

# CI HW1

Sohrab Namazinia

97522085

## A1.

I have used the normal perceptron algorithm to implement that. So, every time you run my code, you get a triple of different weights that work true for the problem.

Every part of my code has comments, but i explain my code here too. My code consists of 2 (and one segment for import statements) sections that I will explain both of them.

### Cell#1:

First i have set the data for this question, 1 as logical 1 and -1 as logical 0

I have separated input and outputs, put them in numpy arrays.

I have added bias to input.

I have set learning rate, number of iterations and randomly initialize weights

I have printed the weights before update for easy comparison to after update.

### Cell #2:

Now I have a for loop with the size of my iteration count.

In each epoch, there is a for loop on the dataset.

In the inner for loop, first I have calculated the inner product of that input data with weight vector.

Then, I passed the result to the sign function to create 1 or -1.

After that, I have computed the error that is the real output of that input data - what we calculated in the previous step.

I have updated the weight vector with the perceptron weight update formula. It means i have added  $\text{error} * \text{input data} * \text{learning rate}$  to the previous value of the weight vector.

After this "while" loop finishes, the weights are ready to use.

I have printed the final weights.

Finally, I have tested the four possibilities of input data with the resulting weights and printed the result. All the answers work true.

## A2.

My code has two Cells for this problem (actually 3 cells, but one is just for importing). My code has been explained by comments, but i explain it here too.

### Cell #1:

First I have fetched the data from 'data.txt'.

I have splitted the input and output into different numpy arrays (X, Y)

I have converted the 0's in Y to -1. Because I want to use the sign function as activation.

I have added bias to input for further simplicity in calculation.

I have initialized weight vectors randomly, also set epoch count, and learning rate.

I have printed the weights before training.

I have defined an array called error\_storage. I will append errors to this array so that finally I plot it.

There is a while loop which repeats n times which n is the epoch counts. In each while loop, first of all, they have decreased the learning rate a bit. Then for every data in the dataset (X), I have calculated the product of w and x (y) and desired response (d). Then I have passed y to the activation function (sign) and subtract the result from d to get the error. If the absolute value of error is 2 ( $1 - (-1) = 2$ ,  $(-1) - (+1) = 2$ ), I have increased the epoch error by one. epoch\_error is the count of misclassifications in that epoch. So I will append it to error\_storage so that finally I plott it. Now i just update the weights:  $w += \text{learning rate} * \text{error} * \text{input data vector}$ .

I have printed the resulting weights after calculations. (bias, W1, W2).

### Cell #2:

Now time for plotting:

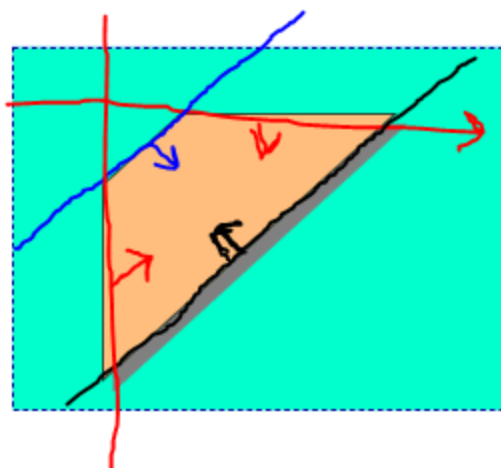
I have plotted the classification result first. Blue dots represent class = [0] and red dots class = [1]. The green line is the decision boundary which has the formula " $\text{bias} + W0 * x + W1 * y = 0$ ". After learning, there are just a few misclassifications.

Also i have plotted the error function, which is simply plotting the error\_storage array based on the iteration number. As you see, it is decreasing and finally the error function is really close to 0.

