Verifying the benefits pruning of Neural Networks based on Neuron Behavior on a Classification Task

Sambit Ghosh

Research School of Computer Science,
Australian National University
Canberra ACT 0200 AUSTRALIA
U6651968@anu.edu.au

Abstract. In practice when working with neural networks there are a lot of factors that affect the learning of a neural net based. Number of neurons is one of them. It is very hard to determine what the number of neurons should ideally be. Often it is considered best to overestimate the number of neurons and then get rid of them. We do this through various pruning methods. Here we will be investigating one such pruning method on a dataset which it has not been applied on before. The dataset is the observations from a study which investigates which form of navigation is ideal in case of a mobile web search. We show that the technique of pruning based on behavior works in two different ways on this dataset by applying it to two different Neural networks. One simple and one deep network. We will highlight the importance of this technique as well. We conclude that this pruning method is useful for some neural networks but may not be suited for others.

Keywords: Pruning, Classification, LSTM, Deep neural network

1 Introduction

When neural networks are trained it is hard to determine the number of neurons, they will likely be needing to be able to learn all aspects of a dataset. The number is usually overestimated. This causes the network to be unnecessarily bulky. It takes more time to train too. This can be changed with pruning. We take unnecessary neurons and get rid of them. This is done based on some criterions. There are various measures to do this in the relevant literature. But our focus is on the method suggested by T.D. Gedeon in his paper Indicators of hidden neuron functionality: the weight matrix versus neuron behavior [1]. He suggests usage of a measure called *distinctiveness.* This is described in the paper as follows, the distinctiveness of hidden neurons is determined from the neuron output activation vector over the pattern presentation set. That is, for each hidden neuron we construct a vector of the same dimensionality as the number of patterns in the training set, each component of the vector corresponding to the output activation of that particular neuron. This vector represents the functionality of the hidden neuron in (input) pattern space.[1] With this definition of distinctiveness we move to construct a simple neural net which does a classification task. Based on our threshold values for tolerance of similarity we remove one of the two similar neurons and add its weights to the one we did not remove.

By doing this we make the size of the network smaller without worrying about deleting neurons which actually have learnt meaningful data distinctly. To do this we create two neural networks for classification. One is a simple network based with one layer. Second is a deep neural network with Long-Short-Term-Memory. We then apply the technique mentioned above. We will be using dataset from [2]. Which explores which mode of navigation is best for a web search, pagination or scrolling.

1.1 What To Classify And Why?

The data set from [2] provides a set of observations which record the behavior of a subject when interacting with their experiment. The details to this experiment are largely irrelevant to our goal. So, we will go over it briefly.

We want to create a simple testing ground to verify the technique, therefore we will not construct a problem which is quite meaningful in terms of what it does to give us more insight into the data set and meaning to it. Instead we are going to just use the data to give us a testing ground for our verification task. Keeping that in mind, We use sheet01 from the dataset only, which has mostly numeric data about various behaviors of the subject.

We choose to try and classify the attribute “Accuracy”. Accuracy in this case is not a continuous number, instead it is a classification between 0 and 1. This was chosen because it made preprocessing easier, since it was already in the required encoding that out network needs. And since it only has two classes, we can more easily visualize the problem.

2 Methods

2.1 Pre-processing The Data

The dataset already being mostly numeric was a delight. But the first column was not numeric. For our purposes, since we are not trying to find meaningful use of the data, we just discard this column entirely. We also remove the subject column as it is just identifiers. The Time on wrong webpage column is also removed since it had a lot of skewed data.

Next we split the data into 80% for training and 20% for testing. This is enough for the task we want to accomplish since we do not necessarily want the best network possible in which case we could have done cross validation since there is very little data. But our goal is to verify the pruning technique so we can cut corners here to save time and implementation hassle.

Next we normalize the data between 0 and 1. This makes the data standardized. If we do not do this, we will cause the weights in the network to be biased by higher or lower raw numbers which we do not need for our purposes.

We then take this preprocessed data and extract the input features and our target accuracy feature and get them ready to be fed into our network.

2.2 The Networks

2.2.1 Simple Neural Network

This is a simple network and thus we build what could barely be called a neural network. It has 1 hidden layer and an output layer. Since we have 11 input features if we tried to remove neurons from say 7 neurons, we would likely get no similar enough neurons to do our pruning. So, 15 seemed like a good compromise. Which also gives us a use case for this experiment, compression. We can use this pruning technique to compress our dataset while keeping all essential information intact. More on that later. The hidden layer was a linear layer. Activation function relu was used in the hidden neurons and the output neurons simply because it seemed to perform best among other functions, we are not using a multi layered network therefore, this should not matter too much.

2.2.2 Deep Neural Network

For a meaningful deep neural netwok we tried out two different types of network to check which performs better.

