CS391L Machine Learning Assignment 5 Backpropagation

Srinath Tankasala,
st34546

Abstract

In this assignment, an overview of a Neural Network (NN) is presented and is applied to classify digits from the MNIST dataset. A NN with a single hidden layer was implemented. A gradient ascent algorithm was implemented using Backpropagation to minimize the Mean square error (MSE). A stochastic gradient descent was found to do better compared to cumulative gradient descent for large training data. Finally the prediction accuracy of the network with respect to the number of neurons used were studied

I. Introduction

A neural network consists of layers of artificial neural units, also called ”perceptrons” or ”neurons”. Refer Rosenblatt et al. Each perceptron has an activation based on the weighted linear combination of inputs it receives. This is very similar to how a biological neuron responds to input electrical signals. Stacking this neural units in multiple layers gives us a neural network prediction model.
The main difference between ”perceptron” and ”neuron” used in this report is that the ”perceptron” has a binary activation, meaning it gives 1 if input crosses ”threshold” and 0 if not. A neuron however has a smooth activation function, usually sigmoid.

$$u = \sum_{i=1}^{n} W_i x_i + b;\ \ b = bias \tag{1}$$

$$\Rightarrow u = \sum_{i=0}^{n} W_i x_i;\ \ x_0 = 1\ (for\ Bias\ term) \tag{2}$$

$$g(u) = \frac{1}{1+e^{-u}};\ \ activation\ function \tag{3}$$

$$g'(u) = \frac{e^{-u}}{(1+e^{-u})^2} = g(u)\cdot(1-g(u)) \tag{4}$$

By varying the weights to different neurons in the network, we can get different models of decision making. Neural networks are therefore very popularly used in regression and classification problems. The reason they have become so popular is because of the back propagation algorithm. The back propagation algorithm gives a way to tune the weights of all neurons in a given network.

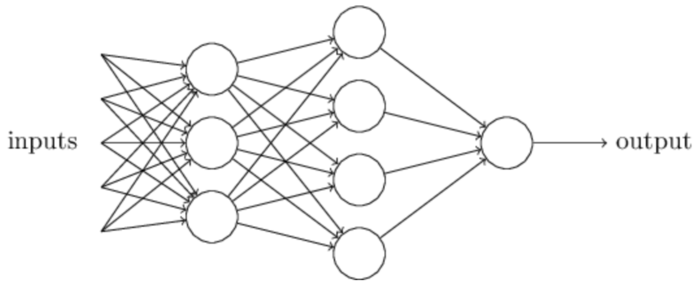

Fig. 1: Sample neural network [1]

Neural Networks (NN) have gained a huge importance in Machine Learning. This is because with enough data, the NN can give a good prediction accuracy for any problem. The bigger advantage is that ”transfer” learning techniques can be applied on pre-trained neural networks like ”AlexNet”, ”ResNet” or ”GoogleNet” to build image or language processors. Neural networks are used heavily in deep learning to obtain complex models with thousands of features.

Here we will build a neural network classifier to recognize handwritten digits in the MNIST dataset. This dataset was already covered in Homework 1 when building a PCA classifier. Hence it’s explanation here is brief.

The MNIST data set consists of grayscale training images and test images for tuning our classifier. The MNIST database has images of size 28x28 pixels. For building the classifier, all pixels in the image are treated as features of the dataset. Thus each image in the dataset has 784 features. There are 10 possible labels for each image ’0’ to ’9’. The data containing the labels is vectorized, meaning a ’0’ is converted to $[1, 0, 0, ..., 0]_{10\times1}$, ’5’ to $[0, 0, 0, 0, 0, 1, 0, ..., 0]_{10\times1}$, etc. This allows us to numerically calculate a classification error.
The pixel levels are normalized from $255 \rightarrow 1$. This is because the bias $(= 1)$ is handled as an extra neuron in our implementation, and normalizing the image will make the bias weight to be on the same scale as that of the image pixels. This scaling will ensure a faster gradient descent.

