OSSuM: A Gradient-Free Approach For Pruning Neural Networks At Initialization

Anonymous authors
Paper under double-blind review

Abstract

Pruning overparameterized neural networks to obtain memory-and-compute-efficient sparse networks is an active area of research. Recent works attempt to prune neural networks at initialization to design sparse networks that can be trained efficiently. In this paper we propose One-Shot Supermasking (OSSuM), a gradient-free, compute-efficient technique to efficiently prune neurons in fully-connected networks. In theory we frame this problem as a neuron subset selection problem, wherein we prune neurons to obtain a better accuracy by optimizing on the cross-entropy loss. In our experiments we show that OSSuM can perform similar to gradient-based pruning techniques at initialization, prior to training. For example, OSSuM can achieve a test set accuracy of 82.4% on MNIST by pruning a 2-layer fully-connected neural network at initialization with just a single forward-pass over the training data. Further, we empirically demonstrate that OSSuM can be used to efficiently prune trained networks as well. We also propose various variants of OSSuM that can be used to prune deeper neural networks.

1 Introduction

With the growth in the size of the neural networks, training them has become prohibitively expensive. Methods that can help tame this complexity have thus been subject of much research. Pruning is one the most common, and effective way of dealing with large networks. Most of the initial works on pruning deal with trained networks, aiming to reduce inference time. However, more recent papers try to use pruning to reduce training complexity. Apart from reducing the computational burden, the existence of sparse networks can often give us insights into the structural characteristics of overparameterized networks, and how gradient-based training behaves on them — a well known such example is the lottery-ticket hypothesis (LTH) (Frankle & Carbin, 2018). LTH demonstrates that at initialization, there exist a subnetwork (lottery ticket or LT) that can be trained in isolation to obtain a classification performance similar to that of the full network. But, it leaves open the question of how to create that subnetwork efficiently without training the full network.

A further dimension to the above question of finding a sparser subnetwork without training the full network is whether we should be looking at structured (e.g. neuron-based pruning) or unstructured (e.g. weight-based pruning) sparsity. On one hand, it is known that neuron pruning is strictly weaker than edge pruning (Malach et al., 2020). But on the other hand, the existing hardware may not necessarily be able to take memory or computational advantage of unstructured sparsity during training.

In this work we tackle the following question — in an overparameterized network, does there exist a *neuron subnetwork at initialization*, of sufficiently small size, that has a performance that outperforms the performance of the full network at initialization. Furthermore, we attempt to find such a subnetwork *without training the full network*, or indeed, without using any gradient steps at all! Notice that if such neuron subnetworks exist at initialization, we have the following attractive properties — 1) the entire network at initialization can be stored by just storing the binary mask and a random seed, and 2) training can be remarkably efficient.

Our contributions can be summarized as follows:

- We give the **first gradient-free** algorithm (OSSuM) that prunes a two-layer network at **initialization** to obtain sparse networks with significantly higher classification performances.

- We demonstrate that even without any scaling or fine-tuning, the above neuron subnetwork has accuracy that is comparable to masks that has been learned via gradient descent, especially at high sparsity.
- For deeper networks, we propose variants of our OSSuM algoithm. Here too, at high to moderate sparsity, the OSSuM variants emerge to be among the most successful ones out of all the considered baselines.
- We also propose OSSuM as a method to prune trained networks and show that it performs much better than the standard default of magnitude pruning, while having a similar computational burden.

2 Related Work

Early works on pruning either use penalties (*e.g.* L_0 or L_1 norms) to enforce sparsity (Chauvin, 1988; Weigend et al., 1991; Ishikawa, 1996) or prune network weights based on some importance criteria (LeCun et al., 1990; Hassibi et al., 1993; Hagiwara, 1993). Mozer & Smolensky (1989) introduce an importance criterion to incorporate the loss function by pruning neurons while Karnin (1990) perform the same by pruning weight edges in the network. These works are sensitive to the scale of the networks since they are designed to be a part of the learning process.

In recent years, large overparameterized networks are shown to be successfully pruned with great results via simple, magnitude-based pruning techniques (Han et al., 2015; Gale et al., 2019; Hayou et al., 2021). Random pruning is also shown to be a strong baseline in the modern pruning literature (Hoefler et al., 2021; Liu et al., 2018). The effectiveness of magnitude-based pruning resulted in works (Guo et al., 2016; Carreira-Perpinán & Idelbayev, 2018) that prune trained networks and then fine-tune them for efficient inference. While the classical methods seek to improve the efficiency of networks at inference, the modern works aim towards finding sparser networks for compute-efficient training and, in turn, faster inference. Zhu & Gupta (2018); Gale et al. (2019) propose techniques that gradually prune the network through out the training process. Evci et al. (2019); Dettmers & Zettlemoyer (2019) introduce methods that update the weight masks dynamically during sparse training. Dai et al. (2019); Ye et al. (2020) suggest techniques to grow the network by adding neurons instead of just pruning.

Frankle & Carbin (2018) show the existence of LTs at initialization and Frankle et al. (2019b;a); You et al. (2019) show the same early in training, obtained using iterative magnitude pruning. Zhou et al. (2019) empirically show that a subnetwork exist at initialization with a good classification performance. They use a gradient-based approach to learn the binary mask at initialization, known as supermask, by optimizing on the loss function. Malach et al. (2020) theoretically show the existence of supermasks in fully-connected networks. Lee et al. (2018); Wang et al. (2020); Tanaka et al. (2020); Verdenius et al. (2020); Lee et al. (2020) introduce various gradient-based techniques to find LTs at initialization. LTs obtained by these algorithms are not shown to be effective supermasks. Frankle et al. (2021) analyze these works and find that iterative magnitude pruning after training networks is better than training pruned networks at initialization.

As pointed out in Blalock et al. (2020), the sparsity in neural networks community suffers from a lack of standardized benchmarks and metrics. Liu et al. (2018) suggests that pruned networks might only perform as good as randomly initialized sparse networks after training with some fixed budget. Most of the works that perform highly sparse pruning use unstructured sparsity. This might not be helpful in making compute or memory efficient networks (Hoefler et al., 2021). Works that focus on structured sparsity rather weight pruning include — coreset-based pruning (Mussay et al., 2019), pruning convolutional filters (Li et al., 2016; Molchanov et al., 2016), sparsity with regularizers (Wen et al., 2016), low-rank approximation (Jaderberg et al., 2014), and tensor factorization (Novikov et al., 2015).

