Neural Network for Handwritten Digits Classification

Suraj Kota, University of Florida

***Abstract*— A neural network based classifier for classification of handwritten digits is designed and the design decisions for the system are studied. The project uses a single layer neural network with Stochastic Gradient Descent(SGD), SGD momentum, batch GD and mini batch GD training to classify.**

***Index Terms*— ReLU, Momentum, Sigmoid, Classification, gradient descent**

I. Introduction

A human brain which is a super computer in itself tries to make sense out of what the eye sees using its very own inbuilt neural system. However, the task of recreating this neural system is a mammoth task to say the least, but certainly not impossible. the concept of neural networks is often used by artificial agents to perceive things around them. One of the functions of the neural network is optical recognition (generalization of handwritten digit recognition). the agent is fed an input dataset of things it is trying to perceive. this is often referred to as the training set.

In our case the expected output is for the recognition of handwritten digits and hence the training set composes mainly of the images of the numerals. After the training phase is completed, another dataset is used to test the model (often referred to as test set) and based on what the system has learnt from the previous phase, it predicts a value for the digit. The accuracy of the model is optimized by continuously training the system with different sets of dissimilar data (so that the system can get to know some of the caveats and exceptional cases) until it reaches a saturation point.

The rest of the paper is organized as follows: Section II gives a brief introduction to Neural Networks and describes the methodology followed to train the network. Section III presents the experimental results and evaluates the design decision best suited for this problem with this dataset. Finally, Section IV concludes with our findings and recommending future research directions.

This work is submitted on 12th December 2016 to fulfill the coursework requirement for Elements of Machine Intelligence course.

Suraj Kota is with the University of Florida, Gainesville, FL 32611 USA (e-mail: suraj.k@ufl.edu).

II. Using Neural Nets to Recognize Handwritten Digits

The power of Artificial Neural Networks (ANNs) lies in creating arbitrary discriminant functions using many different processing elements (PEs) in its topology that help in achieving optimal classification. The topology of ANN is defined by the interconnectivity between PEs. The number and shape of discriminant functions are the resultant of the ANN topology which is subject to change with topology as ANN defines its discriminant functions from its PEs. Each PE in the network receives connections from itself and/or other PEs and the signals flowing on the connections are scaled by adjustable parameters called weights. The PEs sum all these contributions and produce an output that is a nonlinear (static) function of the sum [1]. The PEs' outputs become either system outputs or are sent to the same or other PEs. The activation function determines the output of the unit. There are many activation functions with different properties. The training method using backpropagation used in this paper requires the activation function to be differentiable. Also, as the output of our network(classify as a digit) cannot be reproduced from a linear combination of inputs(pixels of image) we must use a non linear activation function in our neural network. Without a non-linear activation function in the network, a ANN even with multiple layers would behave like a single-layer perceptron because summing these layers would give you just another linear function. We examine the performance of the network subject to two activation functions: Sigmoid and ReLU.

i. Sigmoid function is a smoothed step function that produces approximate 0 for negative input with large absolute values and approximate 1 for large positive inputs,

$$\sigma(z) = \frac{1}{1 + \exp(-z)|} \tag{1}$$

and its derivative is given by,

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z)) \tag{2}$$

ii. ReLU's can "die" if a large enough gradient changes the weights such that the neuron never activates on new data [2]. So we use Leaky ReLU given by,

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases} \quad (3)$$

and its derivative is given by,

$$\frac{df}{dx} = \begin{cases} 1 & x > 0 \\ 0.01 & x \leq 0 \end{cases} \quad (4)$$

A. *Training the network*

In this paper we train a single hidden layer perceptron(Figure 1) on MNIST database, which is a database of 10 handwritten digits. The MNIST dataset (28x28) contains 60k images for training and 10k images for testing. The last 10k images in the training dataset are as validation dataset for cross-validation. We'll use the test data to evaluate how well our neural network has learned to recognize digits.

