An Exploration of the Lottery Ticket Hypothesis through Fisher Pruning and the Lens of Mode Connectivity

Matevz Matjasec
mm2676

Teodora Reu
tr500

Thomas Christie
thwc3

Yumna Naqvi
yfn21

Abstract

Since Frankle & Carbin conjectured the "Lottery Ticket Hypothesis", demonstrating that trainable sparse networks can be found with iterative magnitude pruning (IMP), much research has been focused on developing novel, computationally more efficient pruning algorithms.

We investigate the efficacy of Fisher pruning as an alternative to magnitude pruning in the iterative pruning framework. We find that Fisher pruning performs similarly to magnitude pruning, although the latter tends to find better performing sub-networks at initialisation; *supermasks*. By investigating the difference in weights chosen to prune by the two techniques, we believe that this can be partially explained by the hypothesis that pruning small weights can be viewed as a form of training, as suggested in [1].

We then use the lens of mode connectivity to further investigate the nature of the "winning tickets" produced with the iterative pruning framework. We reproduce the main findings from [2] and investigate whether this linear mode connectivity phenomenon is a general property of "winning" subnetworks by performing instability analysis on sparse subnetworks found with alternative pruning algorithms. While our findings are inconclusive they do hint that the subnetworks found with IMP are not fundamentally different from subnetworks found with alternative pruning algorithms.

Information about our codebase can be found in Appendix A.

1 Introduction

It is not a secret that many neural networks are over-parameterised [3], enabling network compression [4, 5]. Before the Lottery Ticket Hypothesis (LTH) [6], many compression algorithms were proposed [7, 8]. A popular method to reduce the dimensionality of a trained model is neural pruning. Neural pruning involves freezing (i.e. setting to zero) neural components such as weights while maintaining minimal change in the accuracy. Although the sparse networks obtained with various pruning methods worked well, training the pruned networks from scratch resulted in underperforming sparse networks.

In recent years, the LTH [6] has received considerable attention. This work presents a simple algorithm for finding *sparse subnetworks within larger networks that are trainable from scratch*. Their approach, *iterative magnitude pruning (IMP)*, works iteratively in steps. At one step they train the network completely, for a set number of iterations. After this, they pick a threshold t such that $p\%$ of the weights are smaller than t. All weights smaller than t are then set to 0, and the rest of the

Submitted group project report for R252 (Theory of Deep Learning) course .

weights are reset to their initial values. They found that after iteratively removing the majority of the network's weights, the networks still performed well, and in some cases even outperformed the original unpruned network.

This work [6] opened a line of research with scholars studying this procedure from various perspectives: (1) studying other pruning methods, and the reset values for pruned weights and remaining weights [1], (2) studying the apparition of *supermasks* (pruned networks that achieve good performance without any training) [1, 9], (3) studying the stability of such subnetworks [2], and (4) adapting such iterative pruning methods to other network architectures such as BERT [10], and Graph Neural Networks [11].

In this work, we explore the efficacy of using Fisher pruning [12] as an alternative to magnitude pruning in the iterative pruning framework introduced in the LTH paper [6]. We apply it to the same settings as [6], namely LeNet on MNIST, and ResNet-20 on CIFAR10, for a full comparison. We also carry out instability analysis of the networks as proposed by [2]. We replicate the main findings of [2] and extend the analysis by looking at alternative pruning algorithms again applied to the same settings. The contributions of this paper can be summarised as follows:

- Implemented Fisher pruning and evaluated Fisher pruning, magnitude pruning and random pruning from various perspectives: (1) the characteristics of weights chosen to prune by the different methods, (2) the accuracy achieved by the sparse networks identified, (3) the speed of training the sparse networks identified, (4) the appearance of supermasks with the different pruning techniques, and (5) the per-layer sparsity differences which arise as a result of pruning *globally*, rather than *locally* (i.e. layer-wise).
- Investigated whether stability to sample of SGD noise is an always emergent property of "winning ticket" subnetworks or a phenomenon that can only be observed when "winning tickets" are found with IMP by (1) reproducing the main results from the Linear Mode Connectivity and the Lottery Ticket Hypothesis paper [2], and (2) performing instability analysis on two diffent pruning algorithms: Gem-Miner [13], and Fisher pruning [12].

2 Fisher Pruning Background

A good heuristic for deciding which weights to prune from a model is those whose removal will result in the smallest drop in performance. Fisher pruning [12] offers a mathematically justified method of pruning which aligns well with this heuristic. In this section, we summarise the theory underlying Fisher pruning, as detailed in [12].

Formally, let us consider the task of supervised classification. Here, we have a training set comprised of N training examples, which are pairs of inputs $\mathbf{x} \in \mathcal{X}$ and corresponding output classes $\mathbf{y} \in \mathcal{Y}$. This can be interpreted probabilistically, with inputs drawn from some distribution $Q_{\mathbf{x}}$ and corresponding outputs drawn from a conditional distribution $Q_{\mathbf{y}|\mathbf{x}}$ [14].

