Neuron models and

basic learning rules

Project-1

***Team Members***

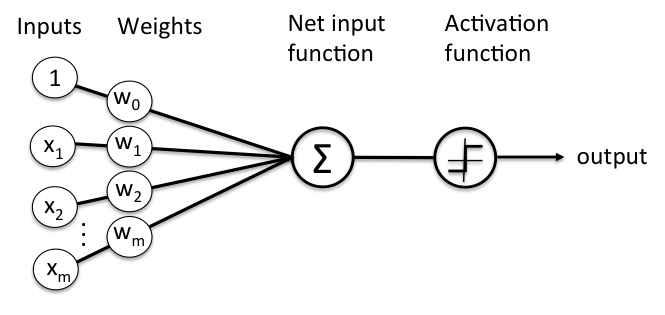
**Atsushi Oba *(s1230128)* Hitoshi Ikuta *(m5221106)***

**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

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.

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.

## 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.

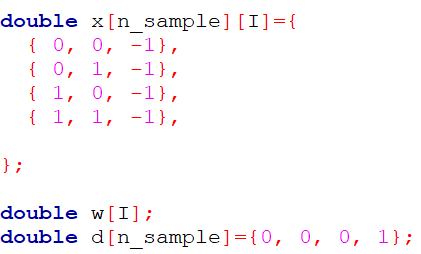


Figure 1. Input to the Network

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.

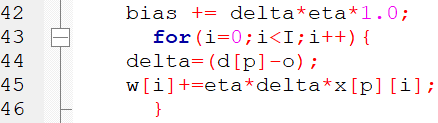


Figure 2. Perceptron learning delta calculation

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.

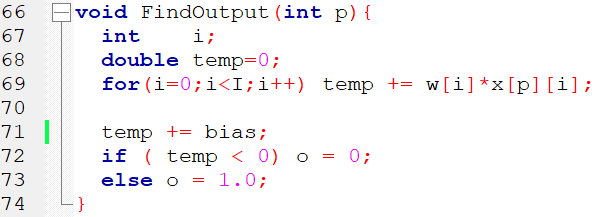


Figure 3. The activation function

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

# 

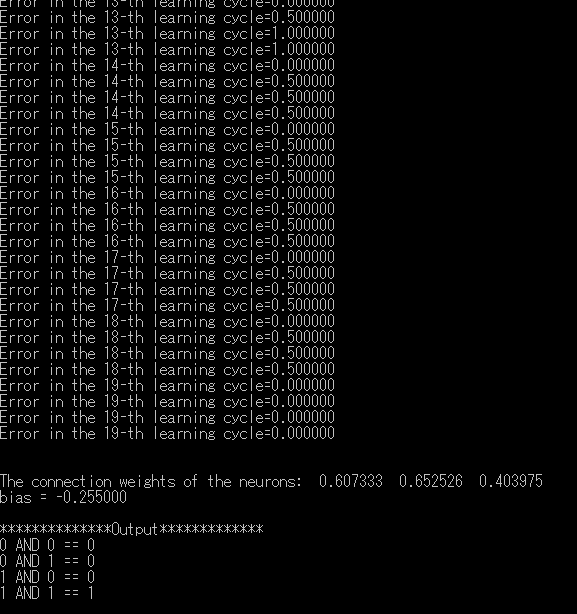


Figure 4. Output

***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

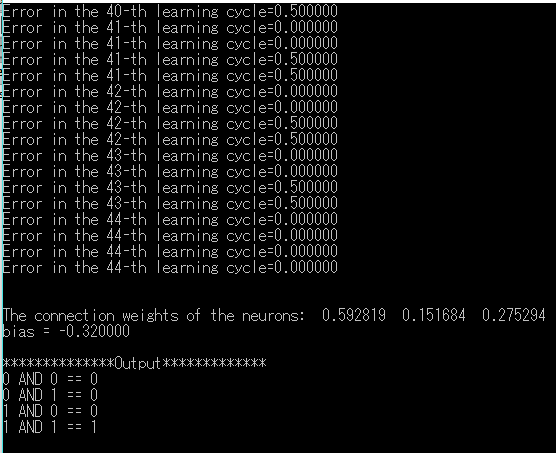
**

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.

