Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Learning a Linear Function via Perceptron and Backpropagation for Neural Network Learning

Objective:

The main goal of part A is to implement a perceptron model that learns a Boolean function (e.g. AND, OR, NAND, NOR) and the objective of part B is to implement the in-class exercise/tutorial on multi-layer perceptron learning.

Background:

For part A, we have written a simple program that uses a single perceptron which uses a Sum of Products (S) operation together with a "step" function (y) for activation:

$$S = \sum_{i=1}^{d} w_i x_i + w_{bias} \qquad y = \begin{cases} 1, & if\ S > 0 \\ 0, & otherwise \end{cases}$$

where *w* are parameters (or weights), *x* are the inputs and *d* is the dimension.

There are two modes in our program such as "Classification mode" and "learning mode". In "classification" mode, it accepts two inputs (0|1) and produces an Output (0|1) that corresponds to that function. In learning mode, it iteratively feeds forward two Inputs, produces an Output, minimizes Error, and uses the below Delta rule to update the three weights which were chosen randomly first with a fixed seed of 1:

$$\Delta w_{i,j} = \eta(t_j - y_j)x_i$$

where *t* is the target and η is the learning rate.

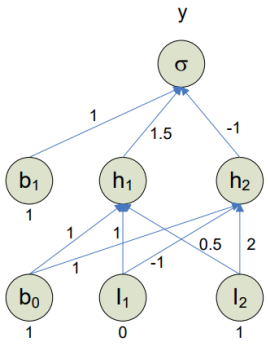
After running through a few Epochs, it produces correct Outputs for all four Input combinations. As a result, it has learned the function. The below output is for AND function:

```
********************** AND **********************
Initial Weights: 0.13 [0.85 0.76]
****************************************
Epoch #1

inputs: [0, 0]
weights: 0.13 [0.85 0.76]
y=1 t=0 ==> incorrect

inputs: [0, 1]
weights: -0.37 [0.85 0.76]
y=1 t=0 ==> incorrect

inputs: [1, 0]
weights: -0.87 [0.85 0.26]
y=0 t=0 ==> correct

inputs: [1, 1]
weights: -0.87 [0.85 0.26]
y=1 t=1 ==> correct

Epoch #2

inputs: [0, 0]
weights: -0.87 [0.85 0.26]
y=0 t=0 ==> correct

inputs: [0, 1]
weights: -0.87 [0.85 0.26]
y=0 t=0 ==> correct

inputs: [1, 0]
weights: -0.87 [0.85 0.26]
y=0 t=0 ==> correct

inputs: [1, 1]
weights: -0.87 [0.85 0.26]
y=1 t=1 ==> correct

Solution took 2 epochs.
Final weights: -0.87 [0.85 0.26]

********** Classify **********
inputs: [0, 1]
weights: -0.87 [0.85 0.26]
Prediction: 0
****************************************
```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

For part B, we have written a program that implements a multi-layer neural network and demonstrates iterative learning. There was an additional hidden layer between the input and output layers in the multi-layer perceptron learning tutorial.

Below is a snapshot of a neural network during training. There are two input units, two hidden layer perceptrons, and a single output unit. Input I1 has a value of 0; input I2 has a value of 1; all bias has value 1. Edges are labeled with their corresponding weights. Learning factor $\eta = 0.5$. Target value is 1.



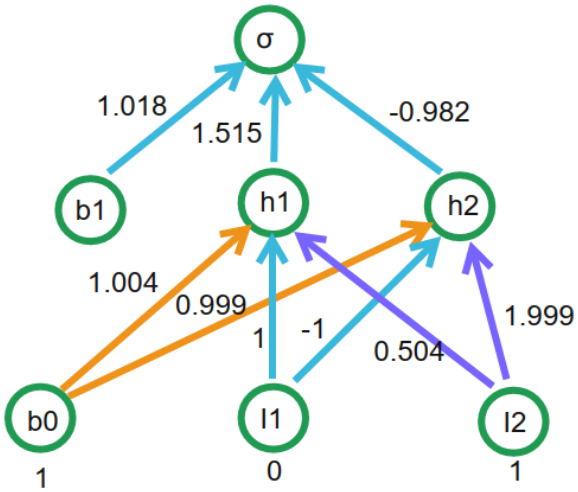
As a form of Supervised Learning, the training process consists of repeatedly presenting labeled examples to the system and iteratively adjusting parameters (weights) until the desired outcome is obtained:

$$\text{WeightUpdate} = \text{LearningRate} \cdot (\text{TargetOutput} - \text{ActualOutput}) \cdot \text{Input}$$

In a multilayer network, there is no target value for perceptrons in the hidden layer; hence their error cannot be directly calculated. Therefore, error is propagated backwards from the output layer.

The below snapshot is the neural network after updating the weights:

Final network with the updated weights
(Manually drawn using the final weights)



Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Visualization: [Display the Output]

Output of Part A:

```
Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin
C:\Users\shuvo\AppData\Local\Programs\Python\Python39\python.exe "C:\Program Files\JetBrains\PyCharm 2021.3\plugins\python\helpers\pydev\pydevconsole.py" --mode=client --port=54776

import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['C:\\Users\\shuvo\\Workspace\\PyCharm\\CIS678', 'C:/Users/shuvo/Workspace/PyCharm/CIS678'])

PyDev console: starting.

Python 3.9.12 (tags/v3.9.12:b28265d, Mar 23 2022, 23:52:46) [MSC v.1929 64 bit (AMD64)] on win32
>>> runfile('C:/Users/shuvo/Workspace/PyCharm/CIS678/project4/project.py', wdir='C:/Users/shuvo/Workspace/PyCharm/CIS678/project4')


********** AND **********


Initial Weights: 0.13 [0.85 0.76]

****************************************

Epoch #1

inputs: [0, 0]
weights: 0.13 [0.85 0.76]
y=1 t=0 ==> incorrect

inputs: [0, 1]
weights: -0.37 [0.85 0.76]
y=1 t=0 ==> incorrect

inputs: [1, 0]
weights: -0.87 [0.85 0.26]
y=0 t=0 ==> correct

inputs: [1, 1]
weights: -0.87 [0.85 0.26]
y=1 t=1 ==> correct

****************************************

Epoch #2

inputs: [0, 0]
weights: -0.87 [0.85 0.26]
y=0 t=0 ==> correct

inputs: [0, 1]
weights: -0.87 [0.85 0.26]
y=0 t=0 ==> correct

inputs: [1, 0]
weights: -0.87 [0.85 0.26]
y=0 t=0 ==> correct

inputs: [1, 1]
weights: -0.87 [0.85 0.26]
y=1 t=1 ==> correct


Solution took 2 epochs.
Final weights: -0.87 [0.85 0.26]

********** Classify **********

inputs: [0, 1]
weights: -0.87 [0.85 0.26]
Prediction: 0


********** OR **********


Initial Weights: 0.13 [0.85 0.76]

****************************************

Epoch #1

inputs: [0, 0]
weights: 0.13 [0.85 0.76]
```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
y=1 t=0 ==> incorrect

inputs: [0, 1]
weights: -0.37 [0.85 0.76]
y=1 t=1 ==> correct

inputs: [1, 0]
weights: -0.37 [0.85 0.76]
y=1 t=1 ==> correct

inputs: [1, 1]
weights: -0.37 [0.85 0.76]
y=1 t=1 ==> correct

****************************************

Epoch #2

inputs: [0, 0]
weights: -0.37 [0.85 0.76]
y=0 t=0 ==> correct

inputs: [0, 1]
weights: -0.37 [0.85 0.76]
y=1 t=1 ==> correct

inputs: [1, 0]
weights: -0.37 [0.85 0.76]
y=1 t=1 ==> correct

inputs: [1, 1]
weights: -0.37 [0.85 0.76]
y=1 t=1 ==> correct


Solution took 2 epochs.
Final weights: -0.37 [0.85 0.76]