The goal is to then construct a model, parameterised by weights $\mathbf{w} \in \mathbb{R}^d$, whose output can be interpreted as a probability distribution $P(\mathbf{y}|\mathbf{x}; \mathbf{w})$ (commonly achieved by applying the softmax function to the output of our model[15]), such that the model's output distribution is close to the true conditional distribution of the dataset, $Q_{\mathbf{y}|\mathbf{x}}$. Typically this is done by updating the model weights to minimise a loss function $\mathcal{L}$, and in this case an appropriate loss function is the cross-entropy loss, defined as:

$$\mathcal{L}(\mathbf{w}) = \mathbb{E}_{\hat{Q}}[-\log P(\mathbf{y}|\mathbf{x}; w)] \tag{1}$$

where the expectation is typically taken with respect to the training data distribution, $\hat{Q}$.

Having such an expression for the loss parameterised by the weights of the model is useful, as we can use it to approximate how much the loss would change as a result of altering, or pruning, a given parameter, using a Taylor expansion. In alignment with the notation used by [12], if we denote the gradient of the loss function as $\mathbf{g} = \nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w})$, and the Hessian matrix as $\mathbf{H} = \nabla^2_{\mathbf{w}}\mathcal{L}(\mathbf{w})$, then we can construct a 2nd order Taylor expansion for the change in loss corresponding to pruning the kth parameter (i.e. setting it to 0). Again using the notation used by [12], if we denote the vector entirely

populated with 0s, except for at its kth entry, where it is equal to 1, as $\mathbf{e}_k$, then the approximation of the change in loss is:

$$\mathcal{L}(\mathbf{w} - w_k \mathbf{e}_k) - \mathcal{L}(\mathbf{w}) \approx -g_k w_k + \frac{1}{2} H_{kk} w_k^2 \tag{2}$$

If we assume that pruning is done once the model has been trained to a good performance, and hence the parameters lie in a minimum in the loss landscape, then the first term is eliminated [16]. This leaves us with the diagonal of the Hessian, H_{kk}, to deal with, which can be approximated by using the Fisher information matrix, F, for efficiency [14]. In the case that the conditional distribution of the model, $P(\mathbf{y}|\mathbf{x}; \mathbf{w})$ is equal to the true conditional distribution of the training data $\hat{Q}_{\mathbf{y}|\mathbf{x}}$, then it has been shown that the Fisher and Hessian matrices are the same [17]. This assumption can be used for our iterative pruning since we prune once the model has been trained to sufficiently good performance.

The diagonal of the empirical Fisher information matrix, $\hat{F}$, calculated using an expectation with respect to the training data distribution, $\hat{Q}$, consisting of N training points, is defined as:

$$H_{kk} \approx \hat{F}_{kk} = \frac{1}{N} \sum_{n=1}^{N} g_{nk}^2 \tag{3}$$

where g_{nk} is the derivative of the loss of the kth parameter with respect to the nth datapoint. Substituting this approximation for H_{kk} into 2, and removing the first term on the RHS due to the assumption of the model being at an optimum in the loss landscape, yields the following approximation for the increase in loss when pruning the kth parameter:

$$\mathcal{L}(\mathbf{w} - w_k \mathbf{e}_k) - \mathcal{L}(\mathbf{w}) \approx \frac{1}{2N} w_k^2 \sum_{n=1}^{N} g_{nk}^2 \tag{4}$$

as derived in [12]. We can easily plug this pruning method into the iterative pruning framework; at each iteration of pruning, we greedily prune the $p\%$ of unpruned weights which correspond to the lowest estimated increase in loss. However, it isn't mathematically clear whether this technique should offer an advantage over magnitude pruning in the iterative pruning framework. This is because the Taylor expansion for the approximated increase in loss assumes that unpruned weights are kept at their same values; with the iterative pruning framework we reset them to their initial values, so the mathematical justification underlying this algorithm is somewhat diminished in this context.

3 Fisher Pruning Evaluation

We evaluate Fisher pruning against two other pruning methods: random pruning and magnitude pruning, on two models. Firstly we use LeNet on MNIST, with each experiment replicated three times to increase the reliability of the results. For the second model, we use ResNet-20 on CIFAR10. Because one pruning run of this model took $\sim$ 9hrs to run on a Google Colab [18] Pro GPU (usually T4 or P100), we decided to experiment with different settings for the ResNet, rather than replicating the same experiment three times. When pruning, we pruned 20% of parameters at each iteration. The other parameters used are listed in table 1 or in the description of each experiment.

Model	Model Type	Dataset	BatchSize	Optimizer	LR	Training Steps	Layers to ignore	Momentum
LeNet	300_100	MNIST	60	Adam	0.0012	50k iterations	-	-
ResNet	20	CIFAR10	128	SGD	0.1	30k iterations	fc.weight	0.9

Table 1: Hyper-Parameter settings for experiments.

