A SIMPLE YET EFFECTIVE METHOD TO PRUNE DENSE LAYERS OF NEURAL NETWORKS

Mohammad Babaeizadeh, Paris Smaragdis & Roy H. Campbell
Department of Computer Science
University of Illinois at Urbana-Champaign
{mb2,paris,rhc}@illinois.edu.edu

ABSTRACT

Neural networks are usually over-parameterized with significant redundancy in the number of required neurons which results in unnecessary computation and memory usage at inference time. One common approach to address this issue is to prune these big networks by removing extra neurons and parameters while maintaining the accuracy. In this paper, we propose NoiseOut, a fully automated pruning algorithm based on the correlation between activations of neurons in the hidden layers. We prove that adding additional output neurons with entirely random targets results into a higher correlation between neurons which makes pruning by NoiseOut even more efficient. Finally, we test our method on various networks and datasets. These experiments exhibit high pruning rates while maintaining the accuracy of the original network.

1 INTRODUCTION

Neural networks and deep learning recently achieved state-of-the-art solutions to many problems in computer vision (Krizhevsky et al. (2012); He et al. (2015)), speech recognition (Graves et al. (2013)), natural language processing (Mikolov et al. (2013)) and reinforcement learning (Silver et al. (2016)). Using large and oversized networks in these tasks is a common practice. Such oversized networks can easily overfit on the training dataset while having poor generalization on the testing data (Sabo & Yu (2008)). A rule of thumb for obtaining useful generalization is to use the smallest number of parameters that can fit the training data (Reed (1993)). Unfortunately, this optimal size is not usually obvious and therefore the size of the neural networks is determined by a few rules-of-thumb (Heaton (2008)) which do not guarantee an optimal size for a given problem. One common approach to overcome overfitting is to choose an over-sized network and then apply regularization (Ng (2004)) and Dropout (Srivastava et al. (2014)). However, these techniques do not reduce the number of parameters and therefore do not resolve the high demand of resources at test time.

Another method is to start with an oversized network and then use pruning algorithms to remove redundant parameters while maintaining the network's accuracy (Augasta & Kathirvalavakumar (2013)). These methods need to estimate the upper-bound size of a network, a task for which there are adequate estimation methods (Xing & Hu (2009)). If the size of a neural network is bigger than what is necessary, in theory, it should be possible to remove some of the extra neurons without affecting its accuracy. To achieve this goal, the pruning algorithm should find neurons which once removed result into no additional prediction errors. However, this may not be as easy as it sounds since all the neurons contribute to the final prediction and removing them usually leads to error.

It is easy to demonstrate this problem by fitting an oversized network on a toy dataset. Figure 1 shows a two-dimensional toy dataset which contains two linearly separable classes. Hence, only one hidden neuron in a two-layer perceptron should be enough to classify this data and any network with more than one neuron (such as the network in Figure 2-a) is an oversized network and can be pruned. However, there is no guarantee that removing one of the hidden neurons will maintain the network's performance. As shown in the example in Figure 1, removing any of the hidden neurons results into a more compact, but under-performing network. Therefore, a more complicated process is required for pruning neural networks without accuracy loss.

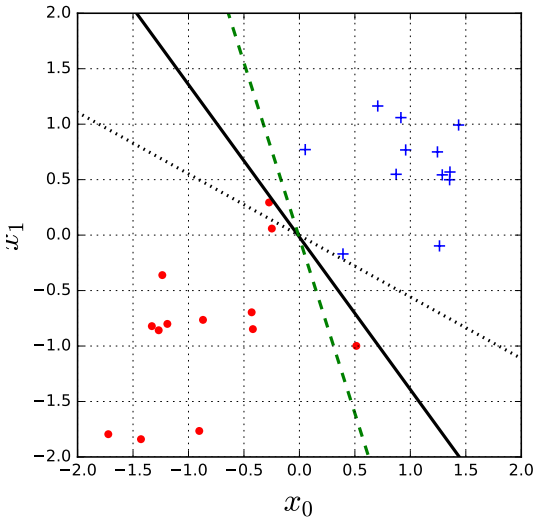

Figure 1: Effect of pruning on accuracy. Bold line represents discriminator learned by a 2-2-1 MLP (Figure 2-a) on a toy data set. Dash line and dotted line show the results after pruning one of the hidden neurons. As it can be seen removing a hidden neuron will result into accuracy drop.

