A. Proofs

A.1. Maximum path kernel trace

We consider a single hidden-layer network, $\boldsymbol{f} : \mathbb{R}^D \to \mathbb{R}^D$, with N hidden units and D inputs and outputs. The incoming and outgoing weights of each hidden unit are initialized by sampling from $\mathcal{N}(0, 1)$. The number of connections in the unpruned network is M, while the target number of connections in the pruned network is $m < M$. The corresponding network density is $\rho = m/M$.

Using the notation of Section 2, the path kernel trace maximization problem is to select the m edges that form a set of paths P such that the following function is maximum:

$$\sum_{p=1}^{P} \sum_{i=1}^{m} \left(\frac{\pi_p(\boldsymbol{\theta})}{\theta_i} \right)^2 p_i, \text{ given } m = \rho \times M \tag{12}$$

The following lemma identifies the architecture that maximizes the previous expression, under the following two assumptions:

1. For simplicity, we assume that m is a multiple of $2D$, so that we can form a fully-connected network with an integer number of hidden units.

2. Because all hidden units follow the same weight distribution, we assume that the sum of squares of the top-d (out of D) outgoing weights is approximately the same for all hidden units – and denote that constant by K_d. This assumption is reasonable if D is large and $d \gg 1$. Also, because the incoming weights in a hidden unit follow the same distribution with the outgoing weights, K_d also approximates the sum of squares of the top-d (out of D) incoming weights at each hidden unit.

Lemma 1: *The maximization of the path kernel trace results in a fully-connected network in which only $n = \frac{m}{2D}$ of the hidden-layer units remain in the pruned network – all other units and their connections are removed. So, given a target network density, the network that maximizes the path kernel trace has the narrowest possible hidden-layer width.*

Proof: Consider a path p defined by the set of edge-weights $\{\theta_p^{[1]}, \theta_p^{[2]}\}$ at the first and second hop, respectively. We can re-write the optimization problem as,

$$\max \sum_{p=1}^{P} \left[(\theta_p^{[1]})^2 + (\theta_p^{[2]})^2 \right], \text{ given } m = \rho \times M \tag{13}$$

Let us denote as d_{in} the number of incoming connections of a hidden unit, and d_{out} the number of outgoing connections. The total degree of that unit is $d_{in} + d_{out}$. We can assume that this quantity is even for the following reason. The total number of edges m is even, and so the total number of units with odd degree has to be even. So, if there is a unit j with odd degree, we can move a connection from another odd-degree unit i to j so that both i and j have even degree. We can repeat this step for every pair of units with odd degree until all hidden units have even degree.

Let us now consider a hidden unit j with in-degree d_{in} and out-degree d_{out}. The path kernel trace contribution of the incoming and outgoing connections of unit j is:

$$\sum_{i=1}^{d_{in}} \sum_{k=1}^{d_{out}} \left[(\theta^{[0]}(j,i))^2 + (\theta^{[1]}(k,j))^2 \right] \tag{14}$$

which is equivalent with:

$$d_{in} \times \sum_{k=1}^{d_{out}} (\theta^{[1]}(k,j))^2 + d_{out} \times \sum_{i=1}^{d_{in}} (\theta^{[0]}(j,i))^2 \tag{15}$$

To maximize this expression, we can select the top-d_{in} incoming connections and the top-d_{out} outgoing connections in terms of squared weights. Then, based on Assumption-2, the previous expression becomes:

$$d_{in} \times K_{d_{out}} + d_{out} \times K_{d_{in}} \tag{16}$$

Suppose that $d_{out} \geq d_{in} + 2$ (recall that $d_{out} + d_{in}$ is even and so it cannot be that $d_{out} = d_{in} + 1$). Then, we can remove the connection with the lowest squared weight of the d_{out} outgoing connections and include an additional incoming connection – the one with the highest $(d_{in} + 1)$-ranked squared weight). This operation will result in the following path kernel trace difference:

$$\begin{aligned} (d_{in}+1) \times K_{d_{out}-1} + (d_{out}-1) \times K_{d_{in}+1} \\ -d_{in} \times K_{d_{out}} - d_{out} \times K_{d_{in}} \end{aligned} \tag{17}$$

$$\begin{aligned} &= d_{in} \times (K_{d_{out}-1} - K_{d_{out}}) \\ &+ d_{out} \times (K_{d_{in}+1} - K_{d_{in}}) \\ &+ K_{d_{out}-1} - K_{d_{in}+1} \end{aligned} \tag{18}$$

This difference is always positive because $d_{out} \geq d_{in} + 2$, and so $|K_{d_{out}-1} - K_{d_{out}}| \leq |K_{d_{in}+1} - K_{d_{in}}|$ and $K_{d_{out}-1} \geq K_{d_{in}+1}$.