First is a simple three layer deep netwok(DNN) made of linear layers along with an output layer. Since the dataset is short and such a deep network can tend to overfit on the data there were dropouts added to the network. Dropouts disconnects some of the nodes randomly in a netwrok. This introduces some noise in the network. This would keep the network form learning too specifically the training data. Batch normalization was also added to help the netwrok generalize more. This reduces the internal covariance shift and introduces some noise in calculation of means in each batch. Each layer of this network had a 100 neurons and used Relu activasion function. Relu was used here specifically because it is designed to work better with multiple layers.

Second deep network was a network with Long-Short-Term-Mermory(lstm). This netwrok had 2 layers of lstm neurons and a fully connected linear layerand another linear layer for output. LSTMs are good for learning trends over time and predicting based on those trends. The decision to have an lstm was made because of the data.While the data is not particularly a time series or sequential data, it is about people’s behaviour with a certain system. This means that there might be certain trends that are in all participants of the experiment which might be picked up by the deep network. People some times tend to do similar things overall and if we can pick up on some of these using an lstm we might have a more useful network. Each layer of this network has 256 neurons. The activation function used is Relu for the fully connected layer the reason desribed already.

Only one of these networks is chosen to perform the pruing on.

2.3 Training The Networks

The loss function was chosen to be Cross Entropy loss as it is well known to be good for classification tasks and it is fast and easy to implement [3,p. 1]. For our purposes this this good enough. To train all the netwroks optimiser Adam [4] was used. This was found out to be best through experimentaion. SGD was also tried.

The simple network is trained for 500 epochs using backprogagation. Not much value is found in training the network as was expected. It was able to reach accuracy of 98% in 500 epochs. Pretty good for what we set out to do.
Then we moved on to training the deep networks.

The simple deep netwrok was trained for 500 epochs using backpropagation. With a learing rate of 0.002 the network had an accuray of 97.16% on training set. This was quite satisfactory for a deep network. But we need to test it on test set to be able to determine which among the two deep netwroks is better.

The LSTM netwrok was trained for only a 100 epochs since it takes much longer to train an lstm. It was also done to prevent overfitting to the data. Unlike in the simple deep network we are not using dropouts in the lstm to regularize it. Initially the network was getting stuck on a plataue but increasing the learing rate to 0.005 helped it reach the solution faster. On the training set the network scored 85.97% accuracy. This was expected as the data is not sequential. But we will have to test it on a the testing set to be sure.

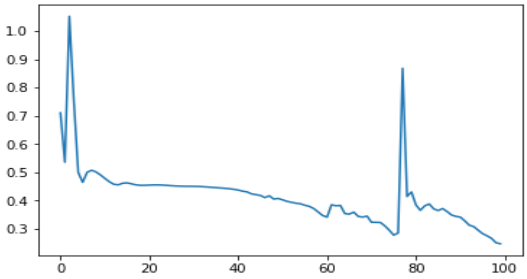

Fig. 1. Loss over epochs for LSTM showing plateau in learning

2.4 Results of training

The Simple netwrok worked well in testing

```
Testing Accuracy: 98.46 %
Confusion matrix for testing:
tensor([[ 8.,  1.],
        [ 0., 56.]])
```

The DNN network performed much poorly on the test data.

```
Testing Accuracy: 67.16 %
Confusion matrix for testing:
tensor([[ 44.,  8.],
        [ 14., 1.]])
```

The LSTM surpringly did better than the DNN to predict on the testing set.

```
Testing Accuracy: 79.10 %
Confusion matrix for testing:
tensor([[51.,  1.],
        [13., 2.]])
```

Based on these results we can safely choose the LSTM netwrok to be our deep netwrok for the pruning process.

2.5 Applying Pruning

Now that we have a trained network we need to prune it using the technique descrided in [1] and pit it against the not pruned network.

To do this we use disticntiveness feature as mentioned in [1]. We do this by taking the activated weights from the neurons and making a vector out of it. This vector describes the fucntion of the neuron. If two neurons are too similar they one of them needs to be taken out. If two neurons are almost opposite to each other, both can be taken out. Distinctiveness is measured using the angular seperation between the vectors. Once we compute that we check if any two neurons are too similar.

It is stated in [1] that from experince it is suggested that a seperation of 15 degrees or less is too similar and one of the neurons can be removed. Although, the weight of the removed neuron must be added to the neuron that remains. If the angle is more than 165 degress than the neurons are opposite of each other and both can be removed. If the angle is more than 165 degress than the neurons are opposite of each other and both can be removed. We will be using that recommended degree of seperation to do our pruning.

Our implementaion of this is that we calculate the the distinctiveness of every pair of neurons and note which ones are similar. Then go back to that list and remove one of the neurons from each pair of similar neurons iteratively through the list. If we encounter a situation where we already removed a neuron and then we leave the other neuron in the pair as it is. The code for this implementaion is availble in the Appendix section 6.1.

For the simple neural network we apply the pruning to the only hidden layer available. For the LSTM network we apply the pruning to the fully connected layer of the network. This is the layer that connect the LSTM layers to the output layer.

3 Results

3.1 Effect of pruning on simple Nerual network

To express this correctly I would like to present a run of the pruned and non pruned netwrok side by side.

