EE456 Section 001 Mini Project Report

ON MLP NN WITH BACKPROPAGATION

Sahin, Efe
PENN STATE UNIVERSITY | NOVEMBER 8, 2022

Overview:

This mini project is about a Multi Layered Perceptron using Backpropagation to update its weights. There are two separate sets of data which are DataSet1_MP1.mat and DataSet2_MP1.mat. Each data set have two separate moons close to each other on cartesian plane. The NN is trained using any coordinate with its respective target value. There are n=2 input neurons, p=20 hidden layer neurons and m=1 output neuron. The Neural Net works by 3 stages as follows:

- Feedforward of input coordinate
- Calculation and backpropagation of error with respect to t
- Adjust weights

The activation function is hyperbolic tangent and the learning rate alpha is decreased linearly from 10^-1 to 10^-5.

Implementation:

The net requires weight vectors and bias vectors for hidden and output layer for the learning process. These weight and bias vectors for middle layer are called V and V0 respectively whereas the weight and bias vectors for output layer are called W and W0 respectively. To initiate the net, the values of V, V0, W, W0 is set according to Nguyen-Widrow initialization method. The following Figure1 shows a portion of code that initializes the mentioned vectors. There are 2 other weight initialization methods in the code too which are only random generators.

```
60      %Nguyen-Widrow
61      B=0.7*(p)^(1/n);
62      Vold=(-0.5-0.5).*rand(n,p) + 0.5;%1xp
63      Vnorm=sqrt(sum(Vold.^2));
64      V=(B*Vold)./Vnorm;
65      V0=((-B)-B).*rand(1,p) + B;%1xp
66      W=(-0.5-0.5).*rand(p,m) + 0.5;%pxm
67      W0=(-0.5-0.5).*rand(1,m) + 0.5;%1xm
```

FIGURE 1: NGUYEN-WIDROW INITILIZATION OF V AND V0 IS DEPENDENT ON CONSTANT BETA

The weight vectors are updated by keeping track of momentum. Keeping track of momentum requires previous weight vectors at time stamp t-1 and t-2. The momentum tracking weight vectors used were W1t, W2t, V1t, V2t. These weight vectors then get used to update the weights W and V according to the momentum constant which linearly decreases exactly as alpha. Momentum update rule is useful as the gradient descent over the error gets to be faster when the weights have been updating in a similar fashion I.e. momentum is dependent on how weights have been increasing or decreasing over each epoch. The following Figure2 shows how the momentum rule has been implemented.

```
106             %--------------
107             % Backprob
108             %--------------
109
110             deltak=(t-y).*tanhD(y_in);%1xm=(1xm-1xm).1xm
111             deltaW=alpha*z'*deltak;%pxm=1x1.px1.1xm
112             deltaW0=alpha*deltak;%1xm=1x1.1xm
113
114             delta_inj=deltak*W';%1xp=1xm.mxp
115
116             deltaj=delta_inj.*tanhD(z_in);%1xp=1xp.1xp
117             deltaV=alpha*x'*deltaj;%nxp=1x1.1xn.1xp
118             deltaV0=alpha*deltaj;%1xp=1x1.1xp
119
120             W=W1t+deltaW+momentum*(W1t-W2t);%pxm=pxm+pxm+1x1*(pxm-pxm)
121             W0=W0+deltaW0;%1xm=1xm+1xm
122             V=V1t+deltaV+momentum*(V1t-V2t);%nxp=nxp+nxp+1x1*(nxm-nxm)
123             V0=V0+deltaV0;%1xp=1xp.1xp
124             if s>=1
125                 if s>=2
126                     W2t=W1t;
127                     V2t=V1t;
128                 end
129                 W1t=W;
130                 V1t=V;
131             end
132         end
```

FIGURE 2: EACH LINE IS COMMENTED WITH THE SIZE CALCULATION WHERE '.' INDICATES MULTIPLICATION. N, P AND M INDICATES NUMBER OF NEURONS ON INPUT, MIDDLE AND OUTPUT LAYERS RESPECTIVELY.