If $d_{in} \geq d_{out} + 2$ we repeat the same process but removing an incoming connection and adding an outgoing connection.

We can continue this iterative process for every hidden unit, strictly increasing the path kernel trace at each step, until $d_{in} = d_{out} = d$ for every hidden unit.

After the completion of the previous process, every hidden unit j will have the same number d_j of input and output connections. So, $2 \times \sum_j d_j = m$. The path kernel trace for the entire network will then be:

$$Tr(\boldsymbol{\Pi}_\theta) = \sum_{j=1}^{n} 2\, d_j\, K_{d_j} \tag{19}$$

We can simplify this expression because $d_j = d$ for all j, based on Assumption-2. The resulting path kernel trace becomes:

$$Tr(\mathbf{\Pi}_\theta) = 2nd\, K_d = m\, K_d \tag{20}$$

The optimization problem can now be written as,

$$d^* = \underset{d}{\operatorname{argmax}}\{m \times K_d\} \text{ such that } d \leq D \tag{21}$$

As d increases so does the sum K_d because m is a constant. Therefore the objective is maximized when d takes the largest possible value D. That is, each of the n hidden units will be connected to all of the input and output units, i.e., $d^* = D$.

So, the optimal width of the hidden layer is $n^* = m/(2D)$, which is the lowest possible width given the target number of edges m. In the next Lemma (A.2), we show that $P^* = n\, D^2$ is the maximum possible number of paths given m.

Open Question: The previous result refers to a single hidden layer. We were not able to generalize it to deeper networks. If we simplify the problem by considering networks in which all edge-weights are equal to the same value, it is easy to show that the subnetwork that maximizes the number of input-output paths also maximizes the path kernel trace (see Appendix A.2).

A.2. MLP architecture with maximum number of paths

Let us consider an ReLU MLP network with $L > 1$ hidden layers, and with N_l hidden layer units at layer $l = 1, ..., L$. Without loss of generality, we can consider the case that the number of both inputs and outputs is equal to D.

Let the number of connections in the un-pruned network be M, the target number of remaining connections after pruning be m, and the target network density be $\rho = m/M$.

For simplicity, assume that m is such that we can form a fully-connected network with the same number k of hidden units at each layer. In other words, we assume that there is an integer k that satisfies the following equation: $m = k\,(2D + (L-1)k)$.

The next Lemma identifies the pruned architecture with the maximum number of paths, given m.

Lemma 2: *In an MLP network with a target number of parameters m, the maximum number of input-output paths results when each hidden layer has the same number of units after pruning, and those units are fully connected with the units of the previous layer.*

Proof:

Base case: Consider a network with two hidden layers. Let the number of units in the two hidden layers of the unpruned network be N_1 and N_2.

We first show that, for a hidden layer l, the number of paths through that layer is maximized by selecting $n_l \leq N_l$ fully-connected units – and pruning all other units. Then we show that the number of paths is maximized when the two hidden layers have the same number of (fully-connected) units after pruning, i.e., $n_1 = n_2$.

Let the number of incoming and outgoing connections for unit i in a hidden layer be $\{d_i, k_i\}$. Then the number of paths from the previous layer to the next layer is given by $P = \sum_{i=1}^{N_l} d_i \times k_i$.

Suppose we can identify two hidden units X and Y that are not fully-connected. The number of paths through X from the previous layer to the next layer is $P(X) = d_x \times k_x$ – and similarly for Y, $P(Y) = d_y \times k_y$. If $P(Y) > P(X)$ we move an edge from X to Y such that the edge is selected from either d_x or k_x, choosing the larger among the two. Similarly, the edge is added to either d_y or k_y, choosing the smaller among the two. This process causes the number of paths to strictly increase, that is, $\Delta P = max\{d_y, k_y\} - min\{d_x, k_x\} > 0$. In the case of a tie, if $P(X) = P(Y)$, we select the unit with the higher number of incoming/outgoing edges so that $max\{d_y, k_y\} \geq min\{d_x, k_x\}$. If there are still ties, we break them moving edges from lower-index units to higher-index units.

We repeat this process, increasing the number of paths in each iteration, until we cannot find any units that are not fully-connected. Then, the layer l will consist of $n_l \leq N_l$ fully-connected units.

