Homework 1

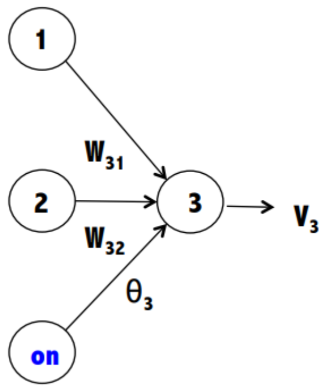
Paden Rumsey
CS578 - Neural Network Design

I. Introduction

Perceptrons are a powerful building block in neural networks. This network of weights and nodes allows for the identification of solutions in n-dimensional space via linear methods (for those problems that have such solutions). This assignment shows the strengths of the single-layer perceptron as well as its weaknesses. The multi-layer perceptron, which I will also show, will illuminate the true strength of the perceptron by countering some of these weaknesses. Howerver, the multi-layer perceptron has some weaknesses of its own. Specifically, I will be modifying the weights using the error produced by comparing the output of the network with its actual target values and using this error to modify the weights to give us the correct solution. By iterating through this process multiple times, and re-using the previous results, we should arrive at the answer.

II. OR Network

A. Architecture



For the architecture, we have a simple perceptron with two input and one output node, with a singular bias. 1 and 2 correspond to V_1 and V_2, respectively. θ_3 corresponds to the bias (b in the table). And the weights correspond accordingly in the table.

B. Handrawn Calculations for the first row:

$\alpha = 0.5$
$bias = -0.5$
$input = 0, 1$
$target = 1$
$activation = y >= 0 \quad V_3 = 1$
$y < 0 \quad V_3 = 0$

y = -0.5 + [0,1] $\cdot [0, 0] = -0.5$
$V_3 = activation(-0.5) = 0$
$\delta_3 = 1 - 0 = 1$
$\Delta W_{31} = 0.5 * 1 * 0 = 0$
$\Delta W_{32} = 0.5 * 1 * 1 = 0.5$
$W_{31} = 0 + 0 = 0$
$W_{32} = 0 + 0.5 = 0.5$
$\Delta b = 1 * 0.5 * 1 = 0.5$
$b = 0 + 0.5 = 0.5$

C. First Two Rows Performed Already

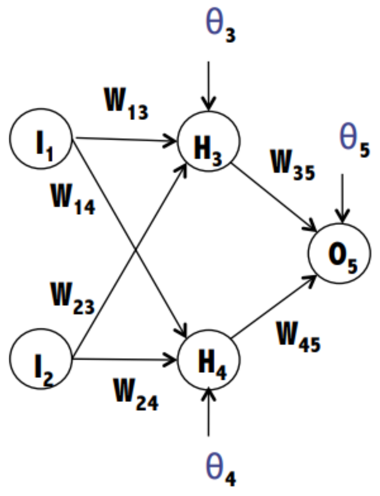
V_1	V_2	t_3	y_3	V_3	δ_3	W_{31}	W_{32}	b
0	0	0	0	1	-1	0	0	-0.5
0	1	1	-0.5	0	1	0	0.5	0.5

D. The Completed Table

V_1	V_2	t_3	y_3	V_3	δ_3	W_{31}	W_{32}	b
0	0	0	0	1	-1	0	0	-0.5
0	1	1	-0.5	0	1	0	0.5	0.5
1	0	1	0.5	1	0	0	0.5	0.5
1	1	1	1	1	0	0	0.5	0.5
0	0	0	0.5	1	-1	0	0.5	-0.5
0	1	1	0	1	0	0	0.5	-0.5
1	0	1	-0.5	0	1	0.5	0.5	0.5
1	1	1	1.5	1	0	0.5	0.5	0.5
0	0	0	0.5	1	-1	0.5	0.5	0.5
0	1	1	0	1	0	0.5	0.5	-0.5
1	0	1	0	1	0	0.5	0.5	-0.5
1	1	1	0.5	1	0	0.5	0.5	-0.5
0	0	0	-0.5	0	0	0.5	0.5	-0.5

III. XOR Network

A. Architecture



This is a network comprised of two input nodes, six weights, one hidden layer (with two nodes) and a single output node. Most values are labeled identically in Appendix A. Xor Final Table Results. The activated hidden values correspond to y_1 and y_2, the $\theta's$ correspond to their numbered biases, and I_1 and I_2 are V_1 and V_2 respectively. O_3 corresponds to V_3 in the table. **The weights are initialized to starting values between (but not including) 0 and 1 using the numpy library's random.rand() function.** The training occurs for 100 epochs, although, the training is re-performed with different random starting weight values until the network converges (i.e. all the values are correct).

