Neuron models and basic learning rules

Project-1

Team Members

Atsushi Oba *(s1230128)*

Hitoshi Ikuta *(m5221106)*

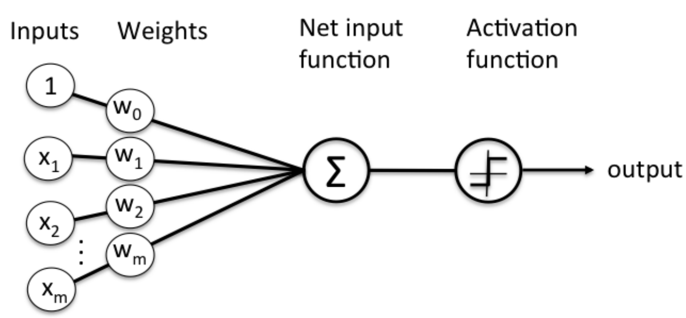
Chowdhury Md Intisar *(m5212106)*

Yunosuke Teshimu *(m5221151)*

Neural Network

Neural network is a coordinative system which is composed of one or more neurons working collectively. A neural work is analogous to our brain. Thus neural network is also called artificial neural network.

In our given project and assignment, we required to study two different kind of learning method for neural network. 1) Delta learning. and 2) Perceptron learning. **Perceptron learning** is the simplest learning method for neural network which works as follows.



The next input function is calculated as

$$z = \sum_{i \in N} (w[i] * x[i]) + bias \tag{1}$$

Here "w" vector is the weight associated with the input vector "x". N is the number of the features of any particular object or entity. Thus the prediction is made as follows.

$$f(x) = \begin{cases} 1.0 & z > 0 \\ 0\,.0 & z < 0 \end{cases}$$

Whereas, in the case of delta learning instead of using step function as activation we use a continuous function. Such as, sigmoid, tanh etc as follows.

$$sigmoid(x) = \frac{1}{1+e^{-x}} \qquad (2)$$

$$\tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \qquad (3)$$

Project -1 Part -1 (Perceptron Learning)

For implementation of project-1 part-1 we have adopted the given code and modified it in certain places. The source code has been attached with the file in a separate folder.

The part 1 of project 1 required to implement ***AND gate*** with perceptron learning. A perceptron represents a hyperplane decision surface in the n-dimensional space of instance. It is to be noted that, some of the classification problem cannot be carried out with the perceptron learning. Perceptron learning can be employed in linearly separable situation. Here, AND gate problem is linearly separable. Although, codes have been provided as separate file, we have included few places of the code where we have made changes to implement the perceptron learning.

```
double x[n_sample][I]={
  { 0, 0, -1},
  { 0, 1, -1},
  { 1, 0, -1},
  { 1, 1, -1},

};

double w[I];
double d[n_sample]={0, 0, 0, 1};
```

Figure 1. Input to the Network

We had to make a slight modification in the teachers signal input. We have replaced all the training label -1 with 0. Since, our model get stuck in local optima and infinite loop with -1. But, after replacing with 0 the model trains with a less error within a reasonable amount of epoch.

```
42      bias += delta*eta*1.0;
43        for(i=0;i<I;i++){
44      delta=(d[p]-o);
45      w[i]+=eta*delta*x[p][i];
46        }
```

Figure 2. Perceptron learning delta calculation

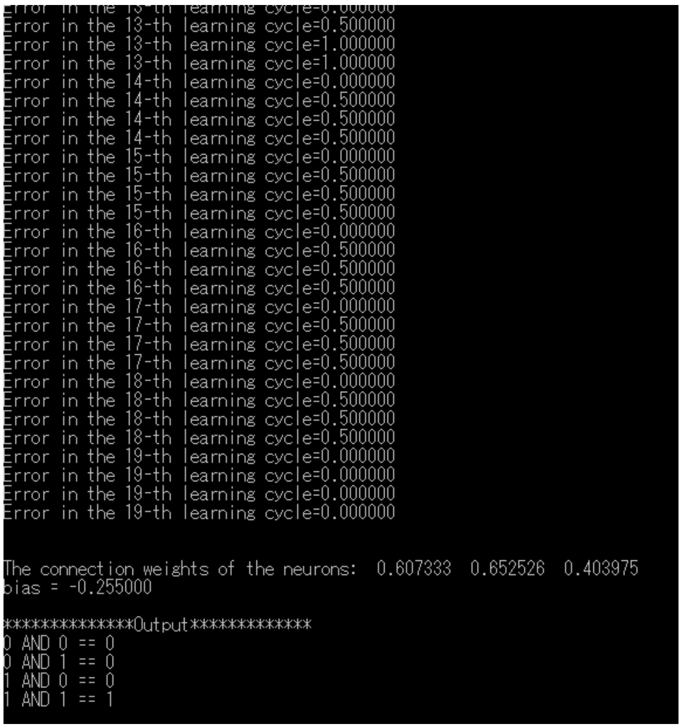
In figure 2. We have depicted how we calculate the delta at line number 44. We have also trained our bias to adjust in each epoch shown in line number 42.

```
void FindOutput(int p){
  int    i;
  double temp=0;
  for(i=0;i<I;i++) temp += w[i]*x[p][i];

  temp += bias;
  if ( temp < 0) o = 0;
  else o = 1.0;
}
```