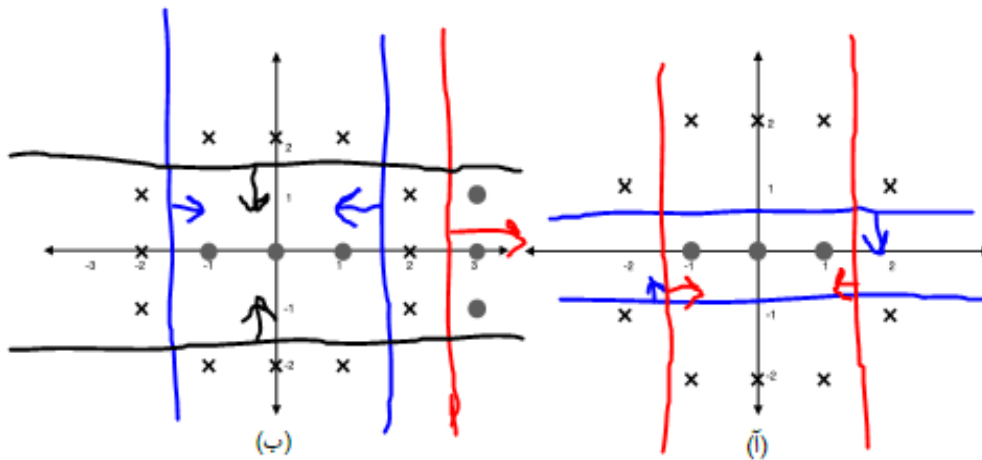
### A3.

How can madaline work for non-separable data? If we have non-separable data, we want to separate some regions from others. These regions are either convex or not. If they are not convex, we can divide them to convex regions. So finally we have some distinct convex regions. Now for each region, we use madaline to distinguish that region. It means, for example, suppose the convex region is the above shape, so with 4 “adaline”s, we can learn each side of this shape.

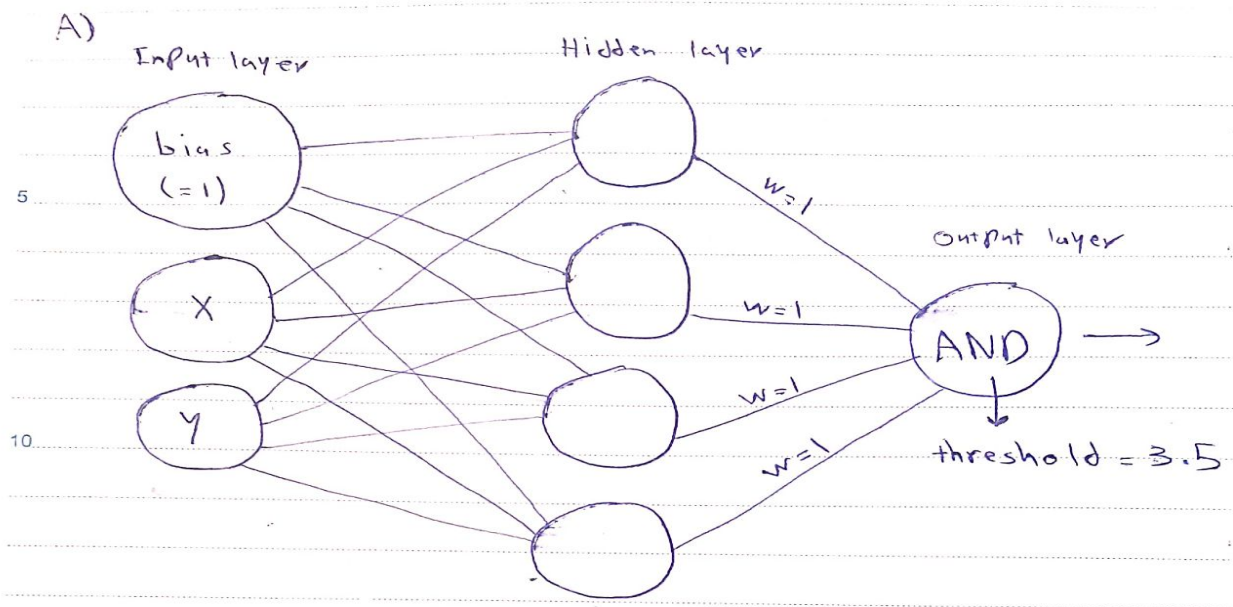
So, we have an input layer and another layer with 4 neurons for 4 sides of this shape. But the output of all of these neurons must be 1 so that we can say a point is inside this shape. So create another layer with one layer that is the output layer. So the previous layer is a hidden layer. The neuron in the output layer should have a functionality like an “And” Function for the outputs from the hidden layer. If all are one, it should be one. Else, 0. So we can put its bias as 3.5 so that it works fine. Now with this network, we can solve one region. So for other regions, we do the same, now we have multiple output neurons, we put all of them as another hidden layer and we add another layer with one neuron as output neuron. This neuron must have the functionality as an “OR” function for the outputs from the hidden layers. because if a point is in one of the regions, so it must result in 1. Totally, our network with 2 hidden layers is constructed.



