Experiment 5: Back propogation for Alphabet

Table of Contents

Name: Ventrapragada Sai Shravani

PRN:17070123120

Batch:Entc(2017-21) G-3

Aim:

Implementation of Alphabet gate using back propagation

Theory:

Back propagation comes under a multiple layer feed forward networks.Back propogation is a supervised learning network, it consists of input, hidden and output layers.

A feedback is provided to the hidden and output layer and input layer does not contain bias since there is no feedback present.

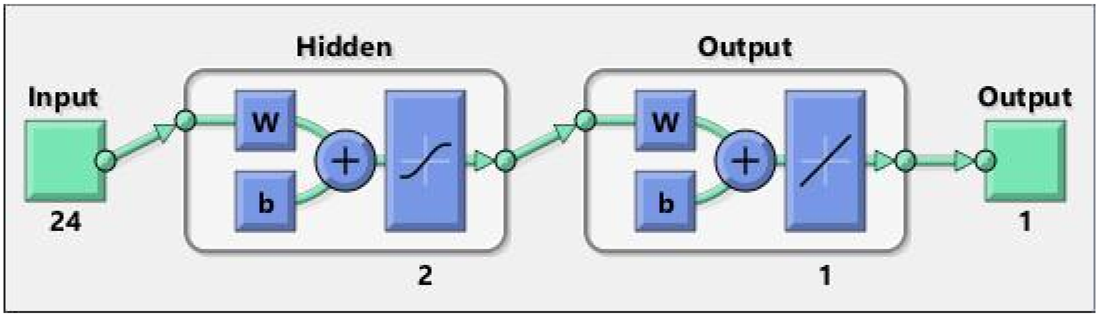
This algorithm can be explained in 2 phases: Feed-forward phase, back-propogation phase of learning where signals are sent in reverse direction.

Gradient descent: Is an optimisation algorithm used to find the value of parameter/ coefficient of a function, that minimises a cost function.

Back Propogation for Alphabet

```
clc;
clear all;
close all;
INPUT = transpose(readmatrix('Trainingcsv2.csv')); %Input 24 features
 of A
TARGET = transpose(readmatrix('Character.csv')); %Alphabet a and b
 character as target
net = feedforwardnet(2,'traingd'); % 2 defines the number of hidden
 layers
```

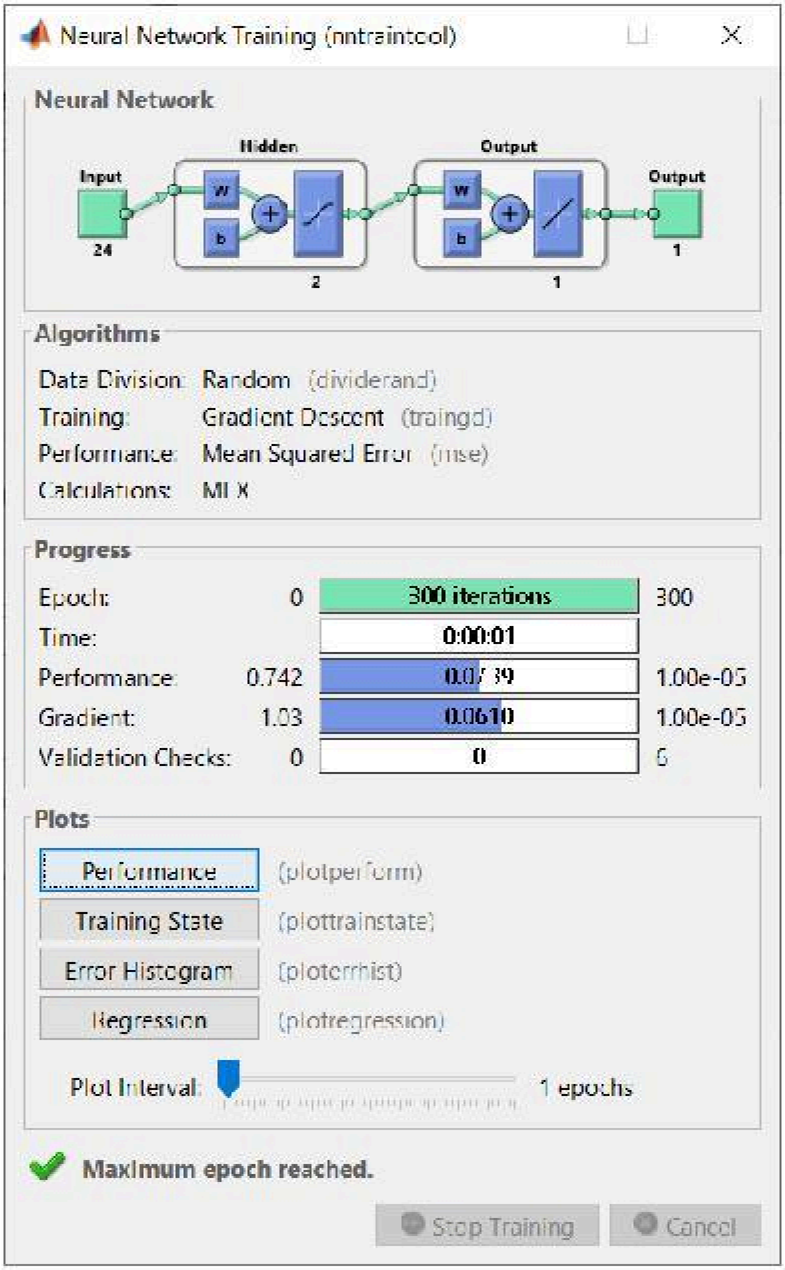
```
% Modifying the defined parameters
net.trainParam.show =50;  % The result is shown at every 50th
 iteration (epoch)
net.trainParam.lr = 0.05; % Learning rate used in some gradient
 schemes
net.trainParam.epochs = 300; % Max number of iterations
net.trainParam.goal = 1e-5; % Error tolerance; stopping criterion

% The goal of "training" a neural network is to find the right weights
 that
%correctly predict what the output of an given input is. There are two
 fundamental
%processes that go into training a neural network.

[net,tr] = train(net,INPUT,TARGET); %Training the network
%tr will contain all the details about the training
view(net)
y = net(INPUT);
```



