On Multiple Winning Tickets and LTH vs SNIP: Final Report

G070 (s1519734, s1512959, s1991337)

Abstract

Pruning a neural network aims to reduce its number of parameters by removing the ones deemed "unimportant" according to a criterion, while at the same time aiming to preserve the performance of the model. Two recent papers have distinguished themselves from the rest. (Frankle & Carbin, 2018) come up with an algorithm (LTH) which preserves or even improves the network's performance after pruning, while (Lee et al., 2018) propose a pruning method (SNIP) which does not require any training whatsoever.

In this report, we find out that there might not be multiple "winning tickets" on the same architecture, for a given weight initialisation. We also compare the performances of the methods in the two papers on a fully-connected architecture and a convolutional one on classification tasks using the MNIST and CIFAR10 datasets, finding out that LTH consistently outperforms SNIP. Finally, we come up with a modified version of SNIP which might have potential on large architectures.

1. Introduction

From small fully connected neural networks such as LeNet300-100 (266K learnable parameters) which were used in the past to modern neural networks such as VGG (Simonyan & Zisserman, 2015) (20M parameters), we can see that the size of state of the art neural networks has been constantly increasing. A growing concern is that those neural networks are more overparametrized than they should be (keeping in mind that overparametrization does improve generalization). This inefficiency, which leads to higher training times and higher storage requirements, is the reason why pruning methods have recently become popular and why there has been a significant effort in the field to come up with novel ones.

In this report, we focus on two recent pruning methods: the one described in "The Lottery Ticket Hypothesis" (the "LTH" method) (Frankle & Carbin, 2018) and the method described in "Single-Shot Network Pruning Based on Connection Sensitivity" (in short, "SNIP") (Lee et al., 2018).

Our research questions are as follows:

1. Our first aim is to discover whether we can find multiple separate "winning tickets" (defined in 2, separate means that they have no weights in common whatsoever) on the same architecture and weight initialisation. As far as we know, this has not been explored yet in any of the recent LTH follow-up papers and the "Lottery Ticket Hypothesis" does not state how many "winning tickets" might exist. If we find multiple distinct "winning tickets", we could investigate what they have in common which would allow us to find them more efficiently. Another motivation for the experiment is to highlight whether certain weights are more important than others in LTH, for a given architecture and weight initialization.

2. The second aim of our project is to investigate how SNIP and LTH fare against each other on two small architectures, LeNet-300-100 and Conv2 (presented in 5), both starting from the same weight initialisations, which as far as we know hasn't been done before. Furthermore, we try to find out whether SNIP can also find subnetworks which can be considered "winning tickets".

3. Finally, we tried to improve SNIP by coming up with an iterative version of it and tested this new method on our two architectures, comparing it with the original one.

2. Background and Related Work

Overparametrization in neural networks is known to improve the neural network's generalization power, but it is not well-understood. However, while some deep learning practitioners try to improve their models by simply "adding more layers", other researchers have instead focused on trying to reduce the size of neural networks while maintaining their performance and generalization power. In general, the core idea of pruning is to define a measure or criterion that ranks the weights in terms of 'importance' (a "saliency criterion") and then remove a certain percentage of the most unimportant weights.

Some time ago, (Cun et al., 1990) and (Hassibi et al., 1993) tried to develop methods of removing unnecessary weights based on the Hessian matrix of the loss function with respect to them. However, computing the Hessian matrix for modern neural networks is intractable due to their size (some approximations do exist (Fairbank & Alonso, 2011), (Martens et al., 2012)). Pruning methods have come a long way since then.

Recently, (Frankle & Carbin, 2018) presented an algorithm that identifies small sub-networks which can achieve better performance than the initial, unpruned networks, in less training time, with as little as 7% of the original network's weights remaining. Such small-subnetworks were aptly called by the authors "winning tickets". However, finding a "winning ticket" is resource-intensive as a model needs to be iteratively pruned and then re-trained, which is very costly for large neural networks such as VGG.

Usually, pruned models have a lower performance than their unpruned counterparts. Thus, it's easy to see why LTH was a breakthroughand thus why it sparked a multitude of follow-up papers, trying to find out why and how LTH works. (Zhou et al., 2019) investigate different pruning criteria which can be applied to the "Lottery Ticket Hypothesis" algorithm and also investigate freezing the pruned weights at their initial values (before training) instead of 0. (Jose et al., 2019) experimentally find out that lottery tickets can be found faster, by using only a percentage of the whole training set (in some classification tasks on CIFAR10 and MNIST). (Frankle et al., 2019), a follow-up paper made by the authors of LTH further refines the algorithm, proving that a winning ticket can be found by resetting the weights (after pruning) to their values at a customly chosen early iteration "k" instead of iteration 0, leading to better results on large neural networks (ResNet-50).

