Report :By Abhigyan Dutta

A MINIMAL GRADIENT FLOW METHOD FOR ADAPTIVE NODE PRUNING IN CONVOLUTIONAL NETWORKS

Abhigyan Dutta · **Pabitra Mitra**

September 2019

Abstract Pruning neural networks provides an effective solution to the problem of large training time and overfitting in convolutional neural networks. Contemporary pruning techniques eliminate weights based on human decided heuristics. Instead, we identify subnetworks having the least gradient flow and prune them altogether. This is done in an adaptive and evolving manner during the learning epochs. Large reduction in the number of weights may be quickly achieved in this process as compared to link dropout strategies. Empirically not only do we observe that the reduced network has comparable accuracy as the original one, but the loss is also significantly better as compared to a Neural Network with a similar architecture.

Keywords Pruning · Adaptive · Dropout · Regularizer

1 Introduction

It is always desirable to reduce the size of a Neural Network for various purposes like faster computation, reducing overfitting, reducing resource requirements, etc without significantly harming accuracy [3] . This is especially important in modern times, as it has been found out that deep learning techniques with a huge number of paramaters, have a large carbon footprint [19].

Contemporary successful ways of reducing parameter count are by pruning the Neural Network used in deep learning. For this, a number of heuristics have been

Abhigyan Dutta*
Department of EECS
Indian Institute of Science
Bengaluru, PA 560012
E-mail: abhigyand@iisc.ac.in

Pabitra Mitra
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur
Kharagpur, PA 721302
E-mail: pabitra@cse.iitkgp.ac.in

proposed as discussed in Related Work section. The question lies in how to decide the heuristic methods to prune a Neural Network. To this end, we propose a novel heuristic method, based on the Adaptive Dropout paradigm [1], to prune the nodes of a Neural Network and thus reduce parameter count significantly.

To summarize the effects of pruning a Neural Network [17] has proposed a conjecture, which states, that given a sufficiently overparametrized and randomly initialized network, there exists a subnetwork within it which can achieve an accuracy comparable to the original overparametrized network. This stronger conjecture has been proven [13], according to which although the above conjecture is true, pruning weights is a computationally hard problem in the worst case, and thus it should be viewed something as similar to weight optimization in a Neural Network. Thus, according to the authors, pruning networks by heuristic approaches is the best way to move forward as it has been shown to work well in practice.

The paper is organized as follows; we first discuss related work in this field, then we propose a novel method to prune Neural Networks and discuss the theoretical aspects of it. Finally, we show some experimental results to demonstrate the effectiveness of our method.

2 Related Work

Most node pruning techniques use heuristics designed by a human to decide on the elimination of weights and nodes. Here we discuss a few different approaches of pruning among many which exist.

A recent successful technique [5] have used weight pruning to lower the parameter count of a Neural significantly Network while also increasing the accuracy of the network. In the paper the authors have introduced the notion of Lottery Ticket Hypothesis which claims that a randomly initialized Neural Network has a sub network, which when trained in isolation, can achieve accuracy comparable to the original large Neural Network. The authors identify these subnetworks as the network formed by a fixed percentage of highest magnitude weights and prune the rest of the Neural Network.

In [12], authors have proposed to weigh a CNN filter by using a factor γ and introducing sparsity in this factor by regularizing it with a sparsity inducing norm l_1 regularizer. An old technique of node pruning[7] has introduced measures which determine the importance of a node in a layer based on either its contribution to nodes in the subsequent layer (Goodness Factor) or how it associates itself with the nodes previous and the subsequent layers (Consuming Energy Factor), another proposed method [8] is to remove nodes based on measures like the relative changes in the activation of a node with respect to different training examples, incoming weights of nodes, outgoing weights of nodes.

A technique [21] inspired by brain functioning suggests evaluating Neural Network nodes by determining how well they integrate with the rest of the Neural Network nodes. This is evaluated by measuring the mutual information of the nodes of the Neural Network.

Another method, [14] similar to our approach proposes to prune nodes by removing nodes with lowest average activation values over all training samples. The difference in our approach would be we allow the Neural Network to decide the most favourable subnetwork by itself while training.

An adaptive technique to determine the importance of weights has been proposed by [17] in which it is adaptively determined, while training decide the importance factor of a weighted connection, update it, and also prune the Neural Network according to the importance factor associated with the weighted connection.

3 Contributon

In this paper, we concentrate our efforts on pruning the nodes of the fully connected layer in a CNN [10], for the classification problem. We know that in general, most of the parameters of a CNN is concentrated in the Fully Connected layers [11], so reducing the number of nodes in the Fully Connected Layer will lower parameter count by significant amounts.

The main contribution of our method is that, we do not decide the heuristics of node pruning; instead we consider the relative importance of the nodes adaptively as the training proceeds. The pruning of nodes also reduces the network's size dynamically, thus decreasing the gradient flow, hence smoothing down the perturbations and regularizing learning over time. This can be seen in the relative smoothness (upon close inspection), for a short time, of the loss function after the complete pruning has been done as compared to un-pruned networks.

