Reducing oversized neural networks using sensitivity

Piyush Amritlal Gupta
College of Engineering and Computer Science, Australian National University
u6476509@anu.edu.au

Abstract. In this paper, we will develop an over-sized single layer neural network that would solve a classification problem, and apply a pruning method called sensitivity to reduce the size of the model. Extending the similar approach, we would be adding more layers to the network to make it a deep learning model and apply the same pruning methods on each layer. Although the pruning can reduce the accuracy of the network, retraining the network could increase the accuracy drastically and provide similar results as its original over-sized network.

Keywords: Neural Networks, Image Segmentation, Classification, Network Reduction, Sensitivity.

1 Introduction

Feed-forward neural networks with error back propagation are used to solve a variety of classification problems. But one of the biggest challenges while designing these types of neural networks is that the size of the network cannot be calculated accurately. The number of neurons for input layer can be guessed from the number and type of inputs, and similarly the number of neurons in the output layer is calculated from the number of classes that the network would divide into [2]. But the real problem arises when selecting the number of neurons for the hidden layer.

Generally, the number of neurons for the hidden layer is guessed and randomly taken. If the network is not learning, then we can just increase the number of neurons in the hidden layer, ranging from single to up to 10% of original neurons and even doubled if necessary [2].

But most of the times the number of neurons for the hidden layer are taken higher than required, because there is a possibility that the underestimated network may be too small and never learn the problem at all [4]. On the other hand, an oversized network may provide faster learning rate [4]. Then once the network has been trained one by one the neuron is removed while at the same time looking after that the accuracy of network does not fall.

This paper works on creating two oversized neural networks, one being a simple two-layer feedforward neural network, and another being a multi-layered artificial neural network based on the former model, which would be referred as deep neural network from here on. One of the network reduction algorithms will be applied on both the networks, and analyse on how well the algorithm works on different models.

2 Data Set

The data set for this paper was taken from UCI repository [8]. The data is referred as 'Image Segmentation Dataset', which was made by Vision Group at University of Massachusetts in 1990. All the instances were drawn randomly from a database of 7 outdoor images [8]. The images were hand segmented for classification of each pixel and each instance is a 3x3 region [8].

It's a multivariate dataset used for classification problem. It consists of 2310 instances, which have already been divided into 210 training set, 2100 testing dataset and 19 continuous attributes. The information of each attribute can be given as follows as per the UCI website [8].

The data will help in identifying the pattern in one of the seven classes – brickface, sky, foliage, cement, window, path or grass.

The dataset was chosen because high number of instances and attributes. It has 19 continuous attributes and no missing values. It has high number of web hits on UCI repository [8] and has been referred by many websites. No pre-processing on inputs were necessary. All the data are continuous and easily usable in the neural network.

Although 210 instances of the training set are sufficient for learning of two-layer neural network, it is far too less for a deep-learning network. So we interchanged the datasets for our multi-layer model, that is the 2100 instances that were originally meant for testing was used for training and the other 210 instances of training set were used for testing.

3 Two-layer Neural Network

3.1 Proposed Model

The two-layer neural network designed to process this dataset has 19 neurons in the input layer based on the number of attributes, 50 neurons in its single hidden layer which were decided randomly based upon the size of the dataset and 7 neurons in output layer based upon its classes. The output will be one of the seven classes that the network will segment the pixel into. Standard back-propagation was applied to the system with sigmoid activation function.

Cross entropy loss method to calculate the loss. Since it is a classification model, the cross entropy would be the most efficient and simple method to calculate the loss [1]. The cross entropy loss function can be given as [7]

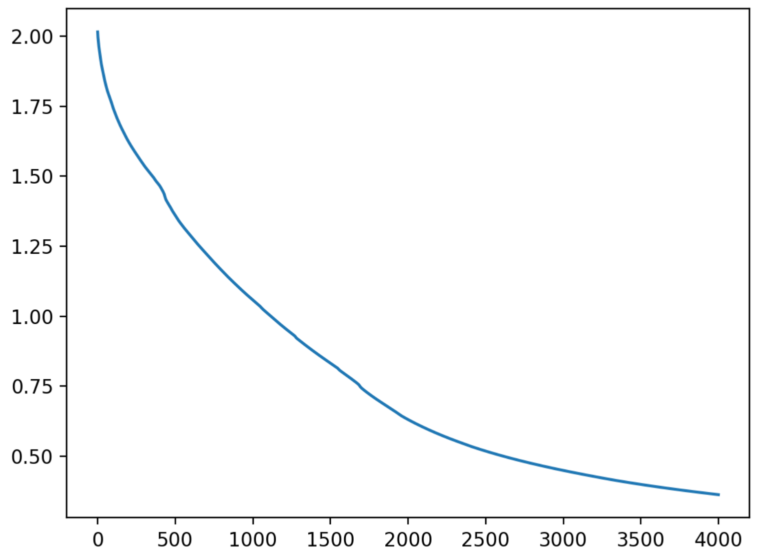
$$\text{loss}(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

The optimiser that we chose was stochastic gradient with learning rate of 0.01.