The main drawback of LTH was the cost involved in reaching a sparse "winning ticket". (Lee et al., 2018) came up with SNIP, an algorithm which prunes a network before training, using only one mini-batch of training data, one forward pass and one backward pass.

We present the algorithms in section 4.

3. Datasets and Task

3.1. Datasets

We chose the same datasets (Frankle & Carbin, 2018) and (Lee et al., 2018) used in their experiments, MNIST and CIFAR10 and used standard pre-processing methods on them.

The MNIST dataset consists of 70.000 28x28 black-and-white images which illustrate hand-written digits (includes all the digits 0-9) out of which, 60.000 are training images and 10.000 are test images, and is considered a classic dataset for evaluating classification models. We created a validation set by sampling 5.000 images uniformly at random from the training set and then removing them from the training set. The dataset was normalised at loading time using torchvision's "Normalize" function, with mean 0.1307 and standard deviation 0.3081.

The CIFAR-10 dataset consists of 60.000 32x32 colour images representing objects from 10 different classes (e.g: "dog", "cat", etc), with 6.000 images per class. There are 50.000 training images and 10.000 test images and as for the previous dataset, we created a validation set by sampling at random, without replacement, 5.000 images from the training set. The dataset was normalised at loading time using torchvision's "Normalize" function, with RGB channel means (0.4914, 0.4822, 0.4465) and standard deviations (0.2023, 0.1994, 0.2010).

3.2. Task

Our tasks are directly derived from the research questions we tried to answer. The algorithms mentioned in this section will all be formally presented in section 4, while the architectures we have used, the exact details of the experiments we have done and their results will be presented in section 5. Our experiments mirror the ones made by (Frankle & Carbin, 2018) and thus have been done using the fully-connected "LeNet-300-100" architecture on MNIST and the convolutional "Conv2" architecture on CIFAR10.

We wanted to investigate whether "winning tickets" are unique for a given architecture and a given initialization. Specifically, we were curious if we can obtain "winning tickets" on the same architecture and weight initialization that have zero weights in common. For this objective, we created the "LTH_flip" algorithm, a slightly modified version of LTH and investigated whether it can produce different "winning tickets" for the two architectures we considered.

We wanted to compare SNIP and the original version of LTH on the two architectures (and respective datasets) we consider. To make the comparison as fair as possible, for a given run and model we used the same weight initialization as the starting point and pruned the same number of weights using the LTH and then the SNIP algorithms. The algorithms will be compared based on the performance of the pruned networks they produce. The performance of such a network will be a function of how fast the network finished training (based on the early stop iteration) and of the top accuracy achieved on the test set.

As we found out that SNIP performs worse than LTH in almost all of the experiments we had made, we tried to see if we can improve it. Thus, we (slightly) modified SNIP obtaining the "iterative SNIP" algorithm and tested it on the two architectures. We then compared the results with the ones obtained by running the original SNIP algorithm, for the same weight initialization on the respective architectures (similar to how we did the SNIP vs LTH comparison).

4. Methodology

4.1. LTH

The "Lottery Ticket Hypothesis" (Frankle & Carbin, 2018) is the novel claim that a randomly-initialized dense neural network contains a sub-network 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. If we adopt the notation in (Frankle & Carbin, 2018), the LTH algorithm is as follows:

1. Let $f(x;\theta)$ be a dense neural network.
2. Randomly initialize the neural network with initial

parameters θ_0 and save them.

3. Train the network for j iterations, arriving at parameters θ_j.

4. Prune $p\%$ of the parameters in θ_j (according to some criterion; in our report and in the original LTH paper the lowest-magnitude weights are pruned), creating a mask m.

5. Reset the remaining parameters to their values in θ_0, obtaining the network $f(x; \theta_0 \otimes m)$ and freeze the pruned weights $1 - m$ to zero.

6. (Optional, if doing iterative pruning): Repeat from step 3. If not, re-train the resulting network and see if it is a "winning ticket".

(Frankle & Carbin, 2018) find that iterative pruning can be used to find smaller "winning tickets" and generally outperforms one-shot pruning a large number of weights (not repeating step 3 in the above algorithm).

4.2. LTH_flip

As mentioned in 3.2, this method attempts to find completely different "winning tickets" on the same architecture and weight initialisation. It is a slightly modified LTH:

1. Initialize a neural network $f(x; \theta_0)$.

2. Apply the above, iterative, version of the LTH algorithm for k iterations obtaining mask m and identifying winning ticket $f(x; m \otimes \theta)$.

3. Compute mask $m' = 1 - m$.

4. Re-initialize the network $f(x; m' \otimes \theta_0)$.

5. Apply LTH algorithm again to the network $f(x; m' \otimes \theta_0)$ and obtain mask s.

6. Train the networks $f(x; m.\otimes\theta_0)$ and $f(x; s\otimes\theta_0)$, decide if they are "winning tickets" or not and compare their performances.