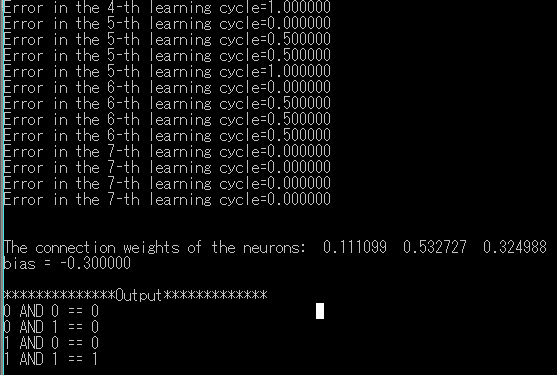


Figure 6. Output

*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

(4)

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

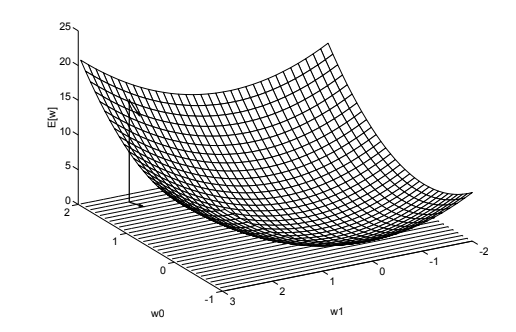


Figure. 7 Weight optimization for error minimization

The formula for gradient descent weight update is

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

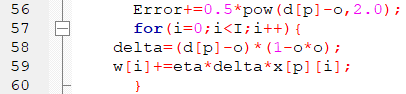


Figure 8. Delta learning

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

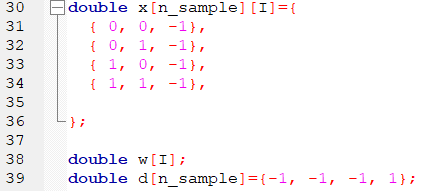


Figure 9. Input data and label

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



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

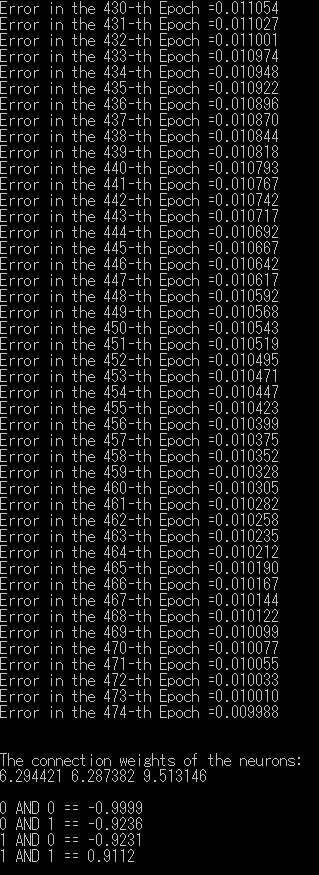


Figure 11. Output by sigmoid function

***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

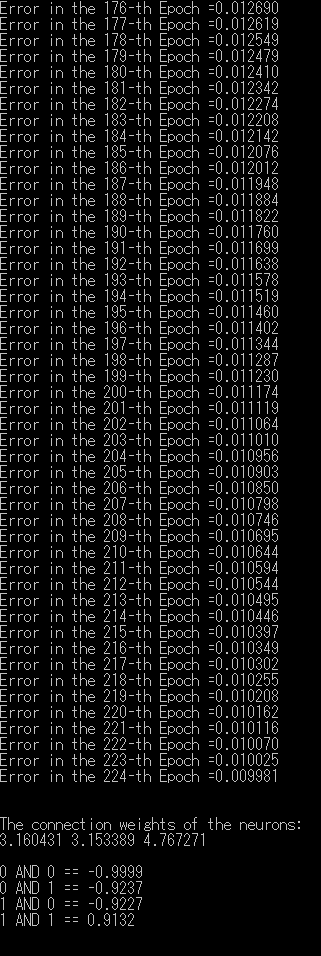


Figure 11. Output by tanh function.