3.1 How Different are the Weights Chosen to Prune by Fisher Pruning compared to Magnitude Pruning?

Before blindly comparing the performance of Fisher pruning with magnitude pruning, it made sense to probe how much the two pruning criteria would differ. In the case of Fisher pruning, we can see in

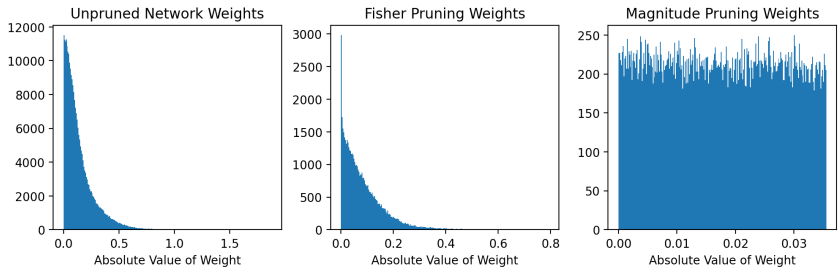


Figure 1: Histograms of absolute values of weights selected to be pruned by Fisher pruning and magnitude pruning on a LeNet trained on MNIST, with 20% of weights to be pruned. The histogram of all the weights in the network is included for comparison. There is a clear difference in weights chosen by Fisher pruning and magnitude pruning, with the former selecting some far larger weights to prune than the latter.

equation 4 that the estimated increase in loss corresponding to pruning weight w_k is proportional to w_k^2. As we greedily prune the weights corresponding to the lowest estimated increase in loss, we might expect that this results in Fisher pruning selecting the smallest weights to prune, and hence doesn't differ much from magnitude pruning.

In order to evaluate this question, we trained a single LeNet on the MNIST dataset for $50,000$ iterations, at which point its test accuracy reached $\sim 98\%$. We then used Fisher pruning and magnitude pruning to select 20% of the weights to prune and plotted a histogram of the absolute values of the weights selected to be pruned, as shown in figure 1. Clearly, Fisher pruning doesn't merely select the $p\%$ of weights with the lowest absolute magnitude but selects weights across a range of magnitudes. For comparison, the largest weight selected to prune by Fisher pruning in the above experiment had a magnitude of 0.665, compared to 0.036 for magnitude pruning. Therefore, it is reasonable to expect that Fisher pruning may behave quite differently from magnitude pruning.

3.2 How does the Accuracy of Pruned Models Change as they become more Sparse?

In figure 2 we plot the test accuracy obtained by pruned networks against the percentage of weights remaining. We do this for random pruning, magnitude pruning and Fisher pruning. For the latter two methods, we experiment with both resetting weights to their initial values between pruning iterations, as well as randomly re-initialising them (rand reinit), to explore the role that initialisation plays in identifying winning tickets.

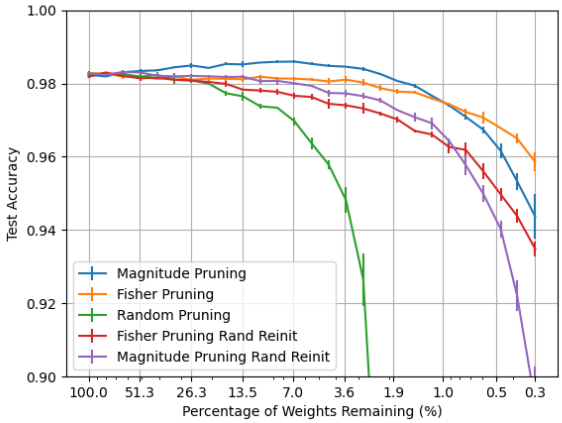


Figure 2: Test accuracy at the end of training for models with varying degrees of sparsity for the pruning methods. This plot is obtained for the LeNet-300-100 architecture on MNIST.

Clearly, both Fisher pruning and magnitude pruning vastly exceed the performance of random pruning, as expected. Fisher pruning and magnitude pruning are both similarly effective, although

they seem to have differing strengths. On the one hand, for networks with a percentage of unpruned weights exceeding 1%, magnitude pruning is more effective. For many sparsities in this range, the winning tickets identified by magnitude pruning actually *exceed* the performance of the unpruned network, whilst Fisher pruning tends to identify tickets that more closely match the performance of the unpruned network. However, once the percentage of unpruned weights drops below 1%, Fisher pruning starts obtaining better results than magnitude pruning, yielding a test accuracy of 96% for networks with only 0.3% of weights remaining unpruned.

Another interesting phenomenon to observe is the role that re-initialisation plays for both Fisher and magnitude pruning, particularly in the very sparse regime. For both Fisher and magnitude pruning, resetting the weights to random values, rather than their initial values, between pruning iterations reduces the accuracy of winning tickets, confirming the importance of re-initialisation as identified in [6]. However, once the percentage of weights remaining drops below 1%, randomly re-initialising weights has a bigger negative impact on magnitude pruning than it does for Fisher pruning. This difference is interesting, as the theory underlying Fisher pruning doesn't take into account the fact that unpruned weights are reset to their initial values.

A final observation is that the results we obtained for magnitude pruning were slightly better than the original LTH paper [6]. However, we noticed that they were pruning *locally* (i.e. pruning $p\%$ of weights from each individual layer), whereas we were pruning *globally*. Global pruning seems to offer an advantage and is explored further in section 3.5.