Step 3 and 4 are equivalent to a pruning iteration in which all of the weights in the winning ticket are pruned. Using the original LTH on this resulting network will allow us to see if it can obtain another "winning ticket" or not.

4.3. SNIP

(Lee et al., 2018)'s SNIP algorithm prunes a model only once, before any training is done. In order to do that, they introduce the auxiliary parameters $\boldsymbol{c}$ which have the purpose of representing the connectivity of their corresponding weight (every weight j in the network has a corresponding auxiliary parameter c_j in $\boldsymbol{c}$, which is 1 if the connection is active and zero otherwise). If θ represents the parameters (weights) of the network, j represents a random model weight and L the loss function, then the algorithm is as follows:

1. First, the network is randomly initialized with weights θ_0 and the corresponding auxiliary parameters $\boldsymbol{c}$ are all initialised to 1.

2. Then, all weights are multiplied with their auxiliary counterparts and a single minibatch of data is pushed through the net. We thus obtain the value of the loss function $L(c \otimes \theta)$.

3. Next, they calculate the derivatives of the loss function with respect to the auxiliary parameters:
$$g_j(\theta) = \frac{\partial L(\boldsymbol{c} \otimes \theta)}{\partial c_j}$$

4. The saliency measure $s_j = \dfrac{g_j(\theta)}{\sum_i g_i(\theta)}$ is then computed.

5. Finally, weights are ranked based on their corresponding saliency measures and the bottom $k\%$ are pruned.

In order to make SNIP robust to architecture changes, (Lee et al., 2018) advise that a variance scaling initialisation method must be used in step 1. This is why we used the Glorot initialisation (Glorot & Bengio, 2010) in all of our experiments.

An important thing to mention is that the derivatives g_j computed in step 3 are not equivalent to the derivatives of the loss function wrt the corresponding weights. The latter represent how the loss changes if the weights are modified by an additive, infinitesimal value δ while the former represent the loss change if the weights are perturbed by a multiplicative δ. This is the reason why methods using the latter (or weight magnitudes) as saliency reasons require the model to be pre-trained.

4.4. Iterative SNIP

Iterative SNIP is a potential improvement for SNIP we came up with after we saw that SNIP consistently underperforms relative to LTH. When we want to prune a large amount of weights from a network, SNIP does this all at once, while LTH reaches the desired pruning target iteratively, allowing the network to gradually adapt to its newly pruned weights. Also, the one-shot version of LTH underperforms compared to its iterative version and this signalled us that there might be some merit in creating an iterative version of SNIP.

The difference between SNIP and iterative SNIP is that for the latter, instead of pruning the model once to reach the target of N pruned weights, we prune the model progressively (using SNIP) according to a schedule of increasing numbers of weights to be pruned $N_1, N_2, \ldots, N$. So, we first prune N_1 weights from the model using SNIP, then we prune N_2 weights using SNIP on the resulting model and so on. As in the case of SNIP, all the pruning is done before any training takes place. Our hope was that this iterative schedule would improve the saliency measure, since at any prune iteration, the weights that had already been pruned would be ignored and thus the measure would be more accurate in ranking the remaining weights.

5. Experiments

In section 5.1, we present our architectures and the hyper-parameters which are common to all of our experiments. In section 5.2, we present how the experiments were carried out and clarify the data derived from them, leaving their motivation (where it is unclear), results and interpretation to section 6.

5.1. Setup

The two architectures that we have used are specified below in the table along with their respective hyper-parameters, which are the same across all the experiments we have done:

Network	Le-Net	Conv-2
Convolutions		64, 64, pool
FC layers	300, 100, 10	256, 256, 10
All/Conv Weights	266K	4.3M / 38K
Batch Size	60	60
Optimizer	Adam	Adam
Learning Rate	$1.2e-3$	$2e-4$
Pruning Rate	fc 20%	conv 10%fc 20%

Table 1. Architectures tested in this paper as a baseline for comparing lottery ticket hypothesis with other algorithms. Convolutions are 3x3. Lenet is from LeCun et al. (1998). Conv-2 is a variant of VGG (Simonyan & Zisserman, 2014).

	LTH	LTH_flip 1	LTH_flip 2
1	3000	7250	7250
2	6000	4250	3500
3	3750	5000	9250
4	4500	5750	3250
5	3500	4250	5000
6	3000	5250	3250
7	2500	3500	4000
8	2500	2750	3000
9	4000	3250	4750
10	3000	4750	3500

Table 2. Average early stopping iterations for the LTH, LTH_flip 1 and LTH_flip 2 models for the LeNet300-100 architecture. LTH_flip 1 corresponds to the LTH_flip model based on the LTH model at prune iteration 7, while LTH_flip 2 represents the remaining one.