Can we classify the following points in the following shapes? The answer is yes. The structure is exactly what we explained in the previous paragraph. Also i have drawn the separator line in the following picture. After that, I will draw my NN Arch.

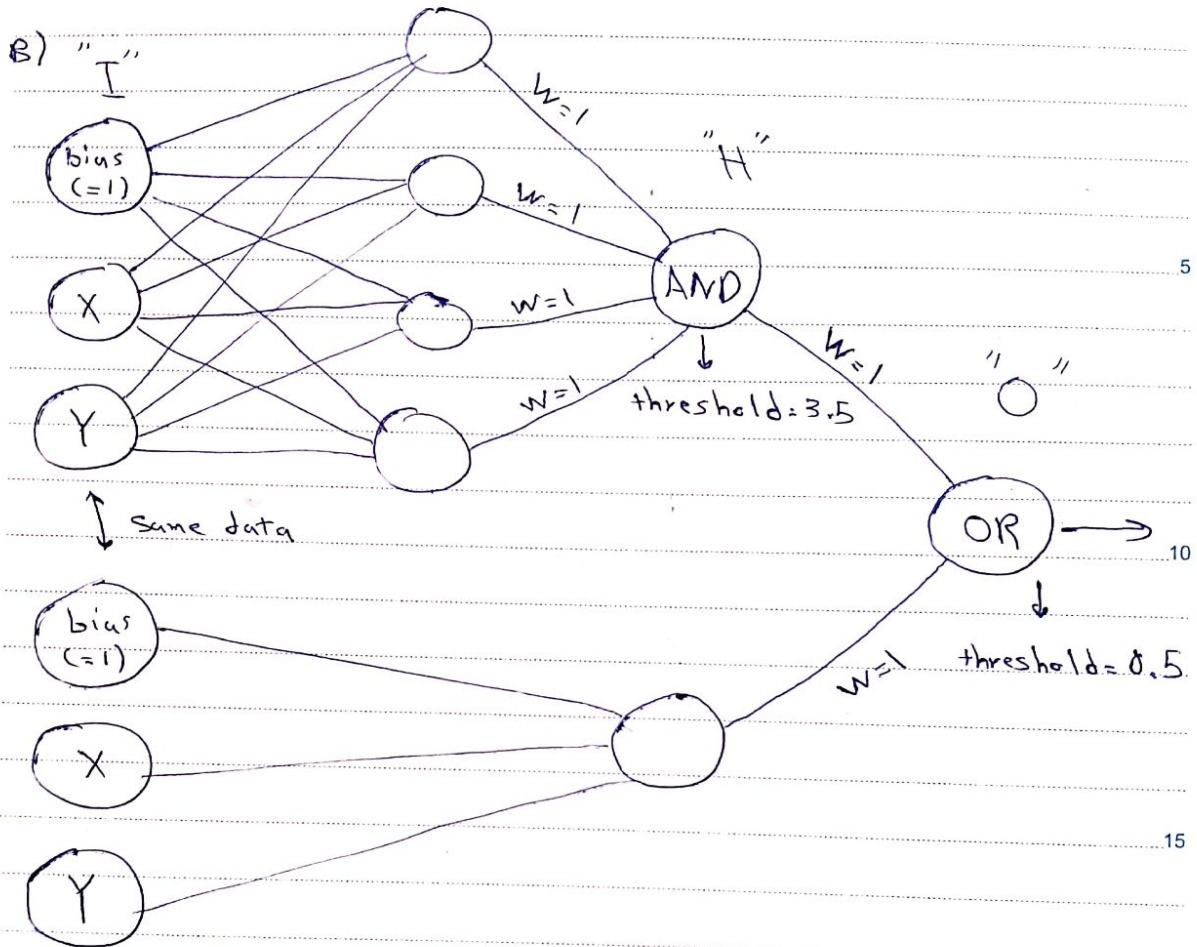


And here is the Arch. of my NN for each problem:




Subject: .....

Year: ..... Month: ..... Day: ..... "H"







**A4.** I have used keras properly for building my network. My code has comments that explain the code properly. But I will explain it here too. My code consists of two cells(actually 3 cells, but one is just for importing) that i explain them here:

#### **Cell #1:**

I have fetched the mnist dataset and get some parameters by that, for example data count, feature count and so on.

I have set the batch size as (1/100) of training data count.

I have written a method for reformatting the data. Because for example the dimensions of the input data are different with what keras methods expect. So in this method, I take training (x, y) and test (x, y) as input and return their refined version as output. For inputs, i reshape them to tuples of shape (data count, feature count) which feature count is  $28 * 28$  that is the number of pixels in each data. So we have 60000 (or 1000 test) data that each one has  $28 * 28$  features. As always in NN, I convert them to float numbers. Also I resize them to a range between 0 and 1. Because they are already between [0, 255] and this may cause late convergence. For output data, there is a keras method called "to\_categorical", I pass it to that. Because our problem is not binary classification and there are 10 possible outputs. What this method does is to convert output to an array of size 10 which one element that is 1, shows the value of that output. Now i just return the results and this is my "reformat\_data()" method.