II. Structure of the Neural network

Given that we are trying to build a NN classifier to identify digits, we will have 10 neurons in the output layer. Each unit in the output layer is a classifier for it’s digit, i.e unit 1 is a classifier for digit ”0”, unit 2 for digit ”1” and so on. Thus each classifier’s output is it’s confidence of that the input is the given digit.
Note that the outputs of the 10 units in the final layer need NOT sum to 1. For example 1 and 7 can look very similar in some images, in that case, both unit 2 and unit 8 can give a greater than 50% confidence. If we want the output layer to give a normalized vector, then use ”softmax” activation function. In this report, the neuron in the output layer with maximum activation/confidence is selected as the label.
The simplest NN will have 1 hidden layer, an input layer and an output layer. This forms the architecture of our NN. If sufficient performance is not obtained from this, then more ”hidden” layers may be needed. However, we find that having 1 hidden layer can give accuracies 94% which is satisfactory. The structure of the NN is shown below:

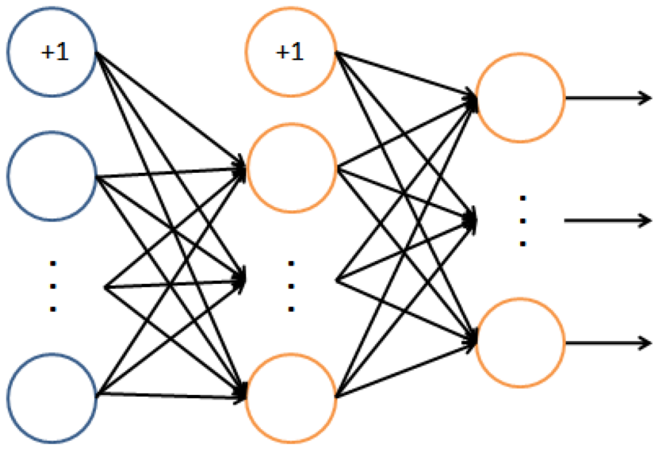

Fig. 2: Neural network layout [img source: Andrew Ng]

If there are "N" inputs, then the input layer will have $n_1 = N + 1$ units, 1 for each image pixel and the extra neuron for the bias term. Thus the 1^{st} layer has a total of 785 neural units.
The number of neurons in the hidden layer can be varied to get desired performance. If we choose to have "n_h" hidden units, then the 2^{nd} layer will have $n_2 = n_h + 1$ neurons, the extra one being for the bias term.
The 3rd layer, i.e. the output layer will have n_3 neurons, 10 in this case.
Given the above, we define the weights of linkage between adjacent layers as a matrix, W.
$W_{ij}^{(l)}$ represents the weight given by neuron i of layer l to the output of neuron j in layer $l-1$. Thus,

$$z_i^{(l)} = \sum_{j=1}^{n_{(l-1)}} W_{ij}^{(l)} u_j^{(l-1)} \tag{5}$$

$$\Rightarrow z_i^{(2)} = \sum_{j=1}^{n_{(1)}} W_{ij}^{(2)} u_j^{(1)} \tag{6}$$

$$u_i^{(2)} = g(z_i^{(2)}) \tag{7}$$

$$z_k^{(3)} = \sum_{i=1}^{n_{(2)}} W_{ki}^{(2)} u_i^{(2)} \tag{8}$$

$$u_k^{(3)} = g(z_k^{(3)});\ \textit{Output of } 3^{rd} \textit{ layer} \tag{9}$$

Where the function $g()$ is the activation function from equation 3. Our goal is to tune the Weight matrices $W^{(2)}$ and $W^{(3)}$ using back propagation and obtain a good classifier.

III. Back propagation algorithm

We measure the performance of the neural network by calculating the total mean square error (MSE) between the 3^{rd} layer and the "vectorized" labels y. Thus the error function is:

$$J(W^{(2)}, W^{(3)}) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{n_3} \left[y_k^{(i)} - \left(u_k^{(3)} \right)^{(i)} \right]^2 \tag{10}$$

where m is the number of training samples being used. Back propagation allows us to calculate the gradient of the cost function in equation 10 w.r.t the matrices $W^{(2)}$ and $W^{(3)}$.
The algorithm to calculate gradient using back propagation is as follows, for a single image sample, i :

- Calculate $\delta^{(3)} = \left[\left(u^{(3)} \right) - y \right] \odot g'(z^{(3)})$, $\odot$ being elementwise product and g'() being the sigmoid gradient from equation 4
- Obtain $\delta^{(2)}$ as, $\delta^{(l-1)} = \left(W^{(l)} \right)^T \delta^{(l)} \odot g'(z^{(l-1)})$, $l = 3$ here
- remove $\delta_1^{(l-1)}$ element, $l = 3$ here
- Calculate gradient w.r.t $W^{(3)}$ as, $\Delta^{(3)} = \delta^{(3)} * \left(u^{(3)} \right)^T$
- Calculate gradient w.r.t $W^{(2)}$ as, $\Delta^{(2)} = \delta^{(2)} * \left(u^{(2)} \right)^T$

A. *Stochastic Gradient descent*

There are two options for gradient descent here. The classic gradient descent for cost function in equation 10 is the cumulative sum of $\Delta^{(l)}$ over all images, m, divided by m, i.e.

