PHEW: PATHS WITH HIGHER EDGE-WEIGHTS GIVE "WINNING TICKETS" WITHOUT TRAINING DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse neural networks have generated substantial interest recently because they can be more efficient in learning and inference, without any significant drop in performance. The "lottery ticket hypothesis" has showed the existence of such sparse subnetworks at initialization. Given a fully-connected initialized architecture, our aim is to find such "winning ticket" networks, without any training data. We first show the advantages of forming input-output paths, over pruning individual connections, to avoid bottlenecks in gradient propagation. Then, we show that Paths with Higher Edge-Weights (PHEW) at initialization have higher loss gradient magnitude, resulting in more efficient training. Selecting such paths can be performed without any data. We empirically validate the effectiveness of the proposed approach against pruning-before-training methods on CIFAR10, CIFAR100 and Tiny-ImageNet for VGG-Net and ResNet. PHEW achieves significant improvements on the current state-of-the-art methods at 10%, 5% and 2% network density. We also evaluate the structural similarity relationship between PHEW networks and pruned networks constructed through Iterated Magnitude Pruning (IMP), concluding that the former belong in the family of winning tickets networks.

1 INTRODUCTION

Generating sparse neural networks through pruning has recently led to a major reduction in the number of parameters, while having minimal loss in performance. Conventionally, pruning methods operate on pre-trained networks. Generally, such methods use an edge scoring mechanism for eliminating the less important connections. Popular scoring mechanisms include weight magnitudes (Han et al. (2015b); Janowsky (1989); Park et al. (2020)), loss sensitivity with respect to units (Mozer & Smolensky (1989)) and with respect to weights (Karnin (1990)), Hessian (LeCun et al. (1990); Hassibi & Stork (1993)), and first and second order Taylor expansions (Molchanov et al. (2016; 2019)). More recent approaches use much more sophisticated variants of these scores (Han et al. (2015a); Guo et al. (2016); Carreira-Perpinán & Idelbayev (2018); Yu et al. (2018); Dong et al. (2017); Guo et al. (2016)).

Further analysis of pruning has showed the existence of sparse subnetworks at initialization which, when trained, are capable of matching the performance of the fully-connected network (Frankle & Carbin (2018); Frankle et al. (2019); Liu et al. (2018)). However, identifying such "winning ticket" networks requires expensive training and pruning cycles. More recently, SNIP (Lee et al. (2018)), You et al. (2019) and GraSP (Wang et al. (2020)) showed that it is possible to find "winning tickets" prior to training – but still having access to at least some training data to compute initial gradients. Furthermore, other work has shown that such subnetworks generalize well across datasets and tasks (Morcos et al. (2019); Tanaka et al. (2020)).

Some network science methods have proposed to construct sparse neural networks but without focusing on their learning performance (Shafiee et al. (2016); Prabhu et al. (2018); Kepner & Robinett (2019)). The closest approach that tackles the same problem with our work is SynFlow (Tanaka et al. (2020)). That work introduced the concept of "layer collapse" in pruning – the state when all the edges in a layer are eliminated while there exists edges in other layers that can be pruned. They proved that iterative pruning on the basis of positive gradient-based scores avoids layer collapse

and introduced an iterative algorithm and an objective function that conserves information flow and avoids layer collapse.

Our goal is to identify sparse subnetworks that perform almost as well as the fully connected network (i.e., to identify a "winning ticket") without *any* training data. Given a fully-connected and initialized architecture, and a target number of learnable parameters, we select a set of input-output paths that will be conserved in the network and prune every remaining connection. The selection of the conserved paths is based strictly on their initial weight values – and not on any training data. Then, the pruned network is trained only once. We refer to this method as PHEW (Paths with Higher Edge Weights). Our main contributions are:

1. We propose to form sparse networks through an input-output path conservation process rather than edge-based pruning (Sec 2).

2. We introduce the Edge-Weight-Product (EWP) metric for selecting paths based on their initial weights, and show that higher EWP-valued paths also have larger loss gradient magnitudes at initialization (Sec 3).

3. We empirically show that sparse networks constructed with paths of higher initial gradients converge faster and perform better (Sec 4).

4. We present a data-agnostic algorithm (PHEW) to construct a sparse network with a given number of parameters using paths of higher EWP values (Sec 5).

5. Finally, we evaluate the structural similarity relationship between PHEW networks and pruned networks constructed through Iterated Magnitude Pruning (IMP), concluding that the former belong in the family of "winning tickets" subnetworks (Sec 6.2).

2 Pruning edges vs. conserving input-output paths

Most pruning algorithms remove individual connections. At higher sparsity levels, this approach can result in the formation of "stub units", i.e., hidden units that do not have any outgoing or incoming connections, but not both. An input unit can also be a stub if it does not have any output edges – and an output unit can be a stub if it does not have any input edges. Stub units do not contribute in learning and they waste model parameters.

To avoid the emergence of stubs during pruning, we construct sparse neural networks by conserving some of the input-output paths of the fully connected architecture. The number of such paths is determined by the target number of parameters. All edges that do *not* participate in any of the conserved paths are then pruned, before the network is trained. Obviously, this approach cannot lead to the emergence of stub units.

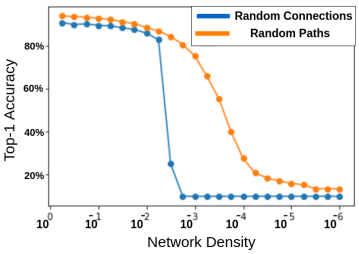

Figure 1: Comparison of random edge pruning and random input-output path selection for different network densities. Architecture: VGG19. Dataset: CIFAR-10.

Network density is the fraction of edges of the fully connected network that are present in the sparse network. In Appendix A.1, we derive the probability of stub units with random edge pruning, as a function of the network density. We also compute the expected number of stub units. Accumulation of stub units in a layer eventually leads to layer collapse. Figure 1 compares the accuracy of two networks with the same number of parameters: one constructed with random edge pruning network and another constructed with random path conservation. For a range of higher density values, the random path method performs marginally better. As the network density decreases, the edge pruning method suddenly fails because of layer collapse. The performance of the random path method on the other hand shows more gradual accuracy loss because it avoids wasting parameters in stub units and preserves the flow of loss gradients during training even at very low density values.