3.2 Training and Testing Accuracy

The initial weights are assigned random values and using the gradient descent method, the weights are adjusted after every epoch until the loss reaches to minimum possible value. After varying different number of epochs for training, it was found that 4000 epochs were good enough for our network to learn and not cause any overfitting. The graph between loss to number of epochs has been given in Figure 1.

Figure 1. Graph between loss rate (y-axis) vs number of epochs (x-axis). As the number of epochs tends to infinity, the loss rate tends to zero, but never truly reaches to zero.



For each class, there are 30 instances available in the training dataset and 300 instances on testing dataset [8]. Therefore, for seven classes, overall 210 classes are available for training and 2100 for testing.

The overall accuracy of our training dataset of 210 instances was found around 91.43%, whereas the testing accuracy was around 87.81%. The results of both, training and testing has been given in Table 1.

Table 1. Results of the classification by Two Layer Network.

Classes	Number of instances identified correctly in training (out of 30)	Number of instances identified correctly in testing (out of 300)
Brickface	29	293
Sky	29	294
Foliage	26	244
Cement	24	196
Window	24	224
Path	30	293
Grass	30	300
Total	192 (210)	1844 (2100)

4 Deep Neural Network

4.1 Proposed Model

The deep neural network is nothing but artificial neural network with multiple hidden layers. Using the above developed two-layer neural network, we added two more hidden layers. So overall our deep neural network has one input layer with 19 neurons, three hidden layers with different number of neurons in each layer (50 in first hidden layer, 30 in second and 60 in third) and one output layer with 30 neurons.

One problem with learning in multi-layer model with gradient-based learning methods and backpropagation is that the gradient of the error function that gives an update to each weights can be vanishingly small. This could refrain the weights from changing weights and restrict the network from learning. This is called vanishing gradient problem.

This problem was observed even in our deep neural network. Our model was only able to provide up to 15% accuracy only. There are various suggested methods available to solve this vanishing gradient problem. One of the methods is using rectifiers for activation functions.

One such rectifier is Rectified Linear Unit or ReLU. Its function can be given as

$$\mathrm{ReLU}(x) = \max(0,x)$$

We tried using ReLU instead of existing sigmoid function from previous model. It did help in solving out vanishing gradient problem and our network started learning just fine.

4.2 Training and Testing Accuracies

Our dataset had 210 instances for training, which is far too less for our deep neural network to learn from. So we interchanged the datasets, and 2100 instances that were originally intended for testing was now being used for training and vice versa. 4000 epochs were fat too less from our previous model, and so we kept on increasing the number of epochs for which the network was trained for, and finally decided to settle for 8000 epochs.

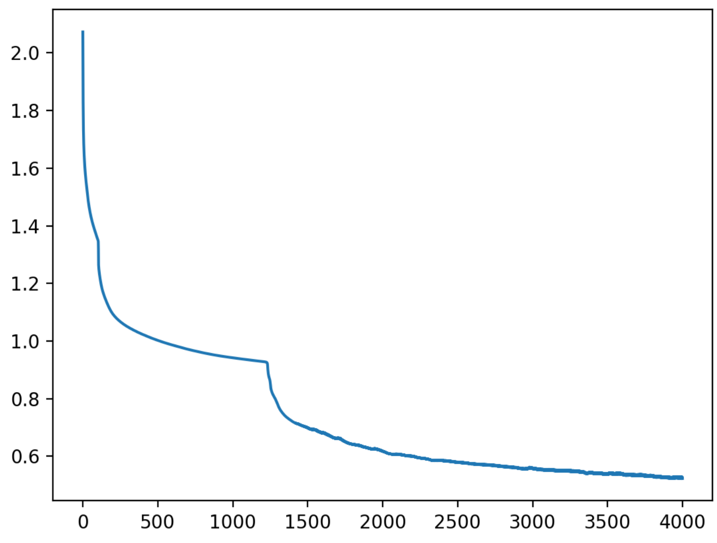
Our deep neural network had training accuracy of about 89.95%, which is same as that of our two-layer network. Considering the already high accuracy our previous model, there wasn't much scope in improvement in training. All it means is that our network is training just as well if not better. Table 2 describes the accuracy for each class.

