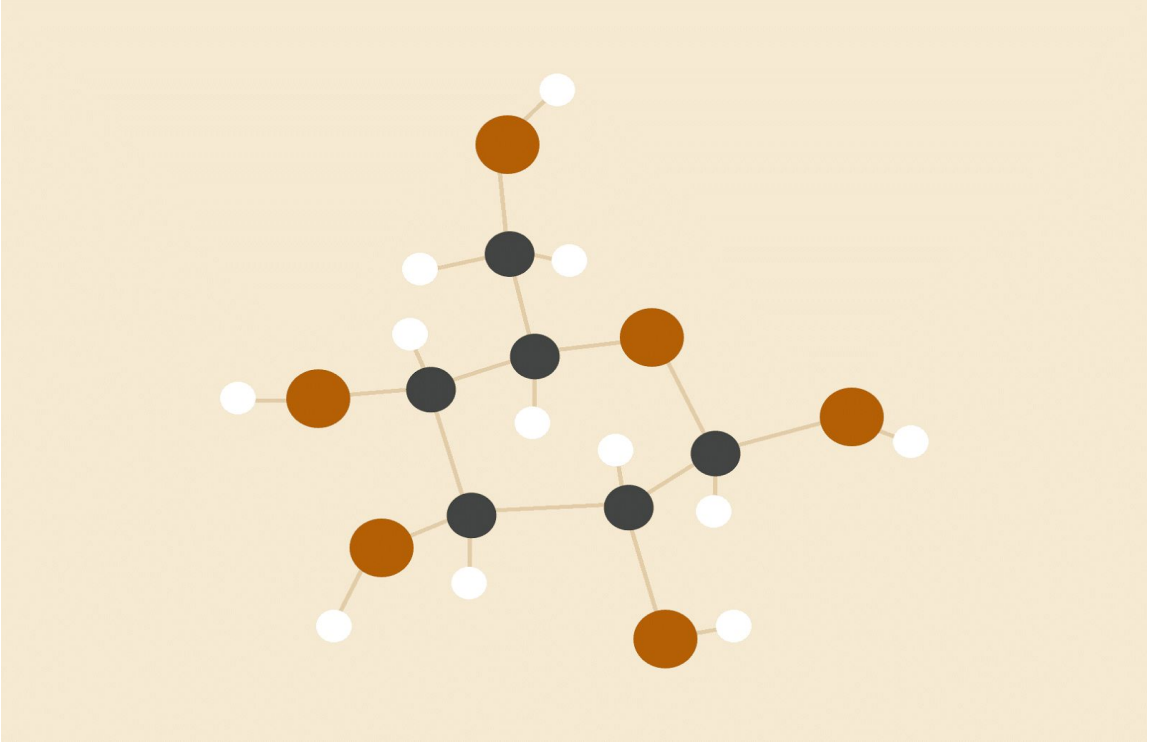
Project 3 Report

MNIST handwritten digit Classification



Adhip Vihan

Person Number: 50134774

08.12.2015
CSE-574
Introduction to Machine Learning

Mr. Sargur N. Srihari

INTRODUCTION

This project involved implementing and evaluating the classification algorithms. The classification task will be to recognize a 28*28 grayscale handwritten digit image and identify it as a digit among 0, 1, 2, … , 9. We were required to do the following three task:

1. Implementing logistic regression, training it on the MNIST digit images and tuning the hyperparameters,
2. Implementing single hidden layer neural network, training it on the MNIST digit images and tuning hyperparameters such as the number of units in the hidden layer.
3. Using a publicly available convolutional neural network package, training it on the MNIST digit images and tuning hyperparameters.

DataSet Used

We used MNIST dataset for this project. The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems.The database is also widely used for training and testing in the field of machine learning.

Algorithms Used

We used two algorithms mainly for the project:

1. Logistic Regression using the Stochastic Gradient Descent.
2. Neural Networks with backpropagation using the stochastic Gradient Descent.

We were provided with an option to choose the ***Mini-Batch Gradient Descent***. Mini-Batch is an efficient method wherein we keep adding the gradient error as we pass through the batch size of roughly 50-100 datasets at a time. And after each batch we update the weights using the learning rate and the gradient error so far.

I have used Stochastic Gradient Descent for the Project since i compared the performance of the two and found that although the Mini-Batch is a faster method, My error was converging faster if i used the stochastic gradient descent.