So why prune nodes dynamically, while training, from Fully Connected Layers, instead of training the CNN with a reduced number of nodes from the beginning? We observe empirically, for a small number of training epochs, the performance of CNN trained by dynamically pruning nodes from the Fully Connected Layer trumps the performance if we start training the CNN with a reduced number of nodes in the Fully Connected Layer from the beginning. Our method gives the added advantage of easily scaling down the CNN to meet resource constraints (without drastically altering performance), without the requirement of re-training the CNN from the start with the scaled down number of nodes (i.e our method prunes the least important nodes). The pruning also has a smoothing and regularizing effect on the loss function. One of the striking features to notice in the experimental loss curve is that the Pruning method always achieves significantly lower loss and somewhat smoother, thus giving us an idea that the Pruning method can be used to a great effect as a learning regularizer. We think that this is due to minimal gradient flow, which prevents significant changes in the loss.

Dropout is one of the simplest and best methods of preventing over-fitting in a Neural Network [18]. In this method, while training, we randomly drop nodes from a layer with some probability (Bernoulli) p but perform prediction using all the nodes after they are scaled according to their probability so as to get an expectation. So it begs the question for the choice of p. Adaptive dropout for training deep neural networks [1] is a proposed method in which the Neural Network, by itself through training, learns the relative importance of a particular node and adjusts the dropout

probability accordingly of every node. Thus, the method has the ability to increase the retention probability of nodes representing important information and vice-versa.

The authors of Adaptive Dropout have claimed that the resulting in mask probabilities, should not be viewed as a distribution over hidden variables, instead it should be viewed as an approximation to a Bayesian posterior distribution over different model architectures. This makes us think whether this can be an effective tool for Bayesian model selection [15] and whether we should eliminate un-necessary parameters, to maximize the likelihood of data for a given model and its parameters. Thus, in our proposed pruning model, we leverage Adaptive Dropout property to prune unimportant nodes and save the Neural Network from permanent damage or drastic performance drop by dynamically pruning the network while training.

4 Adaptive Dropout

In this section, we discuss the method and implementation of Adaptive Dropout. The Adaptive Dropout method banks on the Neural Network to learn the dropout probabilities for each node dynamically while training. This can help the Neural Network combine different unimportant or similar features to a single node freeing up other nodes. Or, in a different interpretation, we can view the Neural Net as increasing the retention probability for a node responsible for passing information about an important feature. The authors have used Adaptive Dropout mainly for training Auto-Encoders [2]. Here, we use the Adaptive Dropout method exclusively for classification tasks.

Using similar notations form the original paper we describe the method of Adaptive Dropout. In the Adaptive Dropout paradigm the nodes from previous layer $l-1$ denoted by subscript i determine the dropout for the nodes of the next layer l denoted by subscript j. In the original Dropout method, while training, we determine the output from a layer as:

$$a_j^{[l]} = m_j^{[l]} g(\sum_j w_{i,j}^{[l-1]} a_i^{[l-1]}) \tag{1}$$

where a stands for the activation output of a layer, and $m_j^{[l]}$ is the dropout mask, over a node, obtained by sampling according to dropout probability p whether a node will be retained or not. The sampling is done as a Bernoulli process, 1 for retention 0 for rejection i.e. $P(m_j^{[l]} = 1) = p$.

In the Adaptive Dropout method:

$$P(m_j^{[l]}) = f(\sum_j \pi_{i,j}^{[l-1]} a_i^{[l-1]}) \tag{2}$$

where f is a suitable function and $\pi_{i,j}^{[l-1]}$ are called Standout weights, and this network mapping the previous layer to dropout probabilities is called the Standout Network. The f is chosen as the sigmoid function here, $\frac{1}{1+e^{-z}}$ since it provides the mapping of $f : \mathbb{R} \to [0,1]$, but it can be replaced by other suitable functions too.

During testing time we replace the feedforward process by taking the expectation of equation (1) as per the standard rules of dropout:

$$\mathbb{E}[a_j^{[l]}] = f(\sum_i \pi_{i,j}^{[l-1]} a_i^{[l-1]}) g(\sum_j w_{i,j}^{[l-1]} a_i^{[l-1]}) \tag{3}$$

If we denote the empirical loss function as $L_S(m,w)$, for easier representation, only dependence on tunable parameters is shown. Thus, we want $P(m|\pi,w)$ to be as close to the best architecture or model indicated by $L_S(m,w)$, while also maximizing the likelihood of data with respect to w. The free energy denoted by $F(P,L_S)$, and the original authors have made an approximation that if we assume $P(m|\pi,w)$ to be close to the true posterior, then the derivative of $F(P,L_S)$ w.r.t P can be approximated to be 0 and all the $\frac{\partial P}{\partial w}$ terms may be ignored, giving us the final approximate gradient with which we can train the Neural Network:

$$-\sum_m P(m|\pi,w) \frac{\partial}{\partial w} log L_S(m,w) \tag{4}$$

To reduce the computational cost, the authors have approximated the Standout Network weights $\pi_{i,j}^{[l-1]}$ as $\alpha w_{i,j}^{[l-1]} + \beta$ where α and β are the Standout Network hyperparameters to be chosen before training. $\alpha = 0$ makes the effect of training on dropout 0 and thus effectively, we have reverted back to the case of sampling from a $Bernoulli(\beta)$

5 Pruning Nodes with Adaptive Dropout

This section describes our method of Pruning Nodes from a Fully Connected Layer using the Adaptive Dropout approach. We hypothesize that nodes of a layer with the highest dropout probability, or lowest retention probability are the least important in terms of its contribution to the Neural Network for some classification task.