Figure 3. The activation function

Program output and Result discussion for **Perceptron** learning


```
Error in the 13-th learning cycle=0.500000
Error in the 13-th learning cycle=1.000000
Error in the 13-th learning cycle=1.000000
Error in the 14-th learning cycle=0.000000
Error in the 14-th learning cycle=0.500000
Error in the 14-th learning cycle=0.500000
Error in the 14-th learning cycle=0.500000
Error in the 15-th learning cycle=0.000000
Error in the 15-th learning cycle=0.500000
Error in the 15-th learning cycle=0.500000
Error in the 15-th learning cycle=0.500000
Error in the 16-th learning cycle=0.000000
Error in the 16-th learning cycle=0.500000
Error in the 16-th learning cycle=0.500000
Error in the 16-th learning cycle=0.500000
Error in the 17-th learning cycle=0.000000
Error in the 17-th learning cycle=0.500000
Error in the 17-th learning cycle=0.500000
Error in the 17-th learning cycle=0.500000
Error in the 18-th learning cycle=0.000000
Error in the 18-th learning cycle=0.500000
Error in the 18-th learning cycle=0.500000
Error in the 18-th learning cycle=0.500000
Error in the 19-th learning cycle=0.000000
Error in the 19-th learning cycle=0.000000
Error in the 19-th learning cycle=0.000000
Error in the 19-th learning cycle=0.000000


The connection weights of the neurons:  0.607333  0.652526  0.403975
bias = -0.255000

**************Output*************
0 AND 0 == 0
0 AND 1 == 0
1 AND 0 == 0
1 AND 1 == 1
```

Figure 4. Output

First Run

Parameter setting

Eta = 0.005 (Lowered the learning rate initially to observe its effect)

Weight = 0.607333 **||** 0.652526 **||** 0.403975

Bias = **-**0.25 (Output by the network)

Total epoch to reach desired error rate = 19

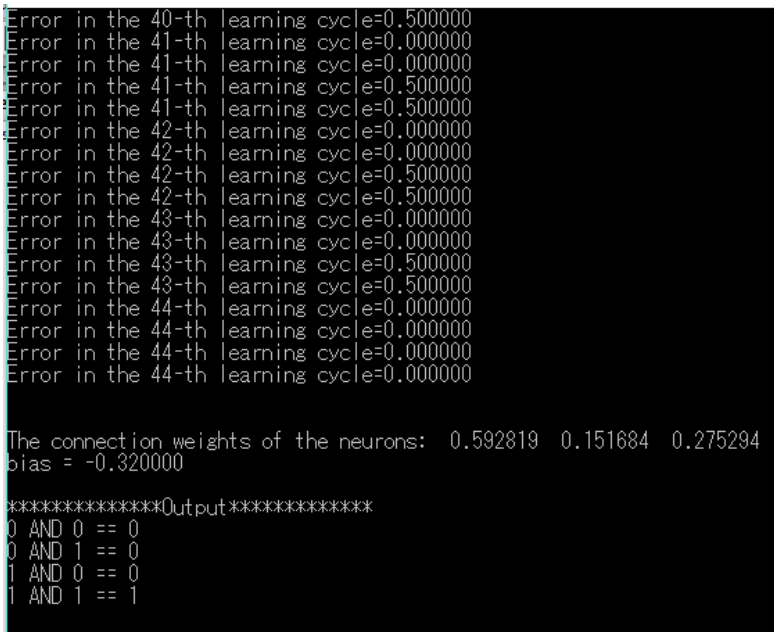


Figure 5. Output

Second Run

Parameter setting

Eta = 0.005 (Lowered the learning rate initially to observe its effect)

Weight = 0.592819 **||** 0.151684 **||** 0.275294

Bias = -0.32

Total epoch to reach desired error rate = 44

Although, we have run the same code with same setting different time, we obtain slightly different weight, bias and epoch rate. This is due to the fact that; we are initializing the weight vector with different numbers in each run.