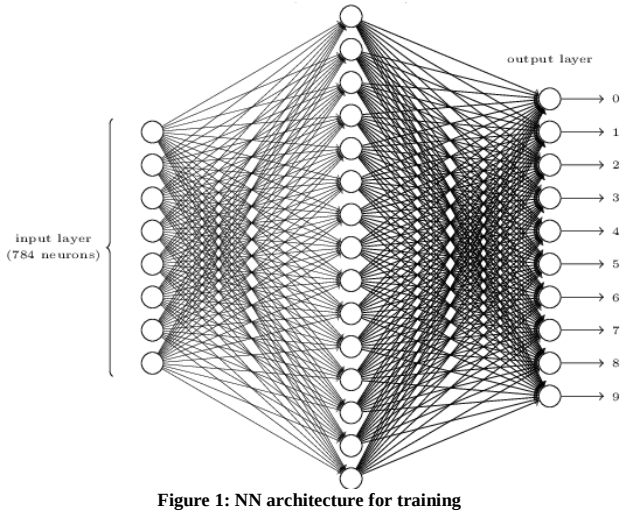


Figure 1: NN architecture for training

To train the network,

i. We first use the loadMNISTImages script to convert the 28x28 image to a 784x1 dimensional vector. Each entry in the vector represents the grey value for a single pixel in the image. We read the corresponding output labels using loadMNISTlabels.m script and convert it each desired output into a 10 dimensional vector, for example, a particular training image represents 2, the corresponding target vector is T=[0,0,1,0,0,0,0,0,0,0].

ii. The 50,000 784x1 images data is presented one by one and an output is obtained from the outermost layer. As we can see figure 1. as this is a classification problem the number of output units we choose is equal to 10 which is enough to compensate the number of classes in data.

iii. An error is obtained by computing the difference between the output and the target response which is used to modify the weights [1]. To quantify how well we're achieving this goal we need a cost function. We use Mean Square error(MSE) as our cost function which is given by,

$$J = \frac{1}{2N}\sum_p \varepsilon_p{}^2 = \frac{1}{2N}\sum_p \left(d_p - y_p\right)^2 \quad (5)$$

The weights are adjusted using the well known backpropagation algorithm.

iv. This procedure is repeated for several epochs, where each epoch consists of presenting all 50,000 images and updating weights in online fashion.

This method of training the network is called Stochastic Gradient Descent(SGD) where we repeatedly iterate through the training set, and each time we find a error, we update the weights of network according to the gradient of the error with respect to that single training example only [3].

The difficult part in NN design is the selection of number and place of the PEs (number of hidden layers & units in these)and other hyper parameters involved like learning rate, choice of activation function, number of epochs required for training or training methodology(batch or incremental). So, in rest of this paper we present how choice of these hyper parameters affect the performance of the network and base our analysis by concentrating on minimizing the cost function using Gradient descent, the computation power needed and performance accuracy of the network with respect to the choices made.

III. Experimental Results and Analysis

As we progress through this section we will build confidence on our design decision of the neural network to improve the recognition of hand written digits.

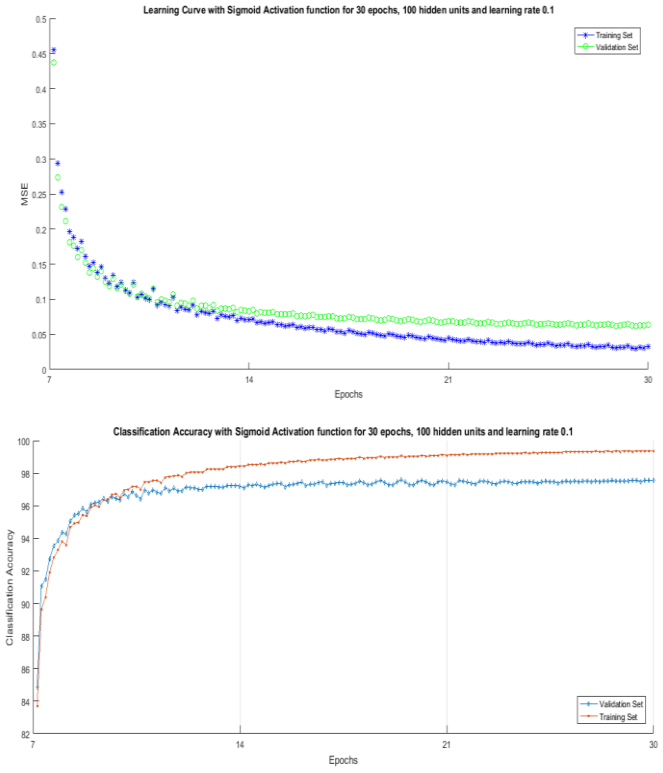
A. *Choice of Activation Function*

In this subsection, we analyze the performance of network with both Sigmoid and Leaky ReLU as activation function. Figure 2 and 3 present the learning curves for training set and validation set MSE over the epochs(each epoch consists of 50k samples), the values are captured after every 10k samples presented.