We want to choose n_1 and n_2 to maximize the total number of paths:

$$D^2 \times n_1 \times n_2, \text{ such that } m = D(n_1 + n_2) + n_1 n_2 \tag{22}$$

Substituting n_2, we want to maximize the function of n_1:

$$D^2 \times n_1 \times \frac{m - Dn_1}{D + n_1} \tag{23}$$

The maximum results when $n_1 = \sqrt{D^2 + m} - D$. Solving for n_2, we get that the corresponding with of layer-2 is $n_2 = \sqrt{D^2 + m} - D = n_1$. Therefore, the number of paths is maximized by having the same number of units $n = n_1 = n_2$ in the two hidden layers.

Inductive step: The induction hypothesis on a network with $L - 1$ hidden layers is: given the target number of connections m, the number of paths for a network with $L-1$ hidden layers is maximized by considering a fully-connected network with the same number of units in each hidden layer, $n \leq N_i, i = 1, ..., L-1$, and $m = n(2D + (L-2)n)$.

Consider now a network with L hidden layers. From the induction hypothesis we know that the number of paths P_{L-1} till the last hidden layer is maximized when each hidden layer contains the same number of units n and is fully-connected. All the units in layer $L-1$ have the same number of incoming paths P_{L-1}/n.

Suppose that the number of selected units in the last hidden layer is $n_L \leq N_L$. The number of paths $P = \sum_{i=1}^{n} P_i \times P_{L-1}/n = P_{L-1}/n \times \sum_{i=1}^{n} P_i$, where P_i is the number of paths from the i^{th} unit in hidden layer $L-1$ to the output units. Using the same approach with the base case, we see that P is maximized when $n_L = n$.

A.3. PHEW and per-layer width

Consider a fully-connected MLP network, $f : \mathbb{R}^D \rightarrow \mathbb{R}^K$, with L hidden layers and N_l units per layer ($l = 1 \ldots L$). The weights are initialized with the Kaiming method, i.e., the initial weight from unit j at layer $l-1$ to unit i at layer l is $\theta^{[l]}(i,j) \sim \mathcal{N}(0, \sigma_l^2)$, where $\sigma_l^2 = 2/N_l$.

Lemma 3: *The expected number of PHEW random walks through each unit of layer l is W/N_l, where W is the required number of walks to achieve the target network density ρ.* In other words, at least in expectation, PHEW utilizes every hidden unit of a layer, resulting in the maximum per-layer width.

Proof: Let us first consider a network with a single hidden layer that has N hidden units. In PHEW, we start the same number of walks from each input unit. So, given the number of walks required to achieve the target density W, the number of walks starting from any input unit is W/D.

Let W_n be the number of walks passing through hidden unit n. Then,

$$\mathbb{E}(W_n) = \sum_{i=1}^{D} \frac{W}{D} \times Q^{[1]}(n,i) \tag{24}$$

where $Q^{[1]}(n,i)$ is the probability that the walk will move from input unit i to a unit n at the first hidden layer. For PHEW random walks this expectation is given by,

$$\mathbb{E}(W_n) = \frac{W}{D} \sum_{i=1}^{D} \frac{|\theta^{[1]}(n,i)|}{\sum_{j=1}^{N} |\theta^{[1]}(j,i)|} \tag{25}$$

The denominator of the previous equation is:

$$\sum_{j=1}^{N} |\theta^{[1]}(j,i)| = N \times \left[\frac{1}{N} \sum_{j=1}^{N} |\theta^{[1]}(j,i)| \right] = N \times \bar{\theta}_N(i) \tag{26}$$

where $\bar{\theta}_N(i)$ is the sample mean of the folded normal distribution. For a sufficiently large sample size N (in practice > 40), the sample mean is approximately equal to the population mean μ. So we replace $\bar{\theta}_N(i)$ with μ for all $i = 1, ..., D$:

$$\mathbb{E}(W_n) \approx \frac{W}{N\mu} \frac{1}{D} \sum_{i=1}^{D} |\theta^{[1]}(n,i)| \tag{27}$$

Similarly we approximate the sample average of the previous equation with μ:

$$\mathbb{E}(W_n) \approx \frac{W}{N\mu}\mu = \frac{W}{N} \tag{28}$$

Hence, the expected number of PHEW walks through any hidden unit is the same.

Similarly, the number of walks through each of the K output units is W/K, where K is the number of output units.

For networks with more than one hidden layer, it is simple to use induction and show that the expected number of random walks using PHEW through any unit at layer l is W/N_l.