We have used the same learning rates as the ones used by (Frankle & Carbin, 2018) (the original LTH paper), who have experimentally validated them. They aimed for small learning rates which would all the while lead to reasonable early stop iterations.

	LTH	LTH_flip 1	LTH_flip 2
1	4500	12000	11250
2	4750	11750	10750
3	3750	12000	14750
4	3250	12500	13500
5	2750	13000	14500
6	2750	13250	11750
7	2500	13750	11500
8	3500	12500	11250
9	3250	13000	11000
10	2500	13000	11750

Table 3. Average early stopping iterations for the LTH, LTH_flip 1 and LTH_flip 2 models for the Conv2 architecture. LTH_flip 1 corresponds to the LTH_flip model based on the LTH model at prune iteration 7, while LTH_flip 2 represents the remaining one.

Beside these, other hyperparameters that we have used in our experiments are:

- The budget of epochs which is set to 25;
- The number of pruning iterations for the LTH experiments, which is set to 10;
- The number of pruning iterations for the LTH flip experiments is set to 9, because we consider flipping the mask as being equivalent to one prune iteration;
- We use weight-magnitude pruning in all LTH and LTH_flip experiments;
- To ensure reproducibility, we use random seeds for each experiment which are then saved.

We have set the pruning rate to 20% for fully connected layers and 10% for convolutional layers because it is specified in the original Lottery Ticket Hypothesis paper (Frankle & Carbin, 2018) that for lower pruning rates of 0.1 or 0.2, the model reaches higher validation accuracy and maintains it to smaller network sizes. Using a higher pruning rate on convolutional layers would soon (after a few number of pruning iterations) make them a bottleneck (since there are significantly fewer conv weights than fc weights in Conv2), making the pruned network perform worse. The outer layer is pruned at a rate of 10% for similar reasons.

We use early-stopping to lower the training time required for the models, due to the limited time and budget (computational resources) that we had. Again, due to these considerations, we evaluate the models on the validation set every 500 iterations (1 iteration = 1 batch). If the validation accuracy does not improve in 4 evaluations, then we early stop the training (the best model is saved by default).