The graph above the learning curve is for the classification accuracy for training and validation set captured at the same time as the MSE. The third figure is the confusion matrix for 10 digits where A*x*= sample is *x* in Actual and P*x*= system Predicted *x*. Also, HU denotes hidden units, LR denotes learning rate and E denotes epochs. This notation is followed for all figures throughout the paper. Figure 2 is for Sigmoid activation whereas Figure 3 is for ReLU activation function.

It is clear from the validation set learning curves that both using either of the function we can train the network for this problem and also works well on test set, which can be seen from classification accuracy graphs and confusion matrix. The learning curves for both training and validation set decrease with epochs because as we present more samples to the network, the network automatically infers rules for recognizing handwritten digits. Furthermore, by increasing the number of training examples, the network can learn more about handwriting, and so improve its accuracy which is reflected in the classification accuracy graph. After a certain point of time, only training error decreases and only training set classification accuracy increases, this is because beyond

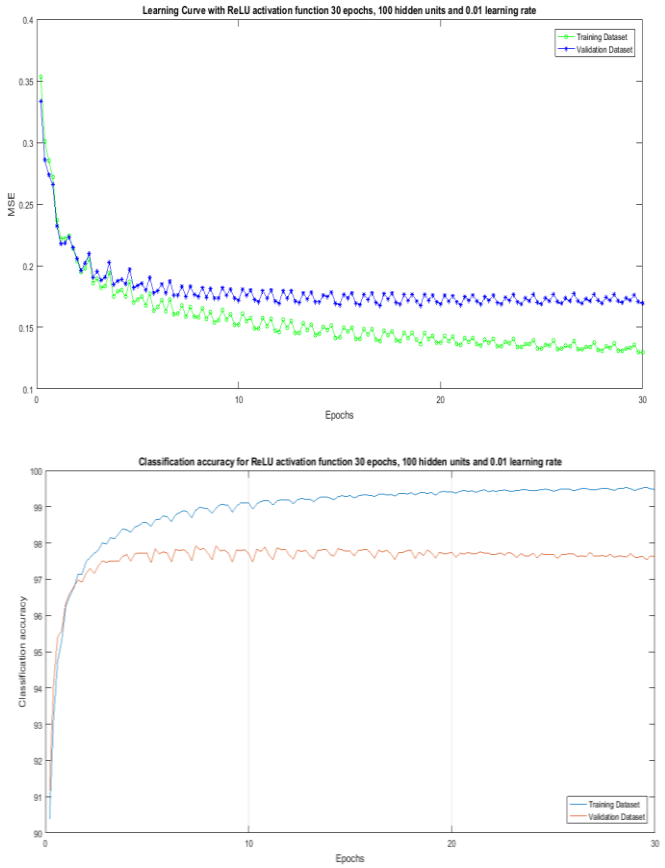
that point, the network tends to overfit the training dataset and there is no improvement for the validation or test dataset. This is the point where learning should stop to improve the generalization performance of the network. But, of both activation functions Sigmoid activation function takes a longer time(more than 20 epochs) to converge(validation set error does not show any improvement and learning should be stopped) than ReLU(almost 10) but achieves a very low MSE of order 0.06, on the other side ReLU stops at 0.16. The bright side of ReLU is inspite of it achieved classification accuracy of 97.9% on test set as opposed to 97.5%. Digit 8 which requires to draw the most number of decision boundaries and maybe called as tougher one to recognize, ReLU performs better than Sigmoid with 97.67%(figure 3-3).



	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Accuracy
A1	964	0	1	1	0	3	8	1	2	0	98.37%
A2	0	1120	4	2	0	1	2	0	6	0	98.68%
A3	0	3	1011	3	0	2	3	6	4	0	97.97%
A4	0	0	3	995	0	1	1	1	6	3	98.51%
A5	1	0	1	0	963	0	7	1	0	9	98.07%
A6	2	1	1	16	1	857	5	0	5	4	96.08%
A7	3	2	4	1	2	2	942	0	2	0	98.33%
A8	0	6	16	5	2	0	0	986	1	12	95.91%
A9	0	0	4	4	5	3	5	3	949	1	97.43%
A10	3	3	2	10	15	5	1	6	1	963	95.44%

Figure 2: Learning Rate, Classification Accuracy and Confusion matrix for Sigmoid Activation function [100HU,30E,0.1LR]

Multiple runs with different hidden units and learning rate have shown that almost same performance with slight difference but ReLU is easy to train in less number of epochs as compared to Sigmoid. One major benefit with ReLU is the reduced likelihood of the gradient to vanish. When *net>0*, the gradient has a constant value in contrast to the gradient of sigmoid function which becomes increasingly small as the absolute value of *net* increases. The constant gradient of ReLUs results in faster learning. So, in rest of the paper we use ReLU as activation function to carry out experiments and evaluate our neural network performance.