PROCEDURE

Multiclass Logistic Regression

We used one-of-k coding scheme where the value of K was 10. Since our image can belong to any of the 10 digits starting from zero.

Multiclass Model for logistic regression is represented by the equation:

$$p(\mathcal{C}_k|\mathbf{x}) = y_k(\mathbf{x}) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

Here the term on the rightmost side gives us the probability of the image belonging to one of the 10 classes.

We have used the softmax function here. It is the function which takes in the values

$$a_k = \mathbf{w}_k^\top \mathbf{x} + b_k.$$

where $\mathbf{w}_k^\top \mathbf{x}$ is the weights assigned to each of the features and X is the feature vector.

In our case the feature vector was 28*28=784 which had values for pixel intensities.

$b_k.$ is the bias term which is a 1x10 vector of Ones.

Since I used stochastic gradient descent, I iterated from the 1 row to the 60000th row.

And after pass through each row the cross entropy function was calculated by

$$E(\mathbf{x}) = -\sum_{k=1}^{K} t_k \ln y_k$$

Where the $\boldsymbol{t}$ is the target vector of size 1x10 containing 1 at the place which defines which class X belongs to.

It is implemented by the following code in MATLAB

```
for i=1:size(phi_total,1)
    j=labelTrain(i,1);
    T(i,j+1)=1;
end
```

Then we have the gradient of error calculated by:

$$\nabla_{\mathbf{w}_j} E(\mathbf{x}) = (y_j - t_j)\,\mathbf{x}$$

This error is used to update the weights after passing over each row, with the help of equation given by:

$$\mathbf{w}_j^{t+1} = \mathbf{w}_j^t - \eta \nabla_{\mathbf{w}_j} E(\mathbf{x})$$

The corresponding MATLAB implementation is given by

```
delta_E=((phi_total(i,:)'*(y(i,:)-T(i,:))));
w_old=w_old-(eta.*delta_E);
error=-sum(sum(T(i,:).*log(y(i,:))));
```

I have used the ***counter*** to check how many times the error reduce while iterating over the rows. If the counter decreases 3 time in continuation then I have increased the learning rate by ***0.005***.

I have kept my initial learning rate as ***0.001***.

Corresponding MATLAB code is given as:

```
if iteration_count>3
   eta=eta+0.005;
   iteration_count=0;
```

The error was converging good with this algorithm and for the final result i was able to reduce the error on the training data to around ***9%.***

Single Layer Neural Network

We have used Single Layer Neural Networks in MATLAB and did the back-propagation.

To implement the neural network the first major thing that we were asked to do was choose the number of Hidden Nodes. the best performance of the neural network i found out for me was with ***300 Hidden nodes***.

So the initial weights were taken as ***785x300*** for the first layer since there are 300 perceptrons in the first layer.

The number of hidden nodes/units. Choosing the number of hidden units is important as they impact the output.Using a very small number of Hidden nodes can cause Underfitting and hence training error may not decrease to desired values. This also means, testing error will be high. on the other hand, having a very large number of Hidden nodes can cause Overfitting where the model is tuned to training set and Error on testing set will be high. Having a large number of Hidden nodes also means higher complexity and higher training time. Hence the program will take significantly longer to train the model. It is generally believed that, M=2/3Number of Features, is a good value for Hidden Units. There

are two other thumb rules for choosing M as follows,the number of hidden nodes must be between number of input features and output classes. The number of hidden nodes must be less than twice the size of input layer. But i chose 300 still to reduce the overfitting.

Now the weights for the final output were chosen to the dimensions of ***300*10***.

The problem that i faced in neural network was that my weights were not getting closer to optimized. I checked my code thoroughly and found out i had no errors but the weights were still not changing after the final iteration.

Turned out to and an interesting find that there is no use of taking initial weights as ***ONES*** since neural network is stuck in the back propagation and the weights are never closed to optimized one.

I solved this problem by initializing all the weights as random and subtracting 0.5 so that they remain as min as possible.