Now i build my NN. The input shape is what we set the input data in "reformat\_data()" method. Also I have used the rectified linear activation function as activation for my hidden layers and softmax for my output layer. And the number of neurons in my hidden layer is equal to the number of pixels, And this number is 10 in the output layer obviously. My structure is to have hidden layers and it works fine.

Now i compile my model with categorical\_crossentropy loss function because this problem is not binary classification.

Now it is time to train my network with train data with the "fit()" method. For me 10 iterations with batch size equal to (1/100) of data count worked fine. I store the result of training in the history variable so that later I plot it.

Finally I evaluate my NN with test data and print its accuracy and loss rate.

#### **Cell #2:**

I have plotted the accuracy and loss of training that are stored in the history variable that i mentioned. The accuracy increases, the loss decreases. I have done it!

## A5.

I have implemented the vectorized version of MLP backpropagation algorithm for NNs. my NN has 3 layers (1 input, 1 hidden and 1 output). In each epoch in our dataset that is fetched from keras i do backpropagation. My code has 3 cells (4 cells with an import cell) that I explain here, then I will show my calculations for finding derivations, etc.

### Cell #1:

I have defined some useful variables that I will need later like feature count or epoch count.

I have a hidden activation method which is ReLU.

Also I have defined its derivative which is 1 for positive values.

Also I have activation for the output layer which is softmax.

I have an evaluate function which does a part of evaluation for test data.

I have get prediction function which returns the argmax of the second activation.

And i have written a one hot encoder function for outputs.

Two methods for computing accuracy and error.

And a method for printing the result in each epoch.

These methods and variables have been used in the next cell.

### Cell #2:

I have loaded the data, shuffle it and transform the input and output into their correct format for my algorithm.

Then I have called the run method which is the core method of my gradient descent.