	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Accuracy
A1	972	1	0	0	1	0	2	2	2	0	99.18%
A2	0	1129	2	2	0	0	0	0	2	0	99.47%
A3	4	2	997	6	2	0	4	6	11	0	96.61%
A4	0	0	5	985	0	5	0	4	8	3	97.52%
A5	0	1	1	0	965	0	3	1	0	11	98.27%
A6	2	1	1	7	0	868	5	1	5	2	97.31%
A7	5	2	0	0	2	4	944	0	1	0	98.54%
A8	2	4	8	2	1	0	0	1004	5	2	97.67%
A9	3	0	2	4	3	1	3	3	951	4	97.64%
A10	2	5	0	3	10	1	0	7	6	975	96.63%

Figure 3: Learning Rate, Classification Accuracy and Confusion matrix for ReLU Activation function [100HU,30E,0.1LR]

B. Choice of hidden units, learning rate and epochs

The choice of these hyper parameters are very crucial in deciding the NN performance. Let us see how,

- *Choice of Learning Rate:* A slower learning rate is always better as it helps reach the minimum error as possible, but the training may take a very long time and if we do not do proper cross validation we may tend to overfit the data. In case of a larger learning rate, the problem of rattling occurs and reaching lowest possible minimum is not possible unless we change the learning rate after a few epochs. Figure 4 shows the learning curves for testing and validation set when ReLU activation is subject to 0.1 learning rate and suffers from rattling. The classification accuracy on test dataset was 64% after training. A learning rate of 0.01 was found accurate for network with 100 hidden units in one layer and 10 epochs when using SGD momentum.

- *Choice of Epochs:* The choice of epochs is crucial because more than required number of epochs often result in overfitting the training data. Problem of overfitting can be seen when the training continues even after the validation error has reached a minima and again starts to increase. As we can see from figure 2 and 3, and will also be clear from further learning curves that, this type of activity does not significantly occur with our network and error for validation set remains constant after a certain time and error keeps on decreasing for training set as it starts to overfit the training dataset. So, It is important that the learning should be stopped at the point where validation set error does not show any improvement as it helps in maintaining good generalization accuracy. But, if less number of epochs are used, the system may have not reached the minimal value and it would still have more scope to learn. This phenomenon is showed in figure 5 when only 10 epochs are used to train the network. We can see that even at the end of training the network is still learning and the validation set curve has scope for reaching a further minimum. Choice of epochs changes as the hyper parameter of the network changes, for the network with 100 HUs 20 epochs was found as a optimal number.

- *Choice of hidden units/PEs in hidden layer:* In order to power the network we have used one hidden layer with multiple PEs, so that each PEs can draw its own linear discriminant function in D dimensional space. More hidden units give better approximation which became evident when comparing a network with 300 hidden units with sigmoid activation function and 0.1 learning rate which gave 98.11% accuracy as opposed to 97.5% with the network having 100 hidden units(figure 2). Figure 6 shows the confusion matrix for the 300 HU Sigmoid network. Same was the behaviour with a 50 unit ReLU network with 96.84% accuracy as compared to 97.9% with 100 unit network as shown in figure 3. Figure 7 shoes the confusion matrix of test dataset for 50 HU ReLU network. Upon several experiments a 80 HUs was found as a optimal number with momentum being applied to the weight updates.

  Literature suggests that instead of having large number of units in one layer we should increase the number of layers which also makes easy for network to learn. Due to the number of hyperparameter choices and training time/computation power involved, only 1 hidden layer NN is studied in this paper.

C. Effects of Training with momentum(SGD momentum)

The performance surface describes how the error changes with the weights. But the performance depends on the topology of the network through the output error. So, when nonlinear PE are utilized to solve a given problem the relationship "performance - weights" becomes nonlinear and there is no guarantee of a single minimum [1]. The performance surface may have *several minima* i.e. the performance surface is non convex.

