Sensitivity based Pruning hidden Neurons

Jaspreet Singh
Australian National University

1. Abstract

There is a lot of research going on which tends to improve the efficiency of the neural networks without hindering the accuracy. Neural networks and deep learning is still under a lot of development and is definitely worth researching and developing many advanced techniques to solve efficiency issues.
This paper aims to apply pruning techniques to Neural networks and deep neural nets using sensitivity which is defined as the change in weights if the . There was a marginal improvement in the average result for the deep neural net as well as single layer neural net. The deep neural net provided better result with sensitivity based pruning. The results for a single layer net for our data set of classifying printed letters was around 89% and for a deep neural net with 3 layers was around 91.5%. A 2% increase from 89% to 91% is significant in neural networks especially when the neural net is already around 89% precise.

2. Introduction

Over the period of time, there are many techniques suggested by many researchers. This paper aims to explore one of few techniques and observe how it impacts the accuracy. This paper also aims to modify the technique and improve it to provide better results than the original technique, if possible.
This paper deals with a classification problem that classify the input image to a character from the list of capital Alphabets. The method of classification will be using a neural network by adding the technique called Sensitivity inside the network. The sensitivity factor is taken from a list network reduction techniques provided by previous research(Frey and Slate, 1991). The NN model will have a big hidden layer so that it can adjust to 26 output classifiers. This paper aims to achieve good accuracy with pruning few neurons from the network and achieve as good accuracy as without pruning the model.

3. Dataset taken

The dataset taken for this paper is a classification of printed capital letters of different typeface. The dataset had 16 features already extracted from a set of 20,000 images of alphabets. Image 1 contains few samples of those images. The target values for the dataset is a set of 26 characters from *A* to *Z*. The dataset is split into 3 parts for training, validation and testing in a proportion of 70:15:15 respectively. It can also be broken down to 2 sets split into 80% for training data and 20% for testing data. the dataset is preprocessed using Z score value which is the value subtracted by the mean value of all the input values divided by the standard deviation of all the inputs. The characters are normalised by subtracting the ASCII value of A. So, the value of output classes varies from 0 to 25 representing the characters from *A* to *Z* (Archive.ics.uci.edu, 2018).

4. Pruning and Backpropagation

Back propagation is considered as time consuming and memory in-efficient. Furthermore, if there are more hidden neurons than required to train the dataset, it can lead to overfitting. This is a major issue for training data as the real world data might be too general/random for the trained neural net for a classification problem i.e., if the model is overfitted to classify cars from images and if a new image is feeded to the model, it might not classify the image as a car image even though it is a car image. Pruning is useful in two ways, one is it can be used to reduce the number of hidden neurons which in turn is useful in efficiency of the neural network. Another reason to use pruning is to create a faster network which will train in less number of epochs and it will be as efficient as the network with more number of hidden neurons(Frey and Slate, 1991). This obviously will work for a network that is sophisticatedly designed and the dataset has a well defined feature set. This level of customisation may sometimes lead to a bad network. So, to the pruning is done with a strict criterion. In our case, we pruned a neuron if the neuron if the sensitivity is less than 0.001 for a neuron.

5. Implementation techniques to prune neuron

There are couple of techniques that can be used to implement pruning of the networks. For removing the hidden neuron x, per se, due to limitations of manipulating heavily abstracted and wrapped neural net objects, we can zero all the weights for all the connections from neuron x to all the output neurons. This way, we can make sure that the hidden

neuron x is not contributing to the output neuron. There are few disadvantages to this approach. One is that even though the hidden neuron is not impacting the output of the neuron, it will take part in back-propagation process which will hinder the error-propagation to other neurons. While we were working on a single layer network or smaller deep nn, the hidden neuron's error will not propagate furthermore which makes this as a hack. Other disadvantage is that the network takes more memory than it should because we are keeping the neuron which is not contributing to the network, the neuron will eat up extra memory. The second method is to create a new hidden layer with less number of neurons and the weights of the new neuron will get replaced with the weights of the to be replaced hidden layer. The only overhead is to creating the new layer again and again might be time consuming. There is also a limitation with pytorch as we cannot replace a hidden layer without affecting the inner workings of a hidden neurons. There are few workarounds like using hooks to simulate and managing the changes yourself which are out of the scope of this paper.

6. Other parts of our network

6.1 Validation record function

The evaluation function calls the neural network with the validation set. The accuracy of the network is recorded and is used to compare with the running loss.

6.2 Loss Evaluation function

The loss is evaluated using the log loss formula. The log loss is a probability that converges towards 1 if the target value is closer to the predicted value.

6.3 Activation function

There are many options for selecting optimizer. This paper chose Sigmoid over other optimizers like ReLU because of how the Sigmoid method works. ReLU can give 0 as the output function for hidden neuron. As Sigmoid consistently outputs, the weight are bound to change more rapidly in every step. So, the network will not be mislead to pruning the hidden neurons because of a bad step.

