Lab 1: Backpropagation with a 3 Neuron Model

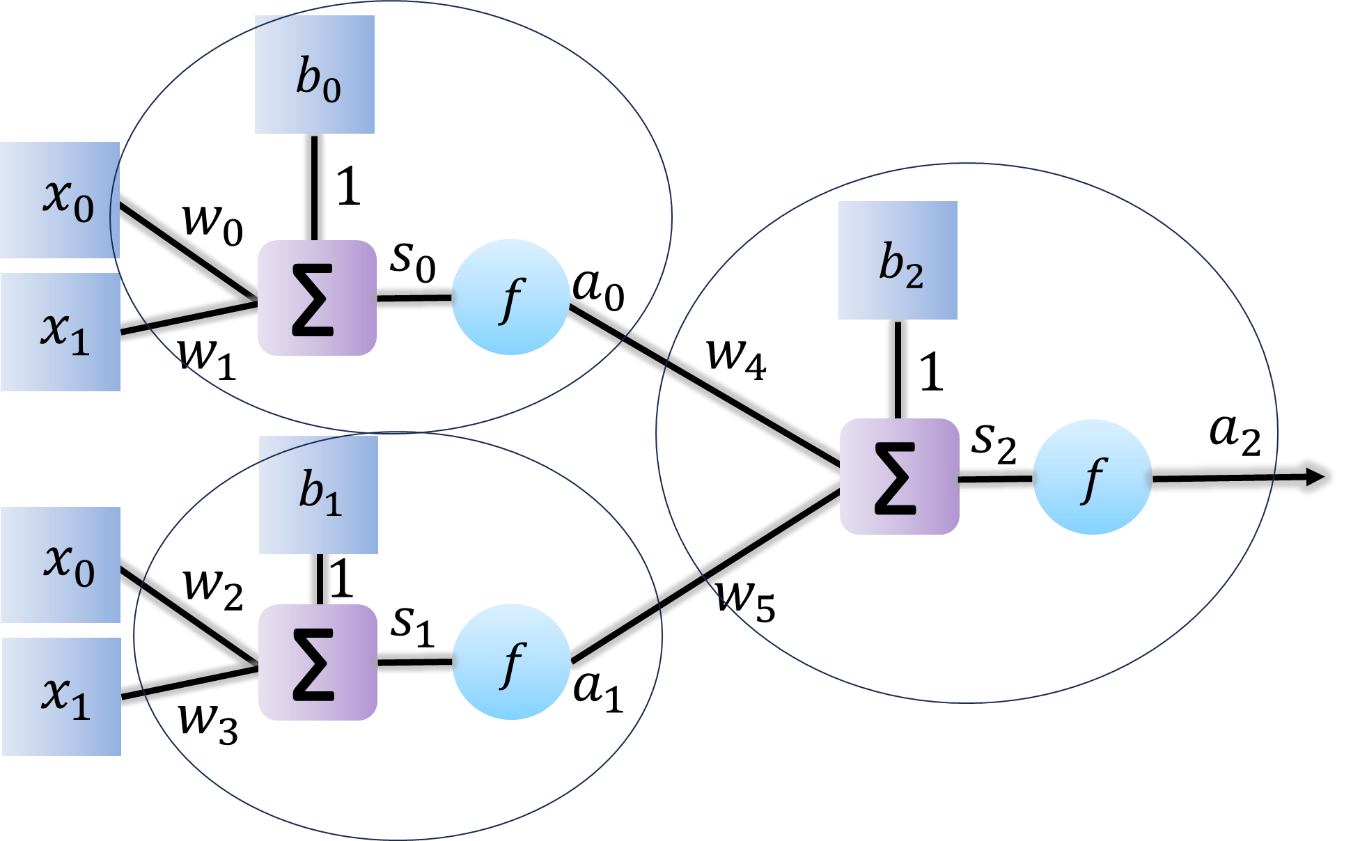
CSC 592: Machine Learning Security and Privacy

**Background**

Backpropagation is the fundamental algorithm for training neural networks by enabling them to learn from data by systematically updating their weights and biases. By computing the gradient of the loss function with respect to each parameter through the chain rule of calculus, backpropagation identifies how each parameter contributes to the model's error. These gradients are then used in conjunction with optimization algorithms, like gradient descent, to minimize the error and improve model performance iteratively. This process allows neural networks to adjust and refine their internal representations, making them capable of capturing complex patterns and relationships in data. Backpropagation is essential for training deep learning models efficiently and has been a driving force behind breakthroughs in areas such as image recognition and natural language processing.

The above paragraph was written using ChatGPT, which was trained using backpropagation. In this lab you will get familiar with the backpropagation algorithm and use it to train a simple 3 neuron model.

**Lab Assignment**

,,,,

Consider the three neuron neural network model shown above (each circle represents one neuron). Note that in lecture we used  to represent both the summation operation for the inputs AND the activation function. However, in the diagram above we have explicitly drawn the summation operation as . This does not change the mathematical operations of the neural network, it is just drawn this way to make the picture clearer.

Derive the forward and back propagation equations for the three neuron network. Then program the neural network using Python. Use the same training data as in homework 1.