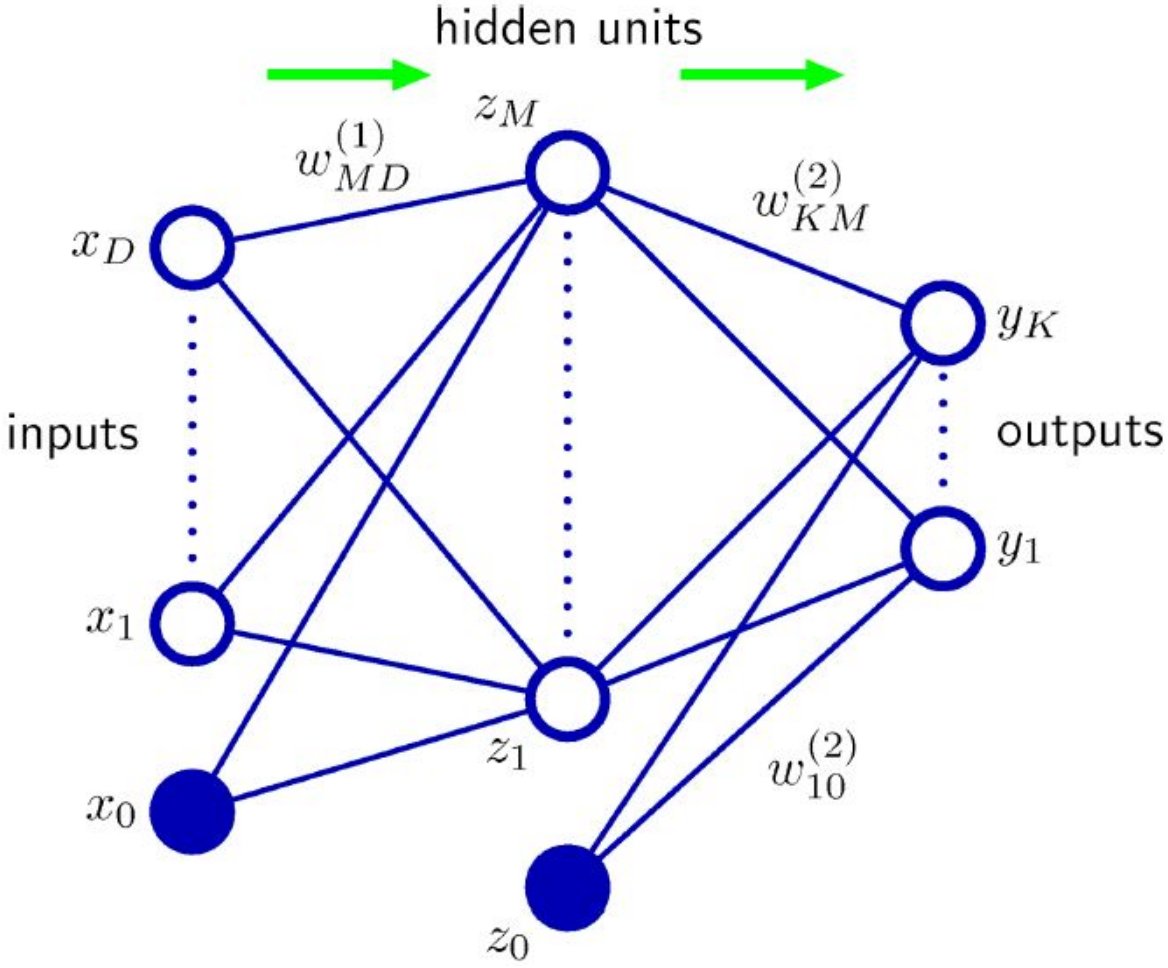
Now for choosing the learning rate ***ETA.***

A low step size means low learning rate, and hence the network learns very slowly. A high step size means a very high learning rate, which causes values of the weights to change drastically, this does not lead to efficient learning of weights. If we keep a xed step size, then it required a large amount of trial and error to nd a perfect or optimum step size.Changing the step size based on the Error gradient is a commonly used approach but it has certain drawbacks. In different cases, we may need large or small step size for small gradient and similarly for large gradient, a small or sometimes large step size may be required. We need a small step size for small gradient near local minima. However, when we initialize weights to small random values, we may need a large step size for small gradient.

So i chose the initial learning rate as ***0.0001.***

I implemented a counter for the number of times the error has gone down, whenever the counter reached 5 i have increased the learning rate by ***0.0005***

Also for the final output i have chosen the error rate as ***0.0000001*** and increased it by ***0.000005*** every time the error goes down 5 times consecutively.