7. Pruning restrictions

There are multiple pruning restrictions that are placed while looking for pruning the neurons. The restrictions are as follows:

1. The pruning is done for the *70%* of the total number of epochs. This ensures that while the net converges towards better classification, the code does not prune the neurons that have learnt the features to classify.
2. The pruning should be done for every 50th or 100th epoch. This ensures that the network learns for a significant amount of time and if a neuron has to be pruned, it is removed from the hidden layer.

8. Versions of implementation

There can be many versions of how to implement this. A few are discussed below:

1. The network starts to learn and as the network learns that a neuron is redundant, the neuron is removed from the hidden layer, the remaining connections are retained as is and the network continues to learn.
2. A variation to the above version is that the network starts to learn and keeps record of how many neurons should be kept to create a network of same efficiency. After a certain accuracy is reached or after a few epochs, the network is re-initialized with the same parameters and one hidden neuron less and retrained. This is opposite to what is proposed in the Network reduction technique where the same network is kept and the neurons are removed(Frey and Slate, 1991).

This paper exploits second variation to find the areas of improvement to the already existing algorithms with the single layer network and the 1st technique the deep neural network.

9. Network Parameters

This network starts with 64 hidden neurons. The bigger network is build for 4 neurons per feature because the number of classes to classify are 26 which is pretty big. Generally a binary classifier does not use these many hidden neurons for 16 input attributes.

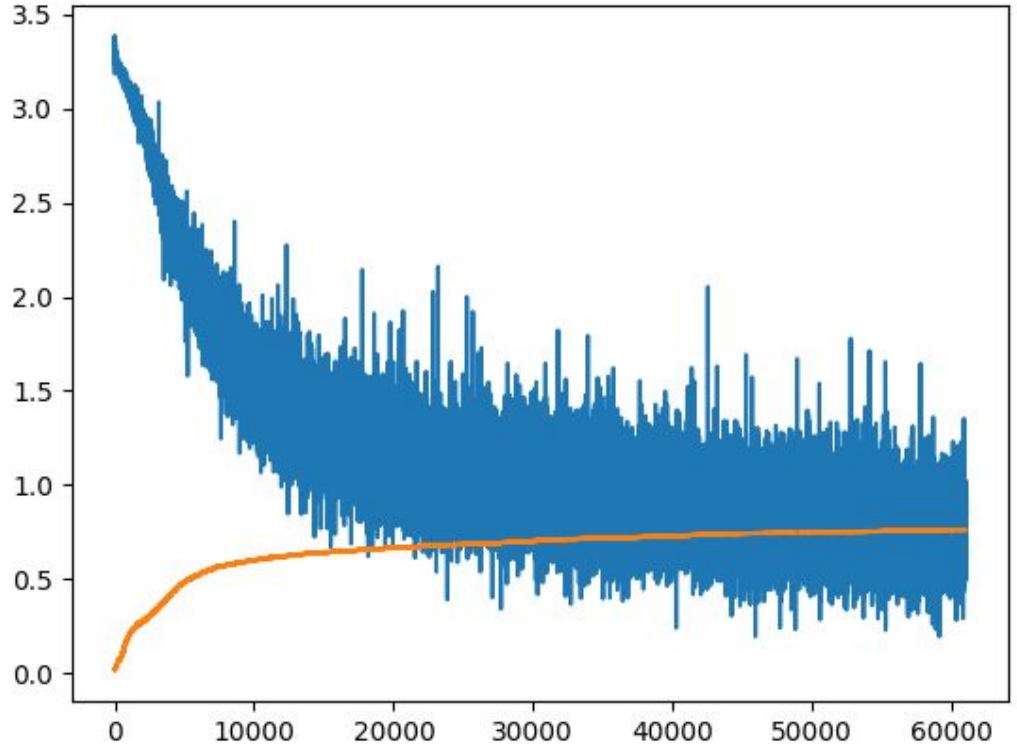
10. Results and comparison

The baseline implementation uses other machine learning techniques apart from neural networks known as Holland-style adaptive classifiers. The best classifier they built had an accuracy of 81.6%(Frey and Slate, 1991). Another contender could have been a neural network with 50 hidden neurons with one layer and Stochastic Gradient Descent. This is a simple configuration but for the purpose of improving other's work, we used adaptive classifiers as our baseline.

The first approach was using ReLU and a neural network with 50 hidden neurons. This basic neural network with 1000 epochs provides an accuracy of 84.7% using ReLU function as activation function, 71% using Sigmoid as the activation function and using SGD as the optimizer. While using ReLU as activation function and RMSProp as the Optimizer and only 200 epochs, the network produced the testing accuracy of 91% which is the best accuracy by a single layer neural net.

