MACHINE LEARNING PROJECT #3 REPORT

- **Gautam Shende (Person# 50245840)**
- **Sujit Singh (Person# 50247206)**

CLASSIFICATION USING NEURAL NETWORK WITH SINGLE HIDDEN LAYER (using RELU)

The four main parameters to be tuned here were

1) Number of input nodes in the hidden layer

2) Number of epochs

3) Batch size

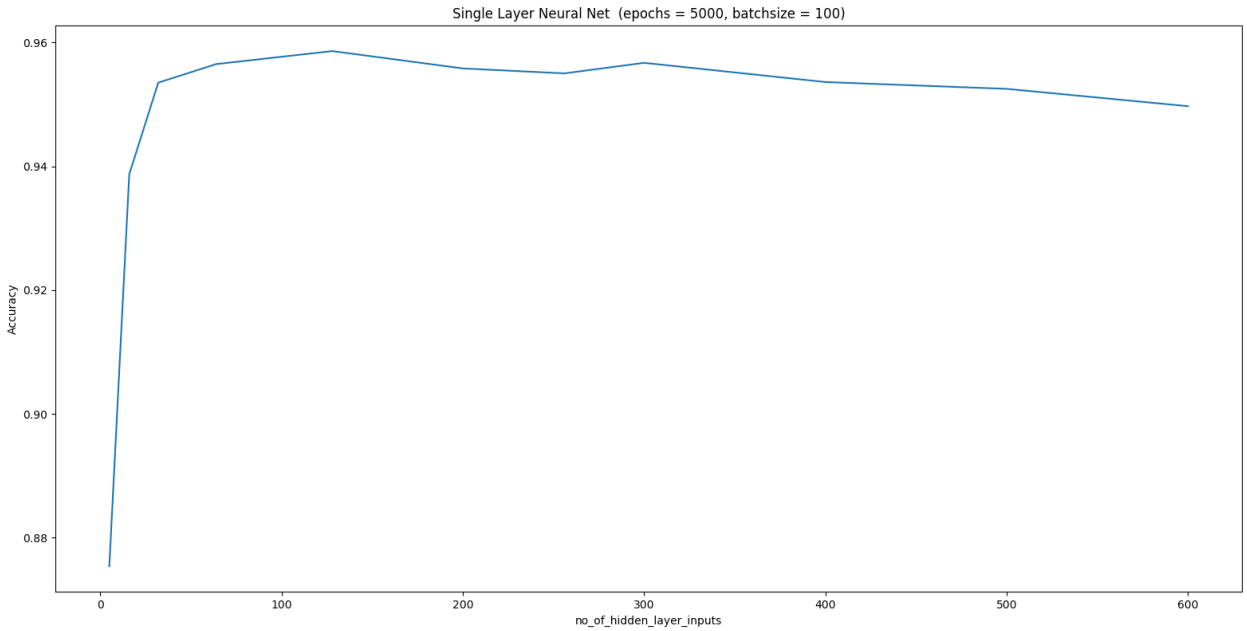
4) Learning Rate

1) Number of input nodes in the hidden layer (N)

Keeping the other parameters constant (epochs = 5000, batch size = 100 and learning rate = 0.5), we had the following

Results for (N) vs accuracy

Nodes in hidden layer	Accuracy
5	0.8754
16	0.9387
32	0.9535
64	0.9565
128	0.9586
200	0.9558
300	0.9567
400	0.9536
500	0.9525
600	0.9497
700	0.9507
256	0.955
800	0.9489
900	0.9464
1000	0.9463



The accuracy was optimum in the range of 100 to 250. It increased upto ~100 and started decreasing from 250 onwards.