Ranganathan & Lewandowski (2020); Taylor et al. (2016) are works that iteratively perform gradient-free network training by updating the weight parameters. In this work we majorly focus on using our one-shot, gradient-free OSSuM algorithm to prune randomly initialized networks with structural constraints (neuron pruning) to obtain subnetworks with significantly higher classification performances.

3 Neuron Scoring

In this section we derive the neuron scoring strategy. We assume the following setup — we have a fully-connected network with only one hidden layer having d neurons. The j^{th} neuron in the hidden layer is represented by a weight vector $\boldsymbol{p}_j \in \Re^m$, where m is the size of the input layer. We denote each input data point as a tuple $(\boldsymbol{x}_i, y_i)$ where $\boldsymbol{x}_i \in \Re^m$ and $y_i \in [K]$, $X = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$. We denote X_α to be the subset of examples with class label α, $|X_\alpha| = n_\alpha$, and $N = \sum_{\alpha=1}^K n_\alpha$. We further assume that the activation function of the hidden layer is given by a function $\phi(\cdot)$. The weights for the final layer is given by the matrix $\boldsymbol{W} \in \Re^{K\times d}$, the individual weights being denoted by w_{kj}. Hence, for the i^{th} data point, the activation of the j^{th} hidden layer neuron is given by $a_j^i = \phi(\boldsymbol{p}_j^t \boldsymbol{x}_i)$. The value generated by the k^{th} output neuron for the input $\boldsymbol{x}_i$ is thus $o_k^i = \sum_{j\le d} a_j^i w_{kj}$. Let Θ denote the entire set of parameters.

The softmax cross-entropy loss for the training data point $(x_i, y_i = \alpha)$ is given as

$$\mathcal{L}(x_i, y_i = \alpha; \Theta) = -\log\left(\frac{\exp(-o_\alpha^i)}{\sum_k \exp(-o_k^i)}\right) = \log\left(1 + \sum_{k\neq\alpha} \exp(-o_k^i + o_\alpha^i)\right).$$

The average cross-entropy loss for the subset X_α is given by

$$\bar{\mathcal{L}}_\alpha(\Theta) = \frac{1}{n_\alpha}\sum_{i\in X_\alpha} \log\left(1 + \sum_{k\neq\alpha} \exp(-o_k^i + o_\alpha^i)\right).$$

We now define the *loss-preserving neuron selection problem* as following: given a target $s \le d$ that represents the number of neurons, let $S \subseteq [d]$ with $|S| = s$ be a selected subset of the hidden neurons. For each i, for each k, define $\tilde{o}_k^i = \sum_{j\in S} a_j^i w_{kj}$ as the activation of the k^{th} neuron in the final layer. Also define

$$\bar{\mathcal{L}}_\alpha^S(\Theta) = \frac{1}{n_\alpha}\sum_{i\in X_\alpha} \log\left(1 + \sum_{k\neq\alpha} \exp(-\tilde{o}_k^i + \tilde{o}_\alpha^i)\right) \quad \text{and}$$
$$\bar{\mathcal{L}}^S(\Theta) = \frac{1}{N}\sum_\alpha n_\alpha \bar{\mathcal{L}}_\alpha^S(\Theta).$$

The optimal set of neurons to be selected can thus be defined as S^* where,

$$S^* = \arg\min_{S\subseteq[d]} \bar{\mathcal{L}}^S(\Theta). \tag{1}$$

In its full generality, the above algorithm is potentially a hard problem. One possible way to try to tackle this selection problem is to see whether it has structural properties e.g. *submodularity*, which would ensure that algorithms for approximate minimization exist. However, given that we want to claim a result for all possible initializations it is not clear how to show such a property. Our approach tackles the above problem by essentially replacing both the logarithmic and the exponential function with their first order approximations. If such approximations hold, then the above problem can be solved easily. We summarize this in Theorem 3.1.

Theorem 3.1. *Take any combination of a class α and data point i, and $S \subseteq [d]$. Note that the choice of S is reflected in the $\tilde{o}$ values. For all such possible combinations, we assume that the following properties hold:*

1. $\log\left(1 + \sum_{k\neq\alpha}\exp(-\tilde{o}_k^i + \tilde{o}_\alpha^i)\right) \approx \sum_{k\neq\alpha}\exp(-\tilde{o}_k^i + \tilde{o}_\alpha^i)$
2. $\forall k \neq \alpha$, $\exp(-\tilde{o}_k^i + \tilde{o}_\alpha^i) \approx 1 + (-\tilde{o}_k^i + \tilde{o}_\alpha^i)$.

Under these assumptions, there exists a one-shot (i.e. single forward pass over the data) algorithm that finds $\tilde{S}^*$ that has a cross-entropy loss similar to S^*.

Proof. By simply substituting the above claims in the expression that we are optimizing (Equation 1), we have that

$$
\begin{aligned}
N\bar{\mathcal{L}}^S(\Theta) &= \sum_{\alpha=1}^{K} n_\alpha \bar{\mathcal{L}}^S_\alpha(\Theta) = \sum_{\alpha=1}^{K}\sum_{i\in X_\alpha} \log\left(1+\sum_{k\neq\alpha}\exp(-\tilde{o}^i_k+\tilde{o}^i_\alpha)\right) \\
&\approx \sum_{\alpha=1}^{K}\sum_{i\in X_\alpha}\sum_{k\neq\alpha}\exp(-\tilde{o}^i_k+\tilde{o}^i_\alpha) \approx \sum_{\alpha=1}^{K}\sum_{i\in X_\alpha}\sum_{k\neq\alpha}(1-\tilde{o}^i_k+\tilde{o}^i_\alpha) \\
&= N(K-1) - \sum_{\alpha=1}^{K}\sum_{i\in X_\alpha}\left((K-1)\tilde{o}^i_\alpha - \sum_{k\neq\alpha}\tilde{o}^i_k\right) \\
&= N(K-1) - \sum_{\alpha=1}^{K}\sum_{i\in X_\alpha}\left(K\tilde{o}^i_\alpha - \sum_{k}\tilde{o}^i_k\right) \\
&= N(K-1) - \sum_{\alpha=1}^{K}\sum_{i\in X_\alpha}\left(\sum_{j\in S} a^i_j(Kw_{\alpha j} - \sum_k w_{kj})\right) \\
&= N(K-1) - \sum_{j\in S}\sum_{\alpha=1}^{K}\left(\sum_{i\in X_\alpha} a^i_j\right)(Kw_{\alpha j} - \sum_k w_{kj})
\end{aligned}
$$