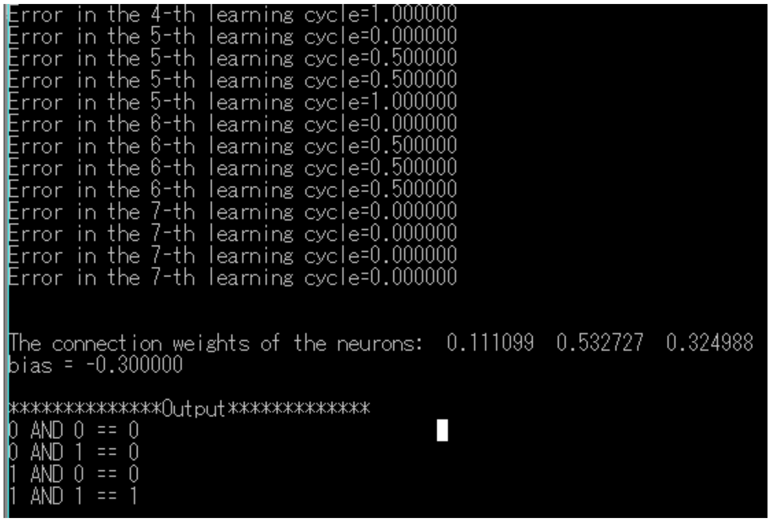


Figure 6. Output

Third Run

Parameter setting changed

Eta = 0.5 (Increased learning rate)

Weights = 0.111099 || 0.532727 || 0.324988

Bias = -0.30 (Derived by the network)

Total epoch to reach desired error rate = 7

Thus, we can see that the learning rate has a great effect on the speed of learning. With large learning rate the network learns faster and converges to the solution faster. But, too large learning rate can skip the optimal solution or global maxima/minima. So, learning rate should be selected and tuned carefully.

Project -1 Part -1 (Delta Learning)

There are many difference between delta learning rule and perceptron learning rule while we implement the same problem using delta learning. First of all, delta learning doesn't use discrete neural. It used continuous neuron, thus the ***activation function is differentiable.*** Delta rule can also be used for cases where data is not linearly separable. Thus, delta learning is slightly better than perceptron learning. But, **it is to be noted that since delta learning doesn't use any threshold like perceptron, its convergences and termination is not always guaranteed**. Delta learning converges towards minimum error asymptotically. The differentiable activation number gives us an opportunity to apply gradient decent on the minimizing the error with respect to the weight vector.

The gradient descent is used for hypothesis space as follows.

Let error measured is

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \qquad (4)$$

Thus our aim is to minimize the function E($\vec{w}$) with respect to weight. Below is a graphical view.

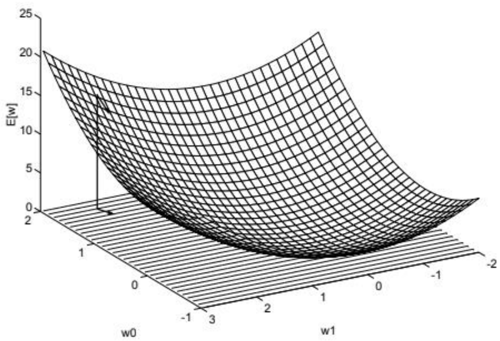


Figure. 7 Weight optimization for error minimization

The formula for gradient descent weight update is

$$\Delta w = eta * (t - o)x_i \ where \ E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \qquad (5)$$

Program output and Result discussion for **Delta** learning

```
56          Error+=0.5*pow(d[p]-o,2.0);
57          for(i=0;i<I;i++){
58      delta=(d[p]-o)*(1-o*o);
59      w[i]+=eta*delta*x[p][i];
60          }
```

Figure 8. Delta learning

From figure 8. Line number 58 we can see delta has been calculated according to the equation 5.

```
double x[n_sample][I]={
  { 0, 0, -1},
  { 0, 1, -1},
  { 1, 0, -1},
  { 1, 1, -1},

};

double w[I];
double d[n_sample]={-1, -1, -1, 1};
```

Figure 9. Input data and label

Figure 9. depicts the input data and label for the given input data.

```
#define tanh(x)     ( (exp(lambda*x) - exp(-lambda*x) ) / (exp(lambda*x) + exp(-lambda*x) ) )
```

Figure 10. Implementation of tanh

We have tested the code using two activation function.