This is based on the fact, that during training, the node is most likely to be dropped out very often and thus there is no gradient flow through it. Thus, we are trying to uncover a subnetwork through which most of the gradient flow occurs and is thus the most important and active in the Neural Network's proper functioning. Since, in the Adaptive Dropout approach, the dropout probabilities are directly related to the weights or the parameters of the Neural Network, a node which is dropped most of the times depends on other nodes to adjust weights of earlier layers accordingly so that its retention probability increases. So we must give sufficient training time, before pruning nodes to make sure that the earlier layers are trained enough to determine the dropout probability of a node which might be suffering from high dropout (due to random initialization), but might be actually important.

We will call this preparatory training time before every pruning of the Neural Network, as pruning period, while pruning ratio is defined as the ratio of nodes pruned from the remainder of the nodes in a layer. Thus, during training after every pruning period, we prune the nodes by the pruning ratio. The nodes pruned are the nodes with the lowest retention or highest dropout probabilities. Pruning period and pruning ratio are hyperparameters which need to be set beforehand.

Following some standard notations, we formally define our training and test process as:

5.1 Training

We define a Neural Network layer l as $f(x; w_{i,j}^{[l-1]}, n^{[l]}, m^{[l]})$ where x is the input to the layer, $w^{[l-1]}$ are the parameters of the layer connecting layer $[l-1]$ to $[l]$, m is the binary mask obtained by sampling from probabilities given by the Standout Network, while n is the pruning mask which prunes the nodes of the layers. Here the mask n is not be confused with the dropout mask, while m is sampled, n is fixed and reflects the pruned network. It is changed only when a pruning of nodes event occurs.

$$a^{[l]} = f(a^{[l-1]}; w^{[l-1]}, n^{[l]}, m^{[l]}) = n_i^{[l]} m_i^{[l]} g(\sum_i w_{i,j}^{[l-1]} a_i^{[l-1]}) \tag{5}$$

The loss function is considered the same as in the aforementioned Adaptive Dropout case along with all the approximations made by the original authors.

After we train for a certain number of epochs called prune period, we adjust $n^{[l]}$ such that the nodes with the lowest probabilities are assigned 0, or in other words they are rejected from the training and testing process from then on, and thus effectively pruned.

5.2 Testing

For testing we use only the nodes which have not been pruned, along with the conventional technique of dropout i.e. replacing the sampling operation in training time with expectations.

$$\mathbb{E}[a_j^{[l]}] = n_j^{[l]} f(\sum_i \pi_{i,j}^{[l-1]} a_i^{[l-1]}) g(\sum_i w_{i,j}^{[l-1]} a_i^{[l-1]}) \tag{6}$$

6 Algorithm

A rough qualitative description of the algorithm will be that we use the dropout probabilities, which is decided by the Adaptive Dropout method described above, to prune the nodes of a Neural Network. We first train the Neural Network for a few epochs, for the dropout probabilities, decided by the Neural Network itself, to start reflecting some sort of determinism in choosing the best nodes for classification by assigning the highest retention probabilities to most important nodes. And then we prune the Neural Network by removing some percentage of nodes with the lowest retention probabilities. We continue this process until we have pruned the desired number of nodes. Here α and β are the parameters associated with Adaptive Dropout. Pruning Ratio γ decides on the number of nodes to prune from the nodes remaining, and Pruning Period P_l is the number of epochs after which the pruning occurs. The rest of the notations are similar to the ones used in describing Adaptive Dropout. Below we provide a rough algorithm of how to use Adaptive Dropout to prune a layer l:

Algorithm 1 Algorithm for Pruning Neural Networks

Input: α, β, epochs, pruning ratio = γ,pruning period = P_t, L_S

Initialize w randomly, set $\pi^{[l-1]} = w^{[l-1]}$, $ratio = 0$, $i \in [l]$, $pSum_i^{[l]} = 0$, $nodesPruned^{[l]} = 0$, $n^{[l]} = 0$
while $e < epochs$ **do**
  if $e \pmod{P_t} = 0$ **then**
    $ratio \rightarrow ratio + \gamma(1 - ratio)$
    for all $i \in layer[l]$) **do**
      procedure PRUNE NODES
        $nodesPruned^{[l]} \rightarrow ratio \times l_{size} - nodesPruned^{[l]}$
        while $n^{[l]} < nodesPruned^{[l]}$ **do**:
          Delete node with least retention chance tracked by $pSum^{[l]}$
          $n^{[l]} \rightarrow n^{[l]} + 1$
        end while
      end procedure
    end for
  else
    for all $i \in layer[l]$) **do**
      $P(m_i = 1) = f(\alpha \sum_j \pi_{j,i}^{[l-1]} a_j^{[l-1]} + \beta)$
      $m_i \sim P(m_i = 1)$
      $a_i = m_i g(\sum_j w_{j,i}^{[l-1]} a_j^{[l-1]})$
      $pSum_i^{[l]} = 0$
      $pSum_i^{[l]} \rightarrow pSum_i^{[l]} + P(m_i = 1)$ ▷ Only a single batch used for training else sum for all minibatches
    end for
  end if
  Update neural network parametrs w using $\frac{\partial}{\partial w} logL_S(m, w)$
end while

7 Experiments

7.1 Experimental Setup:

We use the Fashion MNIST dataset [20] to perform our experiments, since it is more challenging for classification task than MNIST. The code used in the experiment can be found here [5]. Refer to Appendix A for more details about the experimental setup and the code used to perform the experiment.

7.2 General Structure of the CNN Classifier Used

For our experiments performed, we have used one of the following CNN architectures, and we have applied our method of Pruning with Dropout in the last Fully Connected Layer:

Table 1 Structure of CNN's used. We will use the first bigger structure more extensively and we have also described a smaller CNN structure

Description	Parameters
Conv(5,5), Filters-32, Stride-2	800
Pool(5,5), Stride-2	0
Conv(5,5), Filters-64, Stride-2	1600
Pool(5,5), Stride-2	0
Conv(5,5), Filters-128, Stride-2	3200
Pool(5,5), Stride-2, Stride-2	0
FC-512	589824
Standout Network	2
Pruning Mask	0
FC-10 (Binary Crosss Entropy)	5120

Description	Parameters
Conv(5,5), Filters-24, Stride-2	600
Pool(5,5), Stride-2	0
Conv(5,5), Filters-48, Stride-2	1200
Pool(5,5), Stride-2	0
Conv(5,5), Filters-64, Stride-2	1600
Pool(5,5), Stride-2, Stride-2	0
FC-512	294912
Standout Network	2
Pruning Mask	0
FC-10 (Binary Crosss Entropy)	5120

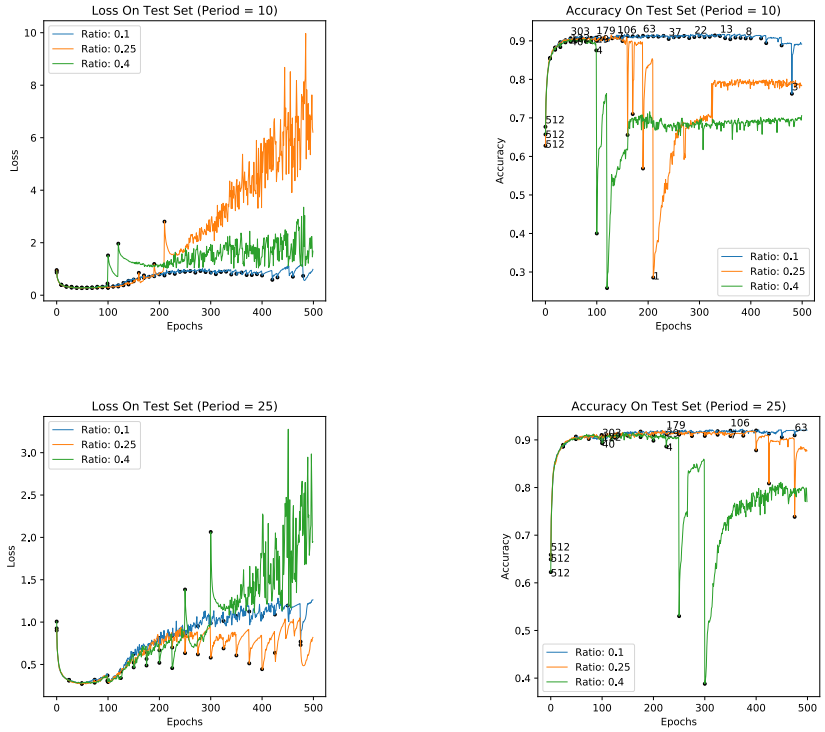

Fig. 1 Here we see the effect of pruning. Loss curve on validation set is on the left, and accuracy curve on the right. The black dots indicate where pruning has occured. To avoid cluttering we show the number of nodes remaining for every 5 periods passed (we just show dots in the loss graph). Pruning is stopped where there is only 1 node remaining.

7.3 Experiment 1: General Visualization and Comparison

We visualize what happens when we prune a Neural Network to get a general intuition of its effect on the loss and accuracy curves on the test set. This will roughly explain the trade-off between the hyperparameters of Pruning Period v/s Pruning Ratio. We continue pruning till there is only a single node left in the Fully Connected Layer.