Notice that in order to minimize the above expression, we need to maximize the second term. Hence we define the OSSuM score for each neuron j as follows:

$$
score_{\text{ OSSuM}}(j) = \sum_{\alpha=1}^{K}\left(\sum_{i\in X_\alpha} a^i_j\right)(Kw_{\alpha j} - \sum_k w_{kj}) \tag{2}
$$

It is clear that given a target cardinality s, if we pick the top-s neurons with the highest $score_{\text{ OSSuM}}(\cdot)$ values, we minimize the above cross-entropy loss expression.

□

While the above derivation is for a specific loss function, the expression derived captures a more general intuition. First, notice that the score is higher for neurons that help differentiate the output of the correct output unit from the wrong one. Furthermore, while the derivation depends on the stated assumptions, the following is true — at higher sparsity, i.e. small $|S|$, assumption (1) is more accurate. Also note that we can derive tighter expressions based on a Taylor series expansion. However, we only work with the first-order expansion that leads to the above Equation 2.

We propose variants of OSSuM using a class-wise score defined in Equation 3. The class-wise ranks of neurons, obtained from the class-wise OSSuM scores, are combined to obtain Linear (Equation 4) and Log OSSuM (Equation 5).

$$
score_{\text{ OSSuM class-wise}}(j,\alpha) = \left(\sum_{i\in X_\alpha} a^i_j\right)(Kw_{\alpha j} - \sum_k w_{kj}) \tag{3}
$$

$$
score_{\text{ Linear OSSuM}}(j) = -\sum_{\alpha=1}^{K} rank(score_{\text{ OSSuM class-wise}}(j,\alpha)) \tag{4}
$$

$$
score_{\text{ Log OSSuM}}(j) = -\sum_{\alpha=1}^{K} \log\left[rank(score_{\text{ OSSuM class-wise}}(j,\alpha))\right] \tag{5}
$$

Algorithm 1 Prune with neuron scores.

```
Input: Data $X$, neuron scoring function $score$, a fully-connected network $f$, pruning parameter $\Delta$.
Output: A pruned subnetwork $\tilde{f}$.
1: $neurons \leftarrow$ a list of hidden neurons in $f$
2: $Scores_i \leftarrow score(neurons_i; X)$ compute $\forall i$
3: $\tilde{f} \leftarrow$ prune bottom-$\Delta$ neurons from $f$ based on $\{Scores_i\}$
4: return network $\tilde{f}$
```

Algorithm 2 OSSuM + PAB.

```
Input: Data $X$, number of hidden layers in subnetwork $L$, number of neurons in $i^{th}$ hidden layer $h_i$, build parameter $b$.
Output: An $L + 1$ layer fully-connected network $f$.
1: $f \leftarrow \emptyset$
2: for $(i = 1;\ L - 1;\ 1)$ do
3:     build two randomly initialized layers to $f$ such that the last hidden layer has $b$ neurons while asserting $f$ to be a valid network for data $X$
4:     prune $b - h_i$ neurons from $f$ using OSSuM and $X$
5:     remove the final layer from $f$ if $i \neq L - 1$
6: end for
7: return network $f$
```

Algorithm 1 provides the generic pruning algorithm for a neural network given a neuron scoring function and training data. Theorem 3.1 is for two-layer networks. For deeper networks, we use a combination of OSSuM + Norm based pruning — the initial layers being pruned using norm of the neurons, followed by the use of OSSuM in the last layer. We also use a variant OSSuM + PAB (prune-and-build), given in Algorithm 2, in which we take as input the size parameters of the full network and the pruned subnetwork, and then we build the desired network using OSSuM on random network.

4 Experiments

In this section we empirically show that OSSuM is effective in pruning fully-connected neural networks both at initialization (prior to training) as well as after training. We compare the variants of our OSSuM algorithm with gradient-based Zhou et al. (2019), magnitude-based Hagiwara (1993); Han et al. (2015), coreset-based (Mussay et al., 2019) and random pruning baselines. We denote a $(l + 1)$-layer fully-connected perceptron network with m input neurons, h_i hidden neurons in the i^{th} hidden layer, and K output neurons by the notation FC-m-h_1-h_2-...h_l-K. In shorthand, we write this as FC-h_1-h_2-...h_l in the remaining text. We use hyperbolic tangent activations and uniform weight initializations in our experiments, if not mentioned explicitly. See Appendix A.1 for more details regarding our experimental setup.

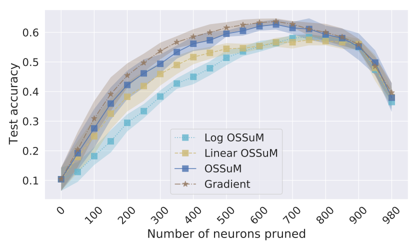
4.1 Baselines

Pruning at initialization: To the best of our knowledge there is no existing gradient-free algorithm that prunes a network at initialization to obtain a sparse network with a higher classification performance. Hence, we propose a gradient-based mask learning technique (Algorithm 3) to learn the mask at initialization, similar to supermasking in Zhou et al. (2019), as a baseline. The major difference is that our baseline is neuron-based supermask when compared to edge-based supermaskin in Zhou et al. (2019). Zhou et al. (2019) first use stochastic gradient descent to learn a continuous mask for a network at initialization, and then discretize this mask to obtain a binary gradient-mask.