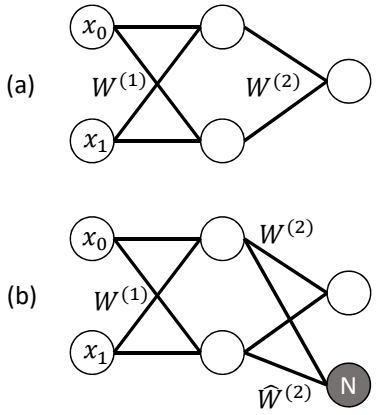

Figure 2: *a)* a simple 2-2-1 MLP; *b)* the same network with one additional noise output. All the hidden units have a linear activation while the output neurons use sigmoid as activation function. The gray neuron is a *noise output* which changes its target in each iteration.

Our goal in this paper is two-fold. First, we introduce NoiseOut, a pruning method based on the correlation between activations of the neurons. Second, we propose an approach which enforces the higher correlation between activations of neurons. Since the effectiveness of NoiseOut hinges on high correlations between neuron activations, the combination of these two methods facilitates more aggressive pruning.

2 Related Work

Optimal Brain Damage (LeCun et al. (1989)) and Optimal Brain Surgeon (Hassibi & Stork (1993)) prune networks based on the Hessian of the loss function. It is shown that such pruning is more effective and more accurate than earlier magnitude-based pruning such as weight decay (Hanson & Pratt (1989)). However, the necessary second-order derivatives require additional computational resources.

Recently, replacing the fully connected layers with other types of layers has been utilized to reduced the number of parameters in a neural network. Deep fried convnets (Yang et al. (2015)) replaces these layers with kernel methods while the GoogLenet (Szegedy et al. (2015)) and Network in Network architecture (Lin et al. (2013)) replace them with global average pooling. Alternatively, Han et al. (2015) proposed a pruning method which learns the import connections between neurons, pruning the unimportant connections, and then retraining the remaining sparse network.

Besides pruning, other approaches have been proposed to reduce the computation and memory requirements of neural networks. HashNets(Chen et al. (2015)) reduce the storage requirement of neural networks by randomly grouping weights into hash buckets. These techniques can be combined with pruning algorithms to achieve even better performance. As an example, Deep Compression(Han et al. (2016)) proposed a three stage pipeline: pruning, trained quantization, and Huffman coding, to reduce the storage requirement of neural networks.

3 Proposed Method

In this section, we describe the details of the proposed method called NoiseOut. First, we show how this method can prune a single neuron and then how it can prune a full network, one neuron at a time.

3.1 PRUNING A SINGLE NEURON

The key idea in NoiseOut is to remove one of the two neurons with strongly correlated activations. The main rationale behind this pruning is to keep the signals inside the network as close to the original network as possible. To demonstrate this, assume there exists u, v, l such that $|\rho(h_u^{(l)}, h_v^{(l)})| = 1 \Rightarrow h_u^{(l)} = \alpha h_v^{(l)} + \beta$, where $h_i^{(l)}$ is the activation of i^{th} neuron in the l^{th} layer. By definition:

$$\begin{aligned} h_k^{(l+1)} &:= \sum_i h_i^{(l)} w_{i,k}^{(l)} + b_k^{(l+1)} = \sum_{i \neq u,v} h_i^{(l)} w_{i,k}^{(l)} + h_u^{(l)} w_{u,k}^{(l)} + h_v^{(l)} w_{v,k}^{(l)} + b_k^{(l+1)} \\ &= \sum_{i \neq u,v} h_i^{(l)} w_{i,k}^{(l)} + \alpha h_v^{(l)} w_{u,k}^{(l)} + \beta w_{u,k}^{(l)} + h_v^{(l)} w_{v,k}^{(l)} + b_k^{(l+1)} \\ &= \sum_{i \neq u,v} h_i^{(l)} w_{i,k}^{(l)} + h_v^{(l)} \big(\alpha w_{u,k}^{(l)} + w_{v,k}\big) + \big(\beta w_{u,k}^{(l)} + b_k^{(l+1)}\big) \end{aligned} \tag{1}$$

This means that neuron u can be removed without affecting any of the neurons in the next layer, simply by adjusting the weights of v and neurons' biases. Note that $\max_{i,j,i \neq j} \big|\rho(h_i^{(l)}, h_j^{(l)})\big| = 1$ is an ideal case. In this ideal scenario, removing one of the neurons results into no change in accuracy since the final output of the network will stay the same. In non-ideal cases, when the highest correlated neurons are not strongly correlated, merging them into one neuron may alter the accuracy. However, continuing the training after the merge may compensate for this loss. If this does not happen, it means that the removed neuron was necessary to achieve the target accuracy and the algorithm cannot compress the network any further without accuracy degradation.