From the above figure, we can clearly see that whenever a pruning event has occurred, there has been a spike, degrading the performance of the Neural Network, but the Neural Network soon recovers if the period of recovery i.e. the epochs between successive pruning is sufficient or when the ratio of pruning is small. There must be proper balancing between these two factors, otherwise a Neural Network might take very long to recover or may not recover at all.

Next, we compare the Neural Network's performance undergoing Pruning till it reaches a fixed number of nodes, and Neural Networks trained that with a fixed number of nodes from beginning and compare it to a standard Fully Connected Layer with 512 nodes with dropout $= 0.5$. As discussed earlier, the Pruning method always achieves significantly lower loss and is somewhat smoother, thus can be interpreted as having a great effect as a learning regularizer.

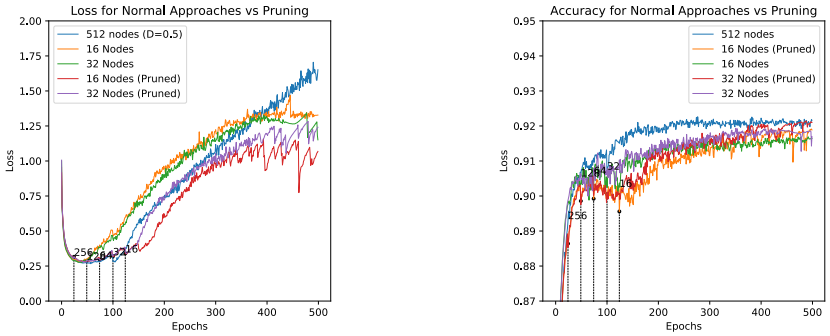

Fig. 2 Performance comparison of adaptively pruned Neural Network vs fixed Neural Networks

7.4 Experiment 2: Comparison of Pruning vs Normal Methods:

In this section, we compare the CNN's performance when we prune the Fully Connected Layer up to a fixed number of nodes versus when we start training, with the fixed number of nodes. We do this for both ReLu and LeakyRelu versions. The training process for the Adaptively Pruned Neural Network follows a very simple scheme: We train the CNN with SGD Optimizer with momentum till it reaches the fixed number of nodes we want to prune up to, then we run another Pruning Period number of epochs for the loss to stabilize, followed by epochs of optimization with Adam Optimizer for fine-tuning. Refer to (Section A) for more details about the experimental setup.

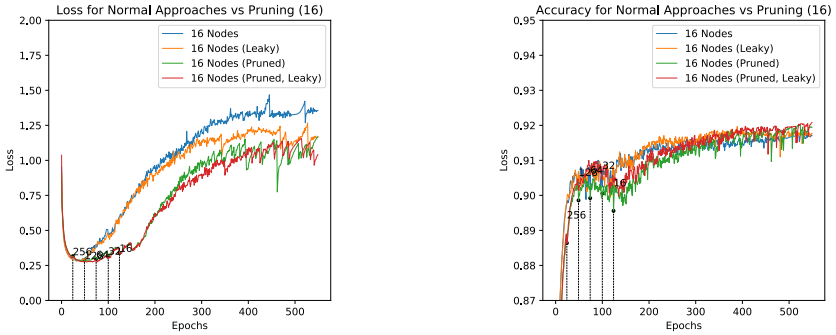

Fig. 3 Fully Connected Layer contains 16 Nodes, wither arrived at by pruning or contains from start.

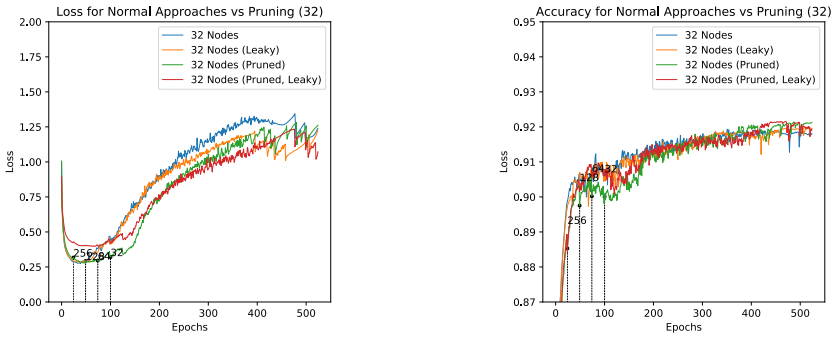

Fig. 4 Fully Connected Layer contains 32 Nodes, arrived at by dynamic pruning or fixed from start

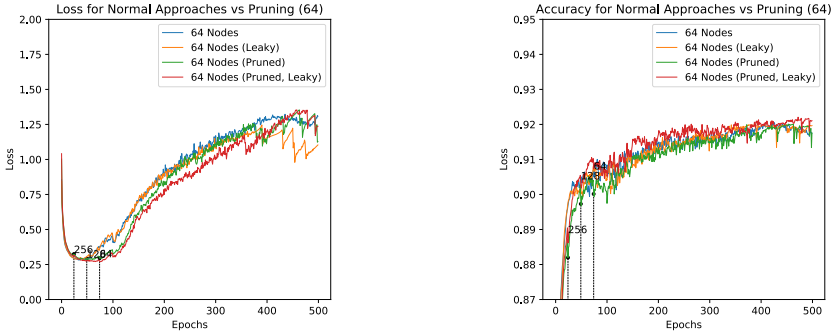

Fig. 5 Fully Connected Layer contains 64 Nodes, arrived at by dynamic pruning or fixed from start