3.3 How does the pruning method chosen affect the training speed of identified tickets?

Figure 3 shows how the test error of pruned networks evolves as they're trained. As was the case in the original LTH paper [6], it is clear to see that with magnitude pruning, up to a certain threshold, sparser networks are able to reach a higher test accuracy than the unpruned network, with fewer iterations of training. Interestingly, with Fisher pruning, models with between $10 - 100\%$ of weights remaining seem to all reach similar accuracy with a similar number of training iterations; the sparsity induced by this technique doesn't seem to offer quite as much of a training advantage as with magnitude pruning. However, we found that more of a gap did become apparent for Fisher pruning once we moved onto the ResNet-20 architecture, seen in section 3.6.

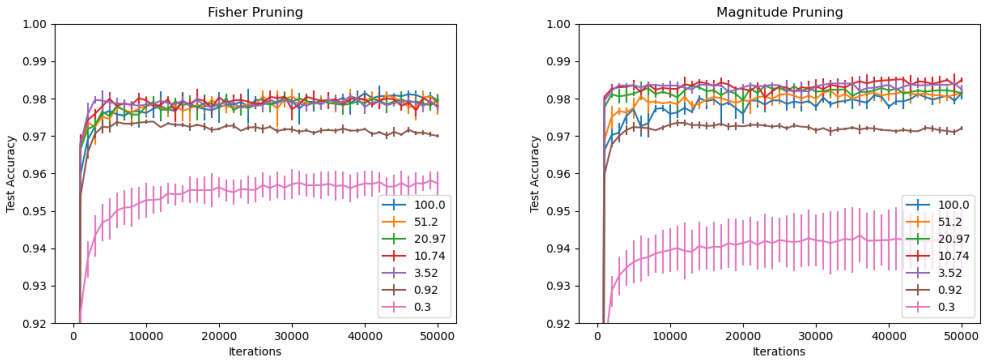


Figure 3: Evolution of test error for networks with various sparsity levels, obtained with Fisher pruning and magnitude pruning on the LeNet architecture. Lines are labelled with the percentage of unpruned weights in the model.

3.4 Supermasks

The appearance of *supermasks*, binary masks which, when applied to untrained networks in order to prune them, achieve better than random test performance, was noted in [1]. In light of this, we investigated how well the masked (but untrained) networks performed with the masks identified over the course of our pruning investigation, as visualised in figure 4.

We obtained similar results to [1] for magnitude pruning, with the untrained network with 2.7% weights remaining achieving over 0.6 accuracy. Fisher pruning also discovered supermasks, with

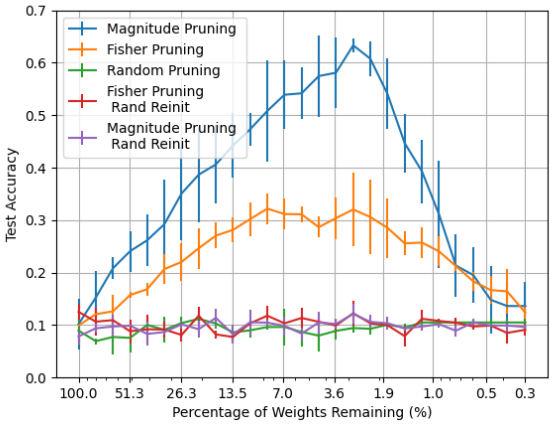


Figure 4: Supermask propagation for the three pruning methods. The model used is LeNet on MNIST.

the untrained network with 2.7% weights remaining to achieve a mean accuracy of 0.32 across the three training runs. We can say, with statistical significance ($p \approx 0$), that this is better than random guessing on the test set.

It is interesting that there is such a gap in the best supermasks identified by magnitude pruning and Fisher pruning. In [1], the authors hypothesise that when pruning the weights with the *smallest* magnitude (i.e. magnitude pruning), this can be viewed as similar to training, as the mask tends to move these weights in the direction they follow during training (i.e. towards zero). We hypothesise that this may also explain the reason that Fisher pruning doesn't produce such "super" super-masks. As demonstrated in figure 1, Fisher pruning doesn't just prune the weights with the smallest magnitude but also chooses some larger weights to prune. We suggest that by resetting these larger weights to 0, we aren't necessarily resetting these weights in the direction they followed during training, hence reducing the power of the supermasks produced. However, it is notable that supermasks are no longer present if we randomly reinitialise the weights of the networks instead of resetting them to their initial values; the reason for this remains unclear.

3.5 Is Global Pruning Uniform Across Layers?

Finally, in order to explain the difference in results obtained by us for magnitude pruning compared to the original LTH paper [6], as described in section 3.2, we noted that we performed pruning *globally*, rather than *locally*. This begs the question - how different are the sparsities of different layers when performing global pooling?