A.4. PHEW and path kernel trace

Consider a fully-connected MLP network, $f : \mathbb{R}^D \rightarrow \mathbb{R}^K$ with L hidden layers and N_l units per layer ($l = 1 \ldots L$), and suppose that the weights are initialized using the Kaiming method.

Lemma 4: *Consider two input-output paths u and b in the previous MLP network: u has been selected with a uniform random walk, while b has been selected with the PHEW random walk process. Then,*

$$\mathbb{E}[\mathbf{\Pi}_{\boldsymbol{\theta}}(b,b)] = 2^L \times \mathbb{E}[\mathbf{\Pi}_{\boldsymbol{\theta}}(u,u)] \tag{29}$$

Proof: The path kernel matrix is given as :

$$\mathbf{\Pi}_{\boldsymbol{\theta}}(p,p') = \sum_{i=1}^{m} \left(\frac{\pi_p(\boldsymbol{\theta})}{\theta_i} \right) \left(\frac{\pi_{p'}(\boldsymbol{\theta})}{\theta_i} \right) p_i p'_i \tag{30}$$

Let path p be a path selected through a random walk process from an input unit to an output unit. The diagonal element that represents the contribution of p to the path-kernel trace is given by

$$\mathbf{\Pi}_{\boldsymbol{\theta}}(p,p) = \sum_{i=1}^{m} \left(\frac{\pi_p(\boldsymbol{\theta})}{\theta_i} \right)^2 p_i \tag{31}$$

Suppose that path p is formed by the edge weights $\{\theta_p^{[l]}\}_{l=1}^{L+1}$. Then,

$$\mathbf{\Pi}_{\boldsymbol{\theta}}(p,p) = \sum_{l=1}^{L+1} \left(\frac{\pi_p(\boldsymbol{\theta})}{\theta_p^{[l]}} \right)^2 \tag{32}$$

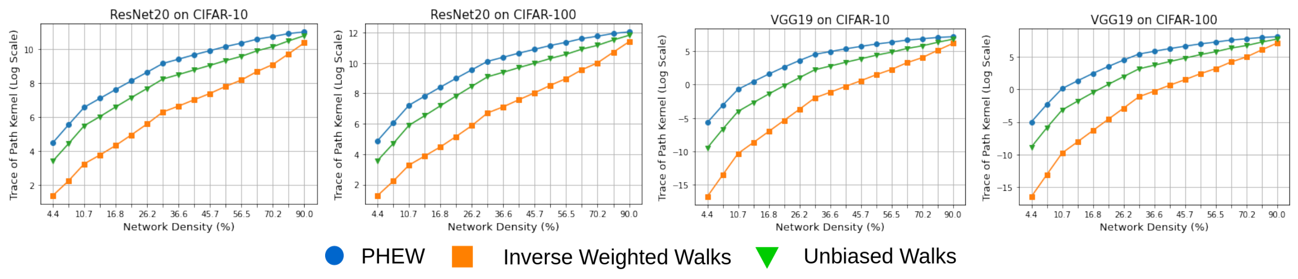

Figure 8. Comparison of the path kernel trace of sparse networks obtained using PHEW, unbiased random walks, and inverse-weighted random walks.

We can observe that $\mathbf{\Pi}_{\boldsymbol{\theta}}(p,p)$ is a linear function with respect to $\{(\theta_p^{[l]})^2\}_{l=1}^{L+1}$. Therefore,

$$\mathbb{E}[\mathbf{\Pi}_{\boldsymbol{\theta}}(p,p)] = \sum_{l=1}^{L+1}\left[\prod_{i=1, i\neq l}^{L+1} \mathbb{E}[\theta_p^{[i]}]^2\right] \tag{33}$$

The probability of selecting unit j at layer l given that the walk is on a unit k at layer $l-1$ is $Q^{[l]}(j,k)$.

We compute the expected value of each squared-weight in a sequential manner. That is,

$$\mathbb{E}[(\theta_p^{[l]})^2] = \sum_{j=1}^{N_l}(\theta^{[l]}(j,k))^2 Q^{[l]}(j,k) \tag{34}$$

Uniform random walks: First, suppose that the random walk is not biased. Denote an unbiased path by u. Then, $Q^{[l]}(j,k) = 1/N_l$. Let us now start the walk with a randomly selected input unit k,

$$\mathbb{E}[(\theta_u^{[1]})^2] = \sum_{j=1}^{N_1}(\theta^{[1]}(j,k))^2\frac{1}{N_1} \approx \sigma_1^2 \tag{35}$$

This is the sample mean of N_l squares from a normal distribution. If N_l is sufficiently large, we can approximate the sample mean with the expected value $\mathbb{E}[(\theta^{[1]}(j,k))^2]$. Note that we are only interested in the expectation of the weight value and not the position of the corresponding connection.