To implement gradient descent we need to do the back propagation

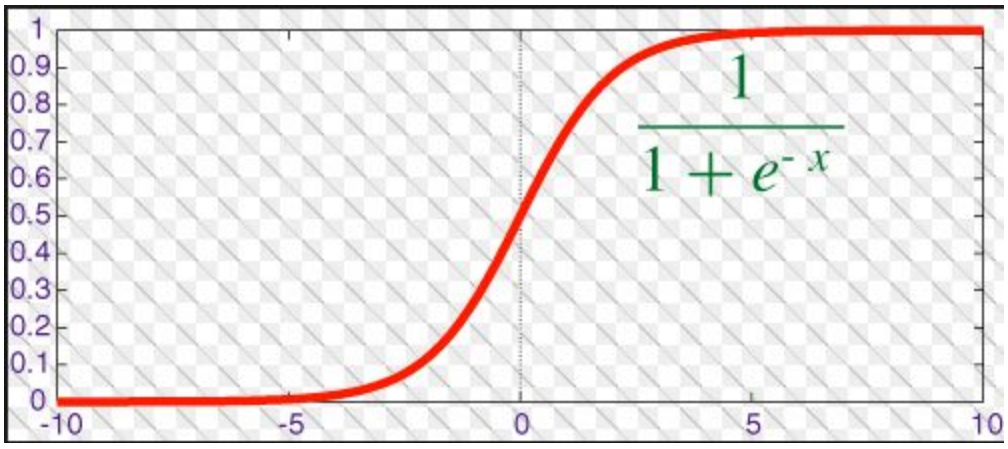
Neural Network can be summed with the equation

$$y_k(x,w) = f\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)})\right)$$

$$z_j = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}\right)$$

where

for the ***h*** i have used the sigmoidal function which is given by:



Here the a^k is given by

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + b_k^{(2)}$$

And the probability ***y*** is given by

$$y_k = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

This is the Soft-Max function used for Multi Class Classification where yk() is the Posterior Probability and where j = 1….K and K is the number of outputs. It is the output predicted by our model based on the weight matrix.

There is a cross-Entropy error function which is used to calculate the error. We continue Gradient descent until error reaches an acceptable value or number of iterations exceeds maximum. It is given by

$$E(w_1 \dots w_k) = -\ln p(T|w_1 \dots w_k) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \ln y_{nk}$$

In Neural Networks, in order to perform gradient descent to minimize error, we perform the BackPropagation Algorithm. Backpropagation is more efficient for nding minima but even this algorithm does not guarantee a global minima. In order to perform back-propagation we must calculate the following values.

$$\delta_k = y_k - t_k$$

then we backpropagate to calculate deltas for hidden units.

$$\delta_j = z_j(1 - z_j)\sum_{k=1}^{K} w_{kj}\delta_k$$

We then calculate the derivatives w.r.t. the rst layer and second layer weights as follows:

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j x_i$$

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k z_j$$

To minimize the error and optimize the weights, we use the gradient descent formula with the derivatives calculated above to calculate new weights $\boldsymbol{w_{ji}}$ ***and*** $\boldsymbol{w_{jk}}$.

$$w^{(\tau+1)} = w^{(\tau)} - \eta \bigtriangledown E(w^{(\tau)})$$

I have used the stochastic gradient descent and used the loop to iterate over the dataset until the error reaches 0.5.

Also I am breaking the loop if my errors reach Infinity to prevent the weights from going NaN.

It is implemented by in MATLAB as:

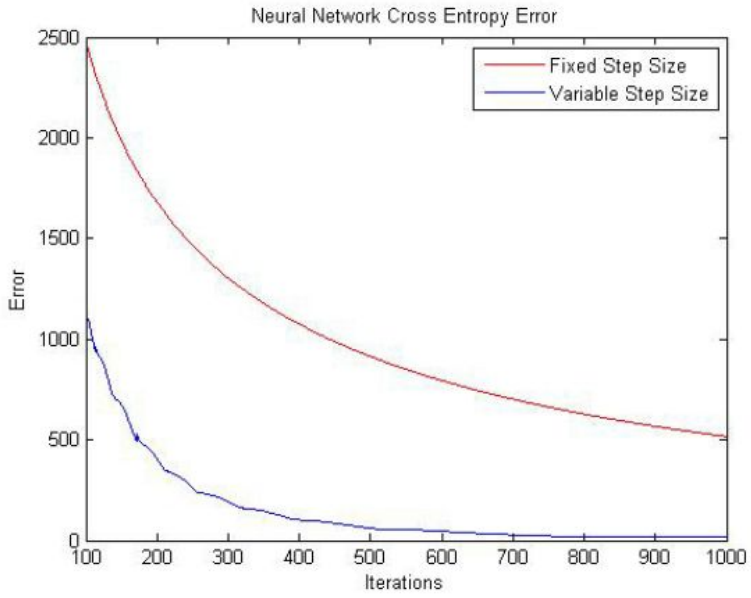
```
while((old_error>0.5&&counter<30000))

if(isnan(error))
break
end
```

