COMP6258 REPRODUCIBILITY CHALLENGE PRUNING NEURAL NETWORKS AT INITIALIZATION: WHY ARE WE MISSING THE MARK?

Pooja Vijayakumar, Tunchanok Ngamsaowaros & Juran Guo
University of Southampton
{pv1n21,tn1u22,jg4n22}@soton.ac.uk

1 INTRODUCTION

Pruning in neural networks often involves training the model and then eliminating parameters without significantly affecting its accuracy. Lee et al. (2019) proposed a method of pruning at initialization, which involves pruning weights of neural networks in a single shot before training. Due to resource constraints in training these large models, pruning neural networks at initialization aims to reduce computation resources while maintaining performance. The original paper, Frankle et al. (2021) compared the efficiency of pruning using SNIP, GRaSP and SynFlow to baselines such as random pruning and magnitude pruning. The performances of these 3 methods to baselines, when pruning is done both at initialization and after training with different ablations, are compared.

2 METHODOLOGY

2.1 PRUNING

Given a network with layers $\ell \in \{1,...,L\}$, weights $w_\ell \in \mathbb{R}^{d_\ell}$ in each layer, pruning produces binary masks $m_\ell \in \{0,1\}^{d_\ell}$, and a pruned subnetwork has weights $w_\ell \bigodot m_\ell$ (Frankle et al., 2021). The sparsity $s \in [0,1]$ of a network represents the portion of weights pruned (Frankle et al., 2021). A mini-batch of 100 samples was randomly selected with 10 images for each label and used for calculating pruning scores. Then, four pruning methods were used.

Magnitude The magnitude of weight is used as its scores, then prune the lowest scores.

SNIP computes gradients g_ℓ for each layer weights and the lowest scores overall were removed with scores calculated as absolute values of element-wise multiplication of weights and the gradients (Frankle et al., 2021). Hence, weights with the highest effect on loss were maintained.

GRaSP computes the Hessian-gradient for each layer weights. Scores are calculated by $z_l = -w_l \odot h_l$ (Frankle et al., 2021). Then prune the highest s scores where s represent the sparse ratio.

SynFlow replaces the weights w_l with $|w_l|$. It computes the sum R of the logits on an input of 1's and the gradients $\frac{dR}{dw_l}$ of R. It issues scores $z_l = |\frac{dR}{dw_l} \odot w_l|$ and removes weights with the lowest scores. It prunes iteratively for 100 iterations (Frankle et al., 2021).

2.2 ABLATIONS DURING INITIALIZATION

To gain more insight into these pruning methods, 3 ablation experiments were conducted as follows.

Random shuffling First, the binary masks m_ℓ used for pruning in each layer were shuffled and then pruning will be done based on the shuffled masks. Frankle et al. (2021) stated that if accuracy stays the same, the per-weight decisions made by the method can be replaced by the per-layer fraction of weights it pruned. Otherwise, the method has determined which parts of the network to prune at a smaller granularity than layers.

Reinitialization This method samples new initial values for the pruned network from the same distribution as the unpruned network to study the sensitivity of the network to the specific initial values of weights as implemented by Frankle et al. (2021).

Inversion To study if the pruning methods preserve the most important weights when pruning to any sparsity under the hypotheses, this ablation inverses the scores and prunes in the same way. Frankle et al. (2021) stated that if the hypotheses behind these methods are correct and they are accurately instantiated as scoring functions, then inverting in this manner should lower performance.

2.3 NETWORKS, DATASETS AND REPLICATES

Frankle et al. (2021) used 4 different networks for the task of image classification on different datasets, and the experiments were repeated five times with different seeds on CIFAR-10 dataset. Due to limited computing resources available, only the results on VGG-16 network on the CIFAR-10 dataset were reproduced and the experiments were not repeated. Frankle et al. (2021) typically used 160 training iterations and depending on the sparsity level of the network, this would take 2 to 3 hours to train. In this paper, the number of training iterations was limited to 80 epochs to reduce the training time. For example, a VGG16 network trained and tested on CIFAR-10 dataset with 90% sparsity took about 2 hours to complete for 160 iterations, while it only took about 1 hour to complete 80 iterations.