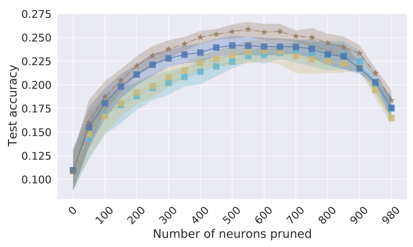
Pruning after training: Most of the pruning literature focuses on pruning after training large over-parameterized networks. One of the most effective methods for pruning post-training is magnitude-based pruning, that uses the magnitude of the weights as the pruning criterion (Han et al., 2015). We use a norm-based pruning approach (Hagiwara, 1993), the neuron pruning counterpart of magnitude-based approach, as a baseline. We also compare OSSuM with coreset-based pruning by Mussay et al. (2019) and random pruning baselines.

Training the pruned networks: Recent works focus on finding memory-and-compute efficient sparse networks at initialization (Lee et al., 2018; Wang et al., 2020; Tanaka et al., 2020) or early during training (You et al., 2019; Frankle et al., 2019a) that can be used for training for various classification tasks. We use variants of OSSuM to obtain sparse subnetworks from a large network without any gradient information, that can be trained in isolation to obtain a network comparable to large trained networks. We use norm-based and random pruning baselines for this setting.

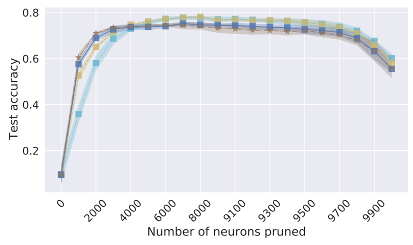
4.2 Two-layer Networks



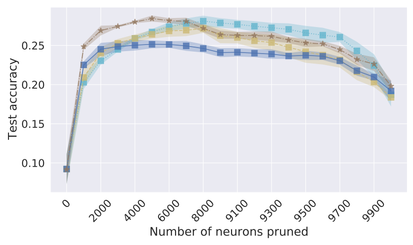
(a) FC-1000 + MNIST



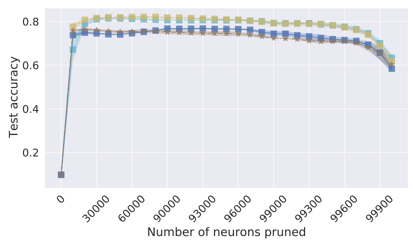
(b) FC-1000 + CIFAR-10



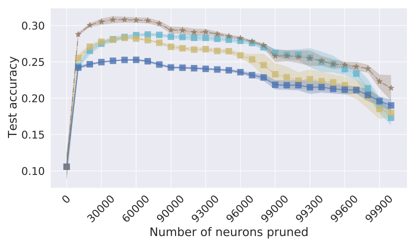
(c) FC-10000 + MNIST



(d) FC-10000 + CIFAR-10



(e) FC-100000 + MNIST

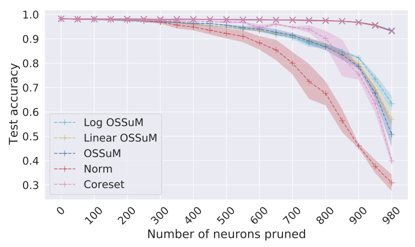


(f) FC-100000 + CIFAR-10

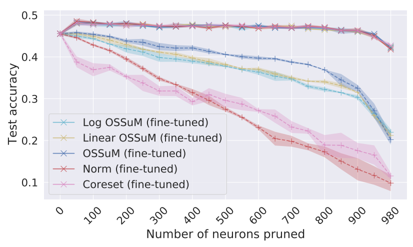
Figure 1: Test accuracies of various two layer pruned networks at different sparsity from randomly initialized full networks (FC-1000, FC-1000-1000, and FC-1000-1000-1000) on MNIST and CIFAR-10 datasets. Each experiment is run for five independent trials and the mean accuracies are reported. The error bands show the standard deviations over five trials. The accuracies of full networks (when 0 neurons are pruned from the hidden layer) at initialization is $\sim 10\%$.

Pruning at intialization: In this experiment setting we consider three different fully-connected architectures, namely, FC-1000, FC-1000-1000 and FC-1000-1000-1000 with MNIST and CIFAR-10 data. Figure 1 shows that OSSuM algorithms perform similar (better in some cases) to the gradient-based masking baseline. OSSuM algorithms obtain maximum test accuracies of $\sim 82\%$ and $\sim 29\%$ over MNIST and CIFAR-10 datasets, respectively.

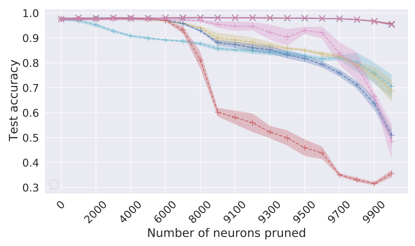
Pruning after training: In this experiment setting, we train the full networks prior to pruning. Figure 2 compares our OSSuM algorithms with coreset-based and norm-based pruning baselines. We do not compare with random pruning since coreset pruning is shown to be better than random pruning by Mussay et al. (2019). We observe that the pruned networks obtained (from trained networks) from our OSSuM algorithms dominate over the baselines *especially at high sparsity*. This effect is more evident with the harder CIFAR-10 dataset. The drop in accuracies of the pruned networks obtained using OSSuM algorithms (towards higher sparsity) is significantly less when compared to the baselines. We also fine-tune the pruned networks to observe that the networks obtained from OSSuM algorithms perform slightly better than other networks at moderate sparsity.