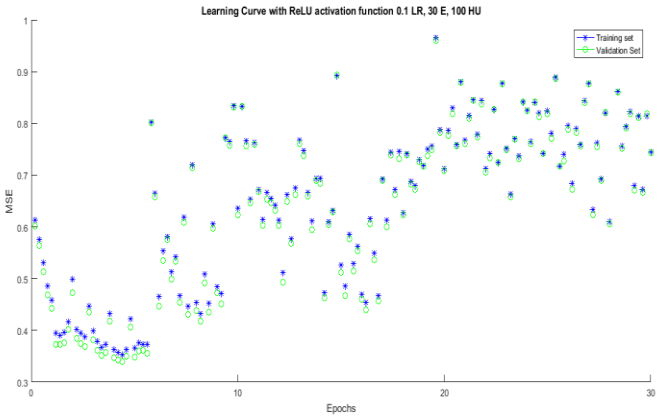


Figure 4: Rattling with higher learning rate

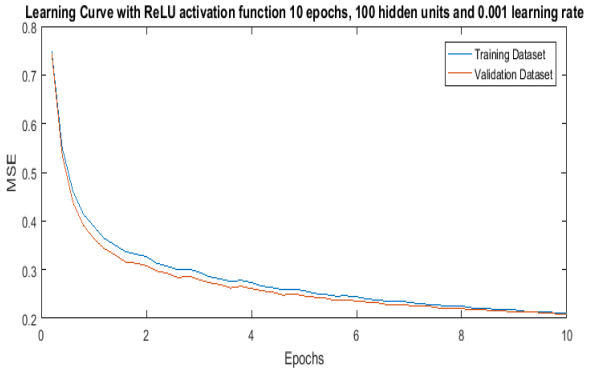


Figure 5: Low Epochs network not converged

This impacts the search scheme, because gradient descent uses local information to search the performance surface. In the immediate neighborhood, local minima are indistinguishable from the global minimum [1]. So the gradient search algorithm may be caught in these sub-optimal performance points "thinking" it has reached the global minimum.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Accuracy
A1	971	0	0	0	1	2	1	1	2	2	99.08%
A2	0	1124	3	3	0	1	1	1	2	0	99.03%
A3	2	2	1011	4	1	0	1	6	5	0	97.97%
A4	0	0	3	996	0	1	0	1	5	4	98.61%
A5	2	0	1	1	963	0	5	1	0	9	98.07%
A6	3	0	0	7	0	870	6	1	2	3	97.53%
A7	5	2	0	1	1	3	944	0	2	0	98.54%
A8	0	4	6	4	1	0	0	1006	2	5	97.86%
A9	1	0	4	5	4	3	0	3	952	2	97.74%
A10	3	2	1	6	13	1	1	6	2	974	96.53%

Figure 6: Confusion matrix for test data set for a 300 HU Sigmoid network with learning rate 0.1 and 50 epochs

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Accuracy
A1	970	1	0	2	0	1	3	2	1	0	98.98%
A2	0	1127	1	2	0	0	1	0	4	0	99.30%
A3	2	6	994	4	5	1	6	10	4	0	96.32%
A4	1	1	4	978	0	7	0	6	12	1	96.83%
A5	1	2	4	0	950	0	6	2	0	17	96.74%
A6	4	2	0	13	2	845	8	2	9	7	94.73%
A7	9	3	0	1	5	6	933	0	1	0	97.39%
A8	1	6	11	2	1	0	3	999	1	4	97.18%
A9	5	3	2	6	3	3	7	16	923	6	94.76%
A10	1	4	0	5	16	4	0	9	5	965	95.64%

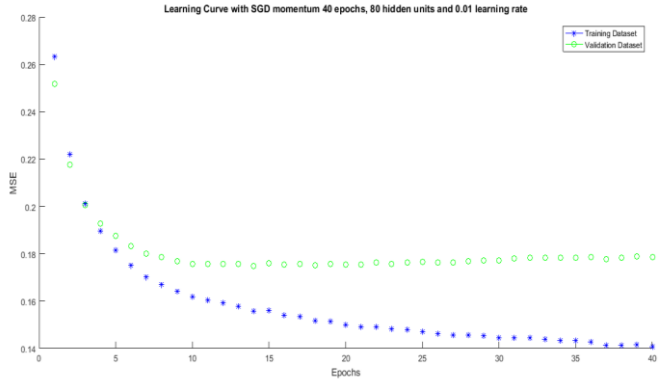
Figure 7: Confusion matrix for test data set for a 50 HU ReLU network with learning rate 0.01 and 50 epochs