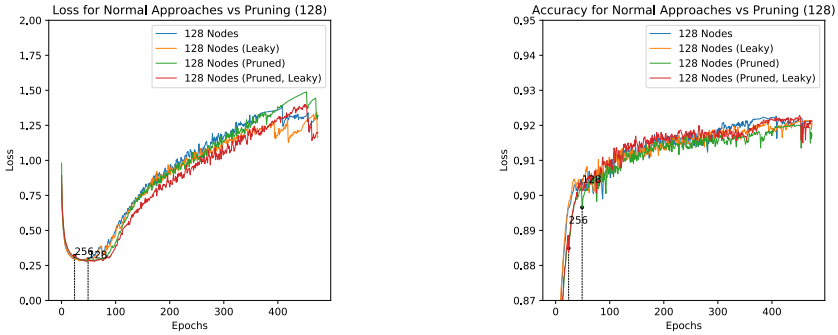

Fig. 6 Fully Connected Layer contains 128 Nodes, arrived at by dynamic pruning or fixed from start

An interesting thing to note from the accuracy curves is that when we use the Leaky Relu activation, except in the initial stages, the accuracy doesn't take much of a hit when a Pruning event occurs as compared to when we use ReLu activation function. Also, the loss curves for Pruned CNN for the most part always lower than the non-Pruned CNN, which suggests this method helps to tackle overfitting and also might perform better on regression tasks.

We present a table showing the highest 10 accuracies reached (form the average of 5 runs) for each of the CNN, we denote Pruned by P, Leaky Relu by L, Dropout ($Bernoulli(p) = 0.5$) by D along with the number of nodes in the Fully Connected Layer. One thing to note from the table is that Adaptive Pruning with Leaky ReLu's outperforms all other training combinations. This is of special interest, since in the loss curve we have noticed that except in the initial stages the training, a Neural Network trained with Leaky ReLu and Adaptive Pruning does not get affected much by the pruning event (although our Neural Network is not big enough to provide a large enough sample set to test this hypothesis).

We now use our method for the second smaller CNN and present the results. Training a smaller CNN for such a small number of epochs leads to a huge variance in each training run's results. So for this method, we select the training which produces the best results and present it here. This results in the effects of smoothing due to pruning significantly visible in the loss function graphs. One can clearly see the graphs of loss for networks that have been pruned are quite smooth near the epochs where pruning occurs.

We present the table for the maximum 10 accuracies reached by the smaller CNN and as we predicted above we see that Adaptive Node Pruning along with the Leaky ReLu activation function clearly outperforms all the other methods. Also, we again see that the loss curve for Adaptive Node Pruning methods is almost always lower than normal methods, which tells us Adaptive Node pruning can be a good way to combat overfitting.

Table 2 Comparison of first 10 accuracies of CNN trained in aforementioned methods

16	0.91784	0.91782	0.9178	0.9177	0.9177	0.91748	0.91746	0.91744	0.91744	0.91742
16 (L)	0.91966	0.919	0.91874	0.9187	0.91862	0.91858	0.91856	0.91846	0.91844	0.91842
16 (P)	0.92016	0.92012	0.92004	0.91996	0.91996	0.91994	0.9199	0.91964	0.9196	0.9196
16 (P, L)	0.92078	0.92074	0.92052	0.9205	0.9205	0.92048	0.92044	0.92044	0.9204	0.92038
32	0.91936	0.91918	0.91916	0.91912	0.91908	0.91904	0.91902	0.91892	0.91886	0.91884
32 (L)	0.91984	0.91972	0.91968	0.91962	0.9196	0.91958	0.91958	0.91956	0.91956	0.91956
32 (P)	0.92142	0.92134	0.92134	0.92128	0.92122	0.92122	0.92116	0.92112	0.92112	0.9211
32 (P, L)	0.92154	0.92152	0.92152	0.9215	0.92148	0.92148	0.92148	0.92146	0.92144	0.92144
64	0.9205	0.92038	0.92018	0.9201	0.9201	0.92006	0.92002	0.92002	0.91984	0.9198
64 (L)	0.91994	0.91992	0.91988	0.91986	0.91986	0.91984	0.91982	0.91982	0.91982	0.9198
64 (P)	0.92002	0.92002	0.91998	0.91998	0.91998	0.91994	0.91994	0.91994	0.91992	0.91986
64 (P, L)	0.92208	0.92176	0.92172	0.9217	0.92162	0.9216	0.92158	0.92152	0.92152	0.92144
128	0.9226	0.92242	0.92238	0.92236	0.9223	, 0.92228	0.92226	0.92224	0.9222	0.9222
128 (L)	0.92138	0.92138	0.92136	0.92132	0.92132	0.92132	0.92132	0.92128	0.92128	0.92126
128 (P)	0.9201	0.92008	0.92006	0.92004	0.92004	0.92004	0.92	0.91998	0.91996	0.91996
128 (P, L)	0.92288	0.92258	0.9224	0.9222	0.92218	0.92216	0.92216	0.92214	0.92212	, 0.9221
512(D)	0.92262	0.92256	0.92254	0.9224	0.92226	0.92214	0.92206	0.92204	0.92204	0.92196