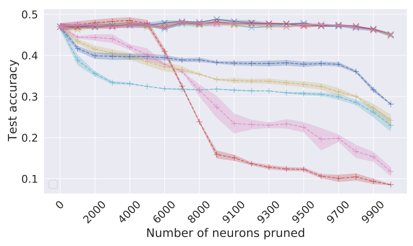
(a) FC-1000 + MNIST



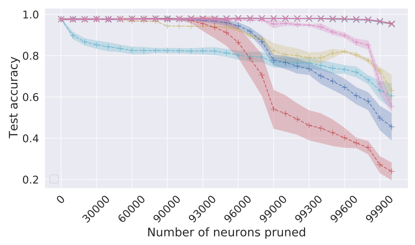
(b) FC-1000 + CIFAR-10



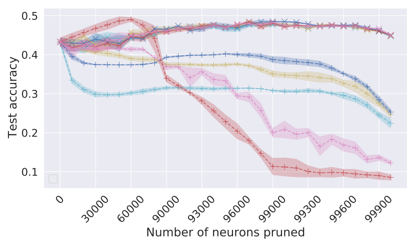
(c) FC-10000 + MNIST



(d) FC-10000 + CIFAR-10

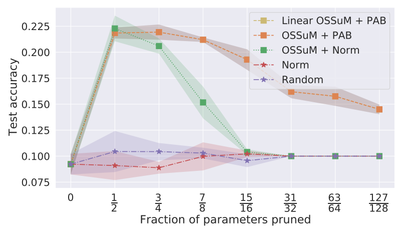


(e) FC-100000 + MNIST

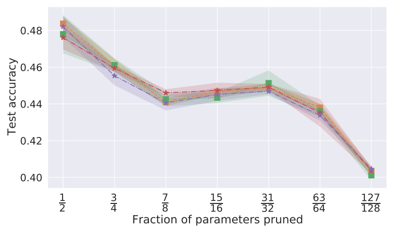


(f) FC-100000 + CIFAR-10

Figure 2: Dashed lines (with marker $+$) show the test accuracies of two layer networks pruned from trained full networks (FC-1000, FC-10000, and FC-100000) at different sparsity. Solid lines (with marker $\times$) show the test accuracies of the pruned networks after fine-tuning. The mean accuracies over three independent trials are reported. Error bands show the standard deviations of accuracies over three trials.



(a) Pruning at initialization

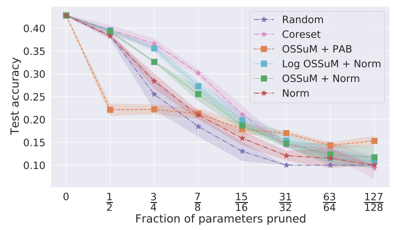


(b) Training after pruning

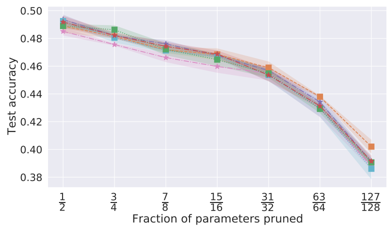
Figure 3: (FC-1000-1000-1000 + CIFAR-10). 3(a) shows the accuracies of sparse networks obtained by pruning the randomly initialized full network at various sparsity. The sparse networks are then trained on CIFAR-10 and compared in 3(b). Mean and standard deviation of accuracies over three independent trials are reported.

4.3 Deeper Perceptron Networks

In this subsection we evaluate the performance of our OSSuM algorithms in pruning fully-connected networks with more than two layers. In all the experiments we prune the same fraction of neurons from each hidden layer.



(a) Pruning after training



(b) Fine-tuning after pruning (post training)

Figure 4: (FC-1000-1000-1000 + CIFAR-10). 4(a) shows the accuracies of sparse networks obtained by pruning the trained full network at various sparsity. The sparse networks are then fine-tuned and compared in 4(b). Mean and standard deviation of accuracies over three independent trials are reported.

Training sparse networks: In this experiment setting we consider randomly initialized full network FC-1000-1000-1000 with CIFAR-10 dataset. First, we use the pruning algorithms to prune networks at initialization to obtain sparse networks. Next, we train the sparse networks on CIFAR-10. In Figure 3, we compare the performance of OSSuM variants after pruning (at initialization) and after training (post-pruning) with other baseline methods (norm and random). OSSuM + PAB prunes the full network at initialization to obtain the best classification performance (Figure 3(a)). In sparsity ranges of $50\% - 75\%$ and $96.8\% - 98.4\%$ OSSuM variants are observed to have an upper hand over the baselines in training post-pruning (Figure 3(b)). The comparison between all the OSSuM variants is shown in appendix (Figure 9). Prior to training the sparse networks, we rescale the weights in a layer with fan_in input neurons so that their maximum absolute value is $1/\sqrt{fan\_in}$ based on Glorot & Bengio (2010); He et al. (2015). In Figure 7 we show that trained OSSuM subnetworks are better than other baselines.

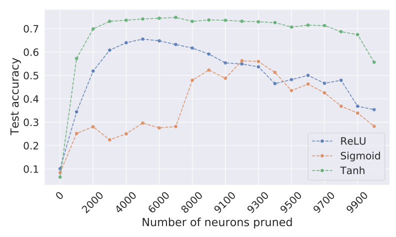
Pruning after training: In this experiment setting, we consider trained FC-1000-1000-1000 with CIFAR-10 dataset. We use our OSSuM algorithms and the baseline methods (coreset, norm, and random) to prune the trained networks, and compare their classification performances at various sparsity (see Figure 4). The pruned networks are then compared after fine-tuning. We find that at very high sparsity OSSuM + PAB and OSSuM + Norm give the best performing subnetwork after pruning the trained full network (Figure 4(a)). At very high sparsity OSSuM + PAB gives the best performing subnetwork obtained by fine-tuning the pruned subnetworks (Figure 4(b)).

4.4 Ablation Studies: Pruning at Initialization

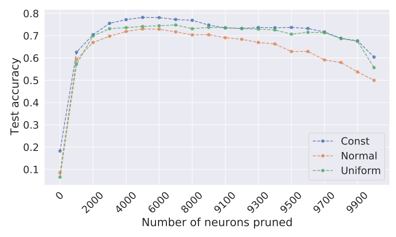
Activation functions: In this experiment we study the effect of OSSuM with various activation functions with FC-10000 on MNIST. We observe that tanh activation is the best for pruning at initialization, followed by ReLU and then sigmoid (see Figure 5(a)). Tanh is significantly better when compared to ReLU and sigmoid at high sparsity.

Network initializations: We use three kinds of initializations for the network (FC-10000 + MNIST) weight parameters for pruning at initialization – 1) Const, where weight parameters can be $-r$ or r with equal probability, 2) Normal, where weight parameters are sampled from the distribution $\mathcal{N}(0, r)$, and 3) Uniform, where weight parameters are sampled from the distribution $\mathcal{U}(-r, r)$ for $r = 1/\sqrt{fan\_in}$. fan_in is the number of input neurons in the layer corresponding to the weight parameter. We infer from Figure 5(b) that Const and Uniform distributions are better than Normal distribution for pruning FC-10000 at initialization. Const is observed to be slightly better than Uniform. But, we use Uniform distribution in our experiments since it is one of the most commonly used weight initializations (Glorot & Bengio, 2010; He et al., 2015).