Moving to the next hop, we observe that the initialized weight distribution for all edges is the same regardless of the previous unit. Hence,

$$\mathbb{E}[(\theta_p^{[l]})^2] = \sigma_l^2 \tag{36}$$

Plugging the values back in the expectation of the path kernel trace,

$$\mathbb{E}[\mathbf{\Pi}_{\boldsymbol{\theta}}(u,u)] = \sum_{l=1}^{L+1}\left[\prod_{i=1, i\neq l}^{L+1} \sigma_l^2\right] \tag{37}$$

PHEW random walks: Now consider a path b that is sampled using the PHEW biased random walk process. Here,

$$Q^{[l]}(j,k) = \frac{|\theta^{[l]}(j,k)|}{\sum_{i=1}^{N_l}|\theta^{[l]}(i,k)|} \tag{38}$$

Let us again start the random walk process with some input unit k,

$$\begin{aligned}\mathbb{E}[(\theta_b^{[1]})^2] &= \sum_{j=1}^{N_1}(\theta^{[1]}(j,k))^2\frac{|\theta^{[l]}(j,k)|}{\sum_{i=1}^{N_1}|\theta^{[l]}(i,k)|}\\ &= \frac{\sum_{j=1}^{N_1}|\theta^{[l]}(j,k)|^3}{\sum_{i=1}^{N_1}|\theta^{[l]}(i,k)|}\\ &= \frac{\frac{1}{N_1}\sum_{j=1}^{N_1}|\theta^{[l]}(j,k)|^3}{\frac{1}{N_1}\sum_{i=1}^{N_1}|\theta^{[l]}(i,k)|}\end{aligned} \tag{39}$$

We approximate this ratio of sample means by the ratio of the corresponding two population means,

$$\mathbb{E}[(\theta_b^{[1]})^2] \approx \frac{2\sqrt{\frac{2}{\pi}}\sigma_1^3 N_1}{\sqrt{\frac{2}{\pi}}\sigma_1 N_1} = 2\sigma_1^2 \tag{40}$$

where the numerator is the expected value of $|X|^3, X \sim \mathcal{N}(0,\sigma_1^2)$, and the denominator is the expected value of $|X|, X \sim \mathcal{N}(0,\sigma_1^2)$.

Moving on to the next hop, we perform the same process as in the case of unbiased random walks, where the expected weight value is only dependent on the initialization distribution of the edges. We know that the initialization distribution is the same for all edges regardless of the unit they are connected to in the previous layer. Hence,

$$\mathbb{E}[(\theta_b^{[l]})^2] \approx 2\sigma_l^2 \tag{41}$$

Plugging this value back into the expected path kernel trace,

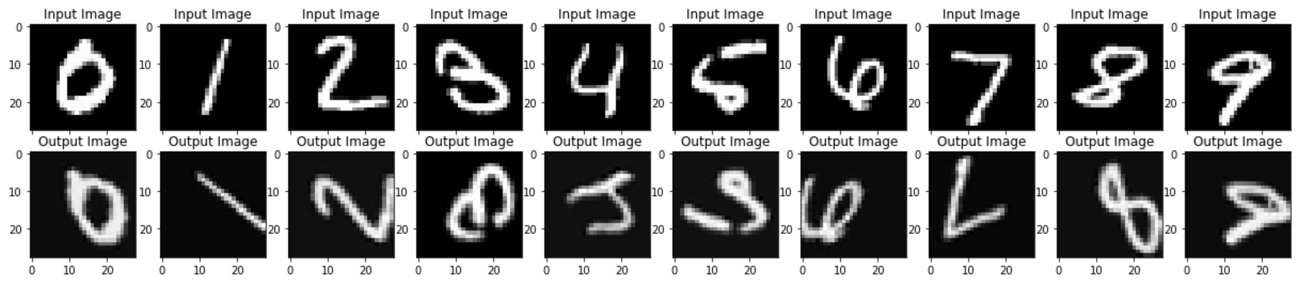

Figure 9. Examples of inputs and outputs for the MNIST image transformation task

$$\mathbb{E}[\mathbf{\Pi}_{\boldsymbol{\theta}}(b,b)] = \sum_{l=1}^{L+1}\left[\prod_{i=1,i\neq l}^{L+1} 2\sigma_l^2\right] = 2^L \sum_{l=1}^{L+1}\left[\prod_{i=1,i\neq l}^{L+1} \sigma_l^2\right] \tag{42}$$