********** Classify **********

inputs: [0, 1]
weights: -0.37 [0.85 0.76]
Prediction: 1


********** NAND **********


Initial Weights: 0.13 [0.85 0.76]

****************************************

Epoch #1

inputs: [0, 0]
weights: 0.13 [0.85 0.76]
y=1 t=1 ==> correct

inputs: [0, 1]
weights: 0.13 [0.85 0.76]
y=1 t=1 ==> correct

inputs: [1, 0]
weights: 0.13 [0.85 0.76]
y=1 t=1 ==> correct

inputs: [1, 1]
weights: 0.13 [0.85 0.76]
y=1 t=0 ==> incorrect

****************************************

Epoch #2

inputs: [0, 0]
weights: -0.37 [0.35 0.26]
y=0 t=1 ==> incorrect

inputs: [0, 1]
weights: 0.13 [0.35 0.26]
```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
y=1 t=1 ==> correct

inputs: [1, 0]
weights: 0.13 [0.35 0.26]
y=1 t=1 ==> correct

inputs: [1, 1]
weights: 0.13 [0.35 0.26]
y=1 t=0 ==> incorrect

****************************************

Epoch #3

inputs: [0, 0]
weights: -0.37 [-0.15 -0.24]
y=0 t=1 ==> incorrect

inputs: [0, 1]
weights: 0.13 [-0.15 -0.24]
y=0 t=1 ==> incorrect

inputs: [1, 0]
weights: 0.63 [-0.15  0.26]
y=1 t=1 ==> correct

inputs: [1, 1]
weights: 0.63 [-0.15  0.26]
y=1 t=0 ==> incorrect

****************************************

Epoch #4

inputs: [0, 0]
weights: 0.13 [-0.65 -0.24]
y=1 t=1 ==> correct

inputs: [0, 1]
weights: 0.13 [-0.65 -0.24]
y=0 t=1 ==> incorrect

inputs: [1, 0]
weights: 0.63 [-0.65  0.26]
y=0 t=1 ==> incorrect

inputs: [1, 1]
weights: 1.13 [-0.15  0.26]
y=1 t=0 ==> incorrect

****************************************

Epoch #5

inputs: [0, 0]
weights: 0.6299999999999999 [-0.65 -0.24]
y=1 t=1 ==> correct

inputs: [0, 1]
weights: 0.6299999999999999 [-0.65 -0.24]
y=1 t=1 ==> correct

inputs: [1, 0]
weights: 0.6299999999999999 [-0.65 -0.24]
y=0 t=1 ==> incorrect

inputs: [1, 1]
weights: 1.13 [-0.15 -0.24]
y=1 t=0 ==> incorrect

****************************************

Epoch #6

inputs: [0, 0]
weights: 0.6299999999999999 [-0.65 -0.74]
y=1 t=1 ==> correct

inputs: [0, 1]
weights: 0.6299999999999999 [-0.65 -0.74]
```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
y=0 t=1 ==> incorrect