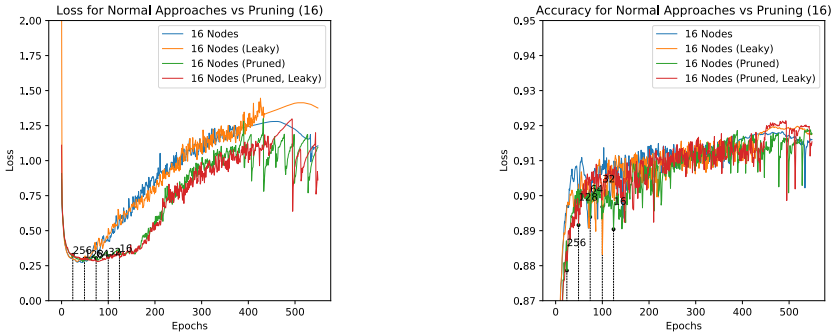

Fig. 7 Fully Connected Layer contains 16 Nodes, arrived at by dynamic pruning or fixed from start

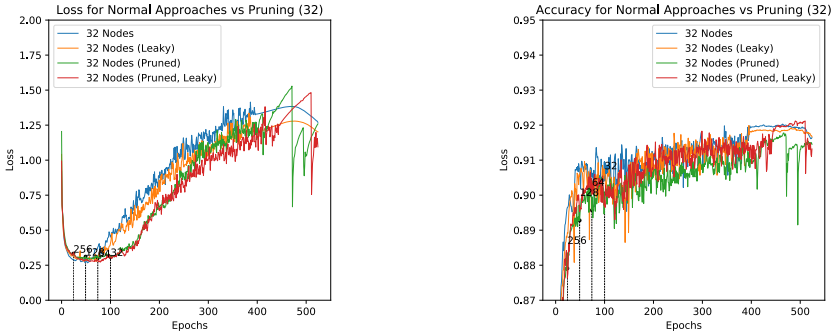

Fig. 8 Fully Connected Layer contains 32 Nodes, arrived at by dynamic pruning or fixed from start

Table 3 Comparison of first 10 accuracies of CNN trained in aforementioned methods

16	0.9183	0.9181	0.9181	0.9181	0.9181	0.918	0.918	0.918	0.9179	0.9179
16 (L)	0.9198	0.9198	0.9197	0.9197	0.9197	0.9196	0.9196	0.9196	0.9196	0.9196
16 (P)	0.919	0.9188	0.9187	0.9186	0.9186	0.9186	0.9186	0.9186	0.9185	0.9185
16 (P, L)	0.9214	0.9213	0.9212	0.9211	0.921	0.921	0.9208	0.9208	0.9208	0.9208
32	0.9203	0.9202	0.9202	0.9201	0.92	0.92	0.92	0.92	0.92	0.92
32 (L)	0.9199	0.9192	0.9191	0.9191	0.919	0.919	0.919	0.9189	0.9189	0.9189
32 (P)	0.9178	0.9177	0.9176	0.9175	0.9172	0.9172	0.917	0.9169	0.9169	0.9169
32 (P, L)	0.9212	0.9212	0.9211	0.9211	0.921	0.921	0.921	0.9209	0.9209	0.9209

8 Discussion and Future Work

Although our experiments are on a smaller scale, few things are very noticeable, like loss curves always being lower than the normal methods, and Leaky Relu activation outperforming other activations. The better loss curves indicate regression tasks might be better for our method. We think our method of pruning is quite easy and straightforward to implement. Unlike most contemporary methods of removing weights to create sparse matrices, we are directly reducing the size of the matrix, which can have direct advantages to computation speed.

We believe that with finer hyperparameter tuning our method can perform even better in terms of both performance and resource consumption (specifically the tradeoff between Pruning Ratio and Pruning Period). We also hypothesize the existence of loss functions, which will reduce the pruning period for the same pruning ratio or increase the pruning ratio for the same pruning period.