NoiseOut follows the same logic to prune a single neuron using the following steps:

1. For each i, j, l calculate $\rho\big(h_i^{(l)}, h_j^{(l)}\big)$
2. Find $u, v, l = \underset{u,v,l,u \neq v}{\arg\max} \big|\rho\big(h_u^{(l)}, h_v^{(l)}\big)\big|$
3. Calculate $\alpha, \beta := \underset{\alpha,\beta}{\arg\min} \big(h_u^{(l)} - \alpha h_v^{(l)} - \beta\big)$
4. Remove neuron u in layer l
5. For each neuron k in layer $l+1$:
   - Update the weight $w_{v,k}^{(l)} = w_{v,k}^{(l)} + \alpha w_{u,k}^{(l)}$
   - Update the bias $b_k^{(l+1)} = b_k^{(l+1)} + \beta w_u^{(l)}$

3.2 ENCOURAGING CORRELATION BETWEEN NEURONS

The key element for successful pruning of neural networks using NoiseOut is the strong correlation between activation of the neurons. Essentially, a higher correlation between these activations means more efficient pruning. However, there is no guarantee that back-propagation results in correlated activations in a hidden layer. In this section, we propose a method to encourage higher correlation by adding additional output nodes, called *noise outputs*. The targets for noise outputs will randomly change in each iteration based on a predefined random distribution. We show that adding noise outputs to the output layer intensifies the correlation between activation in the hidden layers which will subsequently make the pruning task more effective.

3.2.1 EFFECT OF ADDING NOISE OUTPUTS TO THE NETWORK

To demonstrate the effect of adding noise outputs to the network, let us reconsider the toy example described previously in Figure 1, this time with some additional noise outputs in the output layer as shown in Figure 2-b. The result of training this network has been shown in Figure 3. As seen in this figure, the activation of the two hidden neurons has become highly correlated, and each hidden unit converged to the optimal discriminant by itself. This means that either of the extra neurons in the hidden layer can be removed without loss of accuracy.

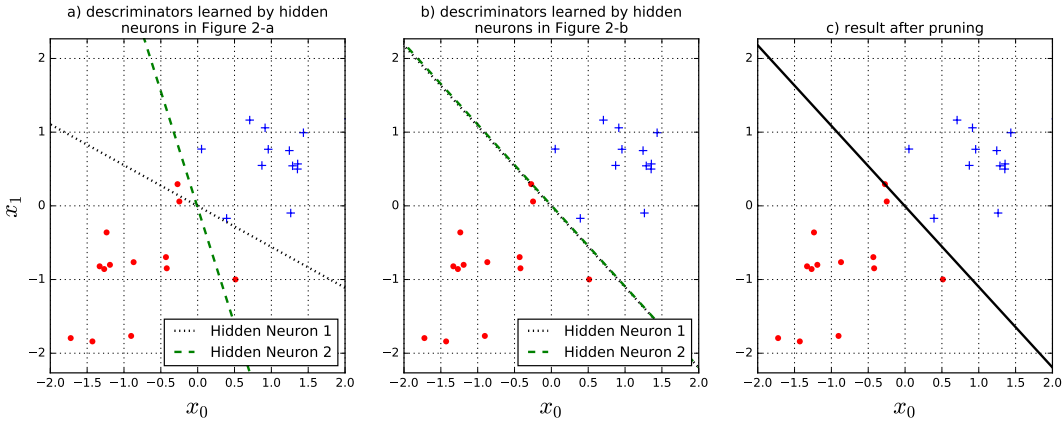

Figure 3: Comparison between discriminators learned with and without noise neurons. As it can be seen with noise neurons the activation of hidden neurons is more correlated; *a)* discriminators defined by each hidden neuron in Figure 2-a; *b)* discriminators defined by each hidden neuron in Figure 2-b; *c)* final discriminator after pruning one hidden neuron in Figure 2-b.