- First we have used **sigmoid** function for the activation function, which was already given in the code.
- Second, as extension we have used ***tanh*** function for comparing the performance.

First run (Sigmoid function)

Parameter Setting

Eta = 0.5

Weights = 6.294421 || 6.287382 || 9.513146

Total epoch to reach desired error rate = 474

```
Error in the 430-th Epoch =0.011054
Error in the 431-th Epoch =0.011027
Error in the 432-th Epoch =0.011001
Error in the 433-th Epoch =0.010974
Error in the 434-th Epoch =0.010948
Error in the 435-th Epoch =0.010922
Error in the 436-th Epoch =0.010896
Error in the 437-th Epoch =0.010870
Error in the 438-th Epoch =0.010844
Error in the 439-th Epoch =0.010818
Error in the 440-th Epoch =0.010793
Error in the 441-th Epoch =0.010767
Error in the 442-th Epoch =0.010742
Error in the 443-th Epoch =0.010717
Error in the 444-th Epoch =0.010692
Error in the 445-th Epoch =0.010667
Error in the 446-th Epoch =0.010642
Error in the 447-th Epoch =0.010617
Error in the 448-th Epoch =0.010592
Error in the 449-th Epoch =0.010568
Error in the 450-th Epoch =0.010543
Error in the 451-th Epoch =0.010519
Error in the 452-th Epoch =0.010495
Error in the 453-th Epoch =0.010471
Error in the 454-th Epoch =0.010447
Error in the 455-th Epoch =0.010423
Error in the 456-th Epoch =0.010399
Error in the 457-th Epoch =0.010375
Error in the 458-th Epoch =0.010352
Error in the 459-th Epoch =0.010328
Error in the 460-th Epoch =0.010305
Error in the 461-th Epoch =0.010282
Error in the 462-th Epoch =0.010258
Error in the 463-th Epoch =0.010235
Error in the 464-th Epoch =0.010212
Error in the 465-th Epoch =0.010190
Error in the 466-th Epoch =0.010167
Error in the 467-th Epoch =0.010144
Error in the 468-th Epoch =0.010122
Error in the 469-th Epoch =0.010099
Error in the 470-th Epoch =0.010077
Error in the 471-th Epoch =0.010055
Error in the 472-th Epoch =0.010033
Error in the 473-th Epoch =0.010010
Error in the 474-th Epoch =0.009988


The connection weights of the neurons:
6.294421 6.287382 9.513146

0 AND 0 == -0.9999
0 AND 1 == -0.9236
1 AND 0 == -0.9231
1 AND 1 == 0.9112
```