The process of training is done by selecting DataSet1_MP1. or DataSet2_MP1.mat. Selected dataset is referred to as dataSetTrain which is shuffled and only 80% percent of it is being used to train and calculate the Training MSE in each epoch. The rest 20% is used to check the Validation MSE in each epoch. At the end, the 100% of the other data set dataSetTest is tested on with the final weights from training.

Results:

In the process of building the code for the net, I first attempted the regular weight update method and regular initialization of weight vectors between range 1 to -1. When it was working with minimum modifications, I experimented with initializing random numbers between (1,-1) and (0.5,-0.5) on weight vectors. My observation was weight vectors randomly initialized between the range (0.5,-0.5) yielded higher accuracies in general. Then I replaced it with another initialization method called Nguyen-Widrow. After I didn't observe much difference between random values in range (0.5,-0.5) and Nguyen-Widrow, I decided to change the weight update method because the was net getting stuck at several local minimums and the net had to be faster at converging global minimum. After I implemented the momentum weight vector update rule, the net got faster as it was able to get high accuracy in training around 96% and 95% for testing. For some cases, I have observed the net converged to almost 100% accuracy, but at very high epochs specifically testing both data after 1200 when rng=21.

To analyze the performance of the final state of the net, I ran it multiple maximum number of epochs with different rng values to see how the net performs overall.

epoch\rng	Rng(1)	Rng(2)	Rng(3)	Rng(4)
10	0.95	0.95	0.95	0.96
50	0.96	0.95	0.96	0.97
250	0.96	0.94	0.96	0.96
500	0.96	0.96	0.96	0.96

FIGURE 3: NGUYEN WIDROW INIT WITH MOMENTUM UPDATE. TRAIN ON DATASET 2, TEST ON DATA SET 1 MEASURING ACCURACY PERCANTAGE

epoch\rng	Rng(1)	Rng(2)	Rng(3)	Rng(4)
10	0.94	0.95	0.95	0.90
50	0.94	0.95	0.94	0.94
250	0.94	0.95	0.95	0.95
500	0.98	0.95	0.94	0.94

FIGURE 4: NGUYEN WIDROW INIT WITH MOMENTUM UPDATE. TRAIN ON DATASET 1, TEST ON DATA SET 2 MEASURING ACCURACY PERCANTAGE

Although the algorithm can reach to high accuracy levels when both dataset is tested, I will show an example that the net can reach also 99% accuracy, the following Figure 5, 6 shows training and validation MSE's and Accuracies during training the net with DataSet1_MP1.mat when rng=2. During training and validation, the net achieved 100% accuracy on both training and validation.

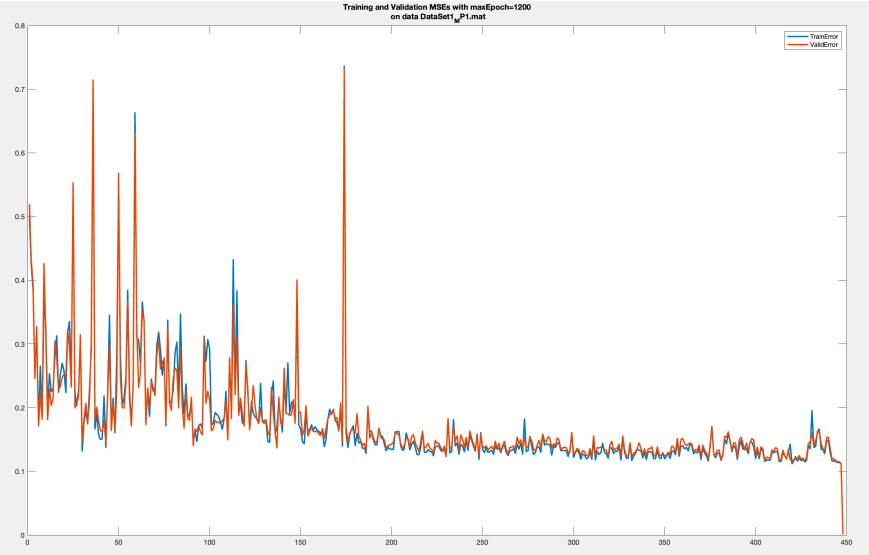

FIGURE 5: DATASET1_MP1.MAT HAS BEEN TRAINED OVER JUST 448 EPOCHS WHERE TRAINING AND VALIDATION ERROR SEEM TO ALTERNATE SIMILARLY TO EACHOTHER TOWARD THE END.