inputs: [1, 0]
weights: 1.13 [-0.65 -0.24]
y=1 t=1 ==> correct

inputs: [1, 1]
weights: 1.13 [-0.65 -0.24]
y=1 t=0 ==> incorrect

****************************************

Epoch #7

inputs: [0, 0]
weights: 0.6299999999999999 [-1.15 -0.74]
y=1 t=1 ==> correct

inputs: [0, 1]
weights: 0.6299999999999999 [-1.15 -0.74]
y=0 t=1 ==> incorrect

inputs: [1, 0]
weights: 1.13 [-1.15 -0.24]
y=0 t=1 ==> incorrect

inputs: [1, 1]
weights: 1.63 [-0.65 -0.24]
y=1 t=0 ==> incorrect

****************************************

Epoch #8

inputs: [0, 0]
weights: 1.13 [-1.15 -0.74]
y=1 t=1 ==> correct

inputs: [0, 1]
weights: 1.13 [-1.15 -0.74]
y=1 t=1 ==> correct

inputs: [1, 0]
weights: 1.13 [-1.15 -0.74]
y=0 t=1 ==> incorrect

inputs: [1, 1]
weights: 1.63 [-0.65 -0.74]
y=1 t=0 ==> incorrect

****************************************

Epoch #9

inputs: [0, 0]
weights: 1.13 [-1.15 -1.24]
y=1 t=1 ==> correct

inputs: [0, 1]
weights: 1.13 [-1.15 -1.24]
y=0 t=1 ==> incorrect

inputs: [1, 0]
weights: 1.63 [-1.15 -0.74]
y=1 t=1 ==> correct

inputs: [1, 1]
weights: 1.63 [-1.15 -0.74]
y=0 t=0 ==> correct

****************************************

Epoch #10

inputs: [0, 0]
weights: 1.63 [-1.15 -0.74]
y=1 t=1 ==> correct

inputs: [0, 1]
weights: 1.63 [-1.15 -0.74]
```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
y=1 t=1 ==> correct

inputs: [1, 0]
weights: 1.63 [-1.15 -0.74]
y=1 t=1 ==> correct

inputs: [1, 1]
weights: 1.63 [-1.15 -0.74]
y=0 t=0 ==> correct


Solution took 10 epochs.
Final weights: 1.63 [-1.15 -0.74]

********** Classify **********

inputs: [0, 1]
weights: 1.63 [-1.15 -0.74]
Prediction: 1


********************** NOR **********************


Initial Weights: 0.13 [0.85 0.76]

****************************************

Epoch #1

inputs: [0, 0]
weights: 0.13 [0.85 0.76]
y=1 t=1 ==> correct

inputs: [0, 1]
weights: 0.13 [0.85 0.76]
y=1 t=0 ==> incorrect

inputs: [1, 0]
weights: -0.37 [0.85 0.26]
y=1 t=0 ==> incorrect

inputs: [1, 1]
weights: -0.87 [0.35 0.26]
y=0 t=0 ==> correct

****************************************

Epoch #2

inputs: [0, 0]
weights: -0.87 [0.35 0.26]
y=0 t=1 ==> incorrect

inputs: [0, 1]
weights: -0.37 [0.35 0.26]
y=0 t=0 ==> correct

inputs: [1, 0]
weights: -0.37 [0.35 0.26]
y=0 t=0 ==> correct

inputs: [1, 1]
weights: -0.37 [0.35 0.26]
y=1 t=0 ==> incorrect

****************************************

Epoch #3

inputs: [0, 0]
weights: -0.87 [-0.15 -0.24]
y=0 t=1 ==> incorrect

inputs: [0, 1]
weights: -0.37 [-0.15 -0.24]
y=0 t=0 ==> correct

inputs: [1, 0]
weights: -0.37 [-0.15 -0.24]
```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
399 y=0 t=0 ==> correct
400
401 inputs: [1, 1]
402 weights: -0.37 [-0.15 -0.24]
403 y=0 t=0 ==> correct
404
405 ****************************************
406
407 Epoch #4
408
409 inputs: [0, 0]
410 weights: -0.37 [-0.15 -0.24]
411 y=0 t=1 ==> incorrect
412
413 inputs: [0, 1]
414 weights: 0.13 [-0.15 -0.24]
415 y=0 t=0 ==> correct
416
417 inputs: [1, 0]
418 weights: 0.13 [-0.15 -0.24]
419 y=0 t=0 ==> correct
420
421 inputs: [1, 1]
