**Commented Working Code:**

import autograd.numpy as np

from autograd import grad

import matplotlib.pyplot as plt

from matplotlib.pyplot import figure

#Simple loop to have user select which version to run

while True:

    select = input("Please enter 'relu' or 'sigmoid' or 'combo' for RELU or Sigmoid or Combo of both respectively \n")

    if select in ["relu", "sigmoid", "combo"]:

        print("Running %s" % select)

        break

def feed\_forward(features, w1, b1, w2, b2, w3, b3, select):

    #my Sigmoid functions

    def sigmoid(x):

        return 1/(1+np.exp(-x))

    #my RELU function

    def relu(x):

        return np.maximum(x, 0)

    #run Relu, Sigmoid, or Combo depending on the user input

    #activation function gets ran on every layer

    if select == "sigmoid":

        #sigmoid

        #hidden layer 1

        hl1 = (np.matmul(w1, features))

        hl1\_bias = np.add(hl1, b1)

        hl1\_act = sigmoid(hl1\_bias)

        #hidden layer 2

        hl2 = (np.matmul(w2, hl1\_act))

        hl2\_bias = np.add(hl2, b2)

        hl2\_act = sigmoid(hl2\_bias)

        #output layer

        output = (np.matmul(w3, hl2\_act))

        targets\_predicted = np.add(output, b3)

        targets\_predicted = sigmoid(targets\_predicted)

    elif select == "relu":

        #relu

        #hidden layer 1

        hl1 = (np.matmul(w1, features))

        hl1\_bias = np.add(hl1, b1)

        hl1\_act = relu(hl1\_bias)

        #hidden layer 2

        hl2 = (np.matmul(w2, hl1\_act))

        hl2\_bias = np.add(hl2, b2)

        hl2\_act = relu(hl2\_bias)

        #output layer

        output = (np.matmul(w3, hl2\_act))

        targets\_predicted = np.add(output, b3)

        targets\_predicted = relu(targets\_predicted)

    else: #combo function

        #relu

        #hidden layer 1

        hl1 = (np.matmul(w1, features))

        hl1\_bias = np.add(hl1, b1)

        hl1\_act = relu(hl1\_bias)

        #relu

        #hidden layer 2

        hl2 = (np.matmul(w2, hl1\_act))

        hl2\_bias = np.add(hl2, b2)

        hl2\_act = relu(hl2\_bias)

        #sigmoid

        #output layer

        output = (np.matmul(w3, hl2\_act))

        targets\_predicted = np.add(output, b3)

        targets\_predicted = sigmoid(targets\_predicted)

    return targets\_predicted

def loss(features, w1, b1, w2, b2, w3, b3, targets\_observed, select):

    '''

    w1 is weights matrix for transition from input to first hidden layer

    b1 is the biases added at the first hidden layer

    w2 is weights matrix for transition from hidden layer 1 to hidden layer 2

    b2 is the biases added to the second hidden layer

    w3 is weights matrix for transition from hidden layer 2 to output layer

    b3 is the biases added to the output layer

    Usage: Calculate the sum of square residuals of the feed forward function

    '''

    #Loss function to

    Targets\_Predicted = feed\_forward(features, w1, b1, w2, b2, w3, b3, select)

    return np.sum((Targets\_Predicted - targets\_observed) \*\* 2)

print('You selected: ' + select)

print('Engines Starting ...')

print('Hold Tight Running Epochs')