$$\frac{\partial J}{\partial W^{(l)}} = \Delta_{cum}^{(l)} = \frac{1}{m} \sum_{i=1}^{m} \left(\Delta^{(l)} \right)^{(i)} \tag{11}$$

Thus the gradient descent step for all weight matrices would be:

$$W^{(l)} = W^{(l)} - \eta \Delta_{cum}^{(l)} \tag{12}$$

where, η above is the learning rate and can be made different for each layer. However, the classic gradient step can be slow as we accumulate the gradient for the same W's and correct it only at the end of traversing the training samples. This can take a long time to converge, especially if the training data set is very large.
To accelerate gradient descent, we use the stochastic gradient descent, where we update W after each training image sample, i.e.

$$W^{(l)} = W^{(l)} - \eta \left(\Delta^{(l)}\right)^{(i)},\ i : 1 \rightarrow m \tag{13}$$

Here W is updated as we go through different images. Thus at the end of m samples, we would have completed a single "pass" over the training data. Multiple "passes" are done through the training data until the cost function is minimized.

IV. Results and Discussion

The back propagation algorithm with stochastic gradient descent was implemented in python. The MSE was minimized over all 60,000 images of the MNIST dataset. The prediction accuracy was calculated over the entire test data set of 10,000 images. For a hidden layer with 15 neurons, the below plots show the behaviour of the different errors:

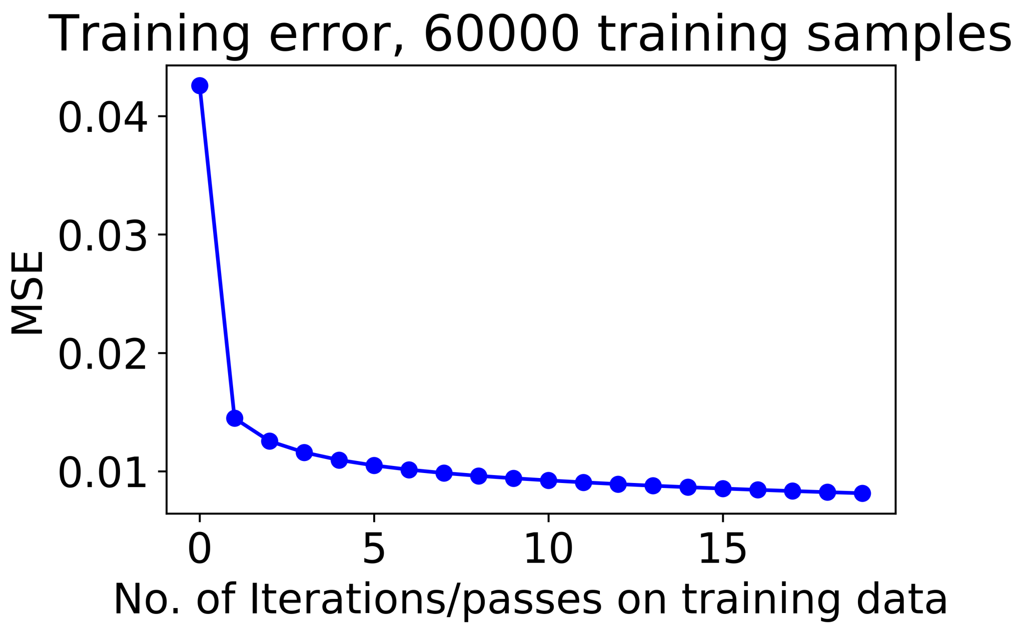

Fig. 3: Training error vs number of passes/iterations

As expected, the gradient descent reduces the MSE cost for each pass over the training data. The training error asymptotically converges to 0. However, the increase in test accuracy slows down as the number of passes increases. In fact, after a large number of passes, the test accuracy will actually *decrease* because the network "overfits" the training data. To prevent this, training was stopped if error fell below a threshold or after a fixed number of iterations. The plot below illustrates this point:

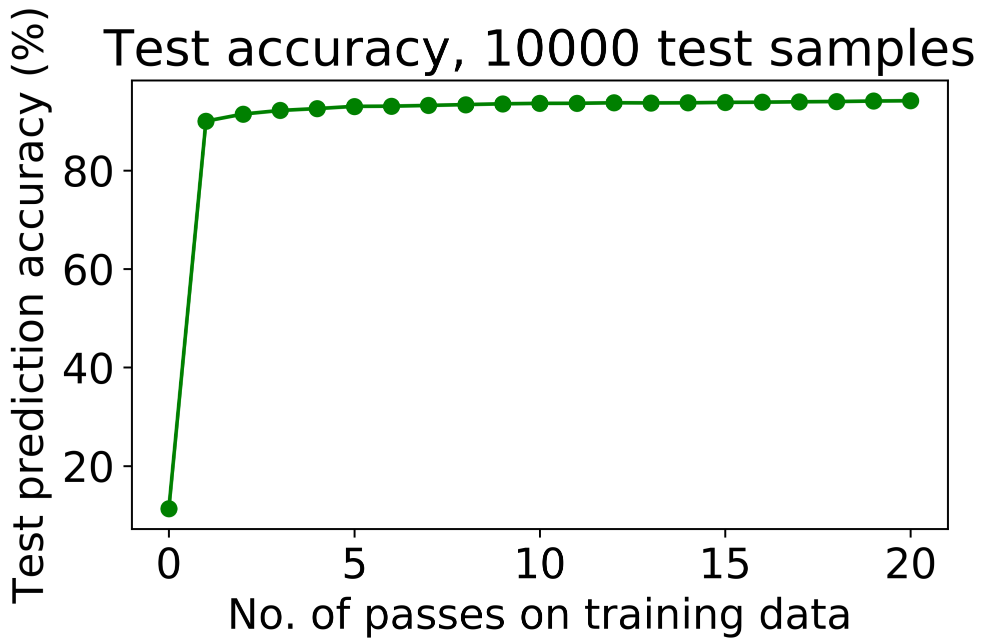

Fig. 4: Prediction accuracy for the test data set

We also study the accuracy of the network as we increase the number of neurons in the hidden layer. Figure 5 plots the final network accuracy, after the back propagation algorithm had converged, with number of neurons.

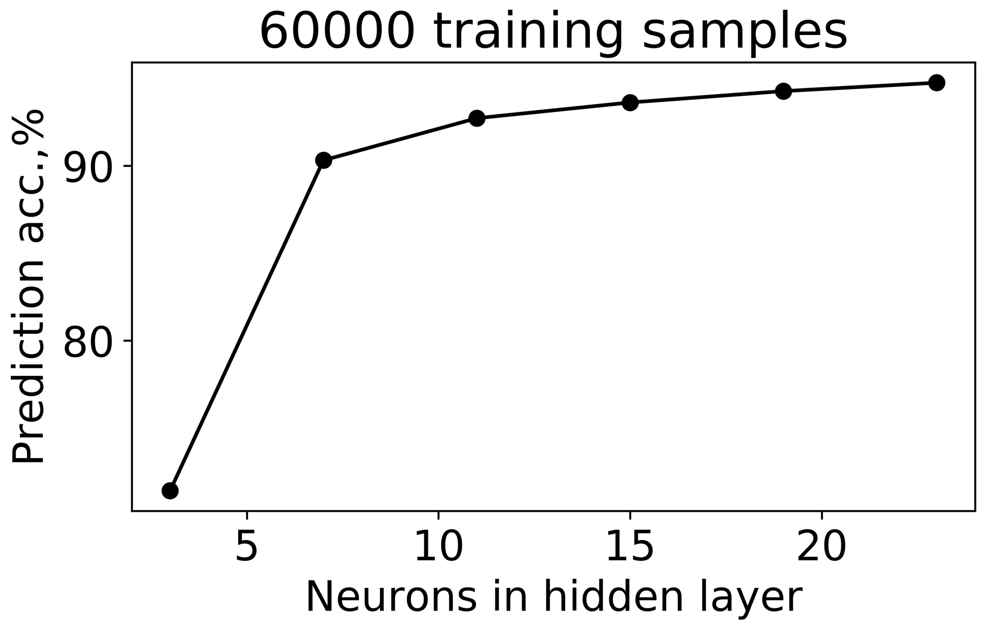

Fig. 5: Prediction accuracy vs number of neurons in hidden layer

Notice that beyond a point we do not get a significant improvement in the prediction accuracy by adding more neurons. The trade off with increasing the number of neurons is that the computational time increases. The best performance was found with 23 neurons giving a prediction accuracy of 95% on the entire test data.

V. Conclusion

In conclusion, a neural network classifier trained with back propagation and stochastic gradient descent was implemented. If we notice that increasing the number of neurons or increasing training accuracy is leading to a decrease in the overall test accuracy, then the model is over-fitting the training data. So we must reduce the number of neurons or stop further training. A large network makes the model more accurate but results in a trade off with computational time The highest classifier accuracy of 95% was observed with hidden neurons = 23 over training data size = 60000.

VI. References

1 http://neuralnetworksanddeeplearning.com/chap1.html

2 Introduction to Machine Learning, Andrew Ng, Coursera