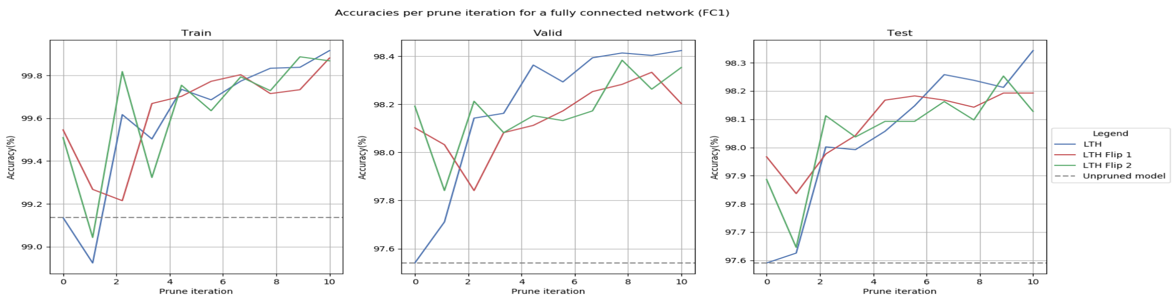

Figure 1. Accuracy%/prune iteration for a FC1 network

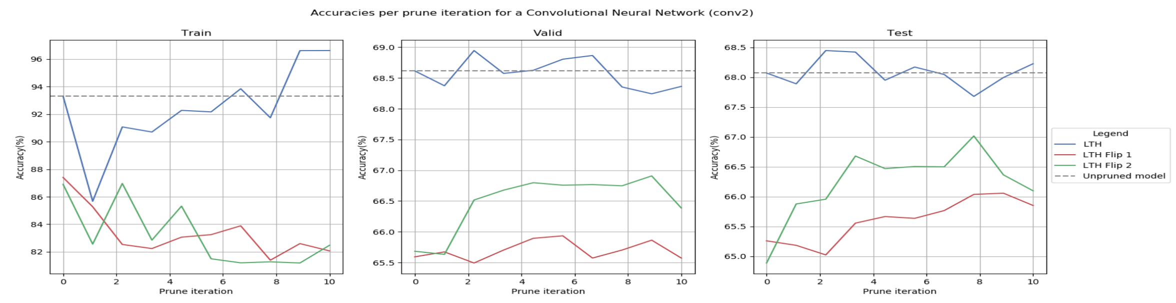

Figure 2. Accuracy%/prune iteration for a Conv2 network

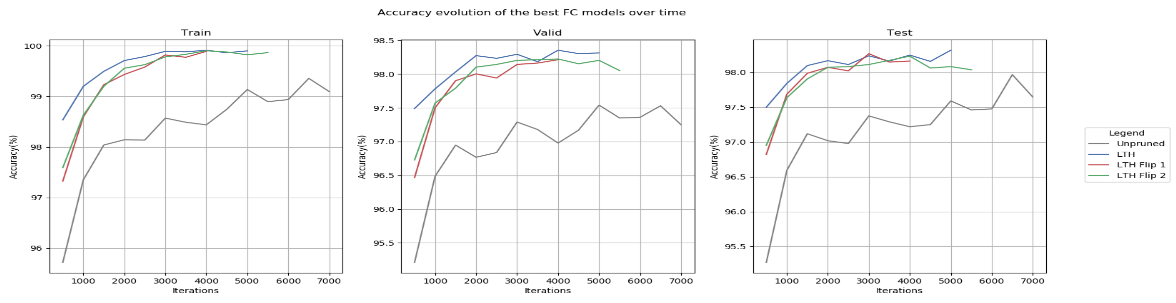

Figure 3. Accuracy evolution over time comparison for the LeNet-300-100 architecture between LTH and LTH_flip models. LTH_flip 1 corresponds to the LTH_flip model based on the LTH model at prune iteration 7, while LTH_flip 2 represents the remaining one. LTH corresponds to the average accuracy of the LTH models over all pruning iterations (1 to 10).

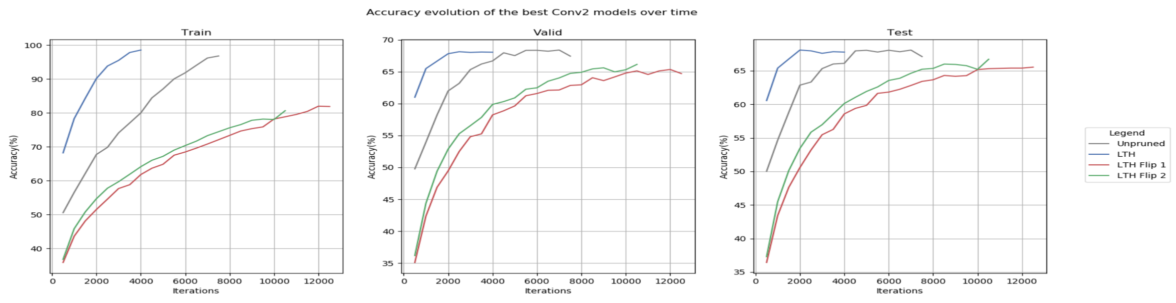

Figure 4. Accuracy evolution over time comparison for the Conv2 architecture between LTH and LTH_flip models. LTH_flip 1 corresponds to the LTH_flip model based on the LTH model at prune iteration 7, while LTH_flip 2 represents the remaining one. LTH corresponds to the average accuracy of the LTH models over all pruning iterations (1 to 10).

5.2. Experiments Description

Using the setup explained in the subsection 5.1 we begin by training our baseline LTH models. We train four total models, two for each specific architecture mentioned previously, FC and Conv2, and each of them on a random seed that we have generated. By doing this we can average the statistics obtained and hence get more confidence in our comparisons. Ideally, we should have used more than two models for generating averages, but this is what our budget allowed us.

As specified in 4.2, LTH_flip requires a mask to begin training from (by flipping it then applying the original LTH). All our LTH_flip experiments use the inverses of masks obtained by the baseline LTH models as starting masks. If the original LTH mask is large, then the initial pruning made by LTH_flip by flipping the mask will prune a large number of weights. In (Frankle & Carbin, 2018) it is specified that doing so would not lead to a "winning ticket". Thus, we should choose LTH masks with a small number of weights as starting masks for LTH_flip.

We calculated that the mask obtained after 7 LTH prune iterations has a number of unpruned weights equal to 20% of the number of weights in the original, unpruned model. This is also the pruning rate that we use in our experiments. Therefore, we have decided to train our LTH_flip models starting from the LTH masks obtained after at least 7 iterations. We train multiple models for each random seed used in LTH, each architecture (both FC and Conv2) and for masks obtained after 7 and 8 LTH prune iterations. Therefore, after training, we will have 8 LTH_flip models in total.

For fair comparison between SNIP and LTH, for a given LTH baseline model obtained after k prune iterations, we train a corresponding SNIP model with the same weight initialisation as the LTH model and we prune the same number of weights. We have 10 LTH base models per architecture (corresponding to the 10 pruning iterations we used) and thus 10 corresponding SNIP models. By using the same respective initialisations, we want to make the comparison between LTH and SNIP fair, by making their start conditions identical. This is also the reason why all the experiments using the SNIP and LTH algorithms share the same hyperparameters.