Firstly here is what the results were for non pruned network:

```
Testing Accuracy: 98.46 %

Confusion matrix for testing:
tensor([[ 8.,  1.],
        [ 0., 56.]])
```

And here is the result after pruning:

```
Testing Accuracy: 98.46 %
Confusion matrix for testing:
tensor([[ 8.,  1.],
        [ 0., 56.]])
```

Looks similar. They were the same. The nework even after pruning performed exactly the same.
In this case 2 pair of neurons were found to be similar and one neuron from each pair was removed.
And yet the network performed just the same. With a few more runs it was clear that the pruning method was working very well there was loss of accuray of less than 1%.
Therefore it is verifired that this method of pruning is very useful for such simple networks. This shows that even simple networks can have unessesary neurons and it might be beneficial to get rid of them to make our network compressed. And one such effective way to get rid of them is using the distinctiveness to prune the similar or opposite neurons.

3.2 Effect of pruing on Deep Neural Network.

Here are the results before pruning:

```
Testing Accuracy: 79.10 %
Confusion matrix for testing:
tensor([[51.,  1.],
        [13.,  2.]])
```

And here are the results after pruning:

```
Testing Accuracy: 79.10 %
Confusion matrix for testing:
tensor([[51.,  1.],
        [13.,  2.]])
```

Just as the simple neural netwrok this also had no change after pruning.
In this case 4 pairs of neurons were found to be similar and one neuron from each was removed.
This is a suspiciously low number of neuron to be found similar in such a large network. This may be due to the lstm layers which learn signifiacntly different things and, since the fully connected layer is just a connection layer from the lstm layers to the output, that layer itself does not learn much itself. Upon further runs it was obseved that certain runs did not produce any similar neurons at all.

From this we can conclude that not all neural networks can be pruned effectively and sometimes all neurons are nessesary. In particular we can make an argument that LSTM networks are probably not the best candidates to prune. Although this would requie futher reseach beyond the scope of this paper to be definitive.

4 Conclusion and Future Work

This method of pruning is very useful in certain cases as we saw. This suggests that even the most simplest neural nets can be overcomplicated. This is interesting it itself. Further, we also saw that the Deep LSTM network does not respond well to pruning. There were hardly any neurons found to be similar based on this pruning technique. This could provide more insight to which sort of neural nets are more succeptable to being overcomplicated.Perhaps the LSTMs have certainn qualities which makes each of their neurons to be unique, or perhaps a different kind of pruning would be more applicable on such networks. This is something that can be explored in further research. It is also interesting that the normal neural net had better predictions than any of the deep networks. Perhaps deep networks need more data to learn properly. It may also be interesting to note that the LSTM performed worse on the training set than the DNN but performed better on the testing set. This might be due to the fact that the DNN dropouts are loosing valuable information or the network migth still be overfitting the training data. It could also be that our hypothesis that people behave in a certain way when interacting with the same system may be true and out LSTM was able to pick up on those trends as LSTMs are made for sequential data. All of these avenues can be explored in a future work.

5 References

[1] T. D. Gedeon, "Indicators of hidden neuron functionality: the weight matrix versus neuron behaviour," Proceedings 1995 Second New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems, Dunedin, New Zealand, 1995, pp. 26-29, doi: 10.1109/ANNES.1995.499431.

[2] im, J., Thomas, P., Sankaranarayana, R., Gedeon, T., & Yoon, H. J. (2016, October). Pagination versus scrolling in mobile web search. In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (pp. 751-760). ACM.

[3] Martinez, M. and Stiefelhagen, R., 2018, October. Taming the cross entropy loss. In German Conference on Pattern Recognition (pp. 628-637). Springer, Cham.

[4] Kingma, D.P. and Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

6 Appendix

6.1 Code Implementation for pruning:

```
#list to store the pair of similar neurons
similar = []
x=0
y=0

#comapare each neuron with each of the other
while x < net.hidden.weight.shape[0]:
    y = 0
    while y < net.hidden.weight.shape[0]:
        u = net.hidden.weight[x]
        v = net.hidden.weight[y]
        angle_sep = angle_between(u.detach().numpy(),v.detach().numpy())
        angle_sep = angle_sep * 180 / math.pi

        #if anglle of seperation is less than 15 degrees we add the pair to
the list
        if angle_sep < 15 and x != y:
            if x < y:
                tup = (x,y)
            else:
                tup = (y,x)
```

```
            flag=1
            for tuple in similar:
                if tup == tuple:
                    flag=0;
            if flag == 1:
                similar.append(tup)
        y =y+1
    x= x+1
# now we remove one of the similar nodes
#and add the weights to the one that remains
for tup in similar:
    u = net.hidden.weight[tup[0]]
    v = net.hidden.weight[tup[1]]
    new_u = v+u
    if u.dim != 0 :
        net.hidden.weight[tup[0]] = new_u
        net.hidden.weight[tup[1]] = 0
```