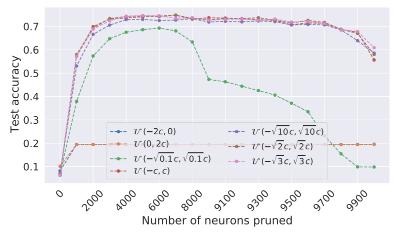
Variance of initialized weights: In this setting we study the effect of variance of the uniform weight intialization distributions for pruning FC-10000 with MNIST datatset at initialization. In Figure 5(c) we find that uniform distributions with zero mean and higher variances work the best.



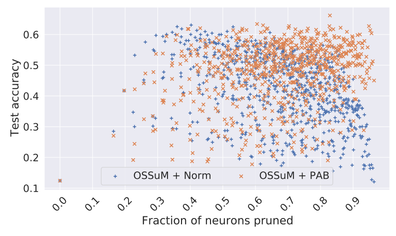
(a) Activation functions



(b) Network initializations



(c) Uniform distributions of weights



(d) Architecture search

Figure 5: Ablation studies. In Figures 5(a)–5(c) we compare the performance of FC-10000 on MNIST dataset with various activation functions, network parameter initializations, and uniform weight distributions, respectively. In Figure 5(d) we compare 730 network architectures searched with our OSSuM algorithms over FC-1000-1000-1000 with MNIST dataset.

Architecture search: We perform pruning using OSSuM on FC-1000-1000-1000 at initialization to obtain a total of 1460 subnetworks. We search using OSSuM over the subnetwork space corresponding to the full network to find out the subnetwork that has the best classification accuracy at initialization for MNIST dataset. We provide the scatter plot in Figure 5(d). Most of the highly sparse networks obtained from OSSuM have over 50% accuracy at initialization. OSSuM + PAB seems to be better than OSSuM + Norm for this task since the former builds each layer using the OSSuM algorithm while the latter uses norm-based approach for pruning the intermediate layers except the final hidden layer. At $\sim 82\%$ sparsity, we observe the maximum accuracy of 66% on MNIST with a four layer network.

5 Discussion

Finding the optimal supermask (neuron mask at initialization) is an NP-hard problem. In this work we consider this problem as a neuron subset selection problem. Using some heuristics, we design OSSuM (the first one-shot, gradient-free, compute-efficient pruning technique) that can prune fully-connected perceptron networks at initialization to obtain subnetworks with significantly higher classification performances. We show that our algorithm can effectively prune two layer networks at initialization to get maximum test set accuracies of 82% and 29% on MNIST and CIFAR-10 datasets, respectively. We also give theoretical backing for our network optimization via neuron pruning. We propose variants of OSSuM that can successfully prune deeper networks with more than two fully-connected layers. OSSuM variants get maximum test set accuracies of 62% and 23% on MNIST and CIFAR-10 datasets, respectively, by pruning four layer networks. We also show that OSSuM can efficiently prune trained networks with significantly lesser drop in accuracy when compared with other baseline pruning methods, at high sparsity. OSSuM can be used to obtain sparse networks that can be used for training. These networks have a slight improvement in classification performance compared to other baseline networks at moderate sparsity. We perform ablation studies with OSSuM to find out that it works the best with uniformly initialized networks with tanh activations. In future work we plan to focus on extending OSSuM for pruning intermediate fully-connected hidden layers and convolutional neural networks.

Reproducibility Statement: We give all the details for implementing our algorithms in Algorithms 1, 2 and Appendix A.1. Moreover, we will make our codes public after the review period. We will submit our codes for reviewing once the discussion phase for all the papers start.

REFERENCES

Davis W. Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag. What is the state of neural network pruning? *CoRR*, abs/2003.03033, 2020. URL `https://arxiv.org/abs/2003.03033`.

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.

Yves Chauvin. A back-propagation algorithm with optimal use of hidden units. In *NIPS*, volume 1, pp. 519–526, 1988.

Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497, 2019.

Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. *CoRR*, abs/1911.11134, 2019. URL `http://arxiv.org/abs/1911.11134`.

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*, 2019a.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Linear mode connectivity and the lottery ticket hypothesis. *CoRR*, abs/1912.05671, 2019b. URL `http://arxiv.org/abs/1912.05671`.

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`.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 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. JMLR Workshop and Conference Proceedings, 2010.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. *arXiv preprint arXiv:1608.04493*, 2016.

M. Hagiwara. Removal of hidden units and weights for back propagation networks. In *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, volume 1, pp. 351–354 vol.1, 1993. doi: 10.1109/IJCNN.1993.713929.

Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pp. 293–299. IEEE, 1993.

Soufiane Hayou, Jean-Francois Ton, Arnaud Doucet, and Yee Whye Teh. Robust pruning at initialization. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=vXj_ucZQ4hA`.

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.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021.

Masumi Ishikawa. Structural learning with forgetting. *Neural networks*, 9(3):509–521, 1996.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

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

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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.

Namhoon Lee, Thalaiyasingam Ajanthan, Stephen Gould, and Philip H. S. Torr. A signal propagation perspective for pruning neural networks at initialization. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=HJeTo2VFwH`.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

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

Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. In *International Conference on Machine Learning*, pp. 6682–6691. PMLR, 2020.

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.

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.

Ben Mussay, Margarita Osadchy, Vladimir Braverman, Samson Zhou, and Dan Feldman. Data-independent neural pruning via coresets. *arXiv preprint arXiv:1907.04018*, 2019.

Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks. *arXiv preprint arXiv:1509.06569*, 2015.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12 (1):145–151, 1999.

Varun Ranganathan and Alex Lewandowski. ZORB: A derivative-free backpropagation algorithm for neural networks. *CoRR*, abs/2011.08895, 2020. URL `https://arxiv.org/abs/2011.08895`.

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.

Gavin Taylor, Ryan Burmeister, Zheng Xu, Bharat Singh, Ankit Patel, and Tom Goldstein. Training neural networks without gradients: A scalable admm approach. In *International conference on machine learning*, pp. 2722–2731. PMLR, 2016.

Stijn Verdenius, Maarten Stol, and Patrick Forré. Pruning via iterative ranking of sensitivity statistics. *CoRR*, abs/2006.00896, 2020. URL `https://arxiv.org/abs/2006.00896`.

