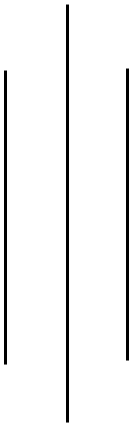
TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

A
LAB REPORT
ON

ARTIFICIAL NEURAL NETWORKS



SUBMITTED BY:

Diwas Adhikari **(076BEI014)**

SUBMITTED TO:

Department of Electronics & Computer Engineering
Pulchowk Campus, IOE

LAB NUMBER

5

SUBMISSION DATE

July 26, 2023

(For course completion of BEI – Artificial Intelligence)

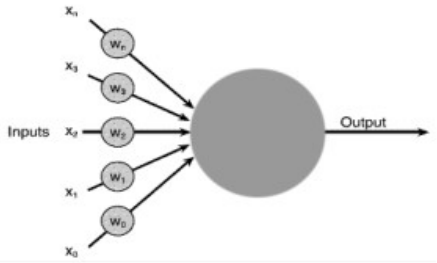
OBJECTIVES

- To understand the working of artificial neural networks (ANN)
- To train neural networks for the operation of basic logic gates.

THEORY

Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the biological nervous system which can process information based on the sensory inputs that it receives from its surrounding. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.

Artificial Neuron
An artificial neuron is a device with many inputs and one output. The neuron has two modes of operation; the training mode and the using mode. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not.



Adaptive Learning
Adaline (Adaptive Linear Neuron) is a type of single-layer neural network. It consists of an input layer, a single processing neuron, and a single output layer. The algorithm works by adjusting the weights on the input connections to the neuron to minimize the difference between the predicted output and the actual output. Adaline uses a linear activation function, which means that the output of the neuron is a weighted sum of the inputs.

Backpropagation
Backpropagation is a more general algorithm that is used to train multi-layer neural networks. It works by propagating the error from the output layer back through the network, adjusting the weights on the connections between the neurons to minimize the error. Backpropagation uses a non-linear activation function, such as the sigmoid or ReLU function, which allows for the modeling of more complex relationships between the inputs and outputs.

Algorithm for Adaline Learning

The training is done using the delta rule which is also known as the least mean squares (LMS) or Widrow-Hoff rule.

1. Initialize the weights and bias (b) to small random values and select a learning rate(a).
2. For each input vector s, set up its target output.
3. Compute the neuron inputs : $y_{in} = b + \sum (x_i w_i)$
4. Use the delta rule to update the bias and weights:

$$b\,(new) = b\,(old) + a*(t - y_{in})$$
$$w_i\,(new) = w_i\,(old) + a*(t - y_{in})*x_i$$

5. Stop if the largest weight change across all the training samples is less than a specified tolerance, otherwise cycle through the training set again.

Algorithm for Madaline Learning with error backpropagation

The backpropagation algorithm provides a computational efficient method for training multi-layer networks. This is also known as madaline (many-adaline) learning.

1. Initialize the weights to small random values and set up a learning rate(a).
2. Feed the training sample through the network and determine the targets.
3. Compute the error for each output unit, for unit 'k':

$$\delta_k = (t_k - y_k) * f'(y_{in}{}^k)$$

4. Propagate the delta error back through the weights of the hidden units where the delta input for the j^{th} hidden unit is: training is done using the delta rule which is:

$$\delta_{in}{}^j = \sum (\delta_k\, w_{jk})$$
$$\delta^{\,j} = \delta_{in}{}^j * f'(y_{in}{}^j)$$

5. Calculate the weight correction term for each output unit for unit 'k':

$$\Delta w_{jk} = a* \delta_k * z^{\,j}$$

6. Calculate the weight correction term for the hidden units:

$$\Delta w_{ij} = a* \delta_j * z^{\,i}$$

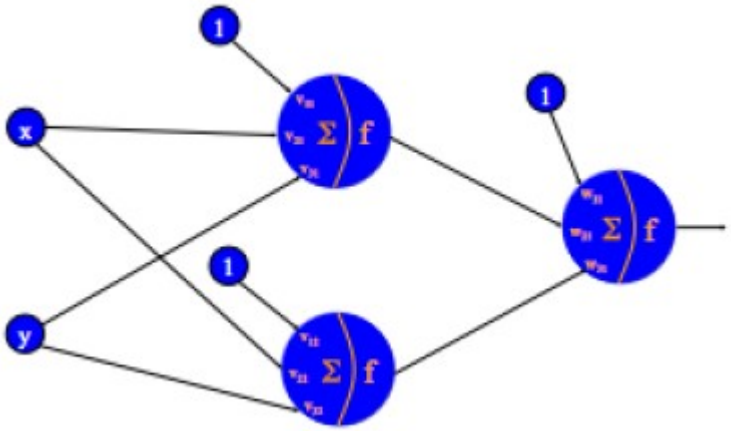
7: Update the weights respectively:

$$w_{ij}\,(new) = w_{ij}\,(old) + \Delta w_{ij}$$
$$w_{jk}\,(new) = w_{jk}\,(old) + \Delta w_{jk}$$