There appear to be no significant changes to accuracies when the model was trained up to 80 epochs compared to 160 epochs. Improvement in accuracy occurred for 160 training iterations when the learning rate was reduced by 0.1 at epochs 80 and 120, as implemented in the original paper. Since the VGG16 network was trained up to 80 epochs, the learning rate was dropped by 0.1 at epochs 40 and 60.

2.4 SPARSITIES

Frankle et al. (2021) used 2 range of sparsities, matching sparsities[1] and extreme sparsities[2]. Hence, both matching and extreme sparsities were used when pruning in this paper. We limit the range of sparsities used to 0.0%, 30%, 60%, 90% and 98% respectively, compared to training the models on a higher range of sparsities as in Frankle et al. (2021) to reduce computational time. However, the range of sparsities used in this reproduction is still broad to allow comprehensive comparison to be made to the results in the original paper using plots of test accuracy versus sparsity.

2.5 BENCHMARK METHODS

Frankle et al. (2021) used two benchmark methods, magnitude pruning and random pruning, to illustrate the highest known accuracies possible at different sparsities. Due to limited computational resources, only one of the baseline methods, magnitude pruning, was implemented. Magnitude pruning was used over random pruning because it produced higher accuracies over random pruning when pruning at initialization for most sparsities on the VGG16 model, according to results obtained in (Frankle et al., 2021).

3 TARGET QUESTIONS

One of the target questions in this reproduction was to understand how pruning methods; SNIP, GRaSP, SynFlow and Magnitude perform pruning when used to prune weights of the VGG16 neural network trained on CIFAR10 dataset. Then, the performances in terms of accuracies of SNIP, GRaSP and SynFlow with Magnitude pruning as a baseline were compared to study if the accuracies produced matched those in the original work for different sparsities of neural networks. Another target question was to understand how inversion, reinitialization and shuffling affect the performance of 4 pruning methods when pruning weights of VGG16 model and compare the results obtained to those in the original work. Finally, this paper also intended to study the resources needed in terms of computational time and hardware requirements to run the reproduction of parts of the original paper.

[1]Matching sparsities are those where magnitude pruning meets full accuracy after training. These are sparsities $\leq$ 93.1% for VGG-16

[2]Extreme sparsities are those beyond matching sparsities

4 REPRODUCTION DETAILS

Frankle et al. (2021) used Pytorch in the implementation of the source codes. In this reproduction, Pytorch was used and the source codes were made available on GitHub[3]

4.1 DATASET, WEIGHT INITIALISATION AND TRAINING HYPERPARAMETERS

Dataset The dataset used for training models pruned by these three methods was transformed and normalised using the same parameters used in the original source codes where the training dataset is randomly cropped, followed by horizontal flips.

Weight initialisation After defining the VGG16 model, weights were initialised using the He initialisation method for all linear and convolution layers. Random weight initialization from a uniform distribution was used for Batch normalisation layers' weights with zero bias, as in the original paper. In addition, the performance of GraSP pruning is unaffected by the use of different methodologies, such as Glorot initialization.

Training Hyperparameters All models were trained over 80 epochs using cross-entropy loss with a loss ratio scheduler. The stochastic gradient descent (SDG) with momentum and weight decay of 0.9 and 0.0001, respectively, was used as an optimiser.

4.2 PRUNING METHODS

The GraSP pruning function was implemented by loosely following the pseudocode and hyperparameters provided by Wang et al. (2020) and the corresponding GitHub repo[4] with modifications. For SNIP and magnitude pruning, functions were implemented by modifying the way the scores were calculated, as described in section 2.1. For SynFlow, the codes from Tanaka et al. (2020) GitHub repository[5] were used with modifications.

Issues and Solutions for SynFlow: When reproducing, by following the instructions in the GitHub repository, the versions of PyTorch and Torchvision Frankle et al. (2021) used were too old to be installed. The solution was to change "===" in the file to ">=".

When calculating SynFlow scores weights, Frankle et al. (2021) sum all the logits on the input of 1s, while Tanaka et al. (2020) shows that the R is calculate by $\mathbb{1}^T(\Pi_{l=1}^{L}|\theta_{\mu}^{[l]}|)\mathbb{1}$, where $\mathbb{1}$ is vector with all elements being 1, θ_{μ} represents masked parameters. However, in implementation by Tanaka et al. (2020), the same method of implementation as Frankle et al. (2021) was used. Hence, this work followed the code to calculate SynFlow scores of weights.