In table 2 we note the layer-wise sparsities of the LeNet networks pruned to 0.3% remaining weights globally. Clearly, pruning doesn't occur uniformly between layers; a far greater proportion of weights are left unpruned in the final layer of the network. We hypothesise that this may explain the superior performance of *global* magnitude pruning compared to *local* magnitude pruning; by not specifying the layer-wise sparsity of networks, we enable pruning to leave a greater proportion of weights unpruned in more critical layers of the network.

	Layer 1	**Layer 2**	**Layer 3**
Fisher Pruning	$0.184(\pm 0.003)\%$	$0.756(\pm 0.021)\%$	$13.800(\pm 0.003)\%$
Magnitude Pruning	$0.146(\pm 0.001)\%$	$0.901(\pm 0.017)\%$	$18.467(\pm 0.643)\%$

Table 2: Mean (and standard deviation) percentage of weights remaining in each layer of the LeNet network when pruned to 0.3% weights remaining globally.

3.6 Performance on Resnet-20

After verifying that Fisher pruning was indeed able to identify lottery tickets in the LeNet-300-100 architecture, we decided to evaluate its performance when applied to a Resnet-20 model trained on

the CIFAR-10 dataset. We also experimented with two different re-initialisation schemes. With the first, after each iteration of pruning, we *reset* unpruned weights to their randomly initialised values, as in the original LTH paper [6]. With the second, we instead *rewound* weights to their values from 2000 iterations into training, as subsequent work [2] found this offered an advantage for more complex architectures.

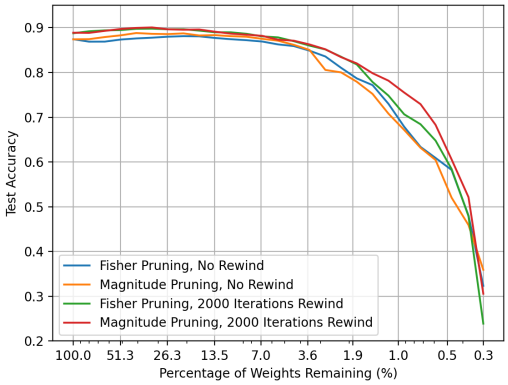


Figure 5: Comparison of test accuracies obtained by Resnet-20 models of different sparsities using both magnitude pruning and Fisher pruning. We also compare *rewinding* weights to their values 2000 iterations into training, rather than *resetting* them to their randomly initialised values. A zoomed-in version of this plot can be found in appendix B.

Figure 5 shows how the test accuracy obtained by the Resnet-20 models changed as they were pruned. Similar to the LeNet architecture, it is clear to see that Fisher pruning and magnitude pruning achieve similar performance across the different sparsities. Also, *rewinding* weights, rather than *resetting* them to their initial values, seems to improve performance, particularly once the networks become particularly sparse, as found in [2].

In terms of the evolution of the test accuracy of winning tickets over iterations of training, the story is similar to the LeNet model. Up to a certain threshold, the pruned models are able to reach *higher* test accuracies with *fewer* iterations of training, as seen in figure 6. For Fisher pruning, this gap in performance compared to the unpruned model seems to be more pronounced than was the case with the LeNet architecture.

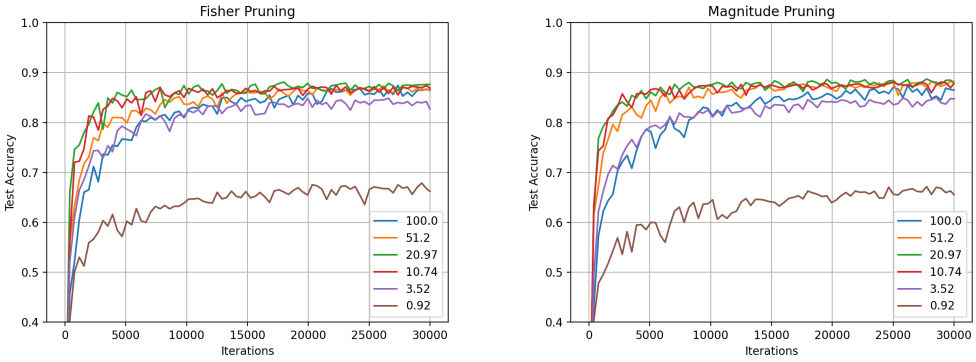


Figure 6: Evolution of test error for Resnet-20 networks with various sparsity levels, obtained with Fisher Pruning and Magnitude Pruning. Note that when pruning the models in this plot, the model weights were *reset* to their initial values in between pruning iterations, as in the original LTH paper [6]. Lines are labelled with the percentage of unpruned weights in the model.