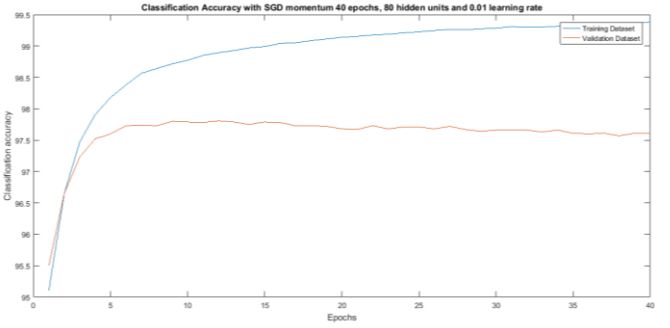
In SGD, Since the gradient for a single data point can be considered a noisy approximation to the overall gradient, this is also called stochastic (noisy) gradient descent and the noise

in the gradient can help to escape from local minima [4]. Another technique that can help the network out of local minima is the use of a momentum term and the equation for back propagation is changed to,

$$\Delta w_{ij}(t) = \mu_i \delta_i y_j + m \Delta w_{ij}(t-1) \quad (6)$$

where 0 < m < 1 is a new global parameter which is determined by trial and error. Momentum simply adds a fraction m of the previous weight update to the current one. When the gradient keeps pointing in the same direction, this will increase the size of the steps taken towards the minimum [4].





	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Accuracy
A1	971	0	1	0	1	1	3	2	1	0	99.08%
A2	0	1122	1	2	0	1	4	1	4	0	98.85%
A3	7	3	1003	0	2	2	4	8	3	0	97.19%
A4	0	0	5	985	0	7	0	6	6	1	97.52%
A5	2	1	0	1	962	1	6	1	2	6	97.96%
A6	2	1	0	11	1	864	8	1	2	2	96.86%
A7	3	3	0	0	2	9	940	0	1	0	98.12%
A8	1	4	10	1	1	0	1	1005	1	4	97.76%
A9	2	1	4	4	7	4	4	4	942	2	96.71%
A10	1	2	1	9	14	4	0	9	6	963	95.44%

Figure 8: Learning Rate, Classification Accuracy and Confusion matrix for network with momentum of 0.8 and ReLU Activation function [80HU,10E,0.01LR]

We carried out experiments with momentum and found that a much faster(at 9 epochs) and smoother convergence is achieved(figure 8) with m as 0.8. Also the accuracy for testset was maintained at 97.57%. even after using 80 hidden units compared to 100 in other runs. This is because the gradient keeps changing direction and momentum helps to smoothen the variations. The error surface may have different curvature along different directions of error surface which leads to formation of narrow valleys. For most points on the surface, the gradient does not point towards the minimum, and successive steps of gradient descent can oscillate from one side to the other, progressing only very slowly to the minimum. The addition of momentum helps to speed up convergence to the minimum by damping these oscillations [4].

D. Training Methodology (SGD vs Batch vs Mini Batch)

This method of training the network where we repeatedly iterate through the training set, and each time we present a sample, we update the weights of network according to the gradient of the error with respect to that single training example only is called Stochastic Gradient Descent(SGD). Whereas in batch gradient descent we iterate through the entire training set before updating the weights—a costly operation as our dataset is very large.

SGD can start making progress right away, and continues to make progress with each sample it looks at. Often, SGD gets error close to the minimum much faster than batch GD. However it may never converge to the minimum, and will keep oscillating around the minimum of cost. But in practice most of the values near the minimum will be reasonably good approximations to the true minimum. For these reasons, particularly when the training set is large, SGD is often preferred over batch GD [3].

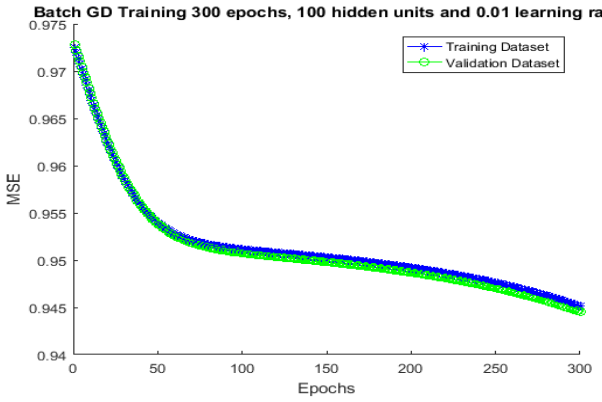
After few experiments upon observing that batch GD takes a very long time to do classification, we increased the number of epochs and hidden units upto 300 and 100 respectively. Each epoch presents 50k images to the network and accumulates the delta weight values over the epoch, then the values are averaged according to batch size and the weight values are adjusted. Even then the classification accuracy came to 24.12% with test set. Because the training was not complete and the rate at which the error decreases is very less, which can be seen from figure 9, that even after 300 epochs the MSE is still in the range of 0.9. A similar situation occurs with the network shown in figure 10. The training was not complete and classification accuracy for test set was 9.8%.