5. Stop the training based on maximum epochs or, keeping the change in weights to a specified tolerance.

Multilayer perceptron network for XOR Gate

Hidden Layer: 2 neurons
Output Layer: 1 neuron



PYTHON CODE & OBSERVATION

1) Adaline Learning

```
import numpy as np ;

# Set up inputs and outputs of AND gate for training the neuron
inputs = [[0,0], [0,1], [1,0], [1,1]] ;
print('Inputs: ', inputs) ;

outputs = np.zeros(len(inputs)) ;
for i in range(len(inputs)):
    if inputs[i][0] == 1 and inputs[i][1] == 1:
        outputs[i] = 1 ;
    else:
        outputs[i] = 0 ;
print('Target outputs: ', outputs) ;

# Initialize weights to random values and select a learning rate
rate = 0.1 ;
tolerance = 0.0001 ;
weights = np.random.randn(2) ;
print('Initial weights: ', weights) ;
bias = np.random.randn(1) ;
print('Initial Bias: ', bias) ;

# Update the bias and weights using delta rule and stop the training in case of
largest weight change is less than the specified tolerance
j = 0 ;
switch = True ;
prev_ms_error = 0 ;
while switch:
    print('Epoch', (j+1)) ;
    print('------------------------------------------------------------') ;
    ms_error = 0 ;
    for i in range(len(inputs)):
        u = np.dot(inputs[i], weights) + bias ;
        error = outputs[i] - u ;
        weights += rate * error * inputs[i] ;
        bias += rate * error ;
        ms_error += np.square(error) ;
        print('Weights: ', weights, 'Bias: ', bias) ;
    if np.abs(ms_error - prev_ms_error) < tolerance:
        switch = False ;
    prev_ms_error = ms_error ;
    print('MSE: ', ms_error) ;
    j += 1 ;
```

```
# Predict outputs from neuron after weight adjustment
threshold = 0.5 ;
predicted_outputs = np.zeros(len(inputs)) ;
for i in range(len(inputs)):
    predicted_outputs[i] = np.dot(weights, inputs[i]) + bias ;
    if predicted_outputs[i] > threshold:
        predicted_outputs[i] = 1 ;
    else:
        predicted_outputs[i] = 0 ;
print('Predicted outputs: ', outputs) ;
```

<The same code can be used for every logic gate by feeding their respective inputs except XOR and XNOR gates which can't be built by a single perceptron>

2) Madaline Learning with error backpropagation (Multi-layer perceptron)

```
# Madaline Learning for XOR Neural Network
import numpy as np ;

# Set up inputs and outputs of XOR gate for training the network
inputs = [[0,0], [0,1], [1,0], [1,1]] ;
print('Inputs: ', inputs) ;
outputs = np.zeros(len(inputs)) ;
for i in range(len(inputs)):
    if inputs[i][0] == inputs[i][1]:
        outputs[i] = 0 ;
    else:
        outputs[i] = 1 ;
print('Target outputs: ', outputs) ;

# Initialize weights to random values and select a learning rate
# A hidden layer must be introduced for classifying the targets of XOR gate.
# Assume that hidden layer has two neurons with an output layer of one neuron
rate = 0.1 ;
tolerance = 0.0001 ;
hiddenLayerWeights = np.random.randn(2,2) ;
outputLayerWeights = np.random.randn(2) ;
print('Initial weights') ;
print('----------------------------------------')
print('Hidden Layer: \n', hiddenLayerWeights) ;
print('Output Layer: ', outputLayerWeights) ;

# Define the activation function - sigmoid function
def sigmoid(x):
    return 1 / (1 + np.exp(-x)) ;

def derv_sigmoid(x):
    return x*(1 - x) ; # 'x' should already be sigmoided first !
```

```
# Train weights accordingly and propagate error backwards to adjust the weights
epochs = 100 ;
du = np.zeros(2) ;
for e in range(epochs):
    print('Epoch', (e+1)) ;
    print('-----------------------------------------------------------') ;
    ms_error = 0 ;
    for i in range(len(inputs)):
        # Feed forward
        u1 = sigmoid(np.dot(inputs[i], hiddenLayerWeights[0][:])) ;
        u2 = sigmoid(np.dot(inputs[i], hiddenLayerWeights[1][:])) ;
        u13 = np.dot(u1, outputLayerWeights[0]) ;
        u23 = np.dot(u2, outputLayerWeights[1]) ;
        u3 = sigmoid(u13 + u23) ;
        error = outputs[i] - u3 ;
        # Error backpropagation
        derror = error * derv_sigmoid(u3) ;
        du[0] = derror * outputLayerWeights[0] * derv_sigmoid(u1) ;
        du[1] = derror * outputLayerWeights[1] * derv_sigmoid(u2) ;
        outputLayerWeights[0] += rate*u1*derror ;
        outputLayerWeights[1] += rate*u2*derror ;
        for j in range(hiddenLayerWeights.shape[0]):
            for k in range(hiddenLayerWeights.shape[1]):
                hiddenLayerWeights[j][k] += inputs[i][k] * rate * du[j] ;
        print('Hidden Layer: \n', hiddenLayerWeights) ;
        print('Output Layer: ', outputLayerWeights) ;
    print('-----------------------------------------------------') ;

# Predict outputs from neural network after weight adjustment
threshold = 0.5 ;
predicted_outputs = np.zeros(len(inputs)) ;
for i in range(len(inputs)):
    u1 = sigmoid(np.dot(inputs[i], hiddenLayerWeights[0][:])) ;
    u2 = sigmoid(np.dot(inputs[i], hiddenLayerWeights[0][:])) ;
    predicted_outputs[i] = sigmoid(np.dot(u1, outputLayerWeights[0]) + np.dot(u2,
          outputLayerWeights[1])) ;
    if predicted_outputs[i] > threshold:
        predicted_outputs[i] = 1 ;
    else:
        predicted_outputs[i] = 0 ;
print('Predicted outputs: ', outputs) ;
```