Perhaps the most interesting difference between the experiments performed on the LeNet model and the ResNet model is that, despite using the same pruning methods on both, the masks generated for the ResNet model weren't *supermasks*. That is, upon applying the mask and resetting the model's

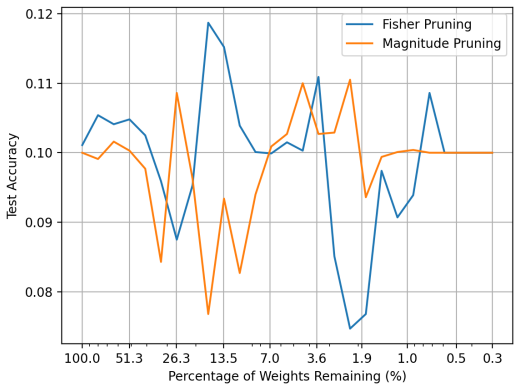


Figure 7: Test accuracy of the Resnet-20 models with a mask applied to the *untrained* network weights over different sparsities. The resulting networks have an accuracy no better than random; no supermasks have been found.

weights to their initial values, this masked model performed no better than the randomly initialised model, as was the case in figure 3. This can be seen in figure 7.

4 Mode Connectivity

In this part of the project, we investigate the LTH through the lens of mode connectivity. Specifically, we consider the implications of the findings in [2], which suggest that subnetworks identified through iterative magnitude pruning (IMP) are only successful when they are stable to stochastic gradient descent (SGD) noise, introduced as a result of factors such as random ordering of data.

In the original LTH paper, IMP fails to find winning tickets for complex model architectures, including ResNet 20 and VGG-19. [2] seek to explain the successes and failures of their IMP experiments with linear mode connectivity. Importantly, they find that subnetworks are only matching when they are stable to SGD noise. Subnetworks that achieve the same performance as their corresponding unpruned networks are referred to as “matching”, rather than "winning tickets" as [2] rewind weights to iteration $k > 0$, instead of resetting them to their random initial values as [19] initially proposed. The study proposes instability analysis to understand if a network is stable to SGD noise, where linear mode connectivity is the method for determining instability.

The methodology proposed by [2] for determining if a network is stable to SGD noise is visualised in figure 8. The process involves creating two copies of a network $\mathcal{N}$, with the same randomly initialised weights W_0. These two copies are then trained with different samples of SGD noise to produce trained weights W_T^1 and W_T^2. The study then linearly interpolates between the resulting trained networks to assess whether the models are connected by a linear path of non-increasing error, a phenomenon termed linear mode connectivity.

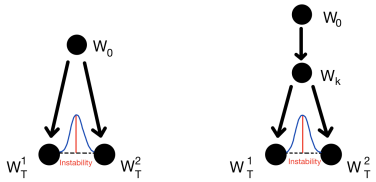


Figure 8: Diagram representing the interpolation experiment, from [2]. The left diagram shows training two copies of neural network $\mathcal{N}$ from the same initialisation but with different samples of SGD noise, whereas the second shows firstly training a model $\mathcal{N}$ to iteration k, then creating 2 copies of this model and training them until completion with different SGD noise [2].

The instability of $\mathcal{N}$ to SGD noise is defined as the maximum increase in error along this linear path p between the two minima [2] (red line in figure 8). $\mathcal{N}$ is considered stable to SGD noise when

instability is ≈ 0. Formally, the error barrier height on the linear path between W_1 and W_2 is defined as $\xi_{sup}(W_1, W_2) - \xi_{ave}(W_1, W_2)$, where $\xi_{sup}(W_1, W_2)$ is the maximum error along the linear path, and $\xi_{ave}(W_1, W_2)$ is the mean error.

Gem-Miner The Gem-Miner algorithm [13] is different to the approach offered by IMP as it focuses instead on finding subnetworks at initialization, rather than the time-consuming process of training, pruning, and re-training that IMP requires. In order to do so, Gem-Miner computes normalised scores associated with each of the weights in the network, such that the scores determine their importance. Backpropagation is then used to update these scores, and the rounded scores are used as a mask.

5 Mode Connectivity Experiments

We investigate whether the correlation between matching subnetworks and their stability is a general phenomenon that can be observed for any pruning algorithm that finds matching subnetworks or a phenomenon that can only be observed when pruning with IMP. [2] find that matching subnetworks are not stable to SGD noise when models are pruned randomly or when they are randomly initialised. We seek to compare IMP against two different pruning algorithms: Gem-Miner, and Fisher pruning.

We start by replicating the main findings outlined in [2], using IMP without rewind on LeNet and ResNet-20. We then extend the findings of [2] by finding subnetworks for ResNet-20 with Gem-Miner and Fisher pruning, linearly interpolating between the discovered subnetworks and performing instability analysis as outlined in [2].

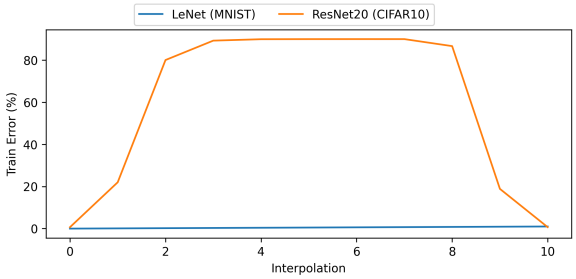
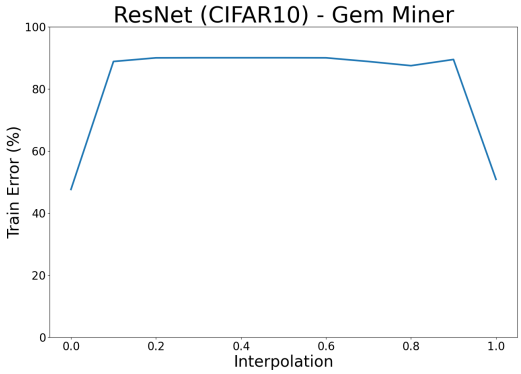


Figure 9: The training error along the linear path p between two equally initialised pruned subnetworks trained with different samples of SGD noise. The figure shows the interpolation experiment on LeNet MNIST and ResNet-20 on CIFAR10. While LeNet is clearly stable to SGD noise, ResNet-20 is not when weights are rewound to iteration 0.