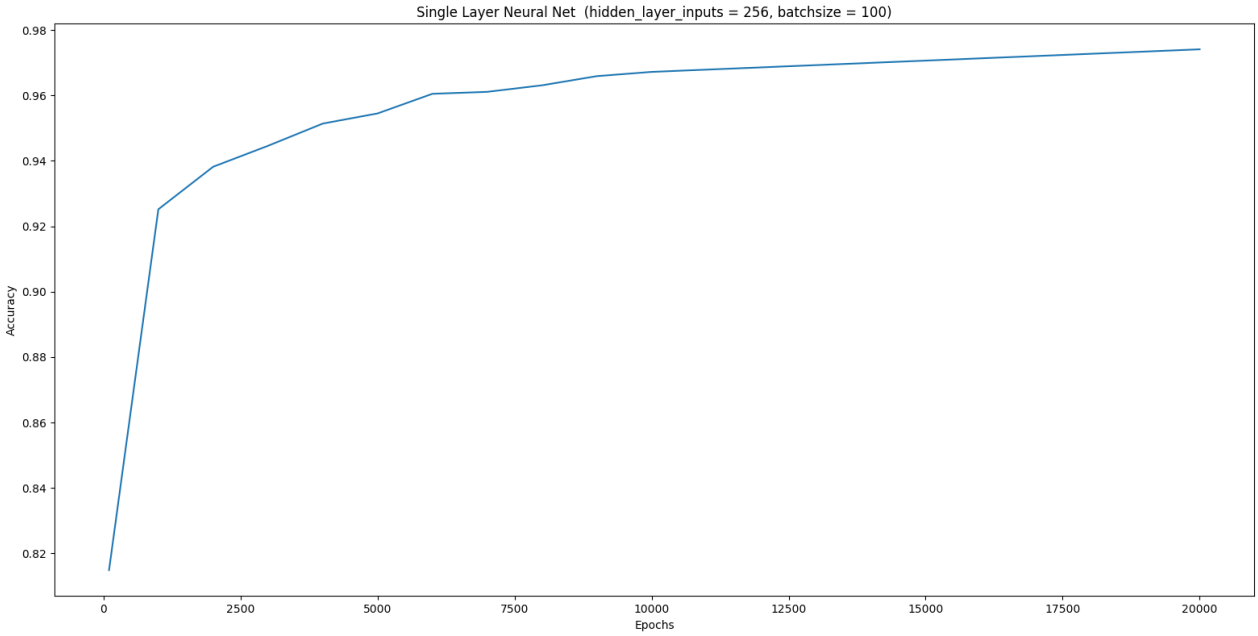
Hence we set number of hidden layer nodes = *256*.

2) Number of epochs

Keeping the other parameters constant (N = 256, batch size = 100 and learning rate = 0.5), we had the following

Results for Epochs vs accuracy

Epochs	Accuracy
100	0.8149
1000	0.9252
2000	0.9382
3000	0.9446
4000	0.9514
5000	0.9545
6000	0.9605
7000	0.9611
8000	0.9631
9000	0.9659
10000	0.9672
20000	0.9741



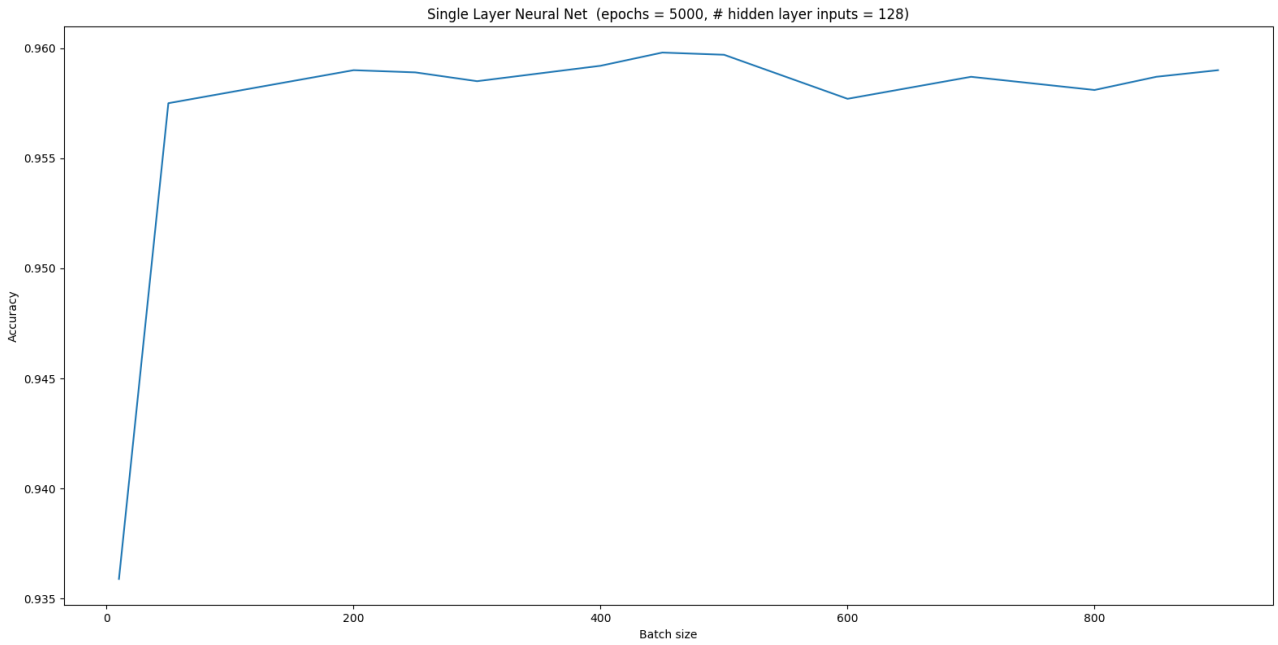
The gain in accuracy was less significant when compared to increase in computation as the epochs went over 15,000 so we set number of epochs = *20,000*.