OBSERVATION

1) Adaline Learning (Unipolar & Bipolar inputs) [Random seed: 123]

- AND Gate

```
Inputs:  [[0, 0], [0, 1], [1, 0], [1, 1]]
Target outputs:  [0. 0. 0. 1.]
Initial weights:  [-1.0856306   0.99734545]
Initial Bias:  [0.2829785]

Epoch 48
-------------------------------------------------------------------------------
Weights:  [0.52909955 0.51174994] Bias:  [-0.22404201]
Weights:  [0.52909955 0.48297914] Bias:  [-0.2528128]
Weights:  [0.50147088 0.48297914] Bias:  [-0.28044148]
Weights:  [0.53107002 0.51257829] Bias:  [-0.25084233]
MSE:  [0.30869008]
Predicted outputs:  [0. 0. 0. 1.]
```

```
Inputs:  [[-1, -1], [-1, 1], [1, -1], [1, 1]]
Target outputs:  [-1, -1, -1, 1]
Initial weights:  [-1.0856306   0.99734545]
Initial Bias:  [0.2829785]

Epoch 13
-------------------------------------------------------------------------------
Weights:  [0.46592766 0.50261976] Bias:  [-0.43976548]
Weights:  [0.52562032 0.4429271 ] Bias:  [-0.49945814]
Weights:  [0.46729681 0.5012506 ] Bias:  [-0.55778165]
Weights:  [0.52622023 0.56017403] Bias:  [-0.49885823]
MSE:  [1.38392525]
Predicted outputs:  [-1, -1, -1, 1]
```

- OR Gate

```
Inputs:  [[0, 0], [0, 1], [1, 0], [1, 1]]
Target outputs:  [0. 1. 1. 1.]
Initial weights:  [-1.0856306   0.99734545]
Initial Bias:  [0.2829785]

Epoch 52
-------------------------------------------------------------------------------
Weights:  [0.43168415 0.46617929] Bias:  [0.26151657]
Weights:  [0.43168415 0.4934097 ] Bias:  [0.28874698]
Weights:  [0.45964103 0.4934097 ] Bias:  [0.31670387]
Weights:  [0.43266557 0.46643424] Bias:  [0.28972841]
MSE:  [0.30950908]
Predicted outputs:  [0. 1. 1. 1.]
```

```
Inputs:  [[-1, -1], [-1, 1], [1, -1], [1, 1]]
Target outputs:  [-1, 1, 1, 1]
Initial weights:  [-1.0856306   0.99734545]
Initial Bias:  [0.2829785]

Epoch 16
-------------------------------------------------------------------------------
Weights:  [0.52828708 0.5003521 ] Bias:  [0.44079033]
Weights:  [0.46957262 0.55906657] Bias:  [0.4995048]
Weights:  [0.52857153 0.50006765] Bias:  [0.55850371]
Weights:  [0.46985724 0.44135336] Bias:  [0.49978942]
MSE:  [1.38423337]
Predicted outputs:  [-1, 1, 1, 1]
```