B. Final Values:

The final network values are represented in Appendix A. on the next page. The network had to be re-ran 119 times in order for the weights to converge after 100 epochs. All floats are truncated to two decimal places to ensure they fit into the table.

IV. Conclusions

While perceptrons are excellent at solving problems that are linearly separable, it does not perform as well on problems that are not. We can introduce a second hyper-plane to help us solve this problem, however, the results are not guaranteed for all initial starting conditions. Sometime the search for the correct weights becomes stuck at that point in the search space. Backpropagation is a technique that will help us solve these kinds of problems. This will help unviel the true strength of neural networks.

Appendix A - XOR Final Table Results

$\mathbf{V_1}$	$\mathbf{V_2}$	$\mathbf{t_3}$	$\mathbf{y_1}$	$\mathbf{y_2}$	$\mathbf{y_3}$	$\mathbf{V_3}$	δ_3	δ_4	δ_5	$\mathbf{W_{13}}$	$\mathbf{W_{14}}$	$\mathbf{W_{23}}$	$\mathbf{W_{24}}$	$\mathbf{W_{34}}$	$\mathbf{W_{45}}$	**b1**	**b2**	**b3**
0	0	0	0.01	0.01	0.00	0	0	0	0	0.21	0.32	0.25	0.35	-0.55	0.32	-0.46	-0.04	0
0	1	1	0.01	0.85	0.27	1	0	0	0	0.21	0.32	0.25	0.35	-0.55	0.32	-0.46	-0.04	0
1	0	1	0.01	0.85	0.27	1	0	0	0	0.21	0.32	0.25	0.35	-0.55	0.32	-0.46	-0.04	0
1	1	0	0.85	0.85	-0.20	0	0	0	0	0.21	0.32	0.25	0.35	-0.55	0.32	-0.46	-0.04	0