#set up training datam

#each row is a case

#columns 0-4 are features

#columns 5 & 6 are targets

features\_and\_targets = np.array(

                                   [[0, 0, 0, 0, 0, 0, 1],

                                    [0, 0, 0, 0, 1, 0, 1],

                                    [0, 0, 0, 1, 1, 0, 1],

                                    [0, 0, 1, 1, 1, 0, 1],

                                    [0, 1, 1, 1, 1, 0, 1],

                                    [1, 1, 1, 1, 0, 0, 1],

                                    [1, 1, 1, 0, 0, 0, 1],

                                    [1, 1, 0, 0, 0, 0, 1],

                                    [1, 0, 0, 0, 0, 0, 1],

                                    [1, 0, 0, 1, 0, 0, 1],

                                    [1, 0, 1, 1, 0, 0, 1],

                                    [1, 1, 0, 1, 0, 0, 1],

                                    [0, 1, 0, 1, 1, 0, 1],

                                    [0, 0, 1, 0, 1, 0, 1],

                                    [1, 0, 1, 1, 1, 1, 0],

                                    [1, 1, 0, 1, 1, 1, 0],

                                    [1, 0, 1, 0, 1, 1, 0],

                                    [1, 0, 0, 0, 1, 1, 0],

                                    [1, 1, 0, 0, 1, 1, 0],

                                    [1, 1, 1, 0, 1, 1, 0],

                                    [1, 1, 1, 1, 1, 1, 0],

                                    [1, 0, 0, 1, 1, 1, 0]], dtype=float)

#shuffle our cases

np.random.shuffle(features\_and\_targets)

#transpose Matrix for mat mul in feed forward

features = np.transpose(features\_and\_targets[:, 0:5])

targets\_observed = np.transpose(features\_and\_targets[:, 5:7])

number\_of\_features, number\_of\_cases = features.shape

print('Number of Features:', number\_of\_features)

print('Number of Cases:', number\_of\_cases)

#set initial weights and biases

#use a seed so others can replicate results

np.random.seed(912312)

losses = []

weights\_1 = np.random.rand(4, 5)

biases\_1 = np.random.rand(4, number\_of\_cases)

weights\_2 = np.random.rand(3, 4)

biases\_2 = np.random.rand(3, number\_of\_cases)

weights\_3 = np.random.rand(2, 3)

biases\_3 = np.random.rand(2, number\_of\_cases)

#set our learning rate

lr = 0.00001

#find slope

#grad loss, variable you want to look at

d\_loss\_by\_d\_w1 = grad(loss, 1)  # w1

d\_loss\_by\_d\_b1 = grad(loss, 2)  # b1

d\_loss\_by\_d\_w2 = grad(loss, 3)  # w2

d\_loss\_by\_d\_b2 = grad(loss, 4)  # b2

d\_loss\_by\_d\_w3 = grad(loss, 5)  # w3

d\_loss\_by\_d\_b3 = grad(loss, 6)  # b3

#create epoch for our back tracking.

#backpropagate to calculate the gradient for each weight

epochs = 10000

for epoch in range(epochs):

    weights\_1 -= lr \* d\_loss\_by\_d\_w1(features, weights\_1, biases\_1, weights\_2,

                                     biases\_2, weights\_3, biases\_3,

                                     targets\_observed, select)

    biases\_1 -= lr \* d\_loss\_by\_d\_b1(features, weights\_1, biases\_1, weights\_2,

                                    biases\_2, weights\_3, biases\_3,

                                    targets\_observed, select)

    weights\_2 -= lr \* d\_loss\_by\_d\_w2(features, weights\_1, biases\_1, weights\_2,

                                     biases\_2, weights\_3, biases\_3,

                                     targets\_observed, select)

    biases\_2 -= lr \* d\_loss\_by\_d\_b2(features, weights\_1, biases\_1, weights\_2,

                                    biases\_2, weights\_3, biases\_3,

                                    targets\_observed, select)

    weights\_3 -= lr \* d\_loss\_by\_d\_w3(features, weights\_1, biases\_1, weights\_2,

                                     biases\_2, weights\_3, biases\_3,

                                     targets\_observed, select)

    biases\_3 -= lr \* d\_loss\_by\_d\_b3(features, weights\_1, biases\_1, weights\_2,

                                    biases\_2, weights\_3, biases\_3,

                                    targets\_observed, select)

    losses.append(loss(features, weights\_1, biases\_1, weights\_2,

                       biases\_2, weights\_3, biases\_3,

                       targets\_observed, select))