For each SNIP model corresponding to an LTH model obtained after at least 7 LTH prune iterations, we train a corresponding iterative SNIP model, for comparison. The reason why we choose those SNIP models is to see how SNIP and iterative SNIP compare when they have to prune a large number of weights (>80%), as iterative SNIP was created with the express intent of outperforming SNIP in these cases. For all iterative SNIP experiments, we use a schedule in which we prune p% of the remaining weights at each iteration. The pruning percentages are roughly the same as for the LTH baseline models: 0.2% for fully-connected layers and 0.1% for convolutional layers.

We have computed tables 6 and 7 in order to see how similar the masks obtained from the two methods are and use them as an additional comparison criterion between them. In these tables, row X corresponds to the pair comprised of the LTH model obtained after the X LTH prune iterations and its corresponding SNIP model. **Difference** is the average number of weights that are unpruned in one of them and pruned in the other, **Overlap** is the average number of weights that remain unpruned in both models, while **Kept** is the average number of weights kept in both the LTH and SNIP model.

Regarding time complexity, we evaluate the two algorithms based on early stopping iterations. Further details will be discussed in section 6.

In order to compare the performances of SNIP to LTH, we compute the differences between the accuracy of the baseline LTH and the ones obtained from SNIP and include them in two sets of graphs, one for each architecture used, in figures 5 and 6. Just like in the LTH flip experiment, in each of the set of graphs we include the accuracy difference from the Training set, Validation set and Test set, although the evaluation will be based purely on the performance on the Test set.

6. Discussion

6.1. LTH vs LTH_flip

In figure 3, we can see that the accuracies of the LTH_flip models are very close to the averaged accuracy of all LTH models for the FC architecture. However, as we can see in table 2, the LTH_flip models converge much slower on average than the LTH models. On the Conv2 architecture, the performance of the LTH_flip models is much lower than the performance of their counterparts, both in terms of best test accuracy and in terms of convergence time (for the latter, see figure 3).

These results are a weak (due to the limited number of experiments) indication that at least on the two architectures we have considered, there are no completely separate "winning tickets" (in terms of shared unpruned weights) obtainable with the LTH algorithm. These results also reinforce the idea that for a certain weight initialisation, the unpruned weights uncovered by the LTH algorithm are more special than the others and that without some unpruned weights or combination of weights present, an LTH-pruned model might never be a "winning ticket".

6.2. LTH vs SNIP

From figures 5 and 6 we observe that the results seem to be architecture-independent: accuracy-wise, the SNIP models' test accuracies lag further behind the corresponding LTH models' test accuracies the more weights are pruned. The SNIP models corresponding to LTH pruning iteration 1 models are the only pair which yield similar test accuracies. However, if we inspect tables 4 and 5 we see that for the FC architecture, the "Iteration 1" SNIP model converges

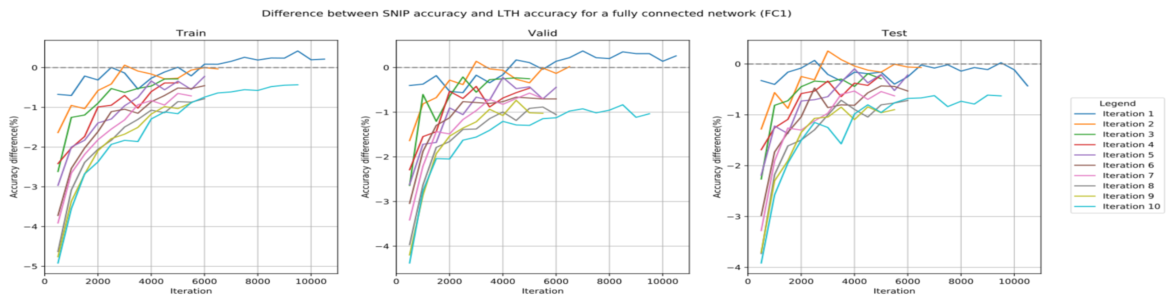

Figure 5. Average accuracy differences between LTH baseline models and their corresponding SNIP model for the LeNet-300-100 architecture.