Figure 11. Output by sigmoid function

Second Run with tanh function

Parameter Setting

Eta = 0.5

Weights = 3.160431 **||** 3.153389 **||** 4.767271

Total epoch to reach desired error rate = 224

```
Error in the 176-th Epoch =0.012690
Error in the 177-th Epoch =0.012619
Error in the 178-th Epoch =0.012549
Error in the 179-th Epoch =0.012479
Error in the 180-th Epoch =0.012410
Error in the 181-th Epoch =0.012342
Error in the 182-th Epoch =0.012274
Error in the 183-th Epoch =0.012208
Error in the 184-th Epoch =0.012142
Error in the 185-th Epoch =0.012076
Error in the 186-th Epoch =0.012012
Error in the 187-th Epoch =0.011948
Error in the 188-th Epoch =0.011884
Error in the 189-th Epoch =0.011822
Error in the 190-th Epoch =0.011760
Error in the 191-th Epoch =0.011699
Error in the 192-th Epoch =0.011638
Error in the 193-th Epoch =0.011578
Error in the 194-th Epoch =0.011519
Error in the 195-th Epoch =0.011460
Error in the 196-th Epoch =0.011402
Error in the 197-th Epoch =0.011344
Error in the 198-th Epoch =0.011287
Error in the 199-th Epoch =0.011230
Error in the 200-th Epoch =0.011174
Error in the 201-th Epoch =0.011119
Error in the 202-th Epoch =0.011064
Error in the 203-th Epoch =0.011010
Error in the 204-th Epoch =0.010956
Error in the 205-th Epoch =0.010903
Error in the 206-th Epoch =0.010850
Error in the 207-th Epoch =0.010798
Error in the 208-th Epoch =0.010746
Error in the 209-th Epoch =0.010695
Error in the 210-th Epoch =0.010644
Error in the 211-th Epoch =0.010594
Error in the 212-th Epoch =0.010544
Error in the 213-th Epoch =0.010495
Error in the 214-th Epoch =0.010446
Error in the 215-th Epoch =0.010397
Error in the 216-th Epoch =0.010349
Error in the 217-th Epoch =0.010302
Error in the 218-th Epoch =0.010255
Error in the 219-th Epoch =0.010208
Error in the 220-th Epoch =0.010162
Error in the 221-th Epoch =0.010116
Error in the 222-th Epoch =0.010070
Error in the 223-th Epoch =0.010025
Error in the 224-th Epoch =0.009981


The connection weights of the neurons:
3.160431 3.153389 4.767271

0 AND 0 == -0.9999
0 AND 1 == -0.9237
1 AND 0 == -0.9227
1 AND 1 == 0.9132
```

Figure 11. Output by tanh function.

Key Observation from the result.

As we can observe delta learning is converging slowly to the solution relative to perceptron, despite the fact that all the hyper parameters are same in both learning. On the other hand, tanh has performed relatively better (224 epochs for tanh) than sigmoid function (447 epochs for sigmoid).

Project -1 Part -2 (Discrete Neuron)

A discrete neuron is analogous to the neuron used in perceptron learning algorithm, that is, a discrete neuron only discrete values (0 or 1/ 1 or -1). Thus the given code has been changed according to the rule of perceptron learning to implement discrete neuron.

In project-1 part-2 we have multiple output neuron. But it is required to train each discrete neuron using perceptron. The input and output array has been changed accordingly as follows.

```

double x[n_sample][N] = {
        { 10, 2, -1},
        { 2, -5, -1},
        { -5, 5, -1},
};

double d[n_sample][R] = {
        { 1, 0, 0},
        { 0, 1, 0},
        { 0, 0, 1},
};
```

Figure 12. Input to single layer discrete neuron.

```
void FindOutput(int index) {

    int i,j;
    double tot=0.0;

    for(i = 0; i < R; i++) {
        tot = 0.0;
        for(j = 0; j < N; j++) {
            tot += w[i][j] * x[index][j];
        }
        tot += bias;
        if ( tot >= 0)
            o[i] = 1;
        else
            o[i] = 0;
    }
}
```

Figure 13. Activation calculation of discrete neurons.

First Run

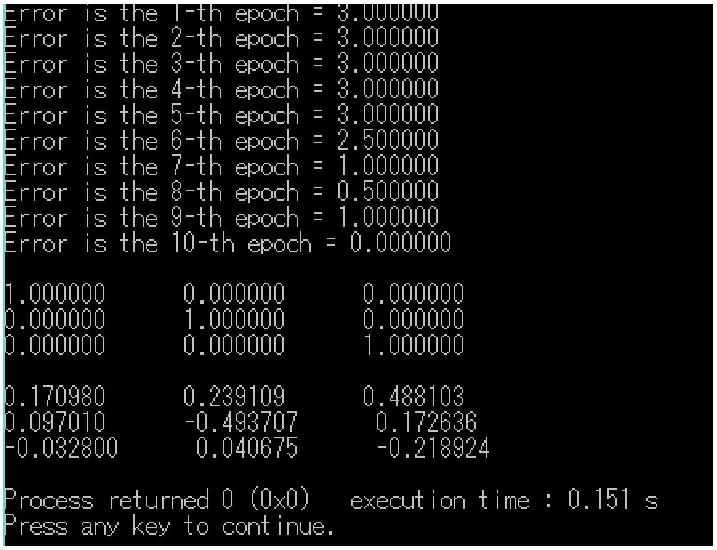
Parameter Setting

Eta = 0.05

Weights -

0.170980	0.239109	0.488103
0.097010	-0.493707	0.172636
-0.032800	0.040675	-0.218924