This claim can be proved formally as well. The key to this proof is that neural networks are **deterministic** at inference time. In other words, the network generates a constant output for a constant input. For noise output, since the target is independent of the input, the training algorithm finds *an optimal constant value* that minimizes the expected error for *every input*. This objective can be presented as:

$$\min_{W} C\Big(f(X;W), \big[Y, \widehat{Y}\big]\Big) = \min_{W} \Big(C\big(f_m(X;W), Y\big) + \sum_{i=0}^{m} C\big(\hat{f}_i(X;W), \widehat{Y}_i\big)\Big) \tag{2}$$

where W is the adjustable parameters (i.e. weights), X is the input, Y is the target, $\widehat{Y}_i$ is the target of noisy outputs at iteration i, C is the cost function, m is the number of iterations. $f_i(X;W)$ and $\hat{f}_i(X;W)$ are the outputs of the neural network in the original and noise outputs respectively, at iteration i. Note that $\widehat{Y}_i$ changes in each iteration according to a random distribution $P_{\hat{Y}}$.

The first part of Equation 2 represents the common objective of training a neural network, while the second part has been added because of the noise outputs. It is possible to adjust the effect of noise outputs based on $P_{\hat{Y}}$. For instance by adding more noise outputs (in the case of Binomial distribution) or adjusting the variance (for Gaussian distribution). Another way of this adjustment is introducing a new multiplier for the second part of the cost. Although $\widehat{Y}_i$ changes in each iteration, the constant value that the network infers for any given input would be the same e.g. θ, due to the independent of $P_{\hat{Y}}$ from X. Therefore:

$$\hat{f}_m(X;W) = J_{1\times n}\theta$$

where $J_{1\times n}$ is the matrix of ones of size $1 \times n$ (n is the number of samples in the dataset). The actual value of θ can be estimated for any given network architecture. As an example, let $W^{(1)}$, $W^{(2)}$ and $\widehat{W}^{(2)}$ be the weights of the first hidden layer, the output layer and the noisy neurons in a 2-2-1 MLP network respectively (Figure 2-b). Assuming linear activation in the hidden neurons and mean square error (MSE) as the cost function, Equation 2 can be rewritten as:

$$\begin{aligned}
\min_{W} C &= \min_{W} C\Big(f(X;W), \big[Y, \widehat{Y}_i\big]\Big) \\
&= \min_{W^{(1)},W^{(2)}} \frac{1}{2}\big(f(X;W) - Y\big)^2 + \min_{W^{(1)},\widehat{W}^{(2)}} \sum_{i=0}^{m} \frac{1}{2}\big(\hat{f}_i(X;W) - \widehat{Y}_i\big)^2 \\
&= \min_{W^{(1)},W^{(2)}} \big(f(X;W) - Y\big)^2 + \min_{W^{(1)},\widehat{W}^{(2)}} \frac{1}{2}\big(\hat{f}(X;W) - \theta\big)^2
\end{aligned} \tag{3}$$