3 WHICH PATHS TO CONSERVE?

Having demonstrated the advantages of conserving input-output paths, we now address the pertinent question : *Of all the input-output paths in the given fully-connected architecture, which paths are better in terms of learning performance?*

We answer this question in two steps. First, we introduce the Edge-Weight-Product (EWP) metric for selecting paths based on their initial weights, and show that higher EWP-valued paths also have larger loss gradient magnitudes at initialization. Second, we empirically show that sparse networks constructed with paths of higher initial gradients converge faster and perform better.

Edge-Weight-Product (EWP) : Given an input-output path $p(i, o)$ of depth L with edges $e_l \in E, l = 1, ..., L$, the path's EWP is defined as: $\Pi(p) = \prod_{l=1}^{L} |w_{e_l}|$.

Claim : *Given two paths between the same input-output units, one with much lower EWP than the other, the path with higher EWP has a higher loss gradient magnitude.*

Let us consider two input-output paths, p_1 and p_2 as shown in Figure 2. The two paths traverse the same unit/edge at each layer, except the units/edges of a single layer. The edges have initial weights $\{w_1, w_{12}, w_{13}, w_4\} \in p_1$ and $\{w_1, w_{22}, w_{23}, w_4\} \in p_2$. We assume that all the weights in p_1 are significantly greater than 0, while the weight w_{23} of p_2 is close to 0, meaning that: (1) $\Pi(p_1) >> 0$, while (2) $\Pi(p_2) \approx 0$. We also assume that the initial loss gradient of that output $\left|\frac{\partial L}{\partial y_{out}}\right|$ is relatively high at initialization.

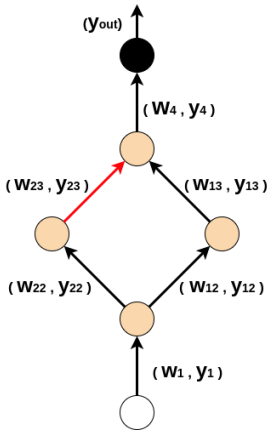

Figure 2: Two paths between the same input-output units – the left path's EWP is much lower than the right path's EWP.

The gradient propagated through $p_1 \& p_2$ is:

$$G(y_4) = \left|\frac{\partial L}{\partial y_4}\right| = \left|\frac{\partial L}{\partial y_{out}} \times \frac{\partial y_{out}}{\partial y_4}\right| = \left|\frac{\partial L}{\partial y_{out}} \times w_4\right| >> 0 \quad (1)$$

The paths now diverge, the major difference between the weight magnitudes w_{23} and w_{13}. For paths $p_1 \& p_2$, the node-gradients $(y_{k3}, k = 1, 2)$ are:

$$G(y_{k3}) = \left|\frac{\partial L}{\partial y_{k3}}\right| = \left|\sum_i \frac{\partial L}{\partial y_i^{l+1}} \times \frac{\partial y_i^{l+1}}{\partial y_{k3}}\right| = \left|\sum_i \frac{\partial L}{\partial y_i^{l+1}} \times w_{ki}^{l+1}\right| \quad (2)$$

The higher node-gradient at y_4, propagates through the paths p_1 & p_2 to the two units y_{13} & y_{23}. The terms representing this propagation in the summation (2) is, $G(y_4) \times w_{13}$ and $G(y_4) \times w_{23}$. Using equation (1), we see that the gradient propagation through the higher edge-weight w_{13} is higher than the gradient though the low edge-weight w_{23}. The higher node-gradient value $G(y_4)$, in a larger proportion is propagated through edge-weights with higher magnitudes. The detailed derivations for the remaining edges and units in $p_1 \& p_2$ are included in Appendix A.2.

Note: The loss gradients propagating through the units participating in the lower-EWP path can be high. This is possible because of other paths with higher edge-weights passing through the units.

4 PATHS WITH HIGHER INITIAL GRADIENT PERFORM BETTER

In this section we argue that paths with higher initial loss gradient form better sparse networks, and show some supporting empirical results. The argument in favor of higher initial loss gradient paths is very general: consider any problem in multi-variate optimization and suppose that, due to various constraints, we can only optimize across a subset of variables – and ignore the rest. In our context, the variables are parameters of the neural network, and the constraint on the number of optimization variables corresponds to the desired pruning ratio. A reasonable heuristic would be to keep the variables with higher initial gradient because they would, presumably, allow us to reach

the optimization objective faster. This is of course only a greedy heuristic – and it would be easy to construct counter-examples. As we show experimentally, however, this heuristic is quite effective in practice.

Together with the previous intuition, some prior work has also utilized gradient-based initialization schemes for training very deep fully-connected networks(Saxe et al. (2013); Poole et al. (2016); Yang & Schoenholz (2017); Xiao et al. (2018)). In GraSP (Wang et al. (2020)), the authors argue that the preservation of gradient propagation by maximizing the gradient norm at initialization leads to larger loss reduction. They also analyzed the effect of maximizing gradient norm in the Neural Tangent Kernel (Jacot et al. (2018)) representation of DNNs and showed advantages in training dynamics and faster convergence.

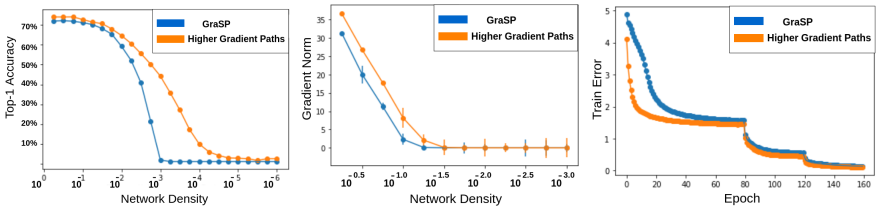

Figure 3: Comparing GraSP with a network formed by a set of paths with higher initial gradient magnitude. The plots show classification accuracy, gradient norm, and training error (10% network density). Network: VGG19. Dataset: CIFAR-100.