Total epoch to reach desired error rate = 10



```
Error is the 1-th epoch = 3.000000
Error is the 2-th epoch = 3.000000
Error is the 3-th epoch = 3.000000
Error is the 4-th epoch = 3.000000
Error is the 5-th epoch = 3.000000
Error is the 6-th epoch = 2.500000
Error is the 7-th epoch = 1.000000
Error is the 8-th epoch = 0.500000
Error is the 9-th epoch = 1.000000
Error is the 10-th epoch = 0.000000

1.000000      0.000000      0.000000
0.000000      1.000000      0.000000
0.000000      0.000000      1.000000

0.170980      0.239109      0.488103
0.097010      -0.493707      0.172636
-0.032800      0.040675      -0.218924

Process returned 0 (0x0)   execution time : 0.151 s
Press any key to continue.
```

Figure 14. Discrete Single layer neuron output

Project -1 Part -2 (Continuous Neuron)

Continuous neuron is composed on continuous function for the activation. Since, continuous function is differentiable thus delta learning rule can be implemented for function neuron. For making a neuron continuous we can use sigmoid function, tanh function, Gaussian, sinusoidal and etc. In our experiment we have used sigmoid (by default given in the code) and tanh function.

First Run ***(Sigmoid activation function)***

Parameter Setting

Eta = 0.05

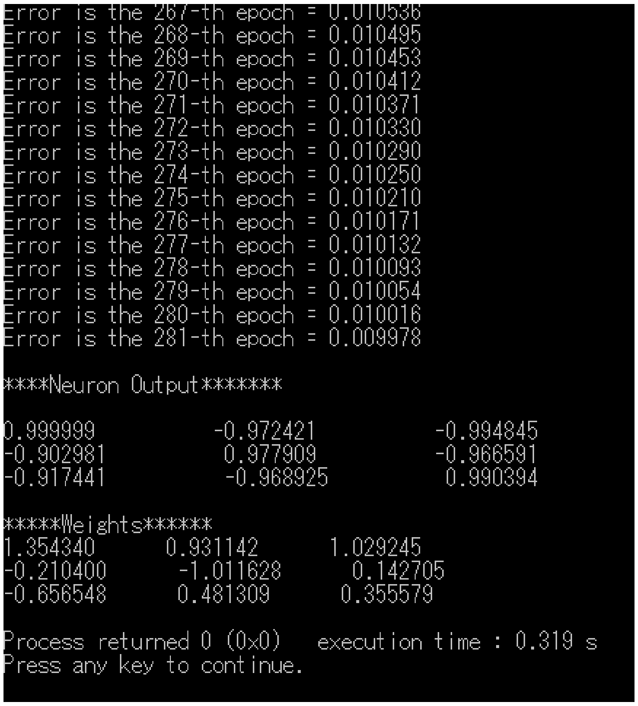
Weights -

1.354340   0.931142   1.029245

-0.210400   -1.011628   0.142705

-0.656548   0.481309   0.355579

Total epoch to reach desired error rate = 281


```
Error is the 267-th epoch = 0.010536
Error is the 268-th epoch = 0.010495
Error is the 269-th epoch = 0.010453
Error is the 270-th epoch = 0.010412
Error is the 271-th epoch = 0.010371
Error is the 272-th epoch = 0.010330
Error is the 273-th epoch = 0.010290
Error is the 274-th epoch = 0.010250
Error is the 275-th epoch = 0.010210
Error is the 276-th epoch = 0.010171
Error is the 277-th epoch = 0.010132
Error is the 278-th epoch = 0.010093
Error is the 279-th epoch = 0.010054
Error is the 280-th epoch = 0.010016
Error is the 281-th epoch = 0.009978

****Neuron Output*******

0.999999          -0.972421          -0.994845
-0.902981          0.977909          -0.966591
-0.917441          -0.968925          0.990394

*****Weights******
1.354340      0.931142      1.029245
-0.210400      -1.011628      0.142705
-0.656548      0.481309      0.355579

Process returned 0 (0x0)   execution time : 0.319 s
Press any key to continue.
```

Figure 15. Continuous single layer neuron output(Sigmoid)