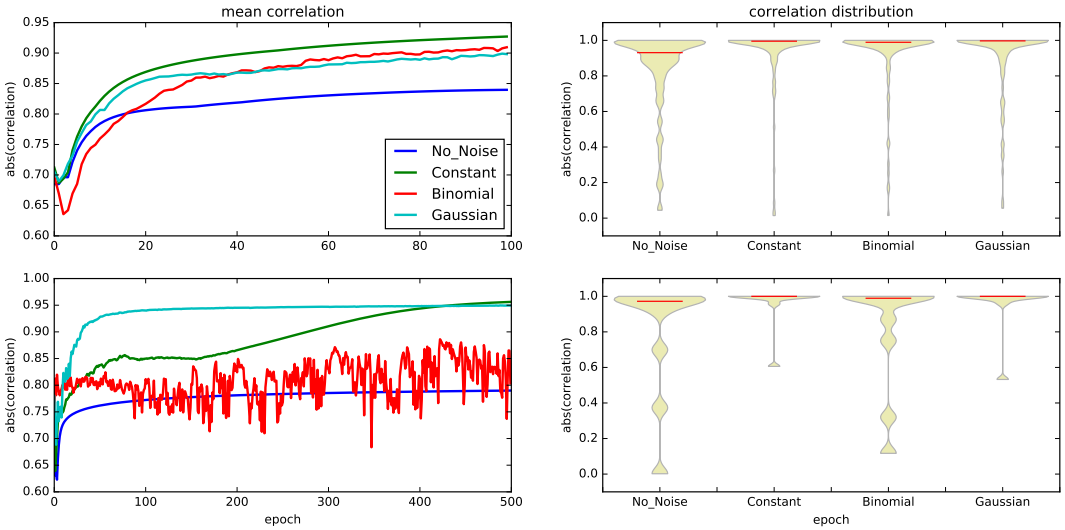

Figure 4: Effect of noise outputs to the correlation of neurons activation in the hidden layers. The top row shows the correlation of two hidden neurons in the network of Figure 2 while the bottom row is the correlation between the two neurons on the first hidden layer of a 6 layer MLP (2-2-2-2-2-2-1). The left column represents the mean correlation *during* training of the network for 100 times. In right column, yellow is the distribution of correlations at the end of the training and small red line shows the median. As it can be seen in these graphs, adding noise outputs improves the correlation between neurons in the hidden layer.

In this particular case, θ can be calculated using derivatives of the cost functions in Equation 3:

$$\begin{aligned}\frac{\partial C}{\partial \theta} &= \frac{\partial}{\partial \theta}\Big[\big(f(X;W)-Y\big)^2 + \frac{1}{2}\big(\widehat{f}(X;W)-\theta\big)^2\Big] \\ &= \frac{\partial}{\partial \theta}\Big[\frac{1}{2}\big(\widehat{f}(X;W)-\theta\big)^2\Big] = \big(\theta - \widehat{f}(X;W)\big) = 0 \\ &\Rightarrow \theta = E\big[\widehat{f}(X;W)\big] = E\big[P_{\hat{Y}}\big] \end{aligned} \tag{4}$$

This means that in this particular network with MSE as the cost function, the final error will be minimized when the network outputs the expected value of targets in noise outputs $\big(E\big[P_{\hat{Y}}\big]\big)$ for any given input.

To demonstrate how outputting a constant value affects the weights of a network, let's consider the network in Figure 2-b. In this case, the output of noisy output will be:

$$\begin{aligned}\widehat{f}(X;W) = XW^{(1)}\widehat{W}^{(2)} &= h_1^{(1)}w_{1,1}^{(2)} + h_2^{(1)}w_{2,1}^{(2)} = \theta \\ &\Rightarrow h_1^{(1)} = \frac{1}{w_{1,1}^{(2)}}(\theta - h_2^{(1)}w_{2,1}) \\ &\Rightarrow \rho_{h_1^{(1)},h_2^{(1)}} = \operatorname{sgn}\left(\frac{-w_{2,1}^{(2)}}{w_{1,1}^{(2)}}\right) \Rightarrow \big|\rho_{h_1^{(1)},h_2^{(1)}}\big| = 1 \end{aligned} \tag{5}$$

Equation 6 means that the activation of the hidden neurons has a correlation of 1 or -1. For more than two neurons it can be shown that the output of one neuron will be a linear combination of other neurons which means the claim still holds.

The same results can be achieved empirically. Since the output of the noise outputs will converge to $E\big[P_{\hat{Y}}\big]$, it seems that there may not be any difference between different random distributions with the same expected value. Therefore, we tested different random distributions for $E\big[P_{\hat{Y}}\big]$ with the same $E\big[P_{Y_N}\big]$, on the network shown in Figure 2-b. These noise distributions are as follows:

Algorithm 1 NoiseOut for pruning hidden layers in neural networks

procedure TRAIN(X, Y) ▷ X is input, Y is expected output
  $W \leftarrow initialize\_weights()$
  for each iteration **do**
    $Y_N \leftarrow generate\_random\_noise()$ ▷ generate random expected values
    $Y' \leftarrow concatenate(Y, Y_N)$
    $W \leftarrow back\_prop(X, Y')$
    while $cost(W) \leq threshold$ **do**
      $A, B \leftarrow find\_most\_correlated\_neurons(W, X)$
      $\alpha, \beta \leftarrow estimate\_parameters(W, X, A, B)$
      $W' \leftarrow remove\_neuron(W, A)$
      $W' \leftarrow adjust\_weights(W', B, \alpha, \beta)$
      $W \leftarrow W'$
  return W