3) Batch Size

Keeping the other parameters constant (N = 256, Epochs = 5000 and learning rate = 0.5), we had the following

Results for batchsize vs accuracy

Batch	Accuracy
10	0.9359
50	0.9575
150	0.9585
200	0.959
250	0.9589
300	0.9585
400	0.9592
450	0.9598
500	0.9597
600	0.9577
700	0.9587
750	0.9584
800	0.9581
850	0.9587
900	0.959



The batch size did not have much effect on the accuracy once over 20, we set it to *500* so probability of covering all inputs is more.

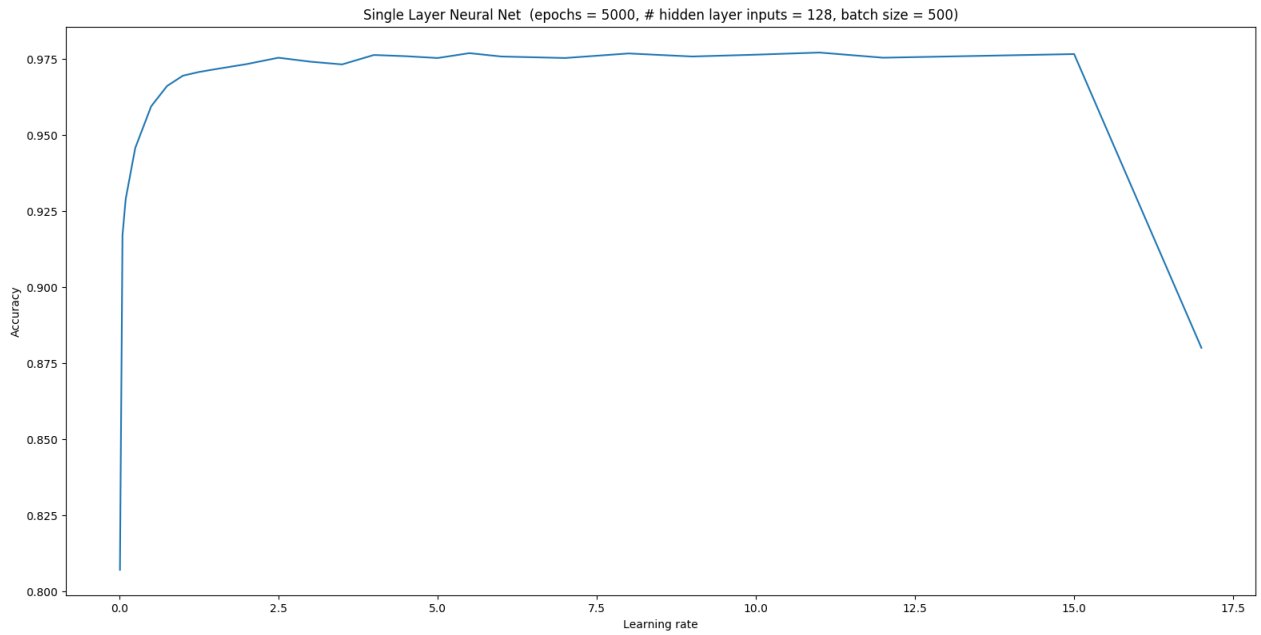
4) Learning Rate:

Keeping the other parameters constant (N = 256, Epochs = 5000 and batch size = 100), we had the following

Results for Learning Rate vs accuracy

Learning Rate	Accuracy
0.01	0.8071
0.05	0.9171
0.1	0.929
0.25	0.9458
0.5	0.9595
0.75	0.9662
1	0.9696
1.25	0.9708
1.5	0.9717
2	0.9734
2.5	0.9755
3	0.9742
3.5	0.9733
4	0.9764
4.5	0.976
5	0.9754
5.5	0.977
6	0.9759
7	0.9754
8	0.9769
9	0.9759
10	0.9765
11	0.9772

12	0.9755
15	0.9767
17	0.8801



The accuracy improved upto learning rate = 2 and then the accuracy almost became constant before starting to drop around 15. We set learning rate = *5.5.*

Activation Function:

After observing the trend in the hyperparameters, we compared the parrallel performance of the 3 sigmoid functions.

We compared the three activation function choices and found out **Rectified Linear Unit (RELU)** was giving the best results when compared to *TANH* and *SIGMOID*.

FINAL RESULTS:

On MNIST Validation Set:

Learning Rate = 5.5, Epochs = 20000, Batch Size = 500, Hidden Layer Nodes = 256, Accuracy = ***0.9792***

On USPS Testing Set:

Learning Rate = 5.5, Epochs = 20000, Batch Size = 500, Hidden Layer Nodes = 256, Accuracy = ***0.5107***

CLASSIFICATION USING LOGISTIC REGRESSION

The three main parameters to be tuned here were

1) Learning Rate

2) Number of epochs

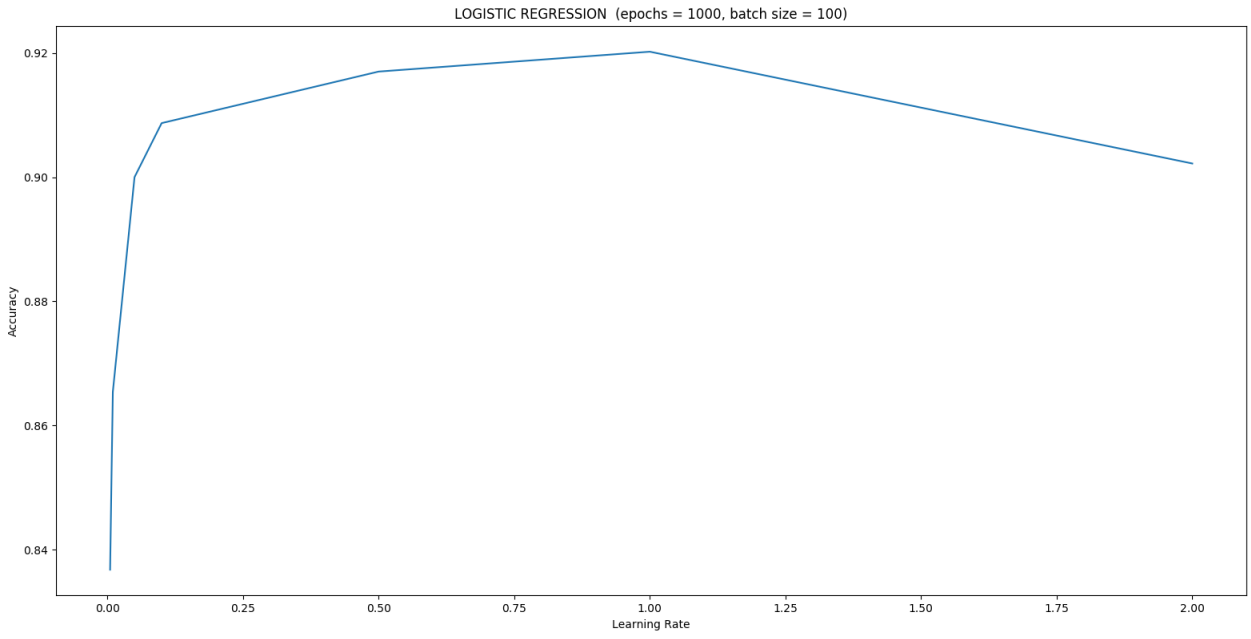
3) Batch size

1) Learning Rate:

Keeping the other parameters constant (Epochs = 1000 and Batch Size = 100), we had the following

Results for Learning Rate vs accuracy

Learning Rate	Accuracy
10	0.8293
8	0.8603
6	0.8824
4	0.8914
2	0.9022
1	0.9202
0.5	0.917
0.1	0.9087
0.05	0.9
0.01	0.8654
0.005	0.8368



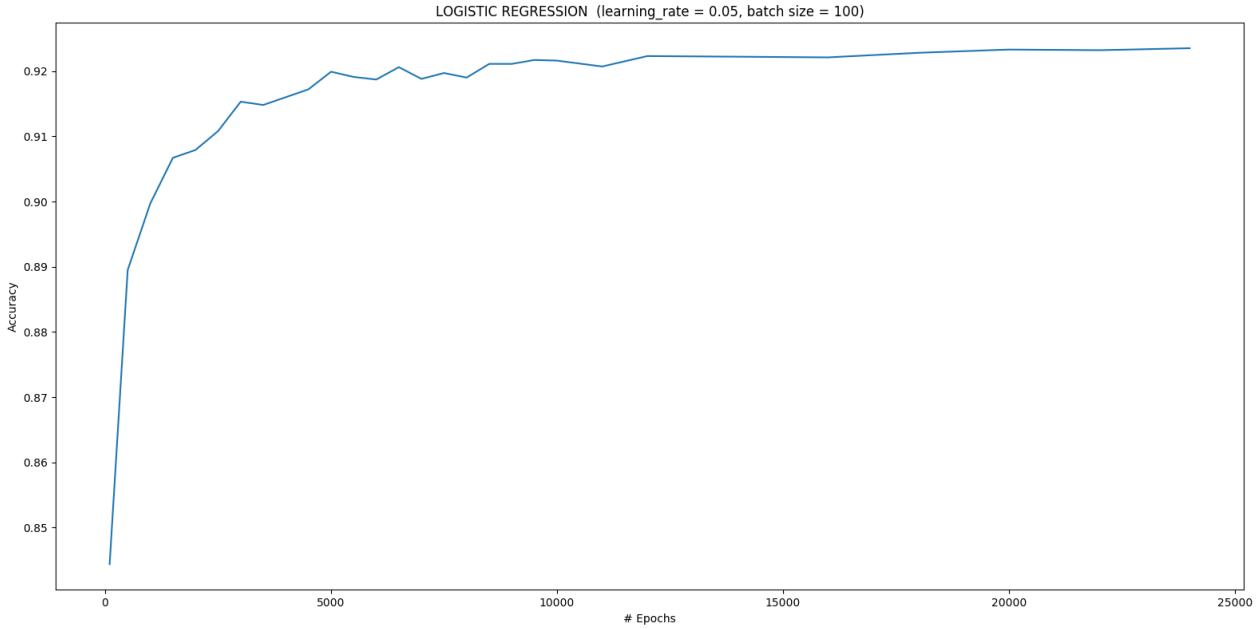
Even though the learning rate was found to be optimal around 1, we chose Learning Rate = *0.5* because we were going to increase the number of epochs later on and didn't want to set it as high as 1 as it would miss the best values due to large jumps.