The following are the neural networks that were implemented with

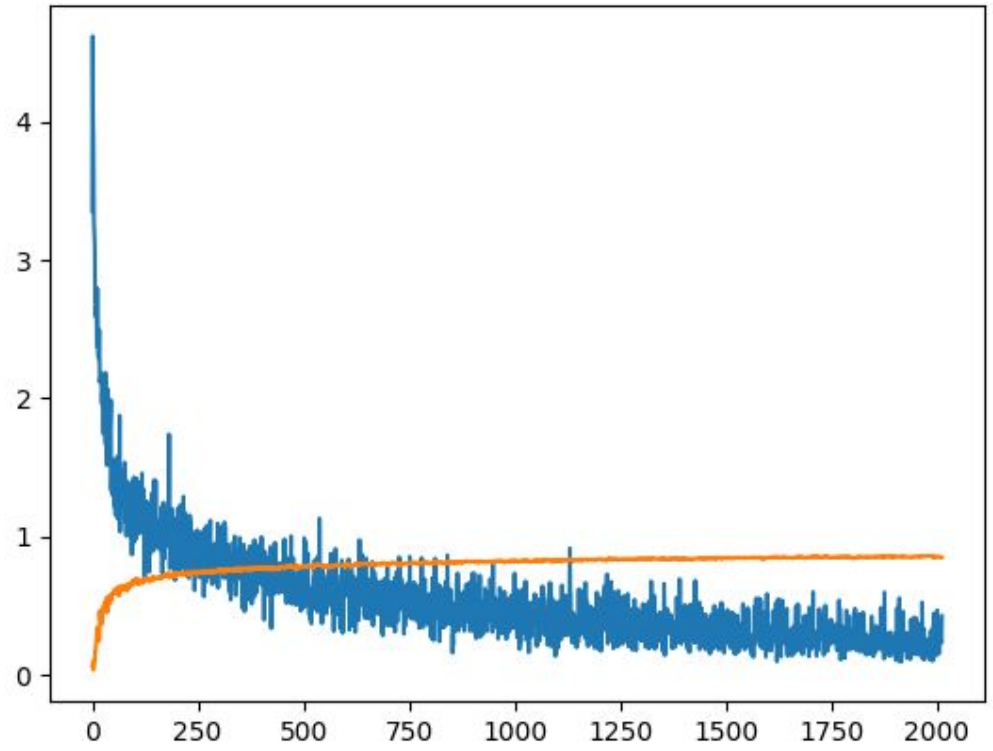
1. When using SGD as the optimizer, the accuracy was around 79%. Figure below shows the loss function.



The blue line is the loss function and the Yellow line denotes the validation function. The x-axis is the number of steps and the y-axis is the loss/accuracy of the result.

SGD did not do a great job because SGD improves with a constant step count and converges slowly towards the local/global minimum. Also, the oscillating value is because when the network trains, the output values changes between 26 output values and the loss is adjusted to different values in backward pass. So, the network has to adjust to classify 26 output classes hence the oscillating graph.

2. When using RMSProp along with Sigmoid function, the accuracy was around 90%. Figure **x** shows the loss function along with the evaluation using validation sets. Figure x shows the accuracy of the testing set. RMSProp resulted in faster learning of the data. The following was a result of 50 epochs as compared to 1000 epochs for SGD optimizer.
   RMSProp converges to the local/global minima/maxima faster than SGD, which means that it will take less number of epochs to train the network and it will converge faster. Thus, the loss falls more steeply in the loss function plot.