- **Gaussian** $P_{\hat{Y}}(x) = \mathcal{N}(0.1, 0.4)$ Normal distribution with mean of 0.1 and standard deviation of 0.4. This noise distribution is appropriate for regression tasks with MSE cost.
- **Binomial** $P_{\hat{Y}}(x) = \mathrm{B}(1, 0.1)$ Binomial distribution with 1 trial and success probability of 0.1. We chose binomial distribution since it generates random classification labels and is appropriate for networks that have to produce binary labels.
- **Constant** $P_{\hat{Y}}(x) = \delta(x - 0.1)$ In this case, the target of the noise outputs is the constant value of 0.1. This is used as an expected-value "shortcut" so that we can examine a stochastic vs. a deterministic approach.
- **No_Noise** no noise output for comparison.

As it can be seen in the top row Figure 4, in a regular network with no noise unit (shown as `No_Noise`), the correlation between the output of hidden neurons $\left|\rho_{h_1^{(1)}, h_2^{(1)}}\right|$ does not go higher than 0.8, while in the existence of a noise output this value approaches to one, rather quickly. This means that the two hidden neuron are outputting just a different scale of the same value for any given input. In this case, NoiseOut easily prunes one of the two neurons.

The same technique can be applied to correlate the activation of the hidden neurons in networks with more than two layers. The bottom row of Figure 4 shows the correlation between the activation of two hidden neurons in the first layer of a six layer MLP (2-2-2-2-2-2-1). As it can be seen in this figure, adding noise outputs helped the neurons to achieve higher correlation compared to a network with no noise output. Binomial noise acts chaotic at the beginning due to its sudden change of expected values in the noise outputs while Gaussian noise improved the correlation the best in these experiments.

3.3 PRUNING A FULLY CONNECTED NETWORK

Algorithm 1 shows the final NoiseOut algorithm. For the sake of readability, this algorithm has been shown for networks with only one hidden layer. But the same algorithm can be applied to networks with more that one hidden layer by performing the same pruning on all the hidden layers independently. It can also be applied to convolutional neural networks that use dense layers, in which we often see over 90% of the network parameters (Cheng et al. (2015)).

This algorithm is simply repeating the process of removing a single neuron, as described in the previous section. The pruning ends when the accuracy of the network drops below some given threshold. Note that the pruning process is happening while training.

4 EXPERIMENTS

To illustrate the generality of our method we test it on a core set of common network architectures, including fully connected networks and convolutional neural networks with dense layers. In all of these experiments, the only stop criteria is the accuracy decay of the model. We set the threshold

Table 1: Results of pruning Lenet-300-100 on MNIST. The accuracy of all the compressed networks are the same as the original network.

Method	Noise Outpus	Layer 1 Neurons	Layer 2 Neurons	Parameters	Removed Parameters	Compression Rate
Ground Truth	-	300	100	266610	-	-
No_Noise	-	23	14	15989	94.00%	16.67
Gaussian	512	20	9	15927	94.02%	16.73
Constant	512	20	7	15105	94.33%	17.65
Binomial	512	19	6	11225	95.78%	23.75
No_Noise	-	13	12	10503	96.06%	20.89
Gaussian	1024	16	7	12759	95.21%	18.58
Constant	1024	18	7	14343	94.62%	17.61
Binomial	1024	19	7	15135	94.32%	25.38

Table 2: Pruning Lenet-5 on MNIST. The accuracy of all the compressed networks are the same as the original network.

Method	Noise Neurons	Dense Layer Neurons	Parameters	Removed Parameters	Compression Rate
Ground Truth	-	512	605546	-	-
No_Noise	-	313	374109	38.21%	1.61
Gaussian	512	3	13579	97.75%	44.59
Constant	512	33	48469	91.99%	12.49
Binomial	512	26	40328	93.34%	15.01

for this criteria to match the original accuracy; therefore all the compressed network have the same accuracy as the original network. For each experiment, different random distributions have been used for $P_{\hat{Y}}$, to demonstrate the difference in practice.

