CISC 867 : Deep Learning

Assignment # 1

Name:- Hanan Fared Mohamed OMara.

ID :- 20 39 8559

[1] Neural Network with 2 input neurons, (x_1, x_2) & Three hidden neurons (h_1, h_2, h_3) and 2 output neurons, (y_1, y_2) & The Network acts as a classifier (Two classes)

The equations which govern Network operation :-

$$h_1 = \sigma(x_1 + 1) \quad (1)$$

$$h_2 = \sigma(x_2 + 1) \quad (2)$$

$$h_3 = \sigma(1 - x_1 - 2x_2) \quad (3)$$