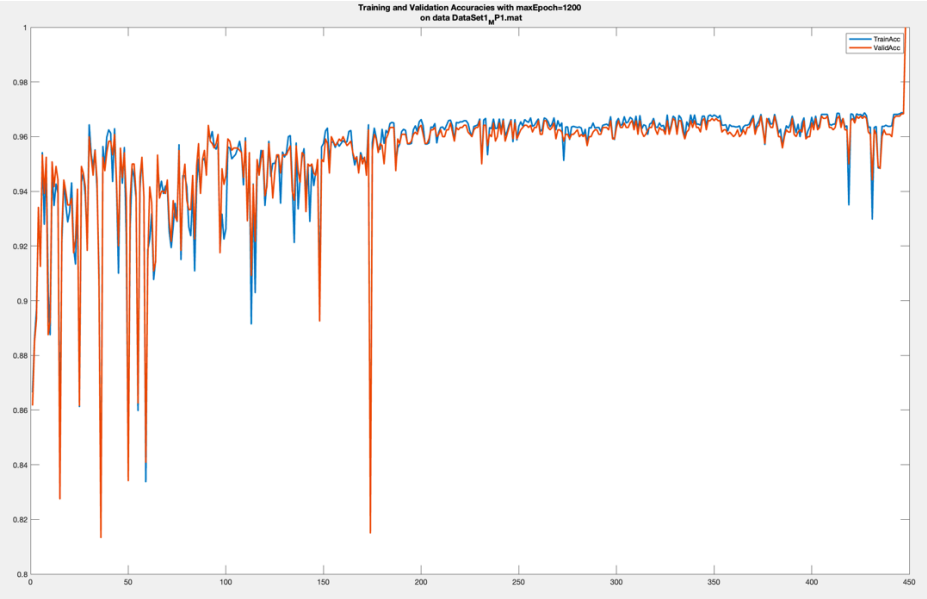

FIGURE 6: DATASET1_MP1.MAT HAS BEEN TRAINED OVER JUST 448 EPOCHS WHERE TRAINING AND VALIDATION ACCURACY SEEM TO ALTERNATE SIMILARLY TO EACHOTHER TOWARD THE END ACHIEVING 100%.

The following Figure 7 shows the decision boundary after testing the final weights on DataSet2_MP1.mat. There can be seen a curve that separates each class. However, as there is an error when some points are in wrong region, the net achieves 98.4667% accuracy and 0.059701 MSE.