4.1 Pruning fully connected and convolutional networks

Table 1 and Table 2 show the result of pruning Lenet-300-100 and Lenet-5 (LeCun et al. (1998)) on MNIST dataset. Lenet-300-100 is a fully connected network with two hidden layers, with 300 and 100 neurons each, while Lenet-5 is a convolutional network with two convolutional layers and one dense layer. These networks achieve 3.05% and 0.95% error rate on MNIST respectively (LeCun et al. (1998)). Note that in Lenet-5 over 98% of parameters are in the dense layer and pruning them can decrease the model size significantly.

As it can be seen in these tables, NoiseOut removed over %95 of parameters with no accuracy degradation. Astonishingly, pruned Lenet-5 can achieve 0.95% error rate with only 3 neurons in the hidden layer which reduce the total number of weights in Lenet-5 by a factor of 44. Figure 6 demonstrates the output of these 3 neurons. This graph has been generated by outputting the activation of hidden layer neurons for 1000 examples randomly selected from MNIST dataset. Then, the data has been sorted by the target class. As it can be seen in this figure, the three neurons in the hidden layer efficiently encoded the output of convolutional layers to the expected ten classes. Obviously, these values can be utilized by the softmax layer to perform the final classification.

To test the effect of pruning of deeper architectures, we prune the network described in Table 4 on SVHN data set.This model which has over 1 million parameters, achieves 93.39% and 93.84% accuracy on training set and test set respectively. As it can be seen in Table 3, NoiseOut pruned more than 85% of the parameters from the base model while maintaining the accuracy.

Table 3: Pruning the reference network in Table 4, on SVHN dataset.

Method	Dense Layer Neurons	Parameters	Removed Parameters
Ground Truth	1024	1236250	-
No_Noise	132	313030	74.67%
Gaussian	4	180550	85.39%
Constant	25	202285	83.63%
Bionomial	17	194005	84.30%

Table 4: Base model architecture for SVHN with 1236250 parameters.

Layer	Filters	Kernel	Weights
conv1	32	3×3	896
conv2	32	3×3	9246
conv3	32	3×3	9246
pool1	-	2×2	-
conv4	48	3×3	13872
conv5	48	3×3	20784
conv6	48	3×3	20784
pool2	-	2×2	-
conv7	64	3×3	27712
conv8	64	3×3	36928
conv9	64	3×3	36928
pool3	-	2×2	-
dense	1024	-	1049600
softmax	10	-	10250

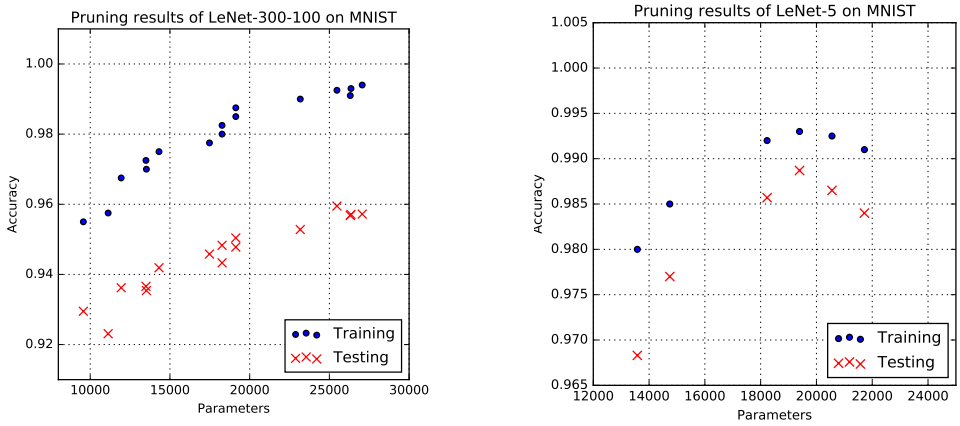

Figure 5: Pruning Lenet-300-100 and Lenet-5 on MNIST data set with various accuracy thresholds. x axis represents the total number of parameters in the pruned network (including weights in the convolutional layers), while y axis shows the accuracy of the model on test and training dataset.

4.2 Effect of NoiseOut on test accuracy

To explore the effect of NoiseOut on the test accuracy, we pruned Lenet-300-100 and Lenet-5 on MNIST with multiple accuracy thresholds, using Gaussian distribution as the target for noise outputs. In each one of these experiments, we measured both the training and test accuracy. As expected, the results which have been shown in Figure 5, indicate that lower accuracy thresholds result into more pruned parameters. However, the gap between training and testing threshold stays the same. This shows that pruning the network using NoiseOut does not lead to overfitting.