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

Andreas S Weigend, David E Rumelhart, and Bernardo A Huberman. Generalization by weight-elimination with application to forecasting. In *Advances in neural information processing systems*, pp. 875–882, 1991.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29:2074–2082, 2016.

Mao Ye, Chengyue Gong, Lizhen Nie, Denny Zhou, Adam Klivans, and Qiang Liu. Good subnetworks provably exist: Pruning via greedy forward selection. *CoRR*, abs/2003.01794, 2020. URL `https://arxiv.org/abs/2003.01794`.

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

Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *arXiv preprint arXiv:1905.01067*, 2019.

Michael H. Zhu and Suyog Gupta. To prune, or not to prune: Exploring the efficacy of pruning for model compression, 2018. URL `https://openreview.net/forum?id=S1lN69AT-`.

A Appendix

A.1 Experimental details

For our experiments, we use a computer with Nvidia Tesla V100 GPU with 32 GB memory and 28 CPUs. Codes for our work will be made public after the review period. We consider fully-connected perceptron networks for our experiments. We use networks with tanh activations since in Section 4.4 we observe them to give the best results when pruned at initialization. We use uniform weight initializations based on results from Section 4.4. We use two main distributions for weight parameter initializations – 1) Uniform $\mathcal{U}(-r, r)$, and 2) Normal $\mathcal{N}(0, r)$, where $r = 1/\sqrt{fan\_in}$. fan_in refers to the number of input neurons to the layer corresponding to the weight parameter (He et al., 2015).

Architectures: In the two-layer network setting we consider three full network architectures each for MNIST and CIFAR-10 – 1) FC-1000, 2) FC-10000, and 3) FC-100000. Note that for MNIST FC-1000 refers to a perceptron network with $784 - 1000 - 10$ neuron architecture (from input to output layer) while for CIFAR-10 it refers to network with $3072 - 1000 - 10$ neuron architecture. In the deeper network setting we consider the full network architecture FC-1000-1000-1000.

Datasets and training: MNIST dataset is normalized assuming it's mean to be 0.13066 and standard deviation (std) to be 0.30810. CIFAR-10 is normalized assuming it's channel-wise mean to be (0.5, 0.5, 0.5) and std to be (0.5, 0.5, 0.5). We use cross-entropy loss for training all of our networks. SGD + momentum (Qian, 1999) is the optimizer that we use. We train the sparse networks using Adam Kingma & Ba (2014) with its default parameters, to avoid the need of any hyper-parameter tuning. To be fair, in all the experiments we do not perform any hyperparameter tuning with respect to any algorithm. Refer to our codes provided in the supplementary files for more details.

Pruning at initialization: We use pruning algorithms to prune randomly initialized networks (supermasking) and then compare OSSuM with gradient-based pruning (Algorithm 3). For MNIST dataset, we learn gradient-mask using the learning rate $\eta = 0.1$ for 10 epochs and then $\eta = 0.01$

Algorithm 3 Gradient supermasking.

```
Input: Data $X$, batch size $b$, number of training steps $T$, a randomly initialized $L$-layer network $f = f_L \circ f_{L-1} \cdots \circ f_1$, a randomly initialized neuron mask matrix $\boldsymbol{M}^0$, number of neurons to be pruned per layer $delete$.
Output: Discretized neuron binary mask $\tilde{\boldsymbol{M}}^T$.
1: freeze weight parameters of $f$
2: masked network at training step $t$, $\tilde{f}^t = f_L \circ (f_{L-1} \odot \boldsymbol{M}^t_{L-1}) \cdots \circ (f_1 \odot \boldsymbol{M}^t_1)$ where masks $\boldsymbol{M}^t_i$'s are trainable parameters
3: for $(t = 1;\ T;\ 1)$ do
4:     uniformly sample mini-batch $\boldsymbol{B}_t$ of size $b$ from $X$
5:     use SGD to update $\boldsymbol{M}^t_i$'s with cross-entropy loss on mini-batch $\boldsymbol{B}_t$
6: end for
7: $\tilde{\boldsymbol{M}}^T \leftarrow$ select bottom-$delete$ neurons from each $\boldsymbol{M}^T_i$ for $i = 1, \cdots, L-1$ and assign corresponding $\boldsymbol{M}^T_i$ values to 0 and the rest to 1.
8: return binary neuron mask $\tilde{\boldsymbol{M}}^T$
```

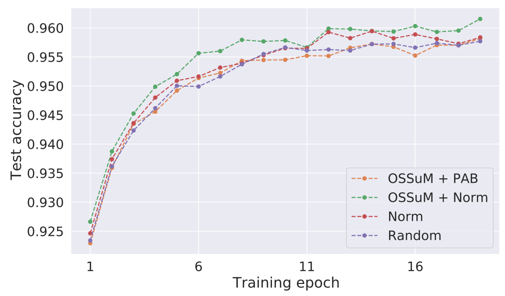
for 5 epochs. For CIFAR-10 dataset, we learn gradient-mask using the learning rate $\eta = 0.1$ for 20 epochs and then $\eta = 0.01$ for 10 epochs.

Pruning after training: We use pruning algorithms to prune randomly initialized networks (supermasking) and then compare OSSuM with gradient-based pruning (Algorithm 3). For two-layer networks, we have the following training schedule (η is the learning rate) for: 1) MNIST – $\eta = 0.1$ for 5 epochs, $\eta = 0.01$ for 5 epochs and then $\eta = 0.001$ for 5 epochs, and 2) CIFAR-10 – $\eta = 0.1$ for 10 epochs, $\eta = 0.01$ for 10 epochs and $\eta = 0.001$ for 10 epochs. We train with SGD and momentum ($= 0.9$). We perform fine-tuning post-pruning. With MNIST dataset the pruned network is fine-tuned for 7 epochs with $\eta = 0.01$. With MNIST dataset the pruned network is fine-tuned for 15 epochs with $\eta = 0.01$. For deeper networks, we use Adam optimizer with default parameters. For this setting networks with MNIST datasets are trained for 20 epochs while that for CIFAR-10 is 50 epochs. Fine-tuning for MNIST is performed for 10 epochs while that for CIFAR-10 is 15 epochs. We use the GitHub code from `https://github.com/BenMussay/Data-Independent-Neural-Pruning-via-Coresets` by Mussay et al. (2019) for implementing the coreset-based pruning.