Another variation on these techniques is called Mini-batch GD which works sometimes even a bit faster than SGD. In Batch GD we use all images in each epoch [5]. Whereas Mini-batch GD does is somewhere in between SGD and batch GD. Here we use a batch of images in each epoch where batch is a parameter called the mini batch size. This is just like batch gradient descent, except that here we use a much smaller batch size. However, even this method takes longer time and larger network to converge than SGD, the performance improvement of this scheme over batchGD is very nice. Figure 11 shows the learning curves, classification accuracy and confusion matrix for 400 HU, 1000 epochs and 0.1 learning rate with a mini batch size of 300. The accuracy for test set was 92.89%, also upon increasing the interations upto 2000, about 97.64% accuracy was achieved.

E. Improving generalization accuracy using Dropout

Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. The term "dropout" refers to

dropping out units (both hidden and visible) in a neural network [6].



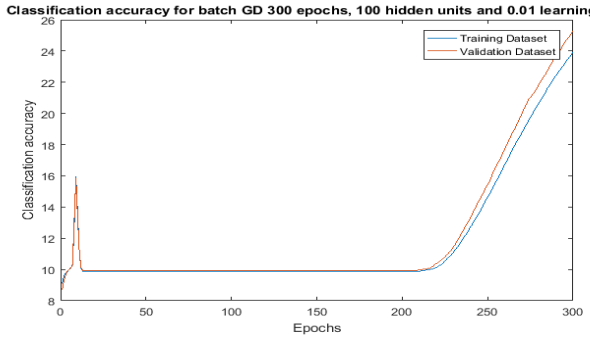


Figure 9: Learning Rate, Classification Accuracy for batchGD and 300E, 100HU,0.01

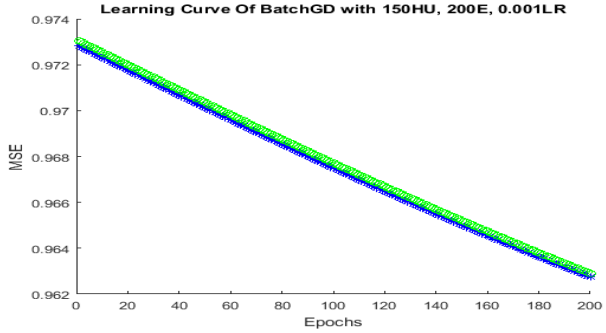
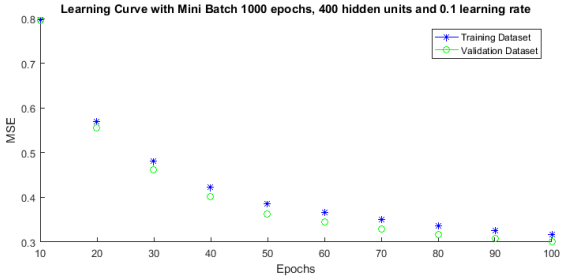
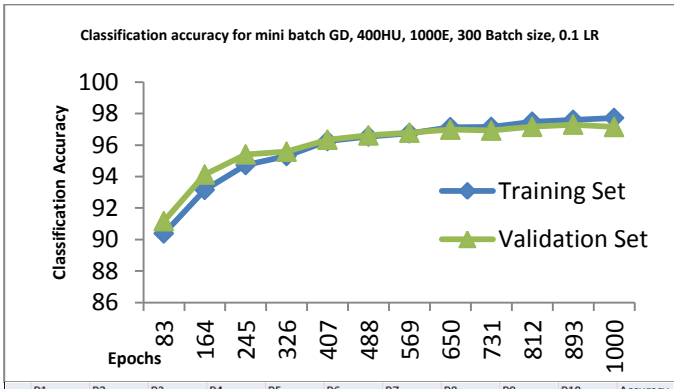


Figure 10: Learning Rate for batchGD and 200E, 150HU,0.001LR

The performance was not improved much from standard SGD, 97.75% accuracy was obtained when using a 20% probability of dropout at either one of the units of hidden layer or output layer weights after every 5000 samples presented to network. The learning curve was similar to that in figure 8 which is shown in figure 12. Accuracy dropped to 97.64% when using a 20% probability of dropout at either one of the units of hidden layer or output layer weights after every 1000 samples presented to network. That means the network in its original algorithm generalizes well.