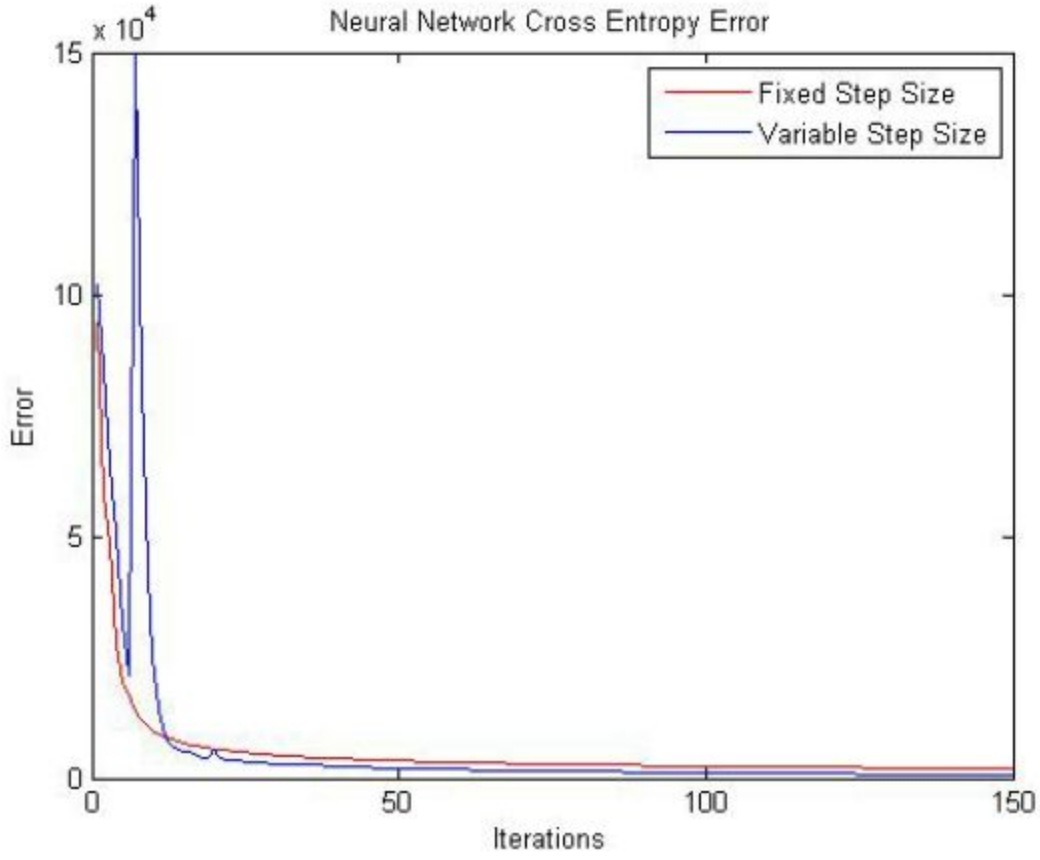
With the help of above all techniques my error finally reached ***0.5*** after around 25000 iterations when it broke out of loop.

And the final result were ***0.065%*** error on the Training set and ***~2.8*** on the testing set.