    #used for testing purposes. If you want to see how the

    #loss backpropagate is calculating a lower gradient uncomment this

    print(epoch, loss(features, weights\_1, biases\_1, weights\_2, biases\_2,

                      weights\_3, biases\_3, targets\_observed, select))

#run feed forward

Targets\_Predicted = feed\_forward(features, weights\_1, biases\_1, weights\_2,

                                 biases\_2, weights\_3, biases\_3, select)

'''

Code to show line graph of the Epochs vs Observed

'''

print('Features : \n', features)

print(' Targets : \n', targets\_observed)

print(' Targets predicted : \n', Targets\_Predicted)

figure(figsize=(10,8), dpi=120)

plt.plot(losses) #plot losses

plt.xlabel('Epochs')  #add x label name

plt.title('Learning Curve using %s Activation Function LR 0.00001' % select) #set title

plt.ylabel('Observed') #add y label name

plt.savefig('%s\_line.png' % select) #save figure

plt.show() #show plot and clear object

'''

Code to show observed vs predicted

'''

N = 22

target1\_predicted = Targets\_Predicted[0, ]

target2\_predicted = Targets\_Predicted[1, :]

target1\_observed = targets\_observed[0, :]

target2\_observed = targets\_observed[1, :]

ind = np.arange(N)

width = 0.35

figure(figsize=(10,8), dpi=120) #set fig size and dpi

plt.subplot(2, 1, 1) #create subplot

plt.bar(ind, target1\_predicted, width, label='Predicted') #create predicted bar

plt.bar(ind + width, target1\_observed, width, label='Observed') #create observed bar

plt.ylabel('Targets 0 or 1') #set y label

plt.title('Closeness of predicted targets for 22 cases - %s' % select) #set title

plt.xticks(ind + width / 2, ind)

plt.legend(loc='best') #set legend place to best/show legend

plt.subplot(2, 1, 2) #set subplot 2 generation

plt.bar(ind, target2\_predicted, width, label='Predicted') #create predicted bar

plt.bar(ind + width, target2\_observed, width, label='Observed') #create observed bar

plt.ylabel('Targets 0 or 1') #set y label

plt.title('Closeness of predicted targets for 22 cases - %s' % select) #set title

plt.xticks(ind + width / 2, ind)

plt.legend(loc='best') #set legend place to best/show legend

plt.savefig('%s\_observation.png' % select) #save figure

plt.show() #show plot

**Results:**

Running combo

You selected: combo

Running Epochs

Number of Features: 5

Number of Cases: 22

Features : [[1. 1. 1. 0. 1. 1. 0. 0. 1. 1. 1. 0. 1. 0. 1. 1. 1. 0. 1. 0. 1. 1.] [0. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1. 0. 1. 0. 0. 1. 1. 0. 1. 0. 1. 0.] [0. 0. 1. 1. 0. 1. 0. 1. 1. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0. 0. 1.] [0. 0. 1. 0. 1. 1. 1. 1. 0. 1. 0. 0. 0. 1. 0. 1. 1. 1. 0. 0. 1. 1.] [0. 0. 0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1.]]

Targets : [[0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0. 0. 0. 1.] [1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0. 0. 0. 1. 1. 1. 1. 0.]]

Targets predicted : [[0.65868495 0.38649967 0.41891895 0.46514931 0.51455131 0.4113774 0.32321588 0.44122416 0.46998526 0.44530997 0.44698204 0.5589786 0.5082679 0.44510287 0.43494124 0.23755726 0.29606325 0.58786062 0.30466912 0.41789123 0.33784127 0.30221717] [0.99665597 0.99996727 0.99999862 0.99946364 0.99998046 0.99997463 0.99968205 0.99996127 0.9999633 0.99993821 0.9999941 0.98550738 0.99991716 0.99993456 0.99993102 0.99999977 0.9999922 0.9990635 0.999993 0.99551305 0.99999134 0.99999597]]

**Graphs:**







