Lusine Kamikyan
Homework 3 Report
BME 595A
Collaborated with Emma Reid

Note: When checking if the gates work, please input the True and False strings as list, i.e

And([True],[False]) or And([True,False,True, False],[True,True,False,False])

Description of each class and its functions:

1. NeuralNetwork class:
   Contains 5 methods: __init__(), getLayer(), forward(), backward(), updateParams()
   It works for any number of layers and any number of inputs with any number of samples for each input. I do not use global variables or helper functions.

- *__init__(self, size_list):*
  The input is self and the number of neurons in each layer
  I create several dictionaries corresponding to activation a, $\bar{a}$, θ, and $\partial E/\partial\theta$. I initialize weight torch tensors with random weights from uniform distribution. I also create $\partial E/\partial\theta$ for each layer to later save derivative values in it.

- *getLayer(self, layer index):*
  Like in the previous homework it returns the weight matrix of the corresponding layer.

- *forward(self, input_tensor):*
  This performs the forward pass through the neural network. I attach the bias to the input tensor, then loop through the layers of the neural network. In each loop I call the *getLayer(layer i) to get the weight matrix,* then the weight matrix Θ_i gets multiplied with the input, then we take the sigmoid of the result. From the result we get the new input to the next layer by again attaching the bias. Then we go back to the next loop. After going through all the layers, we get the final output of the forward pass which is returned by the forward() function. Inside the loop I save the activation a and a_hat in the dictionaries to later use in the backpropagation part.

- *backward(self, target):*
  Accepts the target tensor as input and performs backpropagation. We find the error in each layer, δ using the general formulas from the lecture notes (corrected), and then the $\partial E/\partial\theta$ making sure the first column (bias weights) are removed from the θ weight matrices.

- updateParams(self, eta):
  Updates the weight matrices based on the formula $\theta = \theta - \eta \partial E / \partial \theta$

2. logicGates class:
   Contains 4 classes: And, OR, NOT, XOR
   Each class has the following methods: __init__(), __call__(), forward(), train()
   No global variables were used and no helper functions. And, Or, Xor run for 2 by n vectors and Or runs for 1 by n vector,

- *__init__(self):*
  It initializes the *NeuralNetwork* class. The *AND, OR* and *NOT* only need an input and output layer, but the *XOR* also needs 1 hidden layer with 2 neurons in it.

- *__call__(self,x,y): # only x in case of NOT*
  It calls the forward function of the specific gate. Getting the output from the *forward* we check to see if each element if it is greater than 0.5 (returns True) or less than 0.5 (returns False)

- *forward(self, x,y): # only x in case of NOT*
  The forward method of each gate converts the boolean values into 0 and 1, and makes an input float tensor out of the values and then calls the forward() function of the *NeuralNetwork* class

- *train(self):*
  The train method defines what eta is. Then I define what my input train dataset is and the corresponding labels. Then go through iterations and inside the loop run the forward function of the gate, then the NeuralNetwork backward method, and at the end the updateParams method of the NeuralNetwork class

I have chosen $\eta = 0.5$ (0.2 in case of XOR) and run through 1000 iterations (5000 for Xor) to train them. My dataset is of length 12 (24 for xor), though And, Or, Not gave good training results even with training set length 4, but Xor has a bit more complicated architecture, it required more data and/or different eta and/or higher number of iterations.

Here are the results:

And weights:
Bias: w_0 = -9.0126
Input 1: w_1 = 5.9478
Input 2: w_2 = 5.9478

From the previous homework I had manually chosen them to be:
$w_0 = -1,\ w_1 = 0.6,\ w_2 = 0.6$

Not weights:
Bias: w_0 = -3.0779
Input 1: w_1 = 6.6410
Input 2: w_2 = 6.6410

From the previous homework I had manually chosen them to be:
$w_0 = -1,\ w_1 = 1.1,\ w_2 = 1.1$

Or weights:
Bias: w_0 = 3.7323
Input: w_1 = -7.6747

From the previous homework I had manually chosen them to be:
$w_0 = 0.5,\ w_1 = -1$

Xor weight:
$\theta^1{}_{10} = 2.9027,\ \theta^1{}_{11} = 6.5040,\ \theta^1{}_{12} = 6.5168$
$\theta^1{}_{20} = -7.1720,\ \theta^1{}_{21} = 4.6720,\ \theta^1{}_{22} = 4.6753$
$\theta^2{}_{10} = -4.5412,\ \theta^2{}_{11} = 9.7884,\ \theta^2{}_{12} = -10.4755$

From the previous homework I had manually chosen them to be:
$\theta^1{}_{10} = -1,\ \theta^1{}_{11} = 1.1,\ \theta^1{}_{12} = 1.1$
$\theta^1{}_{20} = 1,\ \theta^1{}_{21} = -0.7,\ \theta^1{}_{22} = -0.7$
$\theta^2{}_{10} = -1,\ \theta^2{}_{11} = 0.73,\ \theta^2{}_{12} = 1.08$

As we can see we have learned different weights through the backpropagation.