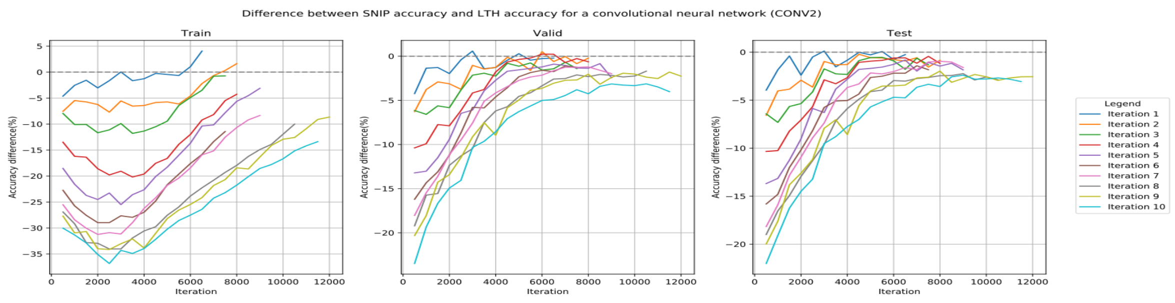

Figure 6. Average accuracy differences between LTH baseline models and their corresponding SNIP model for the Conv2 architecture.

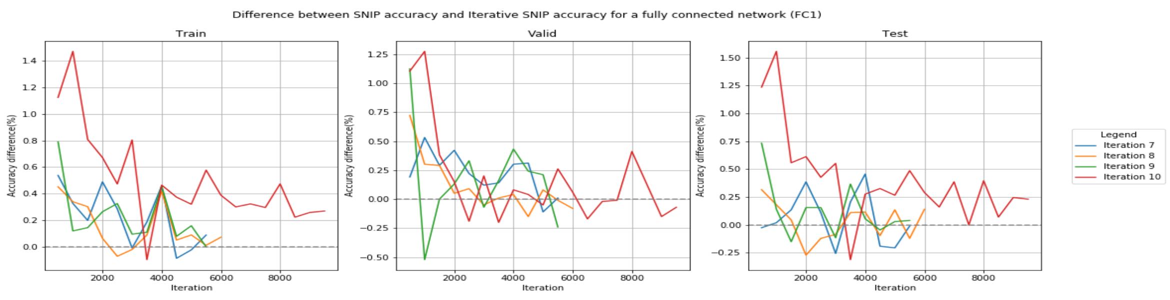

Figure 7. Average accuracy difference between our SNIP models and their iterative SNIP counterparts for the LeNet-300-100 architecture.

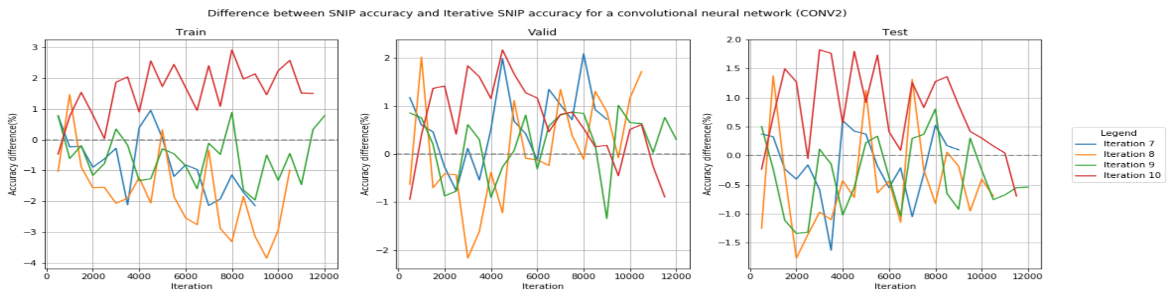

Figure 8. Average accuracy difference between our SNIP models and their iterative SNIP counterparts for the Conv2 architecture.

in twice as many iterations as its LTH counterpart, thus disqualifying it from being a "winning ticket", but that on the Conv2 architecture, the "Iteration 1" SNIP model could be considered a "winning ticket". We have our doubts regarding this particular result, as we can see from table 7 that SNIP and LTH have pruned the exact same weights, which is highly unlikely. After careful investigation, we believe that the accuracy curves are in fact correct, but an error must have happened when saving the number of unpruned weights.

From 7 and 6 we see that there is an inverse correlation between the number of unpruned weights the resulting masks of the LTH-SNIP model-pairs have in common and the pruning amount, which is to be expected given the accuracy results.

Finally, from tables 4 and 5 we see that generally LTH outperforms its SNIP counterpart in terms of convergence speed on both architectures.

6.3. SNIP vs Iterative SNIP

On the FC architecture, from figure 7 we can see that the original SNIP method tends to outperform its iterative counterpart. For the Conv2 architecture, figure 8 tells a slightly different story: the iterative SNIP method tends to outperform the original SNIP on the training set in terms of accuracy, but this accuracy advantage disappears on the validation set. However, at high training iterations, the iterative SNIP seems to outperform the original version, indicating that the iterative SNIP algorithm might yield models which generalise better.

The difference between the results on our two architectures might also indicate the possibility that iterative SNIP outperforms the original SNIP on larger architectures.