Table 2. Results of the classification on training by Deep Neural Network.

Classes	Number of instances identified correctly (out of 300)	Number of error (loss)	Percentage Error (loss)
Brickface	294	6	2
Sky	293	7	2.33
Foliage	265	35	16.6
Cement	227	73	11.67
Window	220	80	26.67
Path	290	10	3.33
Grass	300	0	0
Total	1889 (2100)	211	10.05

The graph between loss and number of epochs for our deep neural network can be given in Figure 2. It is not as smooth as our two-layer model, but it shows that our data is not over-fitting.

Figure 2. Graph between loss rate (y-axis) vs number of epochs (x-axis) for deep neural network. As the number of epochs tends to infinity, the loss rate tends to zero, but never truly reaches to zero.



Testing accuracy had slightly increased, close to 89.05% as compared to 87% in our previous model. Table 3 shows the testing accuracies for every class

Table 3. Results of the classification on testing by Deep Neural Network.

Classes	Number of instances identified correctly (out of 30)	Number of error (loss)	Percentage Error (loss)
Brickface	29	1	3.3
Sky	30	0	0
Foliage	25	5	16.6
Cement	24	6	20
Window	23	7	23.3
Path	30	0	0
Grass	30	0	0
Total	191 (210)	19	9.04

Thus, our deep neural network is working well for our classification problem.

5 Case Study – Support Vector Machine

Another way of classifying the given data is Support Vector Machine, which has already been done by James [5] on the data set that we chose. The error calculated by him is given in the Table 4 [5].

Table 4. Results on testing data set using Support Vector Machine (SVM).

Classes	Number of test patterns in the classes	Number of error (loss)	Percentage Error (loss)
Brickface + Cement	600	18	3
Brickface + Cement +Foliage	900	40	4.4
Brickface + Cement +Foliage + Grass	1200	43	3.6
Brickface + Cement +Foliage + Grass + Path	1500	60	4
Brickface + Cement +Foliage + Grass + Path + Sky	1800	62	3.4
Brickface + Cement +Foliage + Grass + Path + Sky + Window	2100	180	8.6

We can observe that model developed by James [5] performs slightly better than our two-layer network and deep neural network. It is hard to say which model is better, as it would be like comparing apples with oranges. SVM is another supervised learning model that is being used to solve classification problem in this case. Some of the advantages of SVM is that it avoids over-fitting of the data, which is being handled by us in our model. So overall our models are aligned with the SVM model made by James [5].

6 Pruning

The problem with error-back propagation is that the efficient structure of the network cannot be defined before the training starts [2]. The input and output neurons can be easily estimated based on the input attributes and output classifiers, but deciding the number of neurons for the hidden layer can be a bit difficult [2].

That is why it is the best preferred to begin with a number more than the required amount. Even for the network model that was used for this research paper, the number of neurons in hidden layer were “guessed” and started with 50 neurons. It is possible that the network would perform just as well or even better with lesser number of neurons in hidden layer. Thus, it is a good practice to find the optimum size of the network not only to improve the efficiency, but also to record the minimum number of hidden neurons required next time the network will be used for classification [2].

The weights of the back-propagation network after each epoch are updated as

$$\Delta w_{ij} = -\eta \frac{\delta E}{\delta w_{ij}}$$

for some gain factor η [4]. Any pruning algorithm is based on the idea that the synaptic weights which have weight decay, $\delta E/\delta w_{ij}$, is close to zero will experience an exponential time decay and keeping the weight would be like adding the penalty value to the error function [4].

Lot of research papers have described different ways of pruning, like relevance [6], badness [3], distinctiveness [2], etc. But for the purpose of this paper, we will look at sensitivity [4] method for network reduction.

6.1 Sensitivity

The challenge in removing the neurons is to decide which neurons to remove from the layer. Sensitivity observes the changes in the synaptic weights of the layer after certain epochs. The sensitivity is calculated using

$$S = -\frac{E(w^f) - E(w^i)}{w^f - w^i} w^f$$

where w^f is the final value of connection after completion of training phase, w^i is the small random initial value of weight that the network started training with and E is expressed as function of w [5].

A “shadow array” is used that keeps track of all the changes to the synaptic weights, sorts them in the decreasing order of their changes, and then discard the ones at the end of the array, since these will be the ones that very low sensitivity or remove the neurons that have synaptic weights below a certain threshold.