We conclude that a path selected using PHEW will contribute to the path kernel trace much more than a path selected through an unbiased random walk, and this gap increases exponentially with the depth L of the network:

$$\mathbb{E}[\mathbf{\Pi}_{\boldsymbol{\theta}}(b,b)] = 2^L \times \mathbb{E}[\mathbf{\Pi}_{\boldsymbol{\theta}}(u,u)] \tag{43}$$

B. Image transformation task

In this section we present results for a regression task in which the number of inputs is the same with the number of outputs. The task is to perform MNIST image transformations (rotation and shearing). The angle of rotation and the shearing coefficient are class-specific. We 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 9 shows various input-output examples.

In this task, we use MLPs with width equal to 100, 200, 300 and 400. We use 1000 examples per class for training and 20 epochs. The learning rate is 0.001 with an exponential decay factor of 0.95 after every epoch, Adam optimizer, batch size of 32. The loss function is the mean-squared error between the actual output and correct output. Test performance is evaluated using the MSE metric on the entire test set. ReLU in combination with batch-normalization are implemented at each hidden layer.

B.1. Comparison between PHEW and other pruning methods on transformation task

Here, we present results for the image transformation task. Specifically, we compare the performance of PHEW with the two data-dependent methods SNIP and GraSP as well as with the data-agnostic methods SynFlow and SynFlow-L2.

Figure 10 shows that PHEW performs better than the other algorithms in a wide range of network density values. Of course, as the density increases the differences between these methods diminish.

An interesting observation here is that the data-dependent methods SNIP and GraSP perform worse in this task than the data-agnostic methods. This may be due to the change in the loss function from cross entropy to mean squared error for regression. We plan to investigate this phenomenon further by introducing different tasks/loss functions and architectures.

B.2. Number of input-output paths

Here, we present results for the number of input-output paths resulting from different pruning methods in the image transformation task – see Figure 11.

We observe that the sub-networks resulting from SynFlow and SynFlow-L2 indeed have the highest number of input-output paths. This empirical observation provides further evidence for the connection between the magnitude of the path kernel trace and the number of input-output paths (see Appendix A.1).

C. Ablations of SynFlow and SynFlow-L2

The formation of narrow subnetworks through SynFlow was first discovered in the ablation studies conducted in (Frankle et al., 2021). In that work, the edges of a layer are shuffled, creating a uniform distribution of connections across all units in the layer.

In Figures 12 and 13, we observe that for both SynFlow and SynFlow-L2, random shuffling of weights increases performance. The performance increase becomes larger in datasets with more classes. We believe that this is due to the increased complexity of those datasets and the need for a larger layer width to learn disconnected decision regions (Nguyen et al., 2018).

The authors of (Frankle et al., 2021) hypothesised that the cause of this effect is the iterative nature of the SynFlow algorithm along with the number of paths through specific

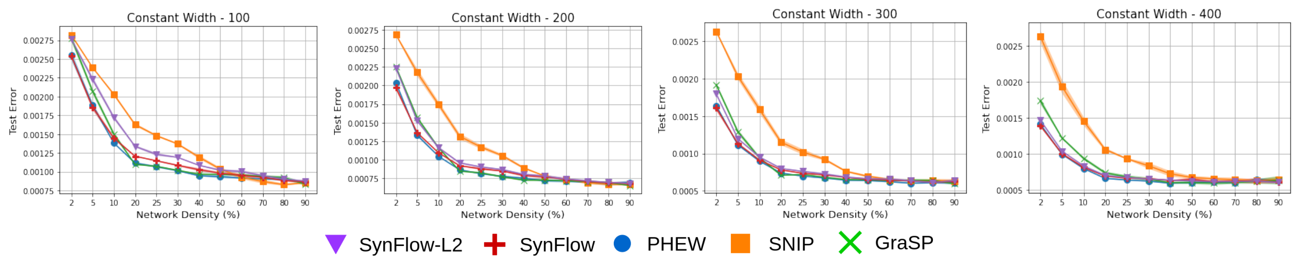

Figure 10. Comparison of PHEW against other pruning-before-training methods for the MNIST image transformation task.