Appendix B - Code

```
import numpy as np


def shift_network_simple(nodes, w, b, output, target):

    if output >= 0:
        result = 1
    else:
        result = 0

    error = calc_error(target, result)

    w[0,0] = shift_weight(0.5, error, nodes[0,0], w[0,0])
    w[1,0] = shift_weight(0.5, error, nodes[0,1], w[1,0])
    b = change_bias(1, error, b)

    return (w, b, result, error)

def shift_network(nodes, hidden, weight1, weight2, bias1, bias2, output, target):

    if output >= 0:
        result = 1
    else:
        result = 0

    error = calc_error(target, result)
    error_3 = 0.5 * error * weight2[0,0]
    error_4 = 0.5 * error * weight2[1,0]

    weight2[0,0] = shift_weight(0.5, error, hidden[0,0], weight2[0,0])
    weight2[1,0] = shift_weight(0.5, error, hidden[0,1], weight2[1,0])


    weight1[0,0] = shift_weight(0.5, error_3, nodes[0,0], weight1[0,0])
    weight1[1,0] = shift_weight(0.5, error_3, nodes[0,0], weight1[1,0])
    weight1[0,1] = shift_weight(0.5, error_4, nodes[0,1], weight1[0,1])
    weight1[1,1] = shift_weight(0.5, error_4, nodes[0,1], weight1[1,1])



    bias1[0,0] = change_bias(0.5, error_3, bias1[0,0])
    bias1[0,1] = change_bias(0.5, error_4, bias1[0,1])
    bias2 = change_bias(0.5, error, bias2)

    return (weight1, weight2, bias1, bias2, result, error, error_3, error_4)

def print_all_simple(nodes, target, output, result, error, weights, bias):
    print(" {}   {}   {}   {}   {}   {}   {}   {}    {}".format(nodes[0,0], nodes[0,1], target

def print_all(nodes, target, hidden, output, result, error, error_3, error_4, weight1, weight2
    print("{}   {}   {}    {}   {}   {}   {}   {}    {}   {}    {}   {}    {}     {}    {}


def calc_error(target, output):
    return target - output
```

```
def shift_weight(alpha, error, node, weight):
    shift = alpha * error * node
    return weight + shift

def change_bias(alpha, error, bias):
    shift = alpha * error
    return bias + shift

def activate_hidden(h):

    if h[0,0] >= 0:
        h[0,0] = .85
    else:
        h[0,0] = .01

    if h[0,1] >= 0:
        h[0,1] = .85
    else:
        h[0,1] = .01

    return h



x1 = np.array([[0,0]], float)
x2 = np.array([[0,1]], float)
x3 = np.array([[1,0]], float)
x4 = np.array([[1,1]], float)
w = np.array([[0],[0]], float)

e = -1
b = -0.5
v3 = 1

i = 0

print(" V1   V2   t3   y3   V3    e   W1   W2    b")


while (i < 100):

    #0 and 1
    output = b + np.dot(x2,w)
    w,b,v3,e = shift_network_simple(x2,w,b,output, 1)
    print_all_simple(x2, 1, output, v3, e, w, b)

    #1 and 0
    output = b + np.dot(x3,w)
    w,b,v3,e = shift_network_simple(x3,w,b,output, 1)
    print_all_simple(x3, 1, output, v3, e, w, b)

    #1 and 1
    output = b + np.dot(x4,w)
    w,b,v3,e = shift_network_simple(x4,w,b,output, 1)
    print_all_simple(x4, 1, output, v3, e, w, b)

    #0 and 0
    output = b + np.dot(x1,w)
    w,b,v3,e = shift_network_simple(x1,w,b,output, 0)
    print_all_simple(x1, 0, output, v3, e, w, b)

    i = i + 1
    j = -1

while (True):

    j = j + 1
    x1 = np.array([[0,0]], float)
    x2 = np.array([[0,1]], float)
    x3 = np.array([[1,0]], float)
    x4 = np.array([[1,1]], float)

    w1 = np.random.rand(2,2)
    w2 = np.random.rand(2,1)
    b1 = np.random.rand(1,2)
    b2 = 0

    print(" V1   V2   t3   y1   y2   y3   V3    e1     e2     e3   W1   W2   W3   W4   W5    W

    i = 0

    while (i < 100):

        #0 and 0
        h = b1 + np.dot(x1, w1)
        hidden = activate_hidden(h)
        output = b2 + np.dot(hidden,w2)
        w1,w2,b1,b2,o5,e,e3,e4 = shift_network(x1,hidden,w1,w2,b1,b2,output,0)
        result1 = o5
        print_all(x1, 0, h, output, o5, e, e3, e4, w1, w2, b1, b2)

        #0 and 1
        h = b1 + np.dot(x2, w1)
        hidden = activate_hidden(h)
        output = b2 + np.dot(hidden,w2)
        w1,w2,b1,b2,o5,e,e3,e4 = shift_network(x2,hidden,w1,w2,b1,b2,output,1)
        result2 = o5
        print_all(x2, 1, h, output, o5, e, e3, e4, w1, w2, b1, b2)

        #1 and 0
        h = b1 + np.dot(x3, w1)
        hidden = activate_hidden(h)
        output = b2 + np.dot(hidden,w2)
        w1,w2,b1,b2,o5,e,e3,e4 = shift_network(x3,hidden,w1,w2,b1,b2,output,1)
        result3 = o5
        print_all(x3, 1, h, output, o5, e, e3, e4, w1, w2, b1, b2)


        #1 and 1
        h = b1 + np.dot(x4, w1)
        hidden = activate_hidden(h)
        output = b2 + np.dot(hidden,w2)
        w1,w2,b1,b2,o5,e,e3,e4 = shift_network(x4,hidden,w1,w2,b1,b2,output,0)
        result4 = o5
        print_all(x4, 0, h, output, o5, e, e3, e4, w1, w2, b1, b2)
        i = i + 1

    if(result1 == 0 and result2 == 1 and result3 == 1 and result4 == 0):
        break


print(j)
```