422 weights: 0.13 [-0.15 -0.24]
423 y=0 t=0 ==> correct
424
425 ****************************************
426
427 Epoch #5
428
429 inputs: [0, 0]
430 weights: 0.13 [-0.15 -0.24]
431 y=1 t=1 ==> correct
432
433 inputs: [0, 1]
434 weights: 0.13 [-0.15 -0.24]
435 y=0 t=0 ==> correct
436
437 inputs: [1, 0]
438 weights: 0.13 [-0.15 -0.24]
439 y=0 t=0 ==> correct
440
441 inputs: [1, 1]
442 weights: 0.13 [-0.15 -0.24]
443 y=0 t=0 ==> correct
444
445
446 Solution took 5 epochs.
447 Final weights: 0.13 [-0.15 -0.24]
448
449 ********** Classify **********
450
451 inputs: [0, 1]
452 weights: 0.13 [-0.15 -0.24]
453 Prediction: 0
454
455 Process finished with exit code 0
456
```

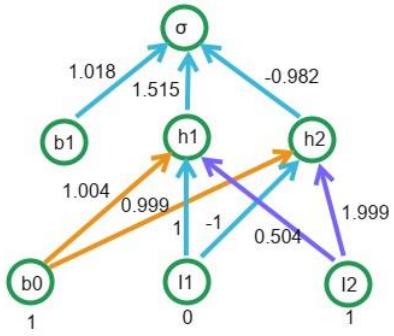
Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Output of Part B:

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
C:\Users\shuvo\AppData\Local\Programs\Python\Python39\python.exe "C:\Program Files\JetBrains\PyCharm 2021.3\plugins\python\helpers\pydev\pydevconsole.py" --mode=client --port=54951

import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['C:\\Users\\shuvo\\Workspace\\PyCharm\\CIS678', 'C:/Users/shuvo/Workspace/PyCharm/CIS678'])

PyDev console: starting.

Python 3.9.12 (tags/v3.9.12:b28265d, Mar 23 2022, 23:52:46) [MSC v.1929 64 bit (AMD64)] on win32
>>> runfile('C:/Users/shuvo/Workspace/PyCharm/CIS678/project4/neural_net_sample.py', wdir='C:/Users/shuvo/Workspace/PyCharm/CIS678/project4')
Step 1: Feed the Inputs forward

Sigma(h1) = { (1.0+1)+(1.0+0)+(0.5+1) } = 1.5    h1 = 0.818
Sigma(h2) = { (1.0+1)+(-1.0+0)+(2.0+1) } = 3.0    h2 = 0.953
Sigma(y) = { (1.0+1)+(1.5+0.818)+(-1.0+0.953) } = 1.274    y = 0.781
Total error in network E = (0.5x(1-0.781)^2) = 0.024


Step 2: Backpropagate the errors

Error(y) = 0.781 x ( 1 - 0.781 ) x ( 1 - 0.781 ) = 0.037
Error(h1) = 0.818 x ( 1 - 0.818 ) x ( 1.5 x 0.037 ) = 0.008
Error(h2) = 0.953 x ( 1 - 0.953 ) x ( -1.0 x 0.037 ) = -0.002


Step 3: Learn

W(h0, y) = 1.0 + (0.5 * 0.037 * 1) = 1.018
W(h1, y) = 1.5 + (0.5 * 0.037 * 0.818) = 1.515
W(h2, y) = -1.0 + (0.5 * 0.037 * 0.953) = -0.982

W(I0, h1) = 1.0 + (0.5 * 0.008 * 1) = 1.004
W(I1, h1) = 1.0 + (0.5 * 0.008 * 0) = 1.0
W(I2, h1) = 0.5 + (0.5 * 0.008 * 1) = 0.504

W(I0, h2) = 1.0 + (0.5 * -0.002 * 1) = 0.999
W(I1, h2) = -1.0 + (0.5 * -0.002 * 0) = -1.0
W(I2, h2) = 2.0 + (0.5 * -0.002 * 1) = 1.999


Parameters at the end

[ 1.018  1.515 -0.982]
[[ 1.004  1.     0.504]
 [ 0.999 -1.     1.999]]

Process finished with exit code 0
```

Final network with the updated weights (Manually drawn using the final weights)



Analysis and Discussion:

We can see from the above visualization part that the output generated using the implemented program satisfies our objective of this project. In this project, we have understood from part A and part B regarding how Perceptron learning works by being able to implement a small model as well as the function of Backpropagation in Neural Network learning by implementing the steps executed for a single data instance.

