6] Yves Chauvin. "A Back-Propagation Algorithm with Optimal Use of Hidden Units". In: *Advances in Neural Information Processing Systems*. Vol. 1. 1989. URL: https://proceedings.neurips.cc/paper/1988/file/9fc3d7152ba9336a670e36d0ed79bc43-Paper.pdf.

[7] Gal Chechik, Isaac Meilijson, and Eytan Ruppin. "Synaptic pruning in development: a computational account". In: *Neural computation* 10.7 (1998), pp. 1759–1777.

[8] Utku Evci et al. *Rigging the Lottery: Making All Tickets Winners*. 2020. arXiv: 1911.11134.

[9] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. *Deep Ensembles: A Loss Landscape Perspective*. 2020. arXiv: 1912.02757.

[10] Jonathan Frankle. *open_lth*. https://github.com/facebookresearch/open_lth. 2020.

[11] Jonathan Frankle and Michael Carbin. *The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks*. 2019. arXiv: 1803.03635.

[12] Jonathan Frankle et al. *Linear Mode Connectivity and the Lottery Ticket Hypothesis*. 2020. arXiv: 1912.05671.

[13] Jonathan Frankle et al. *Pruning Neural Networks at Initialization: Why are We Missing the Mark?* 2021. arXiv: 2009.08576.

[14] Song Han et al. "Learning both Weights and Connections for Efficient Neural Network". In: *Advances in Neural Information Processing Systems*. Vol. 28. 2015. URL: https://proceedings.neurips.cc/paper/2015/file/ae0eb3eed39d2bcef4622b2499a05fe6-Paper.pdf.

[15] Alex Krizhevsky and Geoffrey E. Hinton. "Learning Multiple Layers of Features from Tiny Images". In: *Tech Report, University of Toronto* (2009).

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[17] Yann LeCun. "The MNIST database of handwritten digits". In: *http://yann.lecun.com/exdb/mnist/* (1998).

[18] Yann LeCun and Yoshua Bengio. "Convolutional Networks for Images, Speech, and Time-Series". In: *Michael A. Arbib The Handbook of Brain Theory and Neural Networks* (1995), pp. 255–258.

[19] Yann LeCun, John Denker, and Sara Solla. "Optimal Brain Damage". In: *Advances in Neural Information Processing Systems*. Vol. 2. 1990. URL: https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

[20] Michael C Mozer and Paul Smolensky. "Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment". In: *Advances in Neural Information Processing Systems*. Vol. 1. 1989. URL: https://proceedings.neurips.cc/paper/1988/file/07e1cd7dca89a1678042477183b7ac3f-Paper.pdf.

[21] Behnam Neyshabur, Ruslan Salakhutdinov, and Nathan Srebro. *Path-SGD: Path-Normalized Optimization in Deep Neural Networks*. 2015. arXiv: 1506.02617.

[22] Alex Renda, Jonathan Frankle, and Michael Carbin. *Comparing Rewinding and Fine-tuning in Neural Network Pruning*. 2020. arXiv: 2003.02389.

[23] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2017. arXiv: 1609.04747.

[24] Hidenori Tanaka and Daniel Kunin. *Synaptic-Flow*. https://github.com/ganguli-lab/Synaptic-Flow. 2020.

[25] Hidenori Tanaka et al. *Pruning neural networks without any data by iteratively conserving synaptic flow*. 2020. arXiv: 2006.05467.