The effects of pruning nodes using this method for multilayer Neural Networks need to be investigated, especially the algorithm's adjustments to make the pruning more effective and capture more important features through dropout probabilities. This method can also help uncover the variance due to pruning i.e., whether an earlier layer or later layer causes more degradation in performance due to pruning, and help in uncovering useful insights on the working of Neural Networks and how the importance of various nodes evolve through the training process.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. Ba, J., Frey, B.: Adaptive dropout for training deep neural networks. In: C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K.Q. Weinberger (eds.) Advances in Neural Information Processing Systems 26, pp. 3084–3092. Curran Associates, Inc. (2013). URL http://papers.nips.cc/paper/5032-adaptive-dropout-for-training-deep-neural-networks.pdf
2. Bald, P.: Autoencoders, unsupervised learning, and deep architectures. JMLR: Workshop and Conference Proceedings 27:3750 (2012)
3. Cheng, Y., Wang, D., Zhou, P., Zhang, T.: A survey of model compression and acceleration for deep neural networks. arXiv preprint arXiv:1710.09282 (2016)

4. Dutta, A.: Pruning neural networks with adaptive dropout (github.com/duttaabhigyan/pruning-with-adaptive-dropout). Github Repository (2020)
5. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635 (2019)
6. Google: Google colaboratory (2018)
7. Hagiwara, M.: A simple and effective method for removal of hidden units and weights. Neurocomputing 6 (1994) 207-218 (1994)
8. He, T., Fan, Y., Qian, Y., Tan, T., Yu, K.: Reshaping deep neural network for fast decoding by node-pruning. IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP) (2014)
9. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Y. Bengio, Y. LeCun (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)
10. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems 25 (2012)
11. Li, F.F., Johnson, J., Yeung, S.: Cs231n: Convolutional neural networks for visual recognition. Stanford University (2018)
12. Liu, Z., Li, J., Shen, Z.: Learning efficient convolutional networks through network slimming. 2017 IEEE International Conference on Computer Vision (ICCV) (2017)
13. Malach, E., Yehudai, G., Shalev-Shwartz, S., Shamir, O.: Proving the lottery ticket hypothesis: Pruning is all you need. arXiv preprint arXiv: 2002.00585 (2020)
14. Min, J.T.C., Motani, M.: Dropnet: Reducing neural network complexity via iterative pruning. Proceedings of the 37 th International Conference on Machine Learning (2020)
15. Murphy, K.: Cs340 (machine learning) fall 2007. University of British Columbia (2007)
16. Paszke, A., Gross, S., Chintala, S., et al: Automatic differentiation in pytorch. 31st Conference on Neural Information Processing Systems (2017)
17. Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., Rastegari, M.: Whats hidden in a randomly weighted neural network. arXiv preprint arXiv: 1911.13299 (2020)
18. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15**(56), 1929–1958 (2014). URL http://jmlr.org/papers/v15/srivastava14a.html
19. Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in nlp. 31st Conference on Neural Information Processing Systems (2017)
20. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. CoRR **abs/1708.07747** (2017)
21. Zhang, Z., Qiao, J.: A node pruning algorithm for feedforward neural network based on neural complexity. International Conference on Intelligent Control and Information Processing (2010)

9 Appendix A

We have performed our experiments on Google Colaboratory [6] using GPU's, using the PyTorch Deep Learning Library [16].

We exclusively use Fashion MNIST dataset to perform our experiments [20]. We have performed all our experiments on Google Colaboratory using GPU's and using the PyTorch Deep Learning Library. The code can be found here [4].

For the optimization of Loss function we use the Backpropagation Algorithm with Gradient Descent (variants), more specifically we use SGD optimizer with momentum = 0.9 and Adam Optimizer [9] with default PyTorch hyperparameters ($lr = 0.001, \beta's = (0.9, 0.999), eps = 1e-08, weight_{decay} = 0, amsgrad = False$). The idea is that SGD with momentum is used to arrive at a prospective solution, while the Adam Optimizer is used in the later stages for finer tuning.

For pruning nodes we follow the following scheme, we denote a specific number of epochs as pruning period, and after each pruning period we prune the number of remaining nodes by the pruning ratio. In this way we go on until, we reach the required number of nodes after which we stop pruning. In general, once we reach the required number of nodes, we use the SGD with momentum optimizer for another pruning period and then switch to the Adam Optimizer, but the number of epochs remain the same when we compare our approach to the conventional approach of training from the beginning (i.e. only the epoch at which the optimizer switch take place depends on the final number of nodes, but the total number of epochs remain same).

The training process for the Adaptively Pruned Neural Network follows a very simple scheme: We train the CNN with SGD Optimizer with momentum till it reaches the fixed number of nodes we want to prune up to, then we run another Pruning Period number of epochs for the loss to stabilize, followed by 400 epochs of optimization with Adam Optimizer for fine-tuning. For the normal CNN with fixed number of nodes form the start, we run the same number of total epochs, but now for only 100 epochs the optimization is done with SGD Optimizer with momentum, followed by the rest of the epochs being trained by the Adam Optimizer. For example if we want to prune till 32 nodes, we require 100 epochs, till we reach 32 nodes form 512 nodes (ratio= 0.5, period= 25). Thus for $100+25$ epochs we will use SGD Optimizer with momentum, followed by 400 epochs of Adam Optimizer. For the Vanilla CNN with 32 nodes from the beginning we use SGD Optimizer with momentum for first 100 epochs, followed by 425 epochs of optimization with Adam Optimizer. All the results are averaged over 5 runs with random initialization of the CNN weights (as per scheme followed by PyTorch).