2) Number of epochs:

Keeping the other parameters constant (Learning Rate = 0.05 and Batch Size = 100), we had the following

results for number of epochs vs accuracy

Epochs	Accuracy
100	0.8444
500	0.8895
1000	0.8997
1500	0.9067
2000	0.9079
2500	0.9108
3000	0.9153
3500	0.9148
4500	0.9172
5000	0.9199
5500	0.9191
6000	0.9187
6500	0.9206
7000	0.9188
7500	0.9197
8000	0.919
8500	0.9211
9000	0.9211
9500	0.9217
10000	0.9216
11000	0.9207
12000	0.9223
14000	0.9222
16000	0.9221
18000	0.9228
20000	0.9233
22000	0.9232
24000	0.9235
100000	0.9253

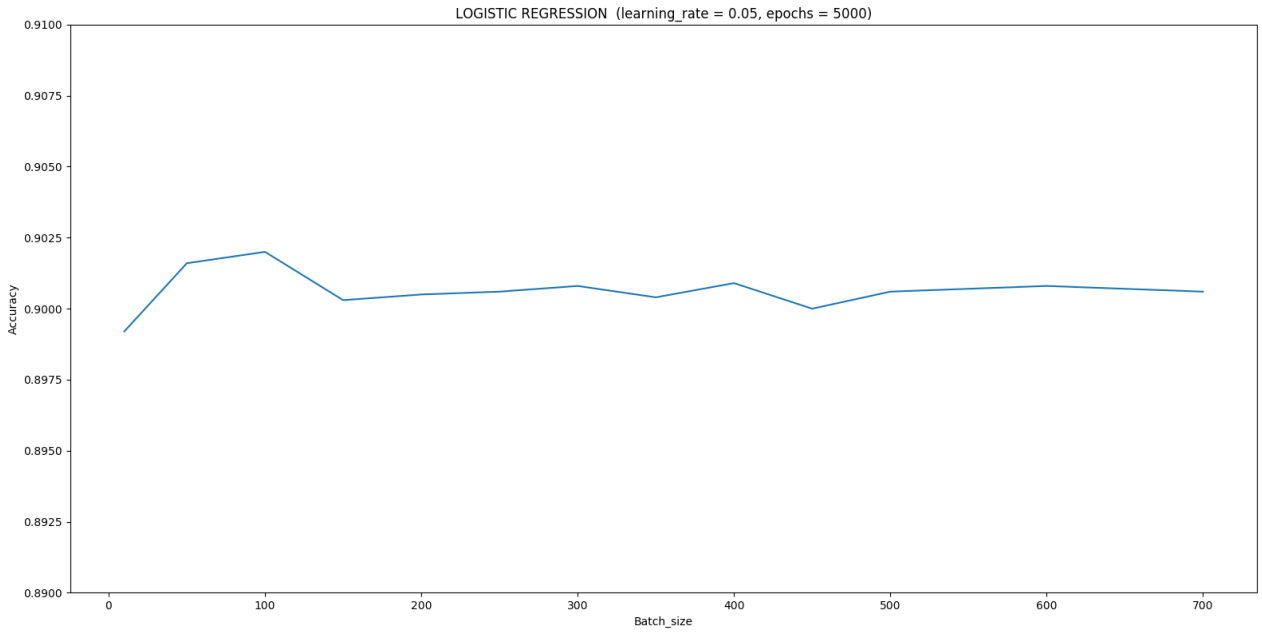


After 10,000 there wasn't any significant improvement in performance when compared to the increase in the resource Requirement. So we set epochs = *10,000*.

3) Batch Size:

Keeping the other parameters constant (Learning Rate = 0.05 and Epochs = 1000), we had the following results for batchsize vs accuracy

Batch size	Accuracy
10	0.8992
50	0.9016
100	0.902
150	0.9003
200	0.9005
250	0.9006
300	0.9008
350	0.9004
400	0.9009
450	0.9
500	0.9006
600	0.9008
700	0.9006



The batch size had no significant effect on the accuracy so we set it to an intermediate value of *100* so that it was cost Efficient and also has decent probability of covering all inputs.