***Second Run** (tanh activation function)*

Parameter Setting

Eta = 0.05

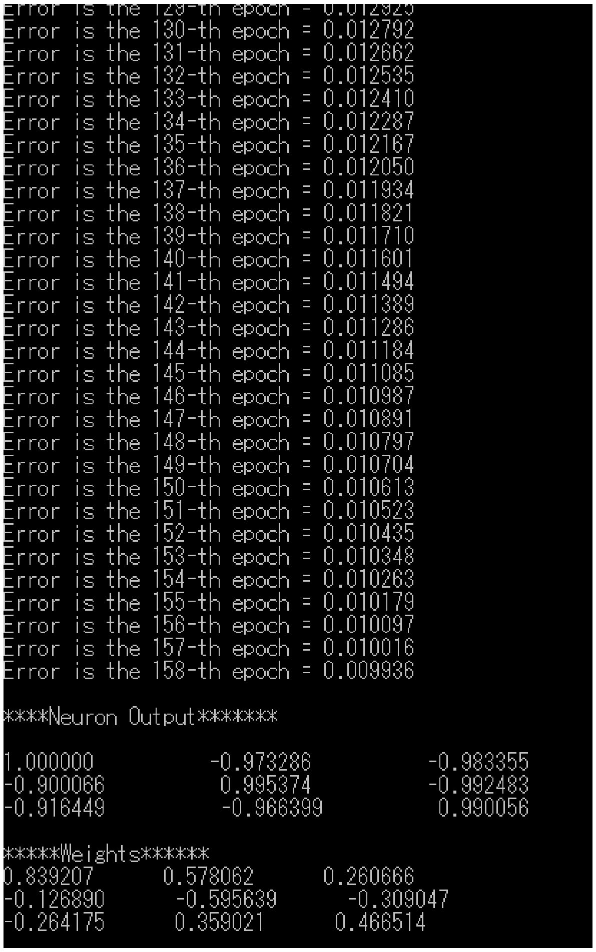
Weights –

0.839207 0.578062 0.260666

-0.126890 -0.595639 -0.309047

-0.264175 0.359021 0.466514

Total epoch to reach desired error rate = 158



```
Error is the 129-th epoch = 0.012925
Error is the 130-th epoch = 0.012792
Error is the 131-th epoch = 0.012662
Error is the 132-th epoch = 0.012535
Error is the 133-th epoch = 0.012410
Error is the 134-th epoch = 0.012287
Error is the 135-th epoch = 0.012167
Error is the 136-th epoch = 0.012050
Error is the 137-th epoch = 0.011934
Error is the 138-th epoch = 0.011821
Error is the 139-th epoch = 0.011710
Error is the 140-th epoch = 0.011601
Error is the 141-th epoch = 0.011494
Error is the 142-th epoch = 0.011389
Error is the 143-th epoch = 0.011286
Error is the 144-th epoch = 0.011184
Error is the 145-th epoch = 0.011085
Error is the 146-th epoch = 0.010987
Error is the 147-th epoch = 0.010891
Error is the 148-th epoch = 0.010797
Error is the 149-th epoch = 0.010704
Error is the 150-th epoch = 0.010613
Error is the 151-th epoch = 0.010523
Error is the 152-th epoch = 0.010435
Error is the 153-th epoch = 0.010348
Error is the 154-th epoch = 0.010263
Error is the 155-th epoch = 0.010179
Error is the 156-th epoch = 0.010097
Error is the 157-th epoch = 0.010016
Error is the 158-th epoch = 0.009936

****Neuron Output*******

1.000000          -0.973286          -0.983355
-0.900066          0.995374          -0.992483
-0.916449          -0.966399          0.990056

*****Weights******
0.839207      0.578062      0.260666
-0.126890      -0.595639      -0.309047
-0.264175      0.359021      0.466514
```

Figure 16. Continuous single layer neuron output (tanh)

Conclusion

From the above experiment with the both ***delta learning of continuous neuron and perceptron learning of discrete learning*** we can conclude following key points

- Perceptron learning converges to the solution faster than
- Perceptron learning is for linearly separable problem, but adding additional neuron/hidden layer can enable non-linear separability.
- Perceptron uses threshold.
- Delta learning converges slowly towards the solution, although increasing the learning rate can accelerate the speed of learning, but in this case we have a risk of missing the global minima. So learning rate has to be chosen carefully.
- Termination of delta learning is not guaranteed, since it progresses towards solution asymptotically.

References

[1] https://towardsdatascience.com/neural-representation-of-logic-gates-df044ec922bc

[2] http://web-ext.u-aizu.ac.jp/~qf-zhao/TEACHING/NN-I/nn-1.html

[3] http://neuralnetworksanddeeplearning.com/chap1.html