Testing data

```
testX = INPUT(:,tr.testInd);
testT = TARGET(:,tr.testInd);
testY = net(testX);
YPredicted = net(INPUT);
YPredicted(:,1:10)
figure(1)
figure(2)
i=imread('C:\Users\acer\Documents\4th year\Machine Learning\MATLAB ML
\Capture.png');
imshow(i);
figure(3)
plotconfusion(TARGET,YPredicted)% plot the network training data
figure(4)
plotperform(tr) %performance plot
figure(5)
plottrainstate(tr) %trainstate plot


ans =

  Columns 1 through 7
```

```
   -0.0725    0.1298    0.1306   -0.0362    0.0056   -0.0057    0.1098

  Columns 8 through 10

    0.1672   -0.0075   -0.0652
```



Algorithms

Data Division: Random (dividerand)
Training: Gradient Descent (traingd)
Performance: Mean Squared Error (mse)
Calculations: MEX

Progress

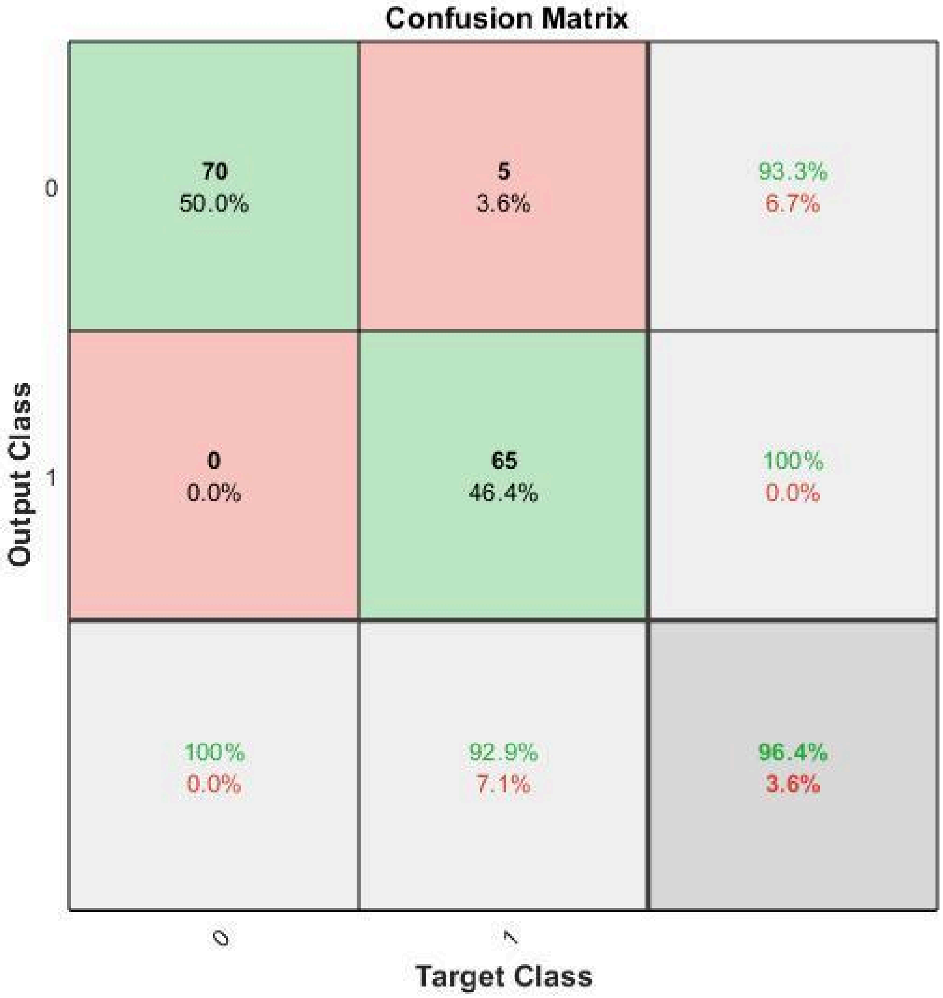
Epoch:	0	300 iterations	300
Time:		0:00:01	
Performance:	0.742	0.0739	1.00e-05
Gradient:	1.03	0.0610	1.00e-05
Validation Checks:	0	0	6

Plots

Performance (plotperform)
Training State (plottrainstate)
Error Histogram (ploterrhist)
Regression (plotregression)

Plot Interval: 1 epochs

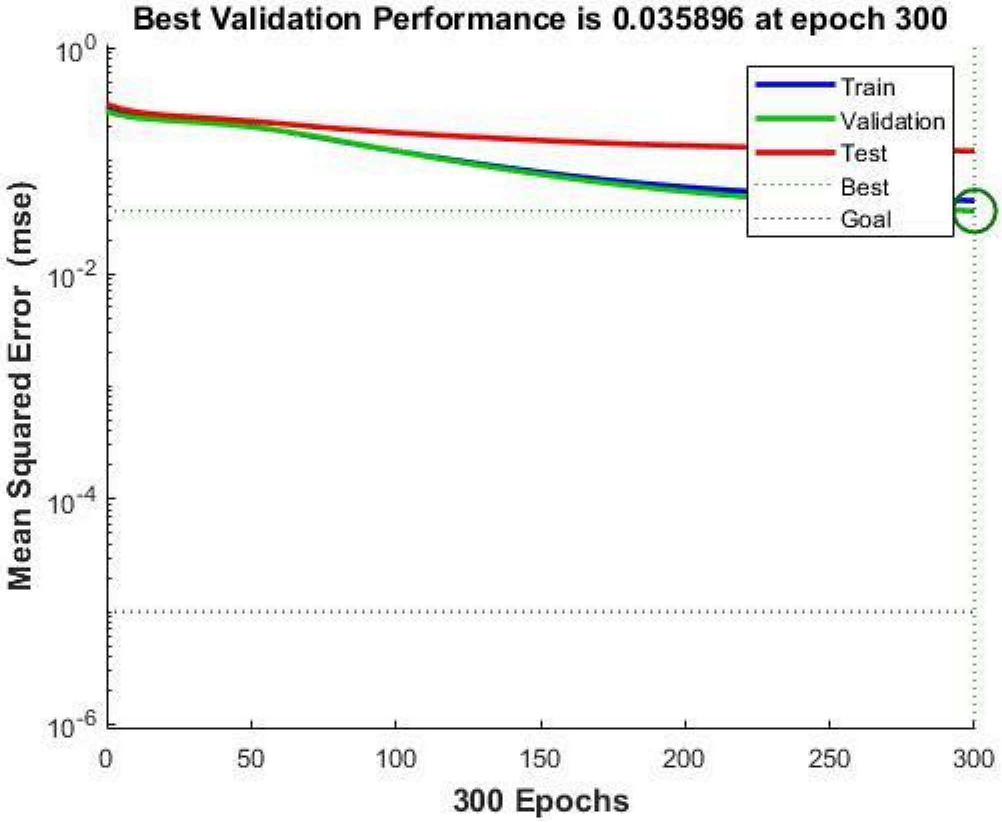
✓ Maximum epoch reached.