FINAL RESULTS:

On MNIST Validation Set:

Learning Rate = 0.5, Epochs = 10000, Batch Size = 100, Accuracy = ***0.9269***

On USPS Testing Set:

Learning Rate = 0.5, Epochs = 10000, Batch Size = 100, Accuracy = ***0.3333***

CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS

We used the open source Tensor flow documentation and resources for using implementing CNN on the MNIST data set. We did not do any parameter tuning and only played with the number of epochs which gave the following results on the MNIST validation set:

Epochs	Accuracy
100	0.89
200	0.94
300	0.96
400	0.91
500	0.95
600	0.97
700	0.98
800	0.98
900	0.98
1000	0.94
1100	0.99
1200	0.97
1300	0.99
1400	0.98
1500	0.96
1600	0.98
1700	0.98
1800	0.99

USPS test accuracy = *0.578667*

No free lunch theorem

The General algorithm can be summarized as:

If algorithm A outperforms algorithm B for some problem, then loosely speaking there must exist exactly as many other problems where B outperforms A.

Comparing it to our results:

Accuracy Data set	Logistic Regression	Single Hidden layer NN	CNN
MNIST Validation	0.9269	0.9792	0.9901
USPS	0.3333	0.5107	0.5786

Thus our implementations did not support the Free lunch theorem in general because on both the datasets, CNN had the best accuracy followed by the Single Hidden Layer Neural Network followed by Logistic Regression.

However, it is not entirely correct to say that it completely violated the free lunch theorem because on the MNIST validation set the CNN and the Single hidden layer NN had very close accuracies so some instances where the CNN was run for less number of epochs would actually have had lesser accuracy on the MNIST validation set when compared to the finely tuned Single Hidden layer NN models but in case of actual testing, the CNN took a huge lead on the USPS data set, so we could have found some instances where the No free lunch theorem was followed.

BACK PROPAGATION IMPLEMENTATION USING SINGLE NEURAL NETS EXTRA CREDIT

The implementation of back propagation using test inputs is done as a part of extra credit.

The backpropagation implemented goes through the following algorithm.

1. Read input data and test labels

2. Declare the number of hidden layers (In our implementation we are considering only one hidden layer)

3. Declare the number of node in each hidden layer

4. Declare weights and biases for each layer in the neural network

After initializing all the required inputs the algorithm is divided into two phases

1. Feed Forward phase:

In feed forward phase we try to find out the the outputs generated from all the layer including the input layer.

An activation function is applied to the output of each and the corresponding error is generated for the output layer. We are using sigmoid function as an activation function.

The error is calculated by the below formula

$$\text{Error} = \tfrac{1}{2}(\sum(\text{target} - \text{predicted\_output})^2)$$

where predicted_output is the output generated from the last layer

2. Feed Backward (Backpropagation phase):

In back-propagation phase we are trying we use equations to reduce errors and set weights.

Back-propagation of the single layer neural nets can be done in the following way:

1. **Back-propagating from output layer to the hidden layer:**
   In this we need to calculate the rate of change of total error w.r.t. weights assigned to the nodes of the hidden layer.
   This can be done by the following equation:

$$\partial E_{total} / \partial w_i = (out_{o1} - target) * out_{o1} * (1 - out_{o1}) * out_{h1}$$

Here E_{total} = total error of all the nodes from the output layer

w_i = weights

out_{o1} = output received from the output layer

out_{h1} = output received from the hidden layer

target = target label

After this, we need to update weights using the error change w.r.t. each weights in the hidden layer using the below formula.

$w_i = w_i - \mu * \partial E_{total} / \partial w_i$

here μ is the learning rate.

2. **Back-propagating from hidden layer to the input layer:**
In this, we need to calculate the rate of change of total error w.r.t. weights assigned to the nodes of the input layer which can be further applied to change weights and converge to the desired target.
This can be done by the following equation:

$$\partial E_{total} / \partial w_i = (out_{o1} - target) * out_{o1} * (1-out_{o1}) * out_{h1} * w_{oi} * input$$

Here E_{total} = total error of all the nodes from the output layer

w_i = weights of the input layer

w_{oi} = weights of the hidden layer

out_{o1} = output received from the output layer

out_{h1} = output received from the hidden layer

target = target label

After this, we need to update weights using the error change w.r.t. each weights in the hidden layer using the below formula.

$w_i = w_i - \mu * \partial E_{total} / \partial w_i$

here μ is the learning rate.

Following these implementation steps the accuracy of the network comes out to be ***48%*** approximately

REFERENCES

1) *https://tensorflow.org*
2) *https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/*

. . .