For our model, we haven’t defined any threshold for pruning. We will be assuming that the neurons which haven’t shown any changes in its weights after certain number of epochs have the lowest sensitivity and hence remove them. We are also limiting to at most one neuron when we will be pruning the neurons (at most one from each layer in case of deep neural network). We will be applying pruning on the single layer of our two-layer network as well as all the hidden layers of our deep neural network and see how our networks perform after pruning.

Additionally, we can’t directly remove the neurons from our network. So we will be making weights of pruned neurons zero manually. But gradient will update the weights of the neurons nonetheless, and so we keep the record of all the neurons that have been pruned and make sure those are assigned zero after every epoch.

6.2 Pruning on two-layer neural network

For our two-layer neural network, we observed the change in weights after certain number of epochs, which varied from 10 to 100, and the neuron which showed no change in weights were removed from the hidden layer. We selected to prune our neurons after certain number of epochs because after pruning, the accuracy of our network dropped drastically, sometimes even dropping to just 14%, and so we wanted our network to retrain before we can remove another neuron.

On varying the interval of pruning, it was found out that 10 epochs were far too less, and the network did not get enough time to retrain. On the contrary, 100 epochs were too long and did not remove enough number of neurons from our network. So the pruning technique worked well between 45 to 70 epochs, and so we decided to prune our hidden neurons after every 50 epochs.

It was observed that anywhere between 0-5 hidden neurons were removed from our network i.e. up to 10% of the hidden neurons were removed. Considering the training situation when 5 neurons were removed, the accuracy was still found to be about 90.96%, which is equal to the testing accuracy of the network with 50 hidden neurons.

Now that the network was trained and pruned, it would be a bad network if it can’t classify the any given inputs accurately. So to test our network after pruning, the same test data set of 2100 instances were passed to the pruned network and observed. The test accuracy with 45 hidden neurons were found to be 88.43%, which is nearly same to the 87.81% accuracy of the network with 50 hidden neurons.

Table 5 shows the accuracy results of our two-layer model after pruning.

Table 5. Results of the classification on data set after pruning on two-layer neural network.

Classes	Number of instances identified correctly in training (out of 30)	Number of instances identified correctly in testing (out of 300)
Brickface	29	294
Sky	30	295
Foliage	25	248
Cement	24	214
Window	23	218
Path	30	289
Grass	30	299
Total	191 (210)	1857 (2100)

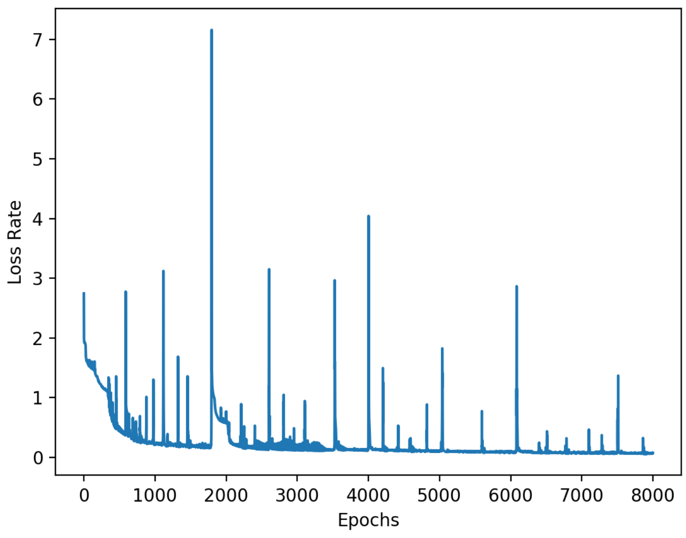
6.3 Pruning on deep neural network

After applying pruning algorithm on the single layer of two-layer model, we will apply the same technique on all the hidden layers of our deep neural network.

All the pruning criteria remain the same from our previous two-layer network model i.e. remove at most one neuron that showed no changes in weights at all. We manually make the weights zero of the pruned neurons for every epoch, and in this case we need to record all the pruned neurons for every hidden layer.

As expected, the accuracy dropped drastically when the neurons were pruned during the training stage. The losses were even increased six folds sometimes, assuming since we are applying the pruning three times and thus increasing the number of neurons that were pruned. So we had to retrain our network again. We tried retraining our network for 50 epochs again, but it was far too less and our network did not retrain enough. So we kept on increasing the interval, and found out 200 epochs was a good time for our network to retrain. The variance in loss can be seen in Figure 2, where it shows a sudden spike in loss when the neurons are pruned, and how retraining reduces the error.

Figure 2. Graph between loss rate (y-axis) vs number of epochs (x-axis). The spikes in the graph show that the loss increases drastically immediately after pruning but decreases after retraining.