The graph for the case is given by:


Neural Network Cross Entropy Error
Fixed Step Size
Variable Step Size
Error
Iterations
2500
2000
1500
1000
500
0
100
200
300
400
500
600
700
800
900
1000

Also by


Neural Network Cross Entropy Error
x 10^4
Fixed Step Size
Variable Step Size
Error
Iterations
15
10
5
0
0
50
100
150

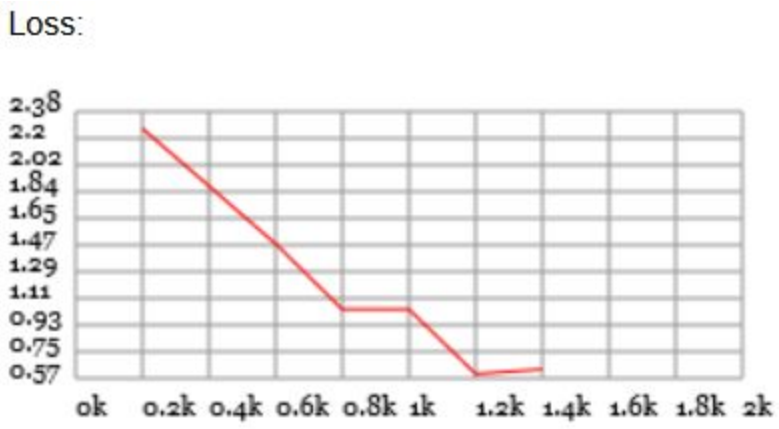
Convolutional Neural Networks

Convolutional Neural Networks are are a special kind of multi-layer neural networks. Like almost every other neural networks they are trained with a version of the backpropagation algorithm. Where they differ is in the architecture.Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.They can recognize patterns with extreme variability (such as handwritten characters), and with robustness to distortions and simple geometric transformations.

We were given Deep-Learning Matlab toolbox which we used to try out the convolutional neural network.

i added it to the MATLAB path and used it. Here are some screenshots of the error rate. I also used the online stanford MNIST demo for visualizing the weights.

It gave out the following results:



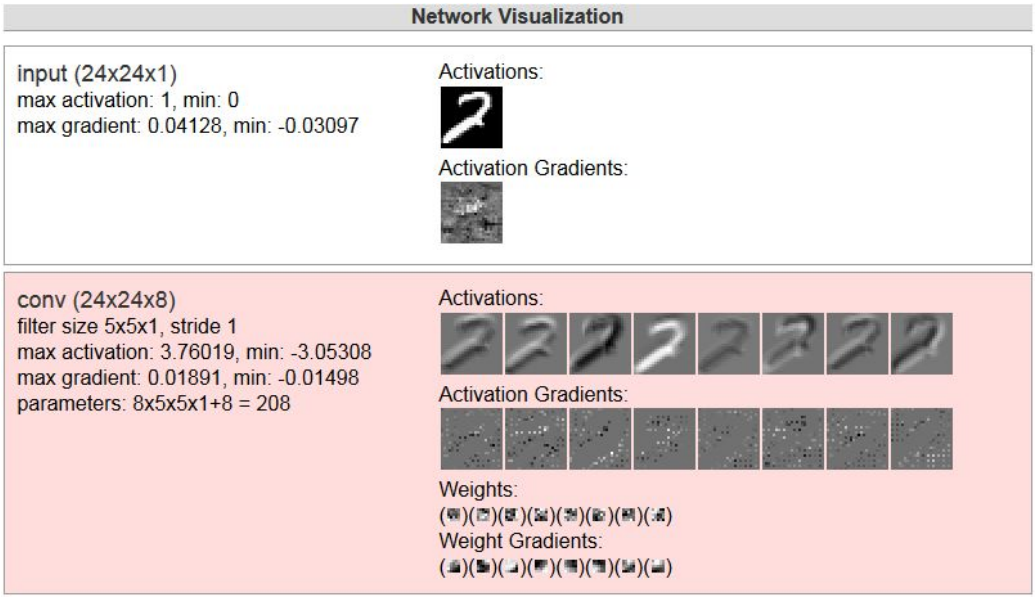
for the values set as

Batch Size = ***20***

Learning rate=***0.01***

Weight decay=***0.001***

Also the network visualisation in the process was:



Predictions on Test Set

CONCLUSION

We Observe from the above results that Neural network has a smaller Error, Error Rate and the Reciprocal Rank of both are same. Hence Neural Network performs better than Logistic Regression. It also clearly show that Neural Network finds a minimum faster than Logistic Regression. Neural Networks Backpropagation algorithm is one of the reasons for faster performance. However, If speed of each Iteration is taken into account, Logistic Regression is faster to perform one iteration.

REFERENCES

1. http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html
2. http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html