Future Work:

In future, we can extend our program by implementing the capability to process Backpropagation in Neural Network learning by implementing the steps executed for whole dataset instead of a single data instance. Furthermore, we can extend our program to create a neural network capable of recognizing handwritten digits.

Source Code (Part A):

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
from perceptron import Perceptron
from pandas import DataFrame


def run_perceptron():
    print('\n\n********************** AND **********************\n\n')
    p = Perceptron(2)
    df = DataFrame([[0, 0], [0, 1], [1, 0], [1, 1]])
    targets = [0, 0, 0, 1]
    p.train(df, targets)
    print('\n********** Classify **********')
    p.classify([0, 1])

    print('\n\n********************** OR **********************\n\n')
    p = Perceptron(2)
    df = DataFrame([[0, 0], [0, 1], [1, 0], [1, 1]])
    targets = [0, 1, 1, 1]
    p.train(df, targets)
    print('\n********** Classify **********')
    p.classify([0, 1])

    print('\n\n********************** NAND **********************\n\n')
    p = Perceptron(2)
    df = DataFrame([[0, 0], [0, 1], [1, 0], [1, 1]])
    targets = [1, 1, 1, 0]
    p.train(df, targets)
    print('\n********** Classify **********')
    p.classify([0, 1])

    print('\n\n********************** NOR **********************\n\n')
    p = Perceptron(2)
    df = DataFrame([[0, 0], [0, 1], [1, 0], [1, 1]])
    targets = [1, 0, 0, 0]
    p.train(df, targets)
    print('\n********** Classify **********')
    p.classify([0, 1])


if __name__ == '__main__':
    run_perceptron()

```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```python
import numpy as np
import pandas as pd
from random import seed, random


class Perceptron:
    __weights = np.zeros((1, 3))
    __epoch = 0
    __learning_rate = 0.5

    def __init__(self, inputs: int):
        seed(1)
        self.__weights = np.array([round(random(), 2) for i in range(0, inputs + 1)])
        #self.__weights = np.array([0.01, 0.08, 0.08])
        print(f'Initial Weights: {self.__weights[0]} {self.__weights[1:]}')

    def __default_step_function(self, inputs):
        product = self.__weights * inputs
        return 1 if product.sum() > 0 else 0

    def __default_update_parameters(self, inputs, t, y):
        mult = self.__learning_rate * (t - y)
        changes = np.array(inputs) * mult
        self.__weights += changes

    def set_hyper_parameters(self, learning_rate=0.5):
        self.__learning_rate = learning_rate

    def train(self, df: pd.DataFrame, targets, epochs=None):
        if len(df) != len(targets):
            raise Exception("Data and Targets don't have same number of rows.")

        converged = False
        while (epochs is None and not converged) or (epochs is not None and epochs != self.__epoch):

            print('\n**************************************\n')
            self.__epoch += 1
            print(f'Epoch #{self.__epoch}')

            correct_predictions = 0
            for i in df.index:
                inputs = df.loc[i].to_list()
                print(f'\ninputs: {inputs}')
                print(f'weights: {self.__weights[0]} {self.__weights[1:]}')

                inputs.insert(0, 1)
                y = self.__default_step_function(inputs)
                print(f'y={y}\tt={targets[i]} ==> ', end='')

                if y == targets[i]:
                    print('correct')
                    correct_predictions += 1
                else:
                    print('incorrect')
                    # update weights
                    self.__default_update_parameters(inputs, targets[i], y)

            converged = correct_predictions == len(df)

        print(f'\n\nSolution took {self.__epoch} epochs.')
        print(f'Final weights: {self.__weights[0]} {self.__weights[1:]}')

    def classify(self, inputs):
        print(f'\ninputs: {inputs}')
        print(f'weights: {self.__weights[0]} {self.__weights[1:]}')
        ins = inputs.copy()
        ins.insert(0, 1)
        y = self.__default_step_function(ins)
        print(f'Prediction: {y}')
        return y