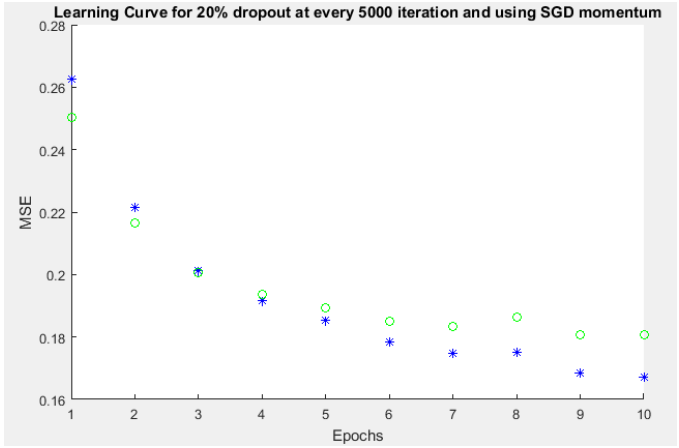


	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Accuracy
A1	961	0	1	2	1	2	9	1	3	0	98.06%
A2	0	1115	3	4	0	0	5	2	6	0	98.24%
A3	13	0	944	8	11	0	15	14	22	5	91.47%
A4	2	0	21	927	2	22	4	13	14	5	91.78%
A5	3	5	4	0	910	0	15	2	3	40	92.67%
A6	14	3	5	44	15	754	21	8	20	8	84.53%
A7	9	3	3	1	8	6	926	0	2	0	96.66%
A8	5	12	23	3	9	1	0	956	0	19	93.00%
A9	10	8	5	12	11	18	16	9	878	7	90.14%
A10	11	8	1	10	39	8	1	8	5	918	90.98%

Figure 11: Learning Rate, Classification Accuracy and Confusion matrix for mini batch GD, 400HU, 1000E, 300 Batch size, 0.1 LR

F. Downsampling and Dropout

The results after downsampling and using the same dropout scheme after every 1000 samples presented were quite satisfactory, downsampling by 2 was used, that is the size of each sample reduced to half, even then the network achieved 97.17% accuracy on test set with the reduced quality dataset. The confusion matrix is shown in figure 13. Network used was 80HU, 10E, 0.01LR with momentum and ReLU activation function.



	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Accuracy
A1	969	0	2	2	0	1	4	1	1	0	98.88%
A2	0	1123	1	4	0	0	4	1	2	0	98.94%
A3	1	2	1008	6	3	0	2	6	4	0	97.67%
A4	1	0	5	993	0	2	0	6	3	0	98.32%
A5	0	0	2	0	961	0	4	0	2	13	97.86%
A6	2	0	0	12	1	864	4	2	4	3	96.86%
A7	5	3	0	0	2	3	944	0	1	0	98.54%
A8	1	6	14	4	1	0	0	991	5	6	96.40%
A9	2	1	5	8	5	1	2	5	943	2	96.82%
A10	2	2	0	6	7	4	1	5	3	979	97.03%

Figure 12: Learning Curve and Confusion matrix for 20% dropout at every 5000 iteration and using SGD momentum

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Accuracy
A1	968	0	0	2	1	1	3	2	2	1	98.78%
A2	0	1119	2	6	0	0	2	0	6	0	98.59%
A3	10	3	994	7	3	0	1	9	5	0	96.32%
A4	0	0	3	987	0	4	0	5	9	2	97.72%
A5	1	1	2	1	949	0	5	1	2	20	96.64%
A6	6	0	0	12	0	857	4	4	4	5	96.08%
A7	4	3	0	0	1	6	941	0	3	0	98.23%
A8	1	6	13	6	1	0	1	994	0	6	96.69%
A9	6	1	3	5	6	1	1	4	944	3	96.92%
A10	5	5	0	5	15	0	1	7	7	964	95.54%

Figure 13: Classification accuracy after dropout, 80HU, 10E, 0.01LR

IV. Conclusion

In this paper, we used the general concept of multilayer NN to classify handwritten digits from MNIST dataset. We discussed two activation functions and explored how the architecture and hyper parameter choices affect the power of neural networks for classification.

References

[1] Neil R. Euliano, W. Curt Lefebvre Jose C. Principe,.: Wiley, 1997, ch. 3.

[2] Chiyuan Zhang. (2014) Mocha Documentation. Web. [Online]. http://mochajl.readthedocs.io/en/latest/user-guide/neuron.html

[3] Andrew Ng. http://cs229.stanford.edu/. [Online]. http://cs229.stanford.edu/notes/cs229-notes1.pdf

[4] Genevieve Orr. (1999, Fall) Neural Networks. [Online]. https://www.willamette.edu/~gorr/classes/cs449/momrate.html

[5] Andrew Ng. Machine Learning by Stanford University. [Online]. https://www.coursera.org/learn/machine-learning/lecture/9zJUs/mini-batch-gradient-descent

[6] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *ARXIV*, 2012.