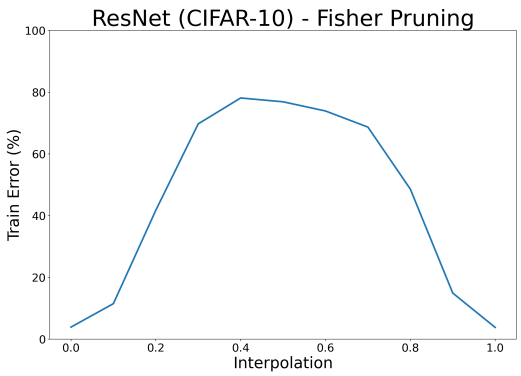
As can be seen in figure 9, the experiment from [2] was successfully reproduced. When rewinding the weights back to their initial values ($k = 0$) and training the two copies of sparse subnetworks with different samples of SGD noise only LeNet is stable. The ResNet-20 has no such linear mode connectivity between the minima found by two subnetworks. This figure matches the result in [2].

As can be seen in figure 10a, the two ResNet-20 subnetworks found with Gem-Miner also do not demonstrate linear mode connectivity when trained from the same initialisation. When moving along the linear path p, the training error increases even more rapidly than it does on ResNet-20 pruned with IMP. This finding suggests that the choice of IMP as a pruning algorithm does have some effect on a network's instability.

Finally, we test the same experiment with Fisher pruning, linearly interpolating between two ResNet-20 subnetworks trained to the same sparsity as the IMP experiments (16.8%). Results can be seen in figure 10b. The error barrier along this linear path is also non-zero, meaning that the network is not stable to SGD noise. However, unlike in the Gem-Miner experiment, the instability levels along the linear path between the two minima are very similar to those of IMP subnetworks in figure 9 .



(a) Train error along the linear path p between two equally initialised pruned subnetworks trained with different samples of SGD noise. The figure shows the interpolation experiment for RESNET-20 on CIFAR10. The algorithm used for pruning was Gem-Miner [13]. RESNET-20 was found to be unstable to SGD noise. The experiment follows the procedure as outlined in [2].



(b) Train error along the linear path p between two equally initialised pruned subnetworks trained with different samples of SGD noise. The figure shows the interpolation experiment for RESNET-20 on CIFAR10. The algorithm used for pruning was Fisher Pruning [12]. RESNET-20 subnetworks were found to be unstable to SGD noise. The experiment follows the procedure as outlined in [2].

Figure 10: ResNet-20 on CIFAR10 experiments.

6 Conclusion

As our project investigated the lottery ticket hypothesis from two perspectives, we have arrived at two sets of conclusions. Fisher Pruning did not seem to offer much of a benefit over magnitude pruning in the evaluation settings we used 1, although it was not much worse, and still far better than random pruning. Whilst it seeks to prune weights whose removal will lead to the lowest increase in loss, the approximation for the increase in loss assumes that the non-pruned weights won't change value. This clearly isn't the case when they are reset to their initial values, as in the iterative pruning framework.

Another interesting finding is related to our supermask experiment in section 3.4, in which the supermasks generated by Fisher pruning were much weaker than magnitude pruning. We explained this by inspecting the histograms of the magnitudes of the pruned weights 1, seeing that Fisher pruning would prune some significantly larger weights than magnitude pruning. We believe that these two observations potentially reinforce the hypothesis proposed in [1] that pruning small weights can be viewed as a form of training, and that perhaps setting the larger weights identified by Fisher pruning to 0 is detrimental to the network's performance, leading to the reduced performance of its discovered "supermasks". We also saw that resetting weights to their initial values was a crucial part of forming supermasks, an observation for which we have no concrete explanation, and which would be interesting to investigate further.

As for the mode connectivity experiments, we have successfully replicated the main findings from the [2] and showed that while IMP subnetworks exhibit linear mode connectivity on small models, they fail to do so on larger models. Our analysis of the instability of subnetworks found with alternative pruning algorithms is very much inconclusive. While the instability of subnetworks found with Gem-Miner hints at the possibility that sparse subnetworks found with IMP may be inherently different from sparse networks found with other pruning algorithms, analysis of the instability of Fisher subnetworks shows the opposite. Both Fisher and IMP subnetworks exhibit similar levels of instability across the linear path between the 2 minima. Further studies should investigate whether the variant of RESNET-20 with lower learning rates or learning rate warmup (hyperparameters proposed by [19] to find matching subnetworks) exhibit linear mode connectivity when pruned with alternative algorithms. This phenomenon has been observed for IMP pruned subnetworks [2]. Further, the instability analysis of non-matching subnetworks could be performed to disentangle the properties of "winning tickets" from non-matching subnetworks.