```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Source Code (Part B):

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```python
import numpy as np
from pandas import DataFrame
import math

learning_rate = 0.5

first_layer_weights = np.array([
    [1, 1, 0.5],
    [1, -1, 2]
])

second_layer_weights = np.array(
    [1, 1.5, -1]
)


def feed_forward(inputs, target):
    h, w = first_layer_weights.shape
    hiddens = [0]*h

    for l in range(0, h):
        print(f'Sigma(h{l + 1}) = ' + '{ ', end='')
        sign = ''
        for i in range(0, len(inputs)):
            print(f'{sign}({first_layer_weights[l][i]}+{inputs[i]})', end='')
            sign = '+'
        print(' } = ', end='')

        sig_h1 = np.dot(first_layer_weights[l], inputs)
        print(sig_h1, end='')

        hiddens[l] = round((1 / (1 + math.e ** -sig_h1)), 3)
        print(f'    h{l + 1} = {hiddens[l]}')

    hiddens.insert(0, 1)
    print('Sigma(y) = { ', end='')
    print(f'({second_layer_weights[0]}+{hiddens[0]})', end='')
    print(f'+({second_layer_weights[1]}+{hiddens[1]})', end='')
    print(f'+({second_layer_weights[2]}+{hiddens[2]})', end='')
    print(' } = ', end='')

    y = np.dot(second_layer_weights, hiddens)
    hiddens[0] = (round((1/(1+math.e**-y)), 3))

    print(f'{y}', end='')
    print(f'    y = {hiddens[0]}')

    E_net = round(0.5 * (target - hiddens[0]) ** 2, 3)
    print(f'Total error in network E = (0.5x({target}-{hiddens[0]})^2) = {E_net}')

    return hiddens, E_net


def backpropagate_errors(hiddens, target):
    errors = [0]*len(hiddens)
    errors[0] = round(hiddens[0]*(1-hiddens[0])*(target-hiddens[0]), 3)
    print(f'Error(y) = {hiddens[0]} x ( 1 - {hiddens[0]} ) x ( {target} - {hiddens[0]} ) = {errors[0]}')

    for i in range(1, len(hiddens)):
        errors[i] = round(hiddens[i]*(1-hiddens[i])*(second_layer_weights[i]*errors[0]), 3)
        print(f'Error(h{i}) = {hiddens[i]} x ( 1 - {hiddens[i]} ) x ( {second_layer_weights[i]} x {errors[0]} ) = {
errors[i]}')

    return errors


def learn(hiddens, errors, inputs):
    hiddens.insert(0, 1)

    for i in range(0, len(second_layer_weights)):
        nw = round(second_layer_weights[i] + (learning_rate * errors[0] * hiddens[i]), 3)
        print(f'W(h{i}, y) = {second_layer_weights[i]} + ({learning_rate} * {errors[0]} * {hiddens[i]}) = {nw}')
        second_layer_weights[i] = nw

    print('')

    h, w = first_layer_weights.shape
    for i in range(0, h):
        for j in range(0, w):
            nw = first_layer_weights[i][j] + (learning_rate * errors[i+1] * inputs[j])
```

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

Prepared by: Abu Naweem Khan & Sayed Muhammad Saifuddin

```
            print(f'W(I{j}, h{i+1}) = {first_layer_weights[i][j]} + ({learning_rate} * {errors[i+1]} * {inputs[j]}) = {
nw}')
            first_layer_weights[i][j] = nw
        print('')


target = 1
inputs = [1, 0, 1]
print('Step 1: Feed the Inputs forward\n')
val_at_nodes, E_net = feed_forward(inputs, target)

print('\n\nStep 2: Backpropagate the errors\n')
errors = backpropagate_errors(val_at_nodes, target)

print('\n\nStep 3: Learn\n')
learn(val_at_nodes[1:], errors, inputs)

print('\n\nParameters at the end\n')
print(second_layer_weights)
print(first_layer_weights)


```

Thank You!!!