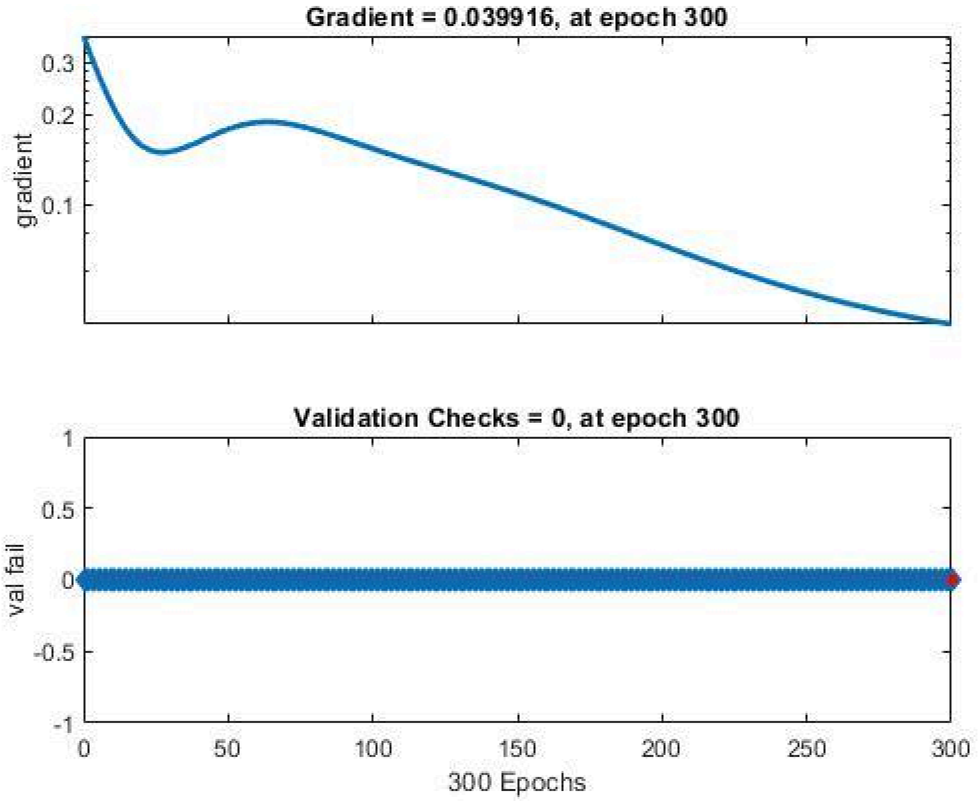


Confusion Matrix

	0	1	
0	70 50.0%	5 3.6%	93.3% 6.7%
1	0 0.0%	65 46.4%	100% 0.0%
	100% 0.0%	92.9% 7.1%	96.4% 3.6%

Output Class
Target Class


Best Validation Performance is 0.035896 at epoch 300
Mean Squared Error (mse)
10^{0}
10^{-2}
10^{-4}
10^{-6}
0
50
100
150
200
250
300
300 Epochs
Train
Validation
Test
Best
Goal


Gradient = 0.039916, at epoch 300
gradient
0.3
0.2
0.1
Validation Checks = 0, at epoch 300
val fail
1
0.5
0
-0.5
-1
0
50
100
150
200
250
300
300 Epochs

Conclusion

In this code we observed graphs of performance, calculated and updated biases, also found final weights after upgradation. Also ploted gradient descent graph in figure 5. And simulated a test data and obtained desired output.

We plotted confusion matrix which shows 4 A samples getting misclassified and B samples getting 100% correctly classified. We also took input as transpose because it is not a feedforward network.

Published with MATLAB® R2020a