The pruning was much more efficient in this model than the other two-layered model. It was able to prune up to 15% neurons in first hidden layer, 70% in second and 40% in third. Second hidden layer saw the most significant changes compared to the other two layers. Overall we were able to reduce our network by 40%, and still able to achieve accuracy of 95.24% after consistent retraining. The training accuracy for each class can be given in Table 6.

Table 6. Results of the classification on training by Deep Neural Network after pruning.

Classes	Number of instances identified correctly (out of 300)	Number of error (loss)	Percentage Error (loss)
Brickface	297	3	1
Sky	300	0	0
Foliage	292	8	2.67
Cement	289	11	3.67
Window	233	67	11.33
Path	290	10	3.33
Grass	299	1	0.33
Total	2000 (2100)	100	4.7

Although the network has learnt extremely well, it shows great results while testing too. It was able to classify images with 93.81%, much better than any other model mentioned in this paper. The results based on the classes can be given in Table 7.

Table 7. Results of the classification on testing data set by Deep Neural Network after pruning.

Classes	Number of instances identified correctly (out of 30)	Number of error (loss)	Percentage Error (loss)
Brickface	29	1	3.3
Sky	30	0	0
Foliage	29	1	16.6
Cement	28	2	20
Window	21	9	23.3
Path	30	0	0
Grass	30	0	0
Total	191 (210)	19	9.04

The number of hidden neurons in each hidden layer were varied, to observe the behaviour of pruning in each layer. The observations have been given in Table 8.

Table 8. Results of the classification on testing data set by Deep Neural Network after pruning.

Number of neurons in hidden layer 1	Percentage of neurons removed from layer 1	Number of neurons in hidden layer 2	Percentage of neurons removed from layer 2	Number of neurons in hidden layer 3	Percentage of neurons removed from layer 3
50	4	50	70	50	30
20	15	60	70	30	30
60	5	20	55	40	25
50	10	20	65	70	30
30	13	30	70	30	36

It was generally the second hidden layer that saw the most pruning. Irrespective of being having the highest number of neurons or lowest or even equal, 50 to 70% of neurons were pruned regardless. On the contrary, the hidden layer 1 saw the least pruning, up to 15% only. This can be because of being close to the input layer it changes its values more often as compared to the other two layers. The third layer showed some consistent amount of pruning, usually in the vicinity of 30%.

7 Conclusion and future work

Thus an oversized two-layer network was made successfully and the same model was extended to create a multi-layered neural network also called as deep neural network. The deep neural network saw vanishing gradient problem, which was solved using rectified linear unit activation function. The models were compared with another technique, called Support Vector Machine and found out that the accuracy of both the models were nearly the same. A pruning technique called sensitivity was studied and applied successfully on both of our networks. Although the pruning method was able to reduce the size of two-layer neural network by only 10%, it was able to reduce our deep neural network by 40%. Furthermore, it was observed that some layers were more responsive to pruning than others.

Using this method of pruning, we could predict the size of the network next time who wants to solve a similar type of classification problem. Also for future scope, we can study on how well the sensitivity works on larger and deeper neural networks that consist of much more hidden layers and compare it with other pruning techniques and see if they work just as same. This pruning technique can further be used for other Deep Learning networks like Convolution Neural Networks (CNNs), Recurrent Neural Networks (RNNs) etc.

8 Reference

1. De Boer, P.T., Kroese, D.P., Mannor, S. and Rubinstein, R.Y., A tutorial on the cross-entropy method. *Annals of operations research*, *134*(1), pp.19-67 (2005)
2. Gedeon, T. D., Harris, D., Network reduction techniques. In Proceedings International Conference on Neural Networks Methodologies and Applications, Vol. 1, pp. 119-126 (1991)
3. Hagiwara, M, "Novel back propagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection," IJCNN, vol. I, pp. 625-630, (1990)
4. Karnin, ED, "A simple procedure for pruning back-propagation trained neural networks," IEEE Transactions on Neural Networks, vol 1., pp. 239-242, (1990)
5. Kwok, J.Y., Moderating the outputs of support vector machine classifiers. *IEEE Transactions on Neural Networks*, *10*(5), 1018-1031, (1999)
6. Mozer, MC, Smolenski, P, "Using relevance to reduce network size automatically,", Connection Science, vol. 1, pp. 3-16, (1989)
7. PyTorch master documentation, http://pytorch.org/docs/master/nn.html, (2018)
8. UCI Machine Learning Repository: Image Segmentation Data Set, http://archive.ics.uci.edu/ml/datasets/Image+Segmentation, (2017)