We illustrate the previous claim empirically in two steps. First we construct a sparse network by selecting the input-output paths with the highest gradient magnitude at initialization, until we reach a target network density. These loss gradients are computed using a small subset of the training data. We then compare the performance of the network formed by these paths with GraSP, which preserves the propagation of gradients instead of maximizing the latter. The "higher gradient paths" approach results in higher classification accuracy, and smoother performance degradation as the density decreases(Figure 3(a)). Note that GraSP results has lower initial gradient norm, as expected. We also compare the training error: larger gradient norm paths result in faster convergence and better training dynamics (Figure 3(b),(c)).

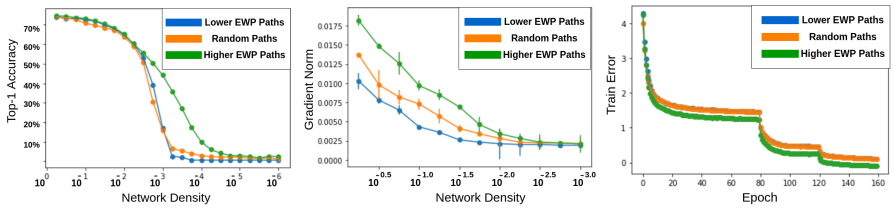

Figure 4: Comparing sparse networks formed by a) lower EWP paths, b) random paths, and c) higher EWP paths in terms of classification accuracy, initial gradient norm, and training error (10% network density). Network: VGG19. Dataset: CIFAR-100.

Another relevant comparison is between sparse networks with the same number of parameters constructed by three different sets of paths: a) paths with lower EWPs, b) random paths, and c) paths with higher EWPs. Figure 4(a) shows that the selection of higher EWP paths results in better learning performance than random paths, and even more better than lower EWP paths. We also show the initial gradient norm and the training error at 10% density (90% pruning ratio). Figure 4(b) shows that the network formed by higher EWP paths has consistently higher gradient norm, and it results in faster training.

5 THE PHEW NETWORK CONSTRUCTION METHOD

Based on the insight of the last two sections, we would like to construct sparse networks that only conserve the input-output paths with higher initial EWP values. One option would be to formulate this as an optimization problem in which we identify the top-k paths in terms of EWP. Those paths,

however, may form bottlenecks on specific connections of high weight. Those bottlenecks could be detrimental for the learning process, as they would need to propagate the gradients of many different paths. In the extreme, we can imagine that the top-k EWP paths may not even traverse all input and output units. So, instead, we would like to use a set of paths that have both relatively high EWP values – and that are more uniformly distributed across all the inputs and outputs, forming a more "balanced" network.

To achieve the previous objective, we select input-output paths probabilistically, based on a *biased random-walk process*. The next-hop of each path, from a unit i to a unit j at the next layer, is selected with a probability that is proportional to the weight of the connection from i to j. Additionally, to maintain a balance of the conserved paths between all inputs and outputs, we create paths in both directions: a) start from an input unit and create a biased random-walk towards an output, and b) start from an output unit and create a reverse biased random-walk towards an input. The random-walk process is described in more detail next:

Bi-directional walks and selection of starting unit: The selection of the starting unit of each random-walk is performed so that all input units and all output units participate in the same number of random-walks. Specifically, 50% of the time we start a forward random-walk from an input, and 50% of the time we start a reverse random-walk from an output. The selection of the starting unit in each case is such that the number of walks that start or terminate at each input or output unit is approximately the same.

Probabilistic bias at each step: Suppose that while constructing a random-walk, we are at unit n_i. The probability that the next-hop of the wak will be unit n_j at the next layer is given by:

$$P(j,i) = \frac{|w(j,i)|}{\sum_j |w(j,i)|} \tag{3}$$

As a result these random-walks are more likely to traverse paths with higher EWP values. At the same time, the walks are stochastic in nature and so they avoid bottleneck connections and units.

The creation of such random-walks continues until we have reached the given, target number of parameters for the sparse network. We refer to a network formed by this process as *PHEW network.* Note that the network formation process does not depend on the training data or task – but it does depend on the initial weights.

5.1 Comparison with unbiased random walks

It is informative to compare such biased random-walks with unbiased random-walks in which the next-hop is selected uniformly at random. In particular, we compare the expected value of the EWP in a biased random-walk versus in a uniform random-walk.

Consider a fully-connected MLP network with L layers and N_l units in each layer. Suppose that the weights are initialized according to the Kaiming method (He et al. (2015)) – meaning that are sampled from a Normal distribution in which the variance is inversely proportional to the width of each layer: $w^l_{i,j} \sim \mathcal{N}(0, \sigma_l^2)$, where $\sigma_l^2 = 2/N_l$. Consider two paths $p_1 \& p_2$, where p_1 has been selected using the previous biased random-walk process while p_2 has been selected with a uniform random-walks. In Appendix A.3 we show that:

$$\mathbb{E}(\Pi(p_1)) = \left(\frac{\pi}{2}\right)^{L/2} \times \prod_{l=1}^{L} \sigma_l >> \left(\frac{2}{\pi}\right)^{L/2} \times \prod_{l=1}^{L} \sigma_l = \mathbb{E}(\Pi(p_2)) \tag{4}$$

As the number of layers increases the ratio between the two expected values becomes exponentially higher: $\left(\frac{\pi^2}{4}\right)^{L/2}$. On the other hand, as the layer width increases the ratio of two values remains the same. Hence, we conclude that, despite the stochastic nature of PHEW random-walks, the average EWP of the selected paths is much greated than the EWP of randomly chosen input-output paths.

5.2 Applying PHEW in convolutional neural networks

A convolutional layer takes as input a 3D vector with n_i channels and transforms it into another 3D vector of n_{i+1} channels. Each of the n_{i+1} units in a layer produces a single 2D channel corresponding to the n_{i+1} channels. A 2D channel is produced applying convolution on the input vector with

n_i channels, using a 3D filter of depth n_i. Therefore each input from a unit at the previous layer has a corresponding 2D kernel as one of the channels in the filter. So, even though MLPs have an individual weight for each edge, in convolutional networks we have a 2D kernel for each edge.