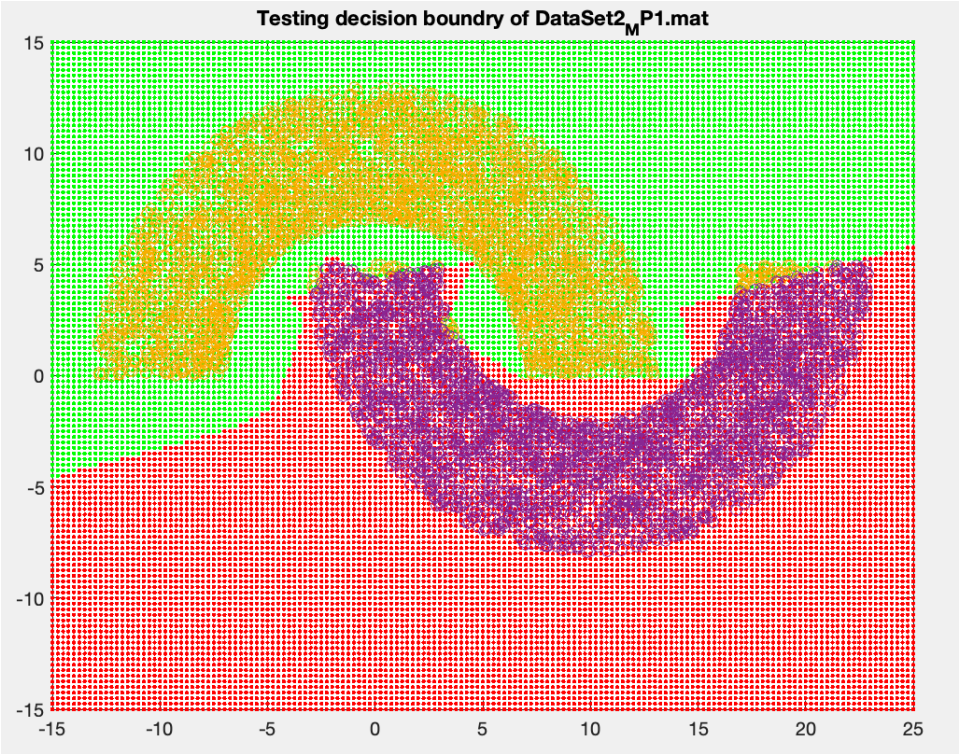

FIGURE 7: THE RESULT OF TESTING DATASET2_MP1.MAT SHOWS THE DECISION BOUNDRY SPLITTING EACH CLASS.

On the other hand, The following Figures 8 and 9 shows data concerning training and validation process on DataSet2_MP1.mat. The following is the result of 1200 epochs when rng=2 where it achieved MSE= 0.008500, Accuracy=99.75%.

