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])
```

```
            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)


```