A random-walk can traverse an edge of a convolutional network in two ways: either traversing a single weight in the corresponding 2D kernel – or traversing the entire kernel with all its weights. Traversing a single weight from a kernel conserves that edge and produces a non-zero output channel. This creates sparse kernels and allows for the processing of multiple input channels at the same unit and with fewer parameters. On the other hand, traversing the entire 2D kernel that corresponds to an edge means that several other kernels will be eliminated. Earlier work in pruning has shown empirically the higher performance of creating sparse kernels instead of pruning entire kernels (Blalock et al. (2020)). Therefore, in PHEW we choose to conserve individual parameters during a random-walk rather than conserving entire kernels.

In summary, PHEW follows a two-step procedure in convolutional networks: first an edge (i.e., 2D kernel) is selected using equation (3). Then a single weight is chosen from that kernel, randomly, with a probability that is proportional to the weight of the sampled parameter. We have also experimented with the approach of conserving the entire kernel, and we present those results as well in the next section.

6 Experiments and Results

In this section we present various experiments conducted to compare the performance of PHEW against other pruning strategies. We compare PHEW against uniform random-walks as well as three state-of-the-art algorithms: SNIP (Lee et al. (2018)), GraSP (Wang et al. (2020)) and SynFlow (Tanaka et al. (2020)). Because PHEW is data-independent, the most relevant comparison is with SynFlow, which also does not require any training data.

We present results both for standard image classification tasks using state-of-the-art convolutional networks as well as an image-transformation task using MLPs (that task is described in more detail in Appendix B). PHEW depends on the initial weights, and so we also present results for different weight initialization schemes. Lastly, we perform a structural network analysis between PHEW networks, and "winning tickets" constructed using the Iterative Magnitude Pruning (IMP) method.

6.1 Classification comparisons

We present results for five pruning algorithms on two architectures (VGG19 and ResNet34) on CIFAR10, CIFAR100 and Tiny ImageNet. We used the standard hyper-parameters to train the original networks and PHEW networks.

For CIFAR-10/100, the network density is set to 10%, 5% and 2% of the fully-connected architecture – those results are shown Table 1. We run each experiment three times and present the mean and standard deviation of test accuracy. For Tiny-ImageNet the network density is set to 20%, 10% and 5% – see Table 2.

Data-Dependent Methods: Figure 5 compares the test accuracy of different pruning algorithms as the network density decreases. It is interesting that PHEW networks consistently outperform even the data-dependent pruning algorithms GraSP and SNIP. At very low network densities, the accuracy of SNIP and GraSP degrades heavily while the accuracy of the data-agnostic methods, PHEW and SynFlow, drops more gradually creating a significant gap between the two groups of methods.

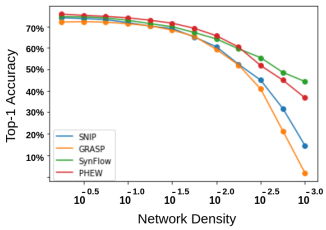

Figure 5: Accuracy comparisons for very low network densities. Network: VGG19. Dataset: CIFAR100

Data-independent Methods : The major contribution of SynFlow is its stability and better performance at very low density values. We observe that at moderate density levels, PHEW performs better than SynFlow. In very low network densities, below $10^{-2.5}$, Figure 5 shows that SynFlow performs better than PHEW. The reason may be that at very low densities the limited conserved random-walks may not be sampling the highest EWP paths.

Dataset	**CIFAR-10**			**CIFAR-100**		
Pruning Ratio/ Density	90%/10%	95%/5%	98%/2%	90%/10%	95%/5%	98%/2%
VGG-19 Baseline	94.30	–	–	74.16	–	–
SNIP	93.56 ±0.04	92.77 ±0.11	91.38 ±0.15	72.33 ±0.17	71.37 ±0.34	62.90 ±1.13
GraSP	93.28 ±0.07	92.47 ±0.15	90.76 ±0.24	72.07 ±0.12	71.28 ±0.21	65.08 ±0.54
SynFlow	93.37 ±0.10	92.63 ±0.08	91.77 ±0.16	72.49 ±0.11	71.77 ±0.19	67.76 ±0.47
Uniform RW	93.21 ±0.11	92.57 ±0.17	91.81 ±0.15	72.53 ±0.09	71.65 ±0.10	67.19 ±0.29
PHEW	**93.80 ±0.08**	**93.18 ±0.14**	**92.24 ±0.18**	**73.30 ±0.11**	**72.51 ±0.13**	**69.17 ±0.26**
Kernel Uniform RW	92.88 ±0.04	92.12 ±0.11	91.59 ±0.20	71.89 ±0.14	71.04 ±0.08	66.46 ±0.17
Kernel PHEW	93.24 ±0.13	92.63 ±0.17	91.77 ±0.19	72.83 ±0.15	72.09 ±0.23	67.71 ±0.29
ResNet-34 Baseline	95.80	–	–	78.04	–	–
SNIP	94.30 ±0.05	93.60 ±0.21	93.17 ±0.31	75.92 ±0.35	74.18 ±0.69	69.20 ±1.43
GraSP	94.11 ±0.21	93.41 ±0.25	93.03 ±0.24	76.18 ±0.27	74.27 ±0.11	70.43 ±0.61
SynFlow	94.48 ±0.10	94.13 ±0.08	93.14 ±0.16	76.04 ±0.31	74.84 ±0.19	72.19 ±0.37
Uniform RW	94.77 ±0.09	94.37 ±0.24	93.38 ±0.19	76.18 ±0.12	75.25 ±0.25	72.36 ±0.31
PHEW	**95.13 ±0.13**	**94.87 ±0.11**	**93.86 ±0.22**	**76.82 ±0.23**	**75.87 ±0.34**	**73.22 ±0.37**
Kernel Uniform RW	94.08 ±0.05	93.14 ±0.16	92.01 ±0.25	75.68 ±0.06	74.45 ±0.16	71.87 ±0.26
Kernel PHEW	94.80 ±0.12	93.46 ±0.17	92.71 ±0.20	75.91 ±0.09	75.42 ±0.27	72.15 ±0.53

Table 1: Accuracy comparisons for sparse VGG19 and ResNet34 obtained through various pruning algorithms on CIFAR-10 and CIFAR-100.