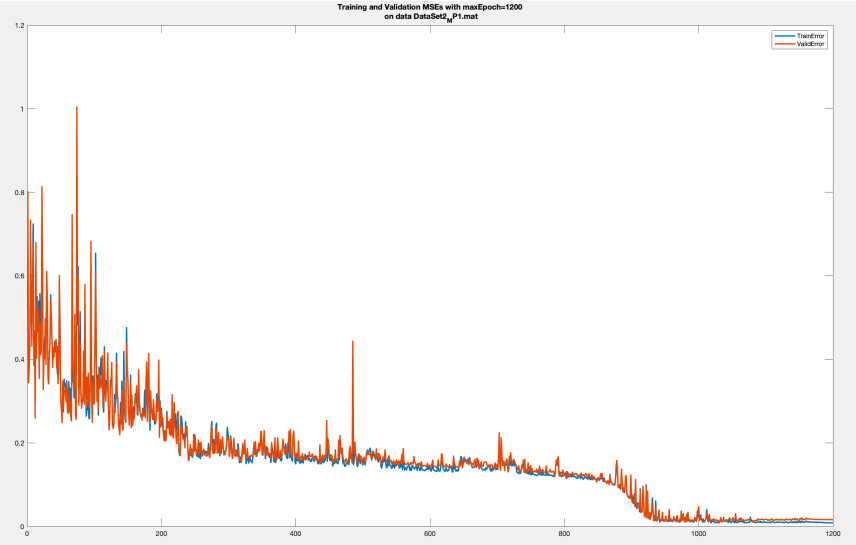

FIGURE 8

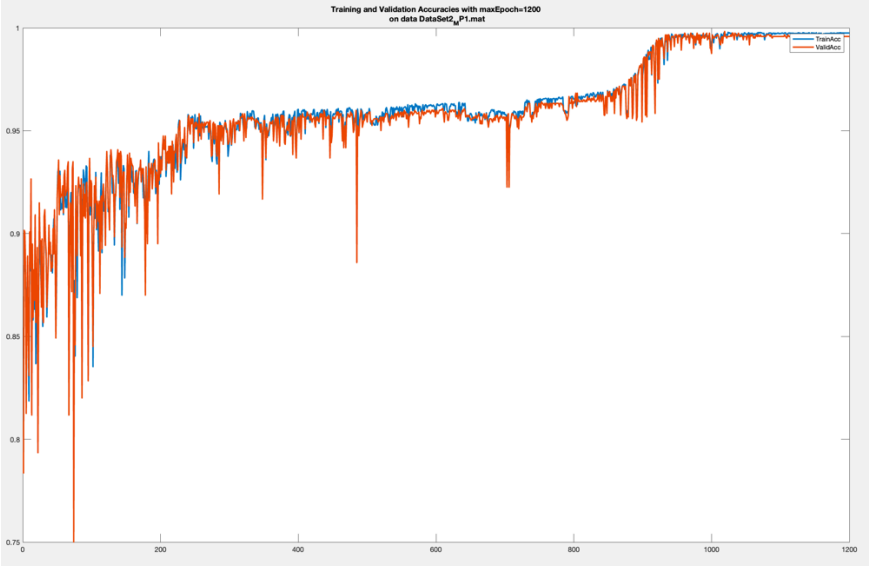

FIGURE 9

The Figure 10 below shows the final testing on DataSet1_MP1.mat where the net achieved MSE=0.004826, Accuracy=99.8833%.

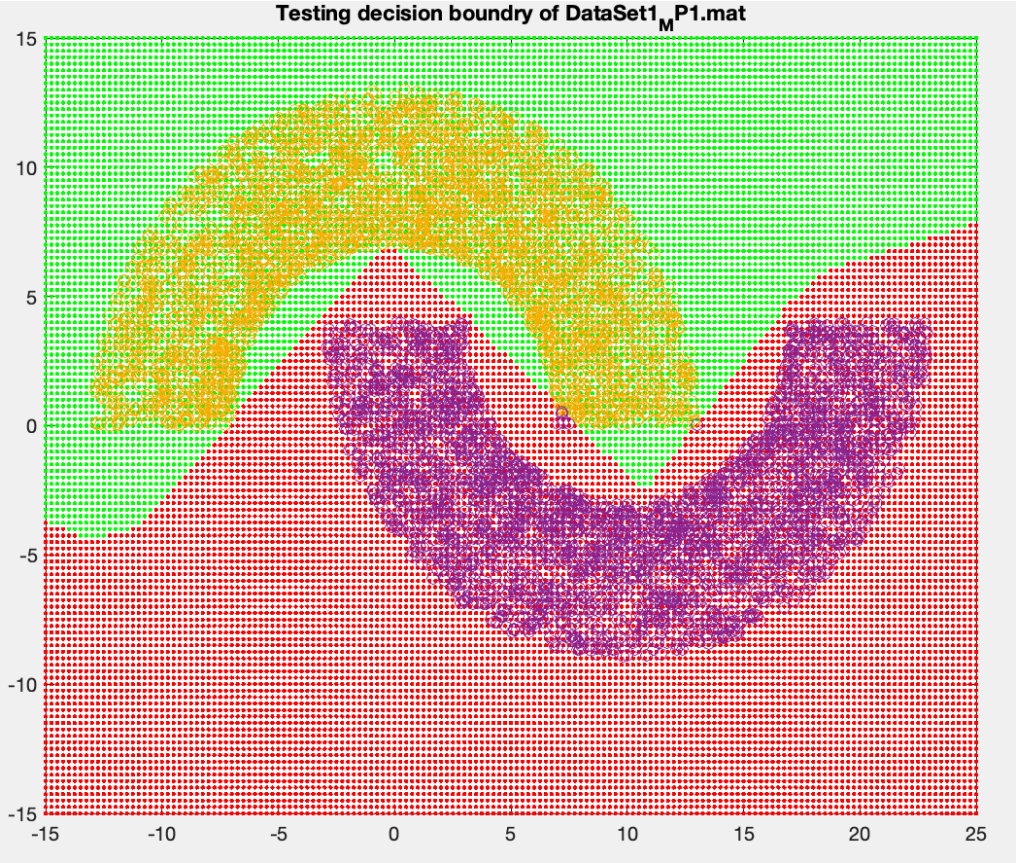

FIGURE 10

Based on the final accuracies of testing, it was expected that training on an easier data then testing on harder data would yield lower accuracy than the other way around (training on harder dataset then test it on easier dataset). One issue I came across was the net being unable to converge at accuracy ranges between (96%-100%) at epochs below 1000. This could be due to not having alpha and momentum term change to way smaller values than where it should be at.