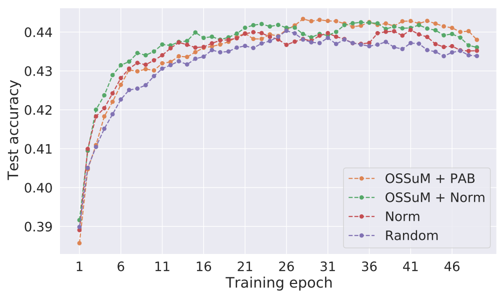
Training the pruned networks: In this setting we use our pruning algorithms to compute supermasks. The weight parameters of the pruned networks are scaled to have a maximum absolute value of $1/\sqrt{fan\_in}$. Scaling is required for having good initializations when training deeper networks (He et al., 2015; Glorot & Bengio, 2010). We use Adam optimizer to train networks on MNIST dataset for 20 epochs and CIFAR-10 dataset for 50 epochs.

Extended observations:

- While training the pruned subnetworks, we find that at some sparsity levels the learning curve of networks obtained from OSSuM algorithms is faster and superior to other baselines. We show two such instances in Figure 6.
- We notice that OSSuM + PAB performs the best while training the pruned subnetworks obtained from deeper networks without rescaling the weights after pruning at most of the sparsity range (see Figure 7).
- Using OSSuM pruning at initialization, we observe that the distribution of weights post-pruning is preserved. Figure 8 shows that at various sparsity mean, variance, minimum value, maximum value, and ratio of positive to negative weight parameters remain approximately the same.
- We compare all the OSSuM variants in Figure 9 where they are used to obtain sparse networks at initialization for training.

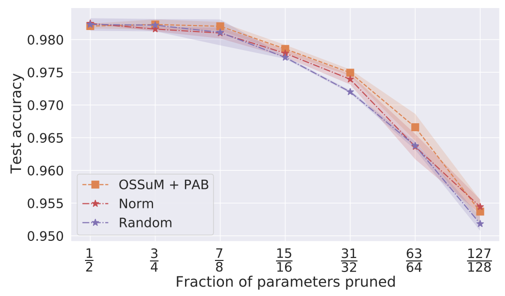


(a) 128× smaller network + MNIST

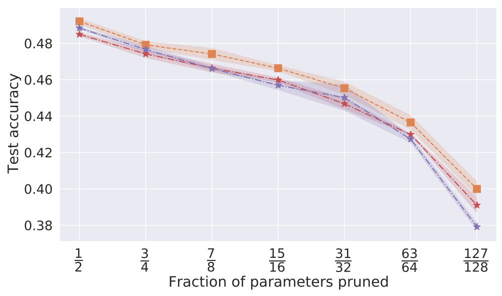


(b) 64× smaller network + CIFAR-10

Figure 6: In Figure 6(a), the learning curve of a pruned subnetwork from FC-1000-1000-1000 (128× compressed network) on MNIST is shown. In Figure 6(b), the learning curve of a pruned subnetwork from FC-1000-1000-1000 (64× compressed network) on CIFAR-10 is shown.

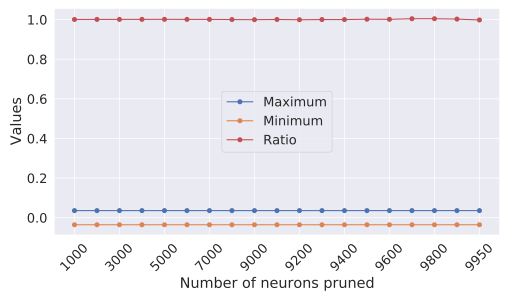


(a) MNIST

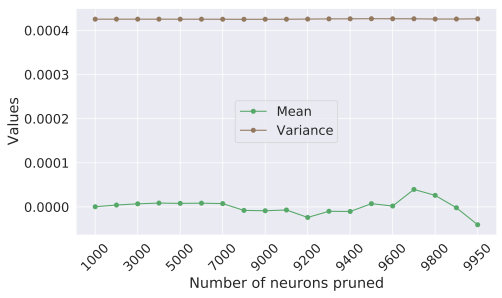


(b) CIFAR-10

Figure 7: FC-1000-1000-1000 full networ. Pruning the full network and training the subnetworks at various sparsity over MNIST and CIFAR-10 data. Mean and standard deviation of accuracies over three independent trials are reported.

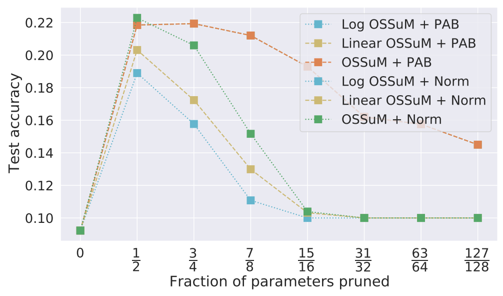


(a) Maximum and minimum

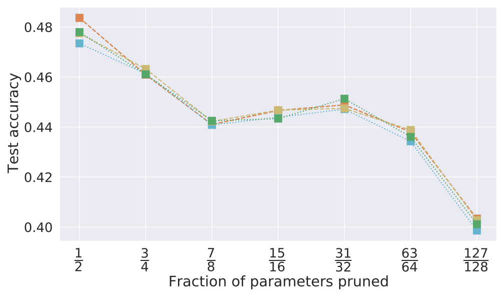


(b) Mean and variance

Figure 8: FC-10000 + MNIST. Figure 8(a) shows the maximum, minimum, and the ratio of the number of positive to negative weight parameters in the hidden layer of pruned networks at various sparsity obtained from OSSuM. Figure 8(b) shows the mean and variance of the weight parameters in the hidden layer of pruned networks at various sparsity obtained from OSSuM.



(a) Pruning at intialization



(b) Training pruned subnetworks

Figure 9: Full network FC-1000-1000-1000 + CIFAR-10. Figure 9(a) shows the performance of all the OSSuM variants when the randomly initialized full network is pruned at various sparsity. Figure 9(b) compares the performance of pruned subnetworks post-training.