Kernel-conserved PHEW variant: We also study a PHEW variant for convolutional neural networks where instead of conserving a single weight of a kernel each time a random walk traverses that kernel, we conserve the entire kernel. This approach reduces the FLOP count immensely by eliminating the operations performed on several 2D feature maps in specific units. We present the comparison for CIFAR10 and CIFAR100 in Table 1, we can observe that at moderate network densities, the kernel-conserved variant performs as well as the other methods – therefore this PHEW variant can be utilized when decreasing the FLOP count is a priority.

	VGG-19			**ResNet-34**		
Density	20%	10%	5%	20%	10%	5%
Baseline	61.93	–	–	67.77	–	–
SNIP	60.23	58.63	55.50	63.65	62.10	56.97
GraSP	60.01	59.64	58.11	63.35	62.34	58.90
SF	60.19	59.38	57.59	63.56	62.38	61.73
Uniform RW	60.10	59.58	58.38	63.73	62.65	61.67
PHEW	**61.29**	**60.10**	**59.78**	**64.37**	**63.31**	**62.68**

Table 2: Accuracy Comparisons on Tiny-Imagenet.

Different weight initializations: The proposed method depends on the initial weights, and so it is important to examine the robustness of PHEW's performance across the major weight initialization methods: Kaiming Normal (He et al. (2015)), Normal $\mathcal{N}(0, 0.1)$, and Xavier uniform (Glorot & Bengio (2010)). Table 3 shows the results of these experiments for CIFAR10 and CIFAR100 on VGG19 and ResNet34. Note that PHEW's performance is quite robust across all these weight initializations. PHEW performs best with Kaiming initialization which is the most widely used technique for training Deep Neural Networks.

Dataset	**CIFAR-10**			**CIFAR-100**		
Density	10%	5%	2%	10%	5%	2%
VGG-19	94.30	–	–	74.16	–	–
Kaiming	93.90	93.28	92.44	73.30	72.51	68.97
Normal	93.56	93.02	92.24	73.07	72.18	68.78
Xavier	93.63	93.16	92.32	73.24	72.09	69.13
ResNet-34	95.80	–	–	78.04	–	–
Kaiming	95.13	94.87	93.86	76.82	75.87	73.22
Normal	95.01	94.71	93.36	76.78	75.58	72.82
Xavier	95.32	94.97	93.29	76.95	75.45	72.41

Table 3: PHEW comparisons for different weight initialization schemes

6.2 ARE PHEW NETWORKS "WINNING-TICKETS"?

The lottery ticket hypothesis (Frankle & Carbin (2018)) posits the existence of sparse subnetworks at initialization that are capable, strictly based on their initial weights, to perform almost as well as the fully-connected architecture. One approach to compute such "winning tickets" is the Iterative Magnitude Pruning (IMP) method (Frankle & Carbin (2018)), which requires a sequence of training and pruning cycles. At the end of the process, the weights of the resulting subnetwork are reverted back to their original initialized values. The key question we ask here is whether PHEW networks are also "winning tickets", despite the fact that they are constructed in a data-agnostic manner.

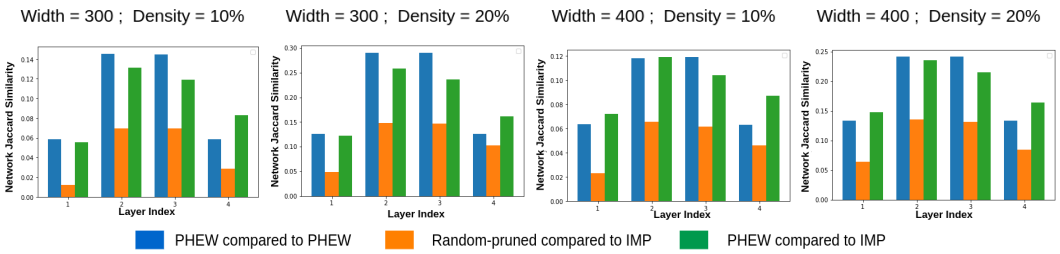

Figure 6: Jaccard Similarity structural comparisons between sparse MLPs on the MNIST image-transformation.

We answer this question by comparing the structural similarity of IMP networks with PHEW networks – as well as with randomly pruned networks. By structural similarity we refer to a comparison of the specific edges that are present in a given pair of networks using the Jaccard similarity metric. Given that PHEW is a stochastic process, we first generate 10 PHEW networks from the same initial weights and calculate their average pair-wise similarity (so that we can quantify how similar PHEW networks are to each other). Then we compare an IMP "winning ticket" network to each of the PHEW networks, and calculate their average structural similarity. We also perform a comparison between the same IMP network and 10 randomly-pruned networks with the same number of parameters – as a reference point.

Figure 6 shows the results of these comparisons for the MNIST image-transformation task on four combinations of 4-layer MLP networks and density values. We find that the average structural similarity between the IMP network and PHEW networks is statistically similar to the average pairwise similarity between PHEW networks. The similarity with randomly pruned networks is much lower, on the other hand. Therefore, the PHEW networks belong in the class of "winning ticket" networks that are constructed by the IMP process – even though they are constructed without any training data.

7 Discussion and Conclusion

In this paper, we proposed an approach called PHEW to create sparse "winning ticket" networks without any training data. We first showed the advantages of conserving input-output paths instead of pruning individual connections. We showed that paths with higher initial edge-weights have larger loss gradient magnitudes initially. We argued, and showed empirically, that the conservation of such paths leads to faster convergence and better learning performance. Finally, we compared the performance of PHEW networks against state-of-the-art pruning-before-training methods and showed that the structural similarity between IMP "winning tickets" and PHEW networks is as high as that between different PHEW networks with the same initial weights.

Possible future directions for this work include: 1) Explore path-based sparse network formation methods that can utilize a limited amount of training data to get even higher performance than PHEW. We have already shown that paths with higher initial loss gradient perform better than other data-dependent pruning methods. 2) Explore how to identify path-based sparse networks that can perform as well as PHEW networks but without any training. 3) Investigate how to dynamically determine the optimal number of parameters in a sparse network at the early stages of training – instead of starting with a pre-determined target number of parameters that may be too low or too high.