7 Individual Contributions

We split this project into two sub-projects - the first investigating Fisher pruning and the second investigating mode connectivity. Thomas and Teodora worked on Fisher pruning, whilst Matevz and Yumna worked on mode connectivity. Individual contributions are as follows:

1. **Thomas** - Software (implemented Fisher pruning and assisted Yumna in writing some mode connectivity code), conceptualisation of the Fisher pruning investigation, running experiments for Fisher pruning and analysing/plotting the results, as well as writing up several sections of the Fisher pruning part of the project and parts of the abstract/conclusion analysing Fisher pruning.
2. **Teodora** - Software (implemented Random Pruning), run multiple experiments on Fisher Pruning and plotted/analysed results, write up the introduction and the evaluation for fisher pruning for some of the experiments. Review.
3. **Matevz** - Proposed the research objective for the linear mode connectivity project and carried out the necessary research. Ran some experiments for mode connectivity (did not end up being used in the report) and plotted figure 9. Analysed results, wrote up linear mode connectivity section, abstract and conclusion referring to linear mode connectivity part of the project.
4. **Yumna** - Software (replicated and implemented mode connectivity work and newer pruning algorithms ultimately not featured in this work), ran multiple experiments for mode connectivity and newer pruning algorithms. Plotted results, edited/contributed to write-up of mode connectivity section.

References

[1] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *Advances in neural information processing systems*, 32, 2019.

[2] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pages 3259–3269. PMLR, 2020.

[3] Yann N Dauphin and Yoshua Bengio. Big neural networks waste capacity. *arXiv preprint arXiv:1301.3583*, 2013.

[4] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. *Advances in neural information processing systems*, 26, 2013.

[5] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838*, 2018.

[6] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[7] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[8] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.

[9] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11893–11902, 2020.

[10] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, 33:15834–15846, 2020.

[11] Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. A unified lottery ticket hypothesis for graph neural networks. In *International Conference on Machine Learning*, pages 1695–1706. PMLR, 2021.

[12] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

[13] Kartik Sreenivasan, Jy-yong Sohn, Liu Yang, Matthew Grinde, Alliot Nagle, Hongyi Wang, Kangwook Lee, and Dimitris Papailiopoulos. Rare gems: Finding lottery tickets at initialization. *arXiv preprint arXiv:2202.12002*, 2022.

[14] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.*, 22(241):1–124, 2021.

[15] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[16] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

[17] Alexander Ly, Maarten Marsman, Josine Verhagen, Raoul PPP Grasman, and Eric-Jan Wagenmakers. A tutorial on fisher information. *Journal of Mathematical Psychology*, 80:40–55, 2017.

[18] Ekaba Bisong. *Building machine learning and deep learning models on Google cloud platform*. Springer, 2019.

[19] Jonathan Frankle and Michael James Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. 2019.

A Technical Details

The code used for the evaluation of Fisher pruning can be found at `https://github.com/Thomas-Christie/lth` while the code for the stability analysis experiments can be found at `https://github.com/yu202147657/lth`, both of them are forks of `https://github.com/facebookresearch/open_lth`, a repository written by Jonathan Frankle, author of the original LTH paper, which facilitated the running of pruning experiments.

B Plots Fisher Pruning

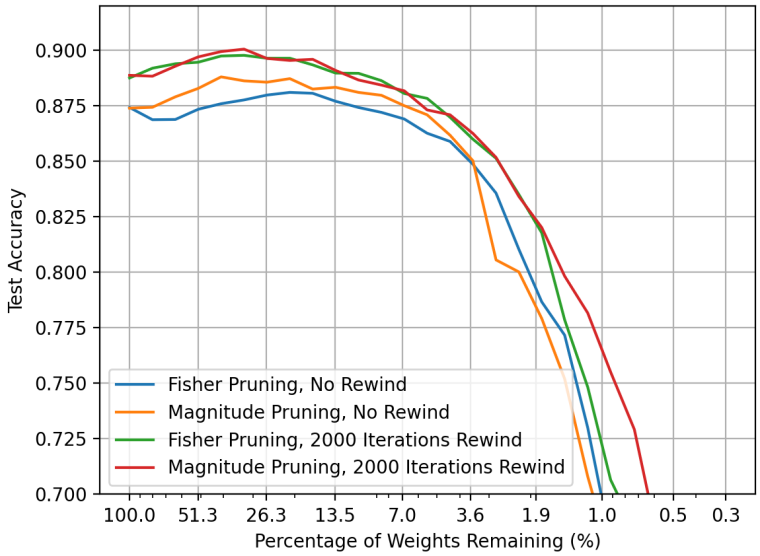


Figure 11: Zoomed in version of figure 5