7. Conclusions

In conclusion, our experiments suggest that multiple, separate "winning tickets" might not exist on the same architecture for a given weight initialization. This observation might suggest that there might be a limited number of "winning tickets" for a given architecture and initialization and also that these "tickets" necessarily share some unpruned weights. A possible further research question would be to figure out if there are certain weights or weight combinations (again for a given architecture and initialization) that if pruned make the corresponding pruned model automatically not be a "winning ticket".

We find out that, in line with the comparative results from (Frankle et al., 2019) on VGG and ResNet, LTH outperforms SNIP on smaller networks such as LeNet-300-100 and Conv2.

The most promising result of this report is the modified version of SNIP we came up with, "iterative SNIP". We found out that iterative SNIP has similar performance to SNIP on our fully connected architecture and that it outperforms it on our convolutional architecture. Future work should investigate whether our modified version of SNIP keeps outperforming the original on larger architectures.

	LTH	SNIP	% remaining
1	3000	8750	80.03%
2	6000	5550	65.06%
3	3750	4750	51.28%
4	4500	5250	41.05%
5	3500	4000	32.86%
6	3000	6250	26.32%
7	2500	4250	21.07%
8	2500	4750	16.86%
9	4000	4000	13.52%
10	3000	8000	10.83%

Table 4. Early stopping iterations per percentage of remaining weights for a fully connected network (FC1)

	LTH	SNIP	% remaining
1	4500	4750	80.12%
2	4750	6500	64.21%
3	3750	6000	51.47%
4	3250	6250	41.27%
5	2750	7750	33.09%
6	2750	6500	26.54%
7	2500	7500	17.10%
8	3500	9500	17.10%
9	3250	10750	13.74%
10	2500	11000	11.04%

Table 5. Early stopping iterations per of percentage remaining weights for a convolutional neural network (CONV2)

References

Cun, Yann Le, Denker, John S., and Solla, Sara A. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pp. 598–605. Morgan Kaufmann, 1990.

Fairbank, Michael and Alonso, E. Efficient calculation of the gauss-newton approximation of the hessian matrix in neural networks. *Neural computation*, 24:607–10, 12 2011. doi: 10.1162/NECO_a_00248.

Frankle, Jonathan and Carbin, Michael. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018. URL http://arxiv.org/abs/1803.03635.

Frankle, Jonathan, Dziugaite, Gintare Karolina, Roy, Daniel M., and Carbin, Michael. The lottery ticket hypothesis at scale. *CoRR*, abs/1903.01611, 2019. URL http://arxiv.org/abs/1903.01611.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In Teh, Yee Whye and Titterington, Mike (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL http://proceedings.mlr.press/v9/glorot10a.html.

Hassibi, Babak, Stork, David G., Wolff, Gregory, and Watanabe, Takahiro. Optimal brain surgeon: Extensions and performance comparisons. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pp. 263–270, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

Jose, Kevin Martin, George, Nisha, and Kairanda, Navami. Experiments on lottery ticket hypothesis. 2019. URL https://navamikairanda.github.io/lottery-ticket-hypothesis/team15_report.pdf.

Lee, Namhoon, Ajanthan, Thalaiyasingam, and Torr, Philip H. S. SNIP: single-shot network pruning based on connection sensitivity. *CoRR*, abs/1810.02340, 2018. URL http://arxiv.org/abs/1810.02340.

Martens, James, Sutskever, Ilya, and Swersky, Kevin. Estimating the hessian by back-propagating curvature, 2012.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Zhou, Hattie, Lan, Janice, Liu, Rosanne, and Yosinski, Jason. Deconstructing lottery tickets: Zeros, signs, and the supermask. *CoRR*, abs/1905.01067, 2019. URL http://arxiv.org/abs/1905.01067.

8. Appendix

	Difference	Overlap	Kept
1	38875.5	174184.5	213060.0
2	52008.5	118529.5	170538.0
3	55351.5	81159.5	136511.0
4	52602.5	56679.5	109282.0
5	47573.0	39917.0	87490.0
6	41562.5	28488.5	70051.0
7	35799.0	20295.0	56094.0
8	30572.0	14351.0	44923.0
9	26018.5	9962.5	35981.0
10	22015.0	6808.0	28823.0

Table 6. Overlap between LTH mask and corresponding SNIP mask per prune iteration for a fully connected network (FC1)

	Difference	Overlap	Kept
1	659475.0	2658477.0	2658477.0
2	1187467.0	2130485.0	2130485.0
3	1610231.0	1707721.0	1707721.0
4	784838.0	584339.0	1369177.0
5	697422.5	400618.5	1098041.0
6	602400.5	278462.5	880863.0
7	506000.0	200877.0	706877.5
8	423640.5	143829.5	567470.0
9	353071.0	102677.0	455748.0
10	291549.5	74643.5	366193.0

Table 7. Overlap between LTH mask and corresponding SNIP mask per prune iteration for a convolutional neural network (Conv2)