REFERENCES

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *arXiv preprint arXiv:2003.03033*, 2020.

Miguel A Carreira-Perpinán and Yerlan Idelbayev. "learning-compression" algorithms for neural net pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8532–8541, 2018.

Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pp. 4857–4867, 2017.

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

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611*, 2019.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances in neural information processing systems*, pp. 1379–1387, 2016.

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*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015b.

Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pp. 164–171, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pp. 8571–8580, 2018.

Steven A Janowsky. Pruning versus clipping in neural networks. *Physical Review A*, 39(12):6600, 1989.

Ehud D Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE transactions on neural networks*, 1(2):239–242, 1990.

Jeremy Kepner and Ryan Robinett. Radix-net: Structured sparse matrices for deep neural networks. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 268–274. IEEE, 2019.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pp. 598–605, 1990.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11264–11272, 2019.

Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *Advances in Neural Information Processing Systems*, pp. 4932–4942, 2019.

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*, pp. 107–115, 1989.

Sejun Park, Jaeho Lee, Sangwoo Mo, and Jinwoo Shin. Lookahead: a far-sighted alternative of magnitude-based pruning. *arXiv preprint arXiv:2002.04809*, 2020.

Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in neural information processing systems*, pp. 3360–3368, 2016.

Ameya Prabhu, Girish Varma, and Anoop Namboodiri. Deep expander networks: Efficient deep networks from graph theory. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 20–35, 2018.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Mohammad Javad Shafiee, Parthipan Siva, and Alexander Wong. Stochasticnet: Forming deep neural networks via stochastic connectivity. *IEEE Access*, 4:1915–1924, 2016.

Hidenori Tanaka, Daniel Kunin, Daniel LK Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *arXiv preprint arXiv:2006.05467*, 2020.

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

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks. *arXiv preprint arXiv:1806.05393*, 2018.

Ge Yang and Samuel Schoenholz. Mean field residual networks: On the edge of chaos. In *Advances in neural information processing systems*, pp. 7103–7114, 2017.

Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Yingyan Lin, Zhangyang Wang, and Richard G Baraniuk. Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*, 2019.

Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9194–9203, 2018.

A Detailed Analysis and Derivations

A.1 Analysis of Random Activation of Connections and Stub Units

In this section we analyze the sparse networks generated with random edge pruning. Edges are pruned by sampling from a Bernoulli distribution with probability equal to the network density. The goal of this analysis is to first understand the relationship between network density and the expected number of stub units. Then as an edge case, show that the probability of encountering at least one stub unit increases with decreasing density.

Let us consider a constant width network with L hidden layers and the number of neurons in each layer N. Let the required connection density be be P/P_{max}. For the sake of simplicity we work with pruning ratio, $x = 1 - P/P_{max}$. Network density and pruning ratio have a complimentary relationship. The probability of a connection j being active $P(m_j = 0) = P/P_{max} = 1 - x$, where m_j is the mask value. Let S be the set of all the stub units in a sparse network with pruning ratio x.

For any neuron in the network, the maximum number of incoming as well as outgoing edges possible is the same, N. By assumption there exist $N \times L$ hidden neurons in the network. The probability of a neuron n_j being a stub unit, $P(n_j \in S)$ and the expected number of stub units, $\mathbb{E}(\|S\|_0)$ are :

$$P(n_j \in S) = (1 - x^N) \times x^N + (1 - x^N) \times x^N = 2 \times x^N(1 - x^N) \tag{5}$$

$$\mathbb{E}(\|S\|_0) = P(n_j \in S) \times N \times L \tag{6}$$

Equation (5) indicates the probability of no outgoing edges or no incoming edges and then subtracts the probability of both these events occurring together. We can see in Figure 7 that the probability value attains a maximum, but for higher width values the probability is mostly an increasing function. Therefore we can see here that the expected number of stub units also increases and attains a maximum.

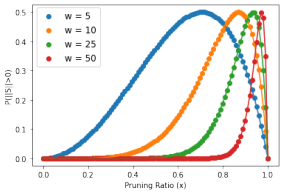

Figure 7: Equation (5)

We extend the above equations to all the neurons present by assuming independence Therefore, for the edge case of encountering at least one stub unit, the probability $P(\|S\|_0 > 0)$ is,

$$P(\|S\|_0 > 0) = 1 - P(\|S\|_0 = 0) = 1 - (1 - P(n_j \in S))^{N \times L} \tag{7}$$

$$P(\|S\|_0 > 0) = 1 - (1 - 2 \times x^N(1 - x^N))^{N \times L} \tag{8}$$

Figure 8 shows the probability of the presence of at least one stub unit in the entire network for varying width values and a depth of three. We can see in Figure 8 if we keep increasing x the probability of the presence of at least one stub unit keeps increasing.

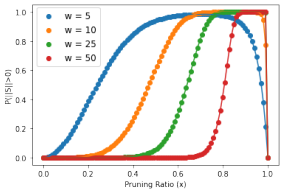

Figure 8: Equation (8)

Inferences : The network density has a complimentary relationship with pruning ratio (x), therefore

1. As the network density decreases, the probability of $n_j \in S$ increases, as a result the expected number of stub units increases in a multiplicative way.

2. We can observe for all width values $P(\|S\|_0 > 0)$ is an increasing function as network density decreases and hits one very quickly. Therefore, the probability of at least one stub unit is very high even for moderate density levels.