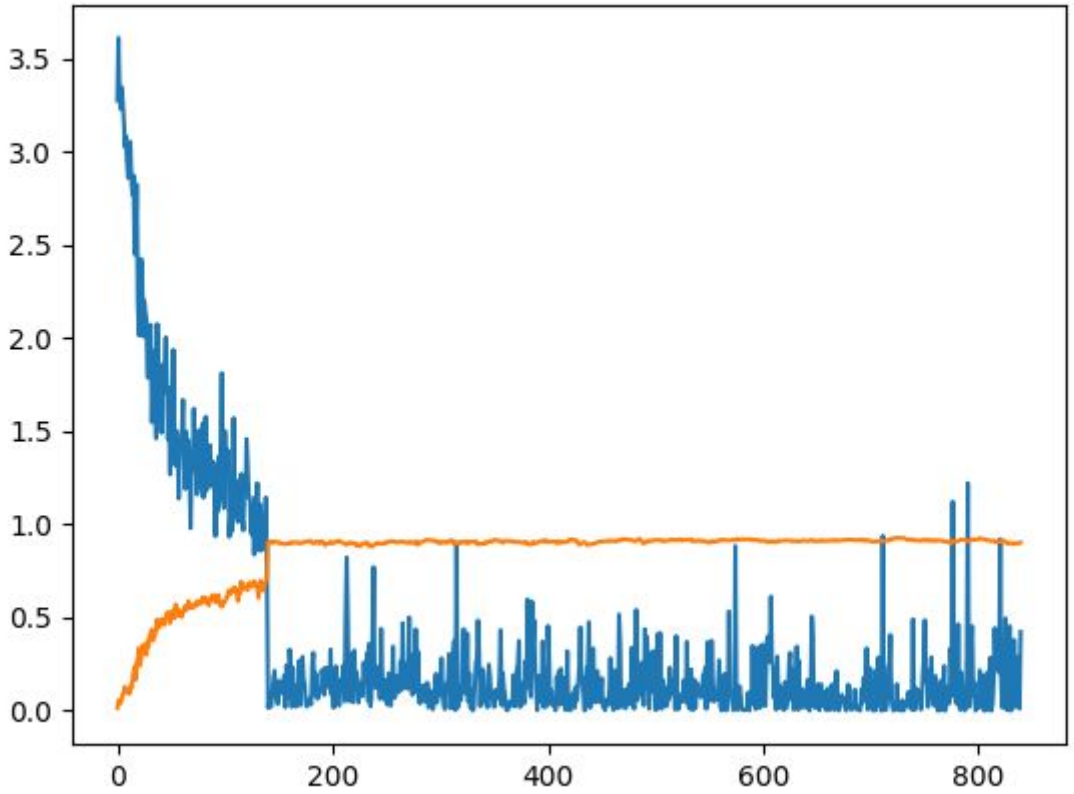


The blue line is the loss function and the Yellow line denotes the validation function. The x-axis is the number of steps and the y-axis is the loss/accuracy of the result.

```
Testing Accuracy: 89.72 %
Confusion matrix for testing:
```

3. We multiplied the number of hidden layers to create a deep neural network with 3 hidden layers and sensitivity based pruning.. The performance of the neural net improved with to an average of around 92% which is a good improvement and the max of around 93% accuracy.
   Because it was a bigger network, one of the results had 9 neurons pruned out of 60 in the first layer, 2nd layer had 1 out of 45 neurons pruned and the third layer had 3 neurons out of 30 neurons. The idea of decreasing number of neurons was taken from the Convolution Neural Nets. The feature extraction mechanism is converts low level features to higher level features.
   This deep neural network works better this way because we are allowing the network to classify and extract features for it. Though it is a simple deep learning network, this fits the best because of the type of dataset we had. It was pre-processed and had the configurations defined already for it.
   The following figures shows the loss function and accuracy of the Deep Neural network with Sensitivity based pruning.

```
Testing Accuracy: 92.07 %
```



The blue line is the loss function and the Yellow line denotes the validation function. The x-axis is the number of steps and the y-axis is the loss/accuracy of the result.

The best the network could perform was with the accuracy of 93%. There are other parameters associated with the pruning as well like badness and distinctiveness that were out of the scope of this research. We were able to improve our performance as compared to the baseline model and previous research by changing the model and doing a couple of modifications and adding sensitivity based pruning for the network reduction.

11. Future work

There are a number of improvements that can be done to this network. While, at the start of the training was used to find for sensitivity, the end of the training can be used to find distinctiveness and badness(Gedeon and Harris, 1991). The network can be perfected even more by tweaking the parameters or updating few functions. We can also tweak the criteria of finding the sensitivity to the neuron using vector angle between the previous step and the current step of the function. More interesting thing would be pruning hidden layer which contains multiple layers. Further work is needed to classify how these techniques can be bad for a neural network and how it can affect the core functionality of the neural networks.
The other improvement could be using TensorFlow or Keras instead of pytorch, which has a more versatile architecture and has a very profound community support.

12. Conclusion

Without hindering the accuracy of the network, we were able to make the network more efficient. The network can be reduced using certain techniques which helps in improving efficiency, robustness, reduce the risk of overfitting and creating better models. These techniques might not suit some models but should work for many networks out there. The accuracy of the baseline network was good but a neural network outperformed the classifier with 10% more accuracy. Also, deep learning network performed better than a single layer network which was expected.

13. References

Archive.ics.uci.edu. (2018). UCI Machine Learning Repository: Letter Recognition Data Set. [online] <http://archive.ics.uci.edu/ml/datasets/Letter+Recognition> [Accessed 23 Apr. 2018]

Gedeon, T. D., and D. Harris. "Network reduction techniques." Proceedings International Conference on Neural Networks Methodologies and Applications. Vol. 1. 1991.

Frey, P. and Slate, D. (1991). Letter recognition using Holland-style adaptive classifiers. Machine Learning, 6(2), pp.161-182.