4.3 Relation to Dropout and regularization

The key point in successful pruning with NoiseOut is a higher correlation between neurons. This goal might seem to be in contradiction with techniques designed to avoid overfitting such as Dropout and Regularization. To investigate this, we pruned Lenet-5 in the presence and absence of these features and demonstrated the results in Figure 7. As it can be seen in this figure, Dropout helps the pruning process significantly, while L2-regularizations causes more variance. It seems that preventing co-adaptation of neurons using Dropout also intensifies the correlation between them, which helps NoiseOut to remove even more redundant neurons without accuracy loss.

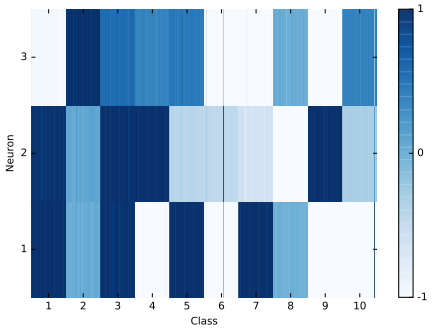

Figure 6: Activation of neurons in a pruned Lenet-5 with only 3 neurons left in the dense layer. The x-axis has been populated by 100 random samples from each class of MNIST, sorted by class. y-axis shows the neuron ID. Note that $tanh$ is the activation function in the hidden layer.

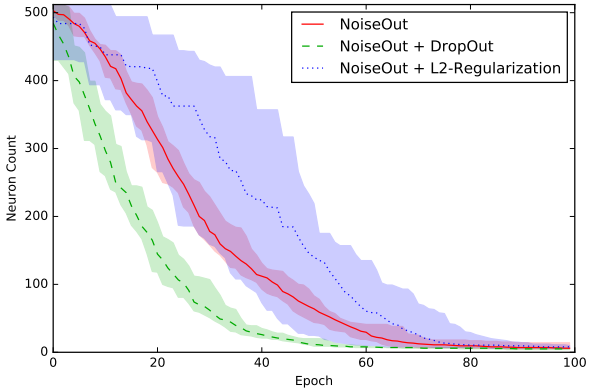

Figure 7: Effect of Dropout and L2-regularization on NoiseOut. The y-axis represents the number of remaining neurons in the dense layer. Note that more than one neuron can be removed in each epoch. In each curve, the bold line is the median of 10 runs and colored background demonstrates the standard deviation.

5 Conclusion

In this paper, we have presented NoiseOut, a simple but effective pruning method to reduce the number of parameters in the dense layers of neural networks by removing neurons with correlated activation during training. We showed how adding noise outputs to the network could increase the correlation between neurons in the hidden layer and hence result to more efficient pruning. The experimental results on different networks and various datasets validate this approach, achieving significant compression rates without loss of accuracy.

References

M Gethsiyal Augasta and T Kathirvalavakumar. Pruning algorithms of neural networks—a comparative study. *Central European Journal of Computer Science*, 3(3):105–115, 2013.

Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. *arXiv preprint arXiv:1504.04788*, 2015.

Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2857–2865, 2015.

Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 6645–6649. IEEE, 2013.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pp. 1135–1143, 2015.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2016.

Stephen José Hanson and Lorien Y Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in neural information processing systems*, pp. 177–185, 1989.

Babak Hassibi and David G Stork. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Jeff Heaton. *Introduction to neural networks with Java*. Heaton Research, Inc., 2008.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Yann LeCun, John S Denker, Sara A Solla, Richard E Howard, and Lawrence D Jackel. Optimal brain damage. In *NIPs*, volume 89, 1989.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 78. ACM, 2004.

Russell Reed. Pruning algorithms-a survey. *Neural Networks, IEEE Transactions on*, 4(5):740–747, 1993.

Devin Sabo and Xiao-Hua Yu. A new pruning algorithm for neural network dimension analysis. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pp. 3313–3318. IEEE, 2008.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

Hong-Jie Xing and Bao-Gang Hu. Two-phase construction of multilayer perceptrons using information theory. *Neural Networks, IEEE Transactions on*, 20(4):715–721, 2009.

Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1476–1483, 2015.