A.2 Analysis of Gradients Along Input-Output Paths

Claim *Given two paths between the same input-output units, one with much lower EWP than the other, the path with higher EWP has a higher loss gradient magnitude.*

Let consider two input-output paths $p_1 \& p_2$, originating from the same input neuron and terminating at the same output neuron as shown in Figure 9. The edge-weights values $(w_1, w_{12}, w_{13}, w_4 \in p_1)$ form the first path and $(w_1, w_{22}, w_{23}, w_4 \in p_2)$ form the second path. The assumption, $\Pi(p_1) = |w_1 \times w_{12} \times w_{13} \times w_4| >> \Pi(p_2) = |w_1 \times w_{22} \times w_{23} \times w_4| \approx 0$. For symplicity and without loss of generality we can assume that a single weight in p_2 is close to zero, $w_{23} \approx 0$. The edges-gradient and the node gradients for (w_4, y_4),

$$G(w_4) = \left|\frac{\partial L}{\partial w_4}\right| = \left|\frac{\partial L}{\partial y_{out}} \times y_4\right| \tag{9}$$

$$G(y_4) = \left|\frac{\partial L}{\partial y_4}\right| = \left|\frac{\partial L}{\partial y_{out}} \times w_4\right| >> 0 \tag{10}$$

Now from this neuron the paths diverge, we can observe that for the gradient propagation through $p_1 \& p_2$, the major difference is the weight magnitudes between w_{23} and w_{13}. For paths $p_1 \& p_2$ the edge-gradient $(w_{k3}, k = 1, 2)$ and the node-gradients $(y_{k3}, k = 1, 2)$ are,

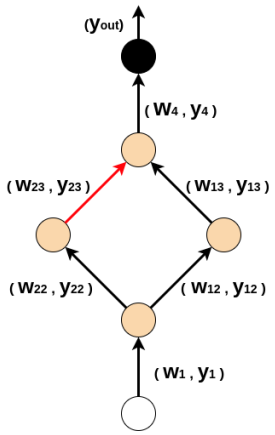

Figure 9: Two paths having extreme EWP values

$$G(w_{k3}) = \left|\frac{\partial L}{\partial w_{k3}}\right| = \left|\frac{\partial L}{\partial y_4} \times \frac{\partial y_4}{\partial w_{k3}}\right| = \left|\frac{\partial L}{\partial y_{out}} \times w_4 \times y_{k3}\right| \tag{11}$$

$$G(y_{k3}) = \left|\frac{\partial L}{\partial y_{k3}}\right| = \left|\sum_i \frac{\partial L}{\partial y_i^{l+1}} \times \frac{\partial y_i^{l+1}}{\partial y_{k3}}\right| = \left|\sum_i \frac{\partial L}{\partial y_i^{l+1}} \times w_{ki}^{l+1}\right| \tag{12}$$

When we analyze the node gradient equations ($G(y_{13})$ and $G(y_{23})$), we can observe,

1. A term in the summation equation (12) for $G(y_{13})$ is $G(y_4) \times w_{13}$. From equation (10) and assumption, the value is very high, therefore the gradient magnitude propagated through the edge-weight $w13$ is relatively higher than the gradient magnitude propagated through other edge-weights.

2. A term in the summation equation (12) for $G(y_{23})$ is $G(y_4) \times w_{23}$, from our assumption, we can see that gradient magnitude value $G(y_4)$ propagating through the edge-weight w_{23} is very low.

The higher node-gradient value in equation (10) in a higher proportion is transmitted through the larger edge-weight w_{13} than the near-zero edge-weight w_{23}. We imply here that the node-gradient $G(y_{13})$ will be relatively higher in magnitude due to the contribution from the higher node-gradient $G(y_4)$ and higher edge-weight w_{13}. Again going back-wards with path p_1 we can observe,

$$\left|\frac{\partial L}{\partial y_{13}}\right| \approx \left|\frac{\partial L}{\partial y_4} \times w_{13}\right| = \left|\frac{\partial L}{\partial y_{out}} \times w_4 \times w_{13}\right| \tag{13}$$

$$\left|\frac{\partial L}{\partial w_{12}}\right| = \left|\frac{\partial L}{\partial y_{13}} \times \frac{\partial y_{13}}{\partial w_{12}}\right| \approx \left|\frac{\partial L}{\partial y_{out}} \times w_4 \times w_{13} \times y_{12}\right| \tag{14}$$

$$\left|\frac{\partial L}{\partial y_{12}}\right| \approx \left|\frac{\partial L}{\partial y_{13}} \times w_{12}\right| \approx \left|\frac{\partial L}{\partial y_{out}} \times w_4 \times w_{13} \times w_{12}\right| \tag{15}$$

The gradient magnitude propagated along high EWP path keeps increasing in a multiplicative manner. But the gradient magnitude being propagated through p_2 vanishes after encountering a single near-zero edge-weight.

Note: The loss gradients propagating through the units participating in the lower-EWP path can be high. This is possible because of other paths with higher edge-weights passing through the units.

A.3 Detailed Derivation of Expectation of EWP

Let us assume a neural network architecture with L layers and N_l neurons in each layer at initialization. The weights have been initialized by sampling from Kaiming Normal distribution, that is $w_{i,j}^l \sim \mathcal{N}(0, \sigma_l^2)$, where $\sigma_l^2 = 2/N_l$. Now let us consider two paths $p_1 \& p_2$, where p_1 has been sampled using biased random walks and p_2 has been sampled using uniform random walks.

Biased Random Walks First we derive the expected value of the edge-weight-product of the path if it is sampled using biased random walks. Let the path sample be p_1, and the weight values sampled at step t be w^t,

$$\mathbb{E}(\Pi(p_1)) = \prod_{t=1}^{L} \mathbb{E}(|w^t|) \tag{16}$$

$$\mathbb{E}(|w^t|) = \sum_{k=0}^{W} |w_{i,k}| \times P(w^t = w_{i,k}) = \sum_{k=0}^{W} |w_{i,k}| \times \frac{|w_{i,k}|}{\sum_{j=0}^{W} |w_{i,j}|} = \frac{\sum_{k=0}^{W} |w_{i,k}|^2}{\sum_{j=0}^{W} |w_{i,j}|} \tag{17}$$

1. For the numerator, we know that $w_{i,j} \sim N(0, \sigma_t^2)$, hence the variance,

$$\mathbb{V}(|w_{i,k}|) = \mathbb{E}(w_{i,k}^2) - (\mathbb{E}(w_{i,k}))^2 = \mathbb{E}(w_{i,k}^2) = \sigma_t^2 \tag{18}$$

2. Similarly for the denominator. $|w_{i,j}| \sim$ folded normal distribution centered at 0,

$$\mathbb{E}(|w_{i,j}|) = \sigma_t \times \sqrt{\frac{2}{\pi}} \tag{19}$$

From equations (18) and (19), we get the following,

$$\mathbb{E}(\Pi(p_1)) = \prod_{t=1}^{L} \mathbb{E}(|w^t|) = \prod_{t=1}^{L} \left(\frac{\sigma_t^2}{\sigma_t \sqrt{\frac{2}{\pi}}} \right) = \left(\frac{\pi}{2}\right)^{\frac{L}{2}} \prod_{t=1}^{L} \sigma_t \tag{20}$$