**Step by Step Guide**

1. Use the following starter code and fill in the TODO:

import sys

def main():

# y = 2x + 0.3

# x0 corresponds to x coordinate, x1 corresponds to y coordinate of a point

# If the given point is below the y = 2x + 0.3 line, the neural network is

# to output a 0, if the point is above the line, it's output is to be 1.

#------create some training data---------

x0 = [1,2,3,4,5,6,7,8,9,10]

x1 = [2.2,4.5,5.6,8.6,10.15,12.44,14.23,16.2,18.4,20.4]

y = [0,1,0,1,0,1,0,0,1,1] # expected outputs of the network (do not confuse it with

# y coordinate of a point

#-------initialize weights and biases

w0 = 0.1

w1 = -0.23

w2 = -0.2

w3 = 0.7

w4 = -0.1

w5 = 0.5

b0 = 0.22

b1 = -0.1

b2 = 0.3

#-------- train the single neuron network

# need 10000 epochs, with 0.001 learning rate

for i in range(0,10000):

loss = 0

#Go through each sample and update the weights

for j in range(0,len(y)):

s0 = x0[j] \* w0 + x1[j] \* w1 + b0 # forward pass

s1 = x0[j] \* w2 + x1[j] \* w3 + b1

a0 = s0

a1 = s1

s2 = a0 \* w4 + a1 \* w5 + b2

a2 = s2

loss += 0.5 \* (y[j] - a2)\*\*2 # compute loss

#TODO: Compute the gradients

dw4 = 0 #TODO

dw5 = 0 #TODO

db2 = 0 #TODO

dw0 = 0 #TODO

dw1 = 0 #TODO

dw2 = 0 #TODO

dw3 = 0 #TODO

db0 = 0 #TODO

db1 = 0 #TODO

w0 = w0 - 0.001 \* dw0 # update weights, biases

w1 = w1 - 0.001 \* dw1

w2 = w2 - 0.001 \* dw2

w3 = w3 - 0.001 \* dw3

w4 = w4 - 0.001 \* dw4

w5 = w5 - 0.001 \* dw5

b0 = b0 - 0.001 \* db0

b1 = b1 - 0.001 \* db1

b2 = b2 - 0.001 \* db2

print('loss =',loss)

# -----test for unknown data, on the trained network----------

x0 = 2.7 # x coord. of point

x1 = 6.0 # y coord. of point

s0 = x0 \* w0 + x1 \* w1 + b0

s1 = x0 \* w2 + x1 \* w3 + b1

a0 = s0

a1 = s1

s2 = a0 \* w4 + a1 \* w5 + b2

a2 = s2

print('output for (',x0,',',x1,')= ',a2)

x0 = 5.3 # x coord. of point

x1 = 10.4 # y coord. of point

s0 = x0 \* w0 + x1 \* w1 + b0

s1 = x0 \* w2 + x1 \* w3 + b1

a0 = s0

a1 = s1

s2 = a0 \* w4 + a1 \* w5 + b2

a2 = s2

print('output for (',x0,',',x1,')= ',a2)

if \_\_name\_\_ == "\_\_main\_\_":

sys.exit(int(main() or 0))

2. If you need some hints to help you complete the assignment two hints are provided in this step.

Hint 1: Use the forward pass and loss functions to help you derive the updates for each of the weights and biases to get and :

s0 = x0 \* w0 + x1 \* w1 + b0

s1 = x0[j] \* w2 + x1[j] \* w3 + b1

a0 = s0

a1 = s1

s2 = a0 \* w4 + a1 \* w5 + b2

a2 = s2

s3 = a0 \* w6 + a1 \* w7 + b3

a3 = s3

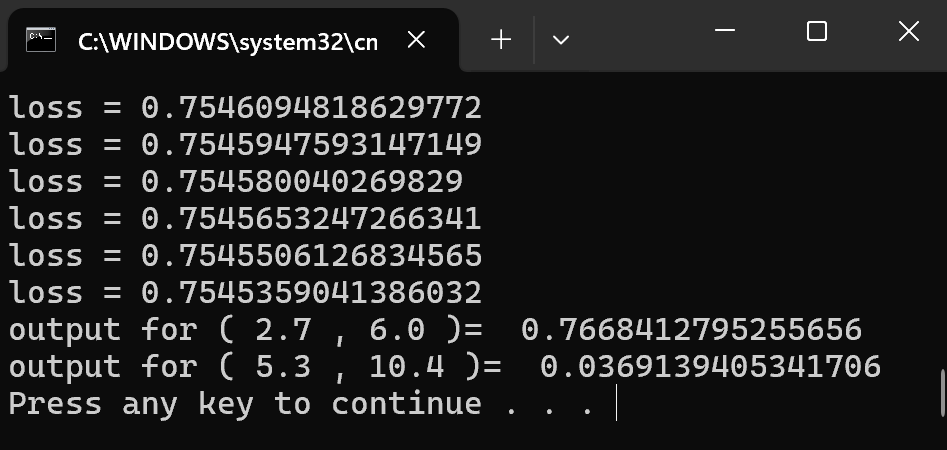
s4 = a2 \* w8 + a3 \* w9 + b4

a4 = s4

If you are still having some trouble or want to double check your calculations, use hint 2.

Hint 2: Backpropagation equations for the three neuron model:

3. After completing the code, you should get the following output:



Submit two deliverables on Brightspace: First your completed code and second a screenshot of the correct code output running on your computer.