In this method, i have first executed forward propagation:

```
z1 = w1.dot(X) + b1
activation_1 = hidden_activation(z1)
z2 = w2.dot(activation_1) + b2
activation_2 = softmax(z2)
```



Then, i have implemented backward propagation:

```
one_hot_Y = one_hot_encode(Y)
diff_2 = activation_2 - one_hot_Y
dw2 = 1 / train_data_count * diff_2.dot(activation_1.T)
db2 = 1 / train_data_count * np.sum(diff_2)
diff_1 = hidden_activation_derivation(z1) * w2.T.dot(diff_2)
dw1 = 1 / train_data_count * diff_1.dot(X.T)
db1 = 1 / train_data_count * np.sum(diff_1)
```

Then i have updated the weights:

```
w1 -= learning_rate * dw1
w2 -= learning_rate * dw2
b1 -= learning_rate * db1
b2 -= learning_rate * db2
```

Finally i have get the predictions and compare them with real outputs, computed the accuracy and error of that epoch.

The above process continues till the number of epochs.

Finally we have our trained bias and weights.

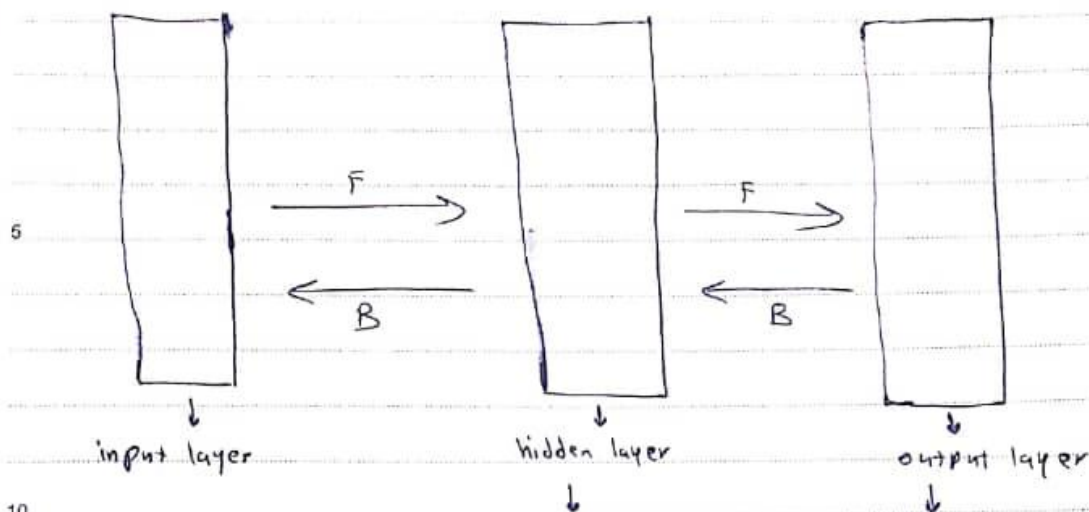
### Cell #3:

Now i have plotted the accuracy and error graphs which x axis is the epoch index. Finally i have made predictions on test data.

Two graphs are shown in output.

In the next picture, you can see the logic of my design for this question.

Subject: \_\_\_\_\_  
 Year: \_\_\_\_\_ Month: \_\_\_\_\_ Day: \_\_\_\_\_



Activation ( $\phi$ ):

ReLU  $\rightarrow$   $\phi' \rightarrow$   $\begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases}$

Activation ( $\phi$ ):

Soft max  $\rightarrow \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$

15 First: forward Propagation, then: Backward Propagation

update rule for weights:

$$w_{ji} = w_{ji} + \eta \times \delta_j \times x_i$$

$\downarrow$  learning rate ( $=0.1$ )       $\downarrow$  local gradient of neuron j.  
 Activated output (explained)

$$\delta_j = \begin{cases} \text{if } j \rightarrow \text{output neuron: } \phi'(v_j)(d_j - y_j) \\ \text{if } j \rightarrow \text{hidden neuron: } \phi'(v_j) \sum_{k \in \text{next layer}} \delta_k w_{kj} \end{cases}$$