- NAND Gate

```
Inputs:  [[0, 0], [0, 1], [1, 0], [1, 1]]
Target outputs:  [1. 1. 1. 0.]
Initial weights:  [-1.0856306   0.99734545]
Initial Bias:  [0.2829785]

Epoch 53
-------------------------------------------------------------------------------
Weights:  [-0.54263415 -0.50763939] Bias:  [1.23000175]
Weights:  [-0.54263415 -0.47987562] Bias:  [1.25776552]
Weights:  [-0.51414728 -0.47987562] Bias:  [1.28625238]
Weights:  [-0.54337023 -0.50909857] Bias:  [1.25702943]
MSE:  [0.30894049]
Predicted outputs:  [1. 1. 1. 0.]
```

```
Inputs:  [[-1, -1], [-1, 1], [1, -1], [1, 1]]
Target outputs:  [1, 1, 1, -1]
Initial weights:  [-1.0856306   0.99734545]
Initial Bias:  [0.2829785]

Epoch 19
-------------------------------------------------------------------------------
Weights:  [-0.47070868 -0.49977969] Bias:  [0.44102789]
Weights:  [-0.52951299 -0.44097538] Bias:  [0.4998322]
Weights:  [-0.47064245 -0.49984592] Bias:  [0.55870274]
Weights:  [-0.52946389 -0.55866735] Bias:  [0.49988131]
MSE:  [1.38396825]
Predicted outputs:  [1, 1, 1, -1]
```

- NOR Gate

```
Inputs:  [[0, 0], [0, 1], [1, 0], [1, 1]]
Target outputs:  [1. 0. 0. 0.]
Initial weights:  [-1.0856306   0.99734545]
Initial Bias:  [0.2829785]

Epoch 48
-------------------------------------------------------------------------------
Weights:  [-0.43608605 -0.45176745] Bias:  [0.73264692]
Weights:  [-0.43608605 -0.47985539] Bias:  [0.70455898]
Weights:  [-0.46293334 -0.47985539] Bias:  [0.67771168]
Weights:  [-0.43642564 -0.45334769] Bias:  [0.70421939]
MSE:  [0.30948087]
Predicted outputs:  [1. 0. 0. 0.]
```

```
Inputs:  [[-1, -1], [-1, 1], [1, -1], [1, 1]]
Target outputs:  [1, -1, -1, -1]
Initial weights:  [-1.0856306   0.99734545]
Initial Bias:  [0.2829785]

Epoch 19
-------------------------------------------------------------------------------
Weights:  [-0.52953156 -0.49969429] Bias:  [-0.44116512]
Weights:  [-0.47066434 -0.55856151] Bias:  [-0.50003233]
Weights:  [-0.52945083 -0.49977502] Bias:  [-0.55881882]
Weights:  [-0.47064636 -0.44097056] Bias:  [-0.50001435]
MSE:  [1.38423069]
Predicted outputs:  [1, -1, -1, -1]
```

2) Madaline Learning for XOR Gate

```
Inputs:  [[0, 0], [0, 1], [1, 0], [1, 1]]
Target outputs:  [0. 1. 1. 0.]
Initial weights
_______________
Hidden Layer:
 [[-0.68704294  0.55549218]
 [ 0.6166009   0.72847796]]
Output Layer:  [2.01137922 0.2656421 ]

Epoch 100
_____________________________________________________________
Hidden Layer:
 [[-0.90727755  0.25213131]
 [ 0.60711352  0.73297355]]
Output Layer:  [ 1.40285816 -0.51430888]
Hidden Layer:
 [[-0.90727755  0.25534824]
 [ 0.60711352  0.73192288]]
Output Layer:  [ 1.40810199 -0.50801427]
Hidden Layer:
 [[-0.90381373  0.25534824]
 [ 0.60572087  0.73192288]]
Output Layer:  [ 1.41155479 -0.50024213]
Hidden Layer:
 [[-0.90795938  0.25120259]
 [ 0.60679398  0.73299599]]
Output Layer:  [ 1.40708228 -0.51056056]
_____________________________________________________
Predicted outputs:  [0. 1. 1. 0.]
```

DISCUSSION & CONCLUSION

In this lab session, we understood the basics and key terminologies associated to artificial neural networks and built these networks based on Adaline learning. Adaline learning introduces error optimization using least squares to converge and adjust weights such that and backpropagation. It is a simple and effective linear regression and classification algorithm that uses the Widrow-Hoff rule to update the weights and biases of a single neuron. For gates like XOR and XNOR gate whose outputs can't be classified by linear boundary, it requires the backpropagation algorithm to train multilayer neural networks with nonlinear activation functions.

SUBMITTED BY:

Diwas Adhikari
076BEI014 – (Roll Number: 14)
Bachelor in Electronics & Communication Engineering (BEI)
076bei014.diwas@pcampus.edu.np
lunaticoda123@gmail.com