4.3 RESOURCES OVERVIEW

All source codes were run on Kaggle and Google Collab as these platforms provided free GPU usage. Torchbearer version 0.5.3 was installed and used to train all models. The GPU used to train all models were NVIDIA Tesla P100 GPU on Kaggle and NVIDIA Tesla T4 on Google Collab. The training and inference of the models were done over the course of 4 days, and using GRaSP, SNIP and magnitude pruning methods, it required an average of one hour to run until completion. SynFlow, on average, required 75 minutes. The lower the sparsity of the network, the higher the training time. To observe the performance of SynFlow on VGG16 on CIFAR10 with 4 different types of ablation methods on sparsities $0, 30, 60, 90, 98$, at least 1500 minutes were required.

5 RESULTS AND DISCUSSION

Magnitude, SNIP and GRaSP The result as shown in Figure 1. When sparsity increases, the accuracy of the pruned model decreases for most sparsities. For the inversion ablation, a drop in

[3] https://github.com/COMP6258-Reproducibility-Challenge/PRUNING-NEURAL-NETWORKS-AT-INITIALIZATION-WHY-ARE-WE-MISSING-THE-MARK-

[4] alecwangcq/GraSP:https://github.com/alecwangcq/GraSP

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

accuracy was to be expected. The test accuracy for inversion ablation for Magnitude and SNIP for all sparsities were at 10% and diverged significantly from the results obtained by Frankle et al. (2021), which were in ranges of 85% to 95%. The experiment was repeated for SNIP pruning method under 98% sparsity for inversion using different seed values, and the result obtained was still at 10%. The plot of accuracy across epochs showed that the accuracy of training data oscillated while the accuracy of validation data remained relatively unchanged. The model appears to have not learnt anything from the training data and was unable to generalize to the validation data. The results obtained using GraSP pruning method are consistent with those of the original research. However, overall accuracies for all sparsities under all ablations were approximately 3% lower.

SynFlow When implementing inversion, the results were not similar to those stated in Frankle et al. (2021). After pruning with inversion on each sparsity, the accuracy remains unchanged. Through repeated re-pruning and re-training, the average accuracy of inverted SynFlow was still as high as unmodified. Two ways to implement inversion were attempted: inverting scores, changing the direction of scores and threshold inequalities. However, the results remain unchanged.

The results obtained for shuffling and reinitialization for all pruning methods show that the model was not sensitive to these ablations when pruning was done at initialization, as stated by Frankle et al. (2021).

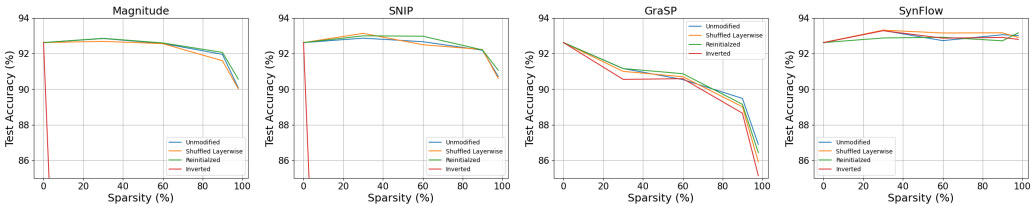

Figure 1: Test Accuracy of Ablations of Pruning Methods

6 Conclusion

The methodology of pruning at initialisation techniques discussed in this project did not require extensive development effort in implementation. However, training and testing the model required significant computing resources and time. Although most of the reimplementation results of the 4 pruning methods with ablations aligned with those reported in the original study, the question of how these pruning methods behave and are able to reach their accuracy remains open. Our result shows no significant difference in accuracy when comparing GraSP and SynFlow pruning with inversion as opposed to without, which raises the question of how the pruning scores are calculated to capture the importance of each weight. For future works, pruning can be done after initialization to study the effect of the 4 ablations on models based on different pruning methods.

References

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=Ig-VyQc-MLK`.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: Single-shot network pruning based on connection sensitivity, 2019.

Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow, 2020.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.