**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.

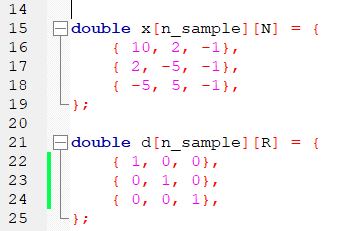


Figure 12. Input to single layer discrete neuron.

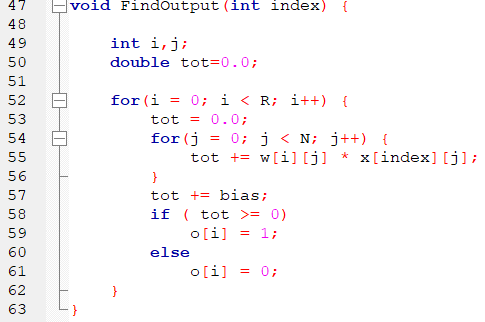


Figure 13. Activation calculation of discrete neurons.

***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

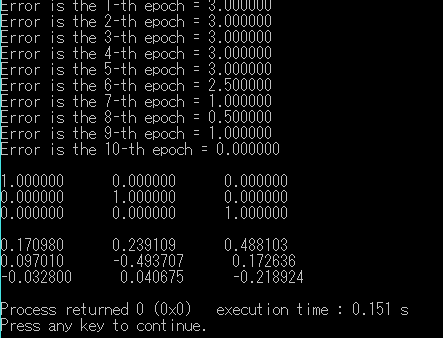


Figure 14. Discrete Single layer neuron output

## 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

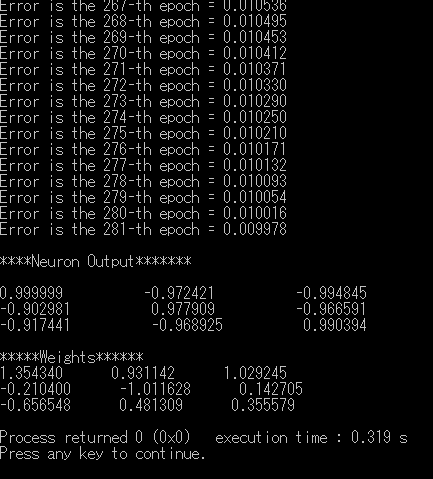


Figure 15. Continuous single layer neuron output(Sigmoid)

***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

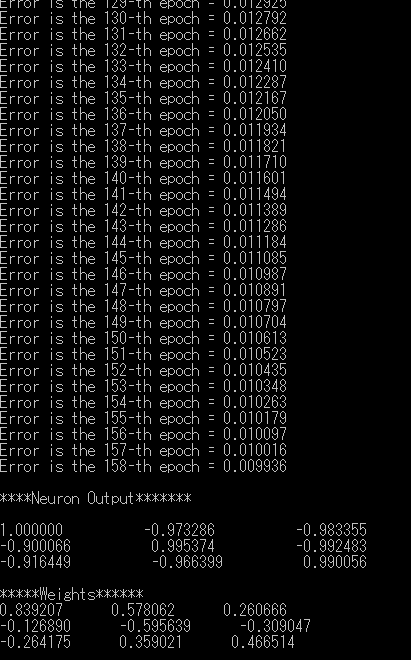


Figure 16. Continuous single layer neuron output (tanh)

# 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>