Uniform Sampling of Paths We generate an expected value for the EWP of the paths when the paths are sampled randomly (Uniformly). We use the mean of the folded normal distribution so as to arrive at the value.

$$\mathbb{E}(\Pi(p_2)) = \prod_{t=1}^{L} \mathbb{E}(|w^t|) = \prod_{t=1}^{L} \left(\sigma_t \sqrt{\frac{2}{\pi}} \right) = \left(\frac{2}{\pi}\right)^{\frac{L}{2}} \prod_{t=1}^{L} \sigma_t \tag{21}$$

Inferences : We can observe the bounds from equations (21) and (20),

$$\mathbb{E}(\Pi(p_1)) = \left(\frac{\pi}{2}\right)^{\frac{L}{2}} \times \prod_{l=1}^{L} \sigma_l >> \left(\frac{2}{\pi}\right)^{\frac{L}{2}} \times \prod_{l=1}^{L} \sigma_l = \mathbb{E}(\Pi(p_2)) \tag{22}$$

As the number of layers increases this difference is more and more profound due to the exponential factor of $L/2$. Similarly, as the width for each of the layers increases, there is no difference in the ratio of two value but the two values themselves decrease due to the inverse nature. Hence, we conclude that even with the randomness in selection of high EWP paths, the EWP of resulting paths is exponentially larger than selecting random paths.

B Regression based Task : Class Specific Transformation

The task chosen for the experiments is that of class specific transformation. We generate the ground truth transformation by rotation and shearing. The angle of rotation and shearing coefficient are class specific. We then crop and pad the rotated image to fit the original dimensions. The classes are rotated in multiples of 30° and sheared with coefficients uniformly sampled between -0.6 and 0.6. Figure 10 shows various examples for the input and ground truth output. We experiment with MNIST and all classes have been used in the single task.

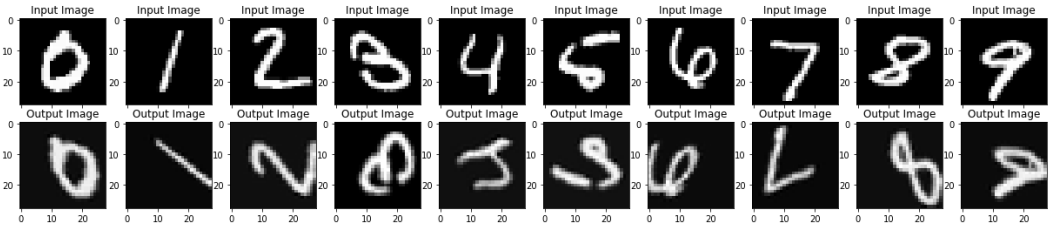

Figure 10: Examples of the input and the output for the transformation task

Network Architectures and Hyper-parameters: We use constant width MLP networks, with width equal to 100, 200, 300 and 400. Only 1000 examples per class were used in training for 20 epochs. A learning rate of 0.001 was used with an exponential decay factor of 0.95 after every epoch, Adam optimizer, batch size of 32, and MSE as the loss. Test performance was evaluated with MSE metric on the entire test set. ReLU in combination with batch-normalization were implemented after each layer, including the output layer.

Results : We present the results for this particular task in two steps, first we compare the performance of the two data-dependent methods SNIP and GraSP with the current state-of-the-art data-independent method SynFlow. Then we compare and analyze the proposed biased random walks based method against SynFlow. The division is created for the sake of clarity in observing the plots as well as understanding the results.

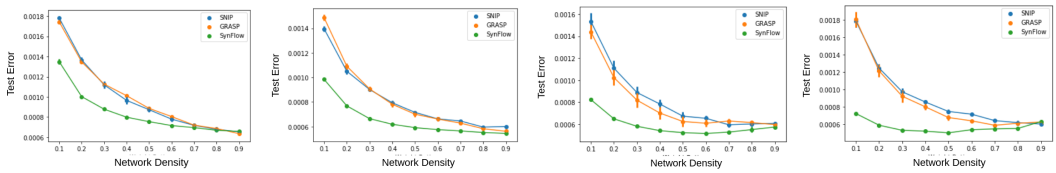

Figure 11: Comparison of SynFlow pruning algorithm with SNIP and GraSP. The row indicates the plots for constant width networks with widths 100, 200, 300 and 400 from left to right.

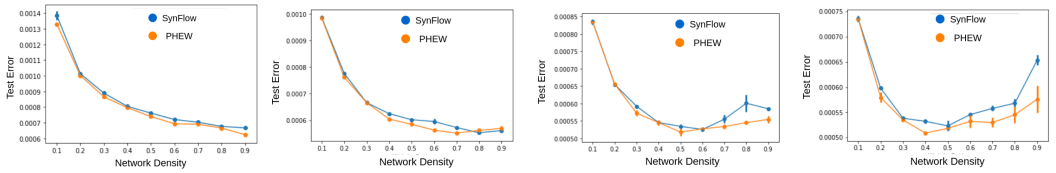

Figure 12: Comparison of the data-agnostic SynFlow pruning algorithm to the biased random walk

Figure 11 shows the comparison between the data-agnostic SynFlow algorithm to the SNIP and GraSP algorithms. Here we can clearly observe that the SynFlow algorithm performs better than the other algorithms for very high level of sparsity, whereas when the density increases the performances start to coincide with each other. This can be explained by the fact that in the cases of very high sparsity also the SynFlow algorithm preserves the effective flow of information.

In the Figure 12, we can see that PHEW networks marginally but consistently performs better than SynFlow networks in terms of test errors. However there exists several cases in which the performance is the same. It is to be observed here that as the size or the capacity of the initialized networks increases the superior performance becomes more and more clear.

Conclusion : The experiments and the results presented in this section further verify the validity of the proposed approach against the state-of-the-art methods. It is to be noted that we are able to show significant improvements and generalization capabilities across data-sets and even tasks.