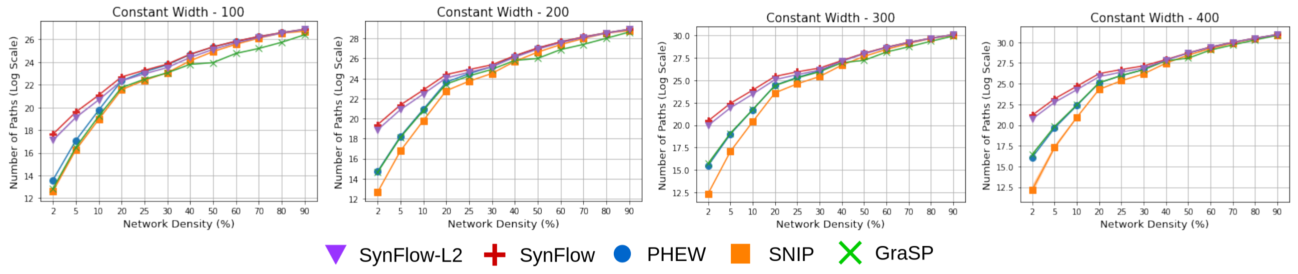

Figure 11. Comparison of the number of paths obtained through PHEW and other pruning methods.

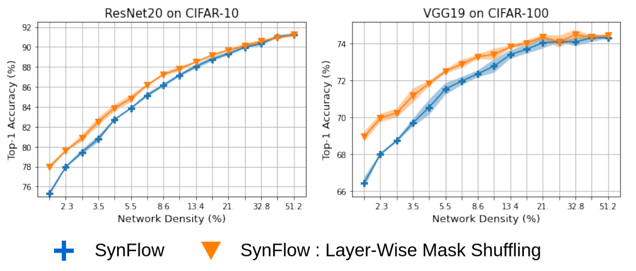

Figure 12. **SynFlow ablation:** Performance comparison of subnetworks obtained using SynFlow and subnetworks obtained by layer-wise randomized SynFlow subnetworks.

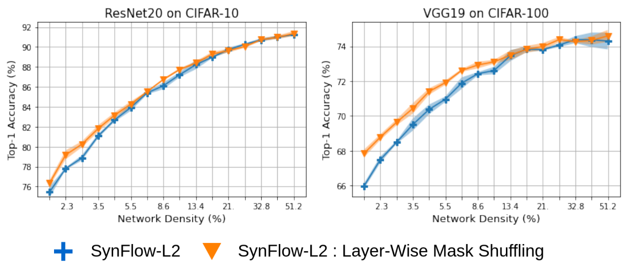

Figure 13. **SynFlow-L2 Ablation:** Performance comparison of subnetworks obtained using SynFlow-L2 and subnetworks obtained by random layer-wise weight shuffling of SynFlow-L2 subnetworks.

units. The scoring function for a specific edge-weight θ_i in SynFlow is $\sum_{p=1}^{P} |\pi_p(\boldsymbol{\theta})| p_i$. This represents the sum of edge-weight-products of all paths through θ_i. Similarly, for SynFlow-L2 the scoring function is $\sum_{p=1}^{P} \pi_p(\boldsymbol{\theta})^2 p_i / |\theta_i|$.

In both of these iterative pruning methods, if a unit has some pruned edges in the first few iterations it becomes more likely to be pruned in subsequent iterations (due to the reduction in the number of paths that traverse that unit). Hence, these iterative methods are more likely to completely eliminate some hidden units. This conjecture is aligned with our observation that by maximizing the number of input-output paths, the SynFlow and SynFlow-L2 methods eliminate many hidden units and result in very narrow layers.

D. Implementation details

In this section we report implementation details and hyper-parameter values for our experiments.

D.1. Networks, datasets and hyperparameters

We present results on three networks and three datasets for classification. The combination of networks and datasets used are as follows:

1. ResNet20 and CIFAR-10
2. ResNet20 and CIFAR-100
3. VGG19 and CIFAR-10
4. VGG19 and CIFAR-100

Network	Dataset	Epochs	Batch-size	Optimizer	Momentum	Learning Rate	Lr Drop	Weight Decay
ResNet20	CIFAR-10	160	128	SGD	0.9	0.1	10x at 1/2 and 3/4 epochs	1e-4
ResNet20	CIFAR-100	160	128	SGD	0.9	0.1	10x at 1/2 and 3/4 epochs	1e-4
VGG19	CIFAR-10	160	128	SGD	–	0.1	10x at 1/2 and 3/4 epochs	5e-3
VGG19	CIFAR-100	160	128	SGD	–	0.1	10x at 1/2 and 3/4 epochs	5e-3
ResNet18	Tiny-ImageNet	200	256	SGD	0.9	0.2	10x at 1/2 and 3/4 epochs	1e-4
VGG19	Tiny-ImageNet	200	256	SGD	0.9	0.2	10x at 1/2 and 3/4 epochs	1e-4

Table 1. Hyperparameters used for various combinations of datasets and networks in the experiments.

5. ResNet18 (Modified) and Tiny-ImageNet

6. VGG19 and Tiny-ImageNet

ResNet20: The ResNet20 architecture was designed for CIFAR-10 (He et al., 2016). The same network is also used for CIFAR-100. The network consists of 20 layers and batch-normalization is performed before activation.

VGG19 : The VGG19 architecture was taken from the original paper (Simonyan & Zisserman, 2014). The network consists of five convolutional blocks followed by max-pooling. The first block consist of two layers with width 64 – that is 64 filters. The second block contained two layers with width of 128; the third, fourth and fifth blocks consist of three layers each with the same width: 256, 512 and 512 respectively. All the filters have dimension 3×3, with the number of channels varying. For CIFAR-10/100 we replace the three linear layers in the original network by a single layer with width equal to the number of classes.

ResNet18 (Modified): ResNet18 was originally proposed for classification on ImageNet (He et al., 2016). A modification of the original architecture was provided by the authors of SynFlow (Tanaka et al., 2020) for Tiny-ImageNet. For a fair comparison we use the same architecture with the SynFlow paper. Specifically, the first convolutional layer uses a filter of dimensions 3×3 with a stride of 1, without the use of max-pooling.

All networks are initialized with Kaiming Normal initialization ((He et al., 2015)).

Datasets: We use three well known datasets for classification: CIFAR10, CIFAR-100 and Tiny-ImageNet, (the smaller version of ImageNet dataset)(Russakovsky et al., 2015). Due to resource limitations we are unable to present results for the full ImageNet dataset. For CIFAR-10/100 we perform channel-wise normalization using mean and standard deviation and use random horizontal flipping. For Tiny-ImageNet we use channel-wise normalization, random cropping of size 64×64 and random horizontal flipping.

Hyperparameters: We provide the hyperparameters for each of the networks and datasets in Table 1. Note that these hyperparameters were tuned for the unpruned network and not for sparse networks.

D.2. Pruning mechanisms

The pruning algorithms that we use as baselines follow a two-step process: first a score is computed for each edge, and then some edges are eliminated based on this score. Below we report various implementation details for each baseline. We use the same training hyperparameters for every baseline.

Random pruning: Upon initialization, all edges are assigned a score. The score follows a uniform distribution with 0-1 range.

Magnitude pruning after training: First, the given dense network is trained. Each edge is assigned a score that is equal to the absolute value of its weight. Then, the network is pruned iteratively, with one epoch of training (fine-tuning) after each pruning iteration. Once the target network density is obtained, the sparse network is fully trained once again. Note that the weights are not reverted back to the initial weights for that second training cycle.

We use the pruning schedule of (Zhu & Gupta, 2017) for iterative pruning. The number of pruning iterations is five for ResNet20 on CIFAR-10, ten for VGG19 on CIFAR-100 and Tiny-ImageNet, and twentyfive for ResNet18 on Tiny-ImageNet.

SNIP: Upon initialization, we compute the SNIP score $\left|\frac{\partial \boldsymbol{L}}{\partial \theta} \odot \theta\right|$ using a random subset of training data. That data is chosen such that they consist of 10 images per class (Lee et al., 2019b). The scores are computed for batches of size 256 for CIFAR-10/100 and 64 for Tiny-ImageNet and summed across batches.

GraSP: Similar to SNIP but the score is given by $\left(H\frac{\partial \boldsymbol{L}}{\partial \theta}\right) \odot \theta$.

SynFlow: Given the desired final network density, we compute the target density at each iteration of the algorithm. Then we compute the SynFlow score given by, $\left|\frac{\partial \boldsymbol{R_{SF}}}{\partial \theta} \odot \theta\right|$ and prune the lowest scoring edges. We repeat this process over 100 iterations, with an exponential decay of the network density, to achieve the target density at the 100^{th} iteration.

SynFlow-L2: Similar to SynFlow but the score is given by

$\left|\frac{\partial \boldsymbol{R_{SF2}}}{\partial \theta^2} \odot \theta\right|$.

PHEW: We first compute the discrete probability distribution for all possible states in both forward and backward directions.

The selection of the starting unit for each random walk in the forward (or backward) direction is performed in a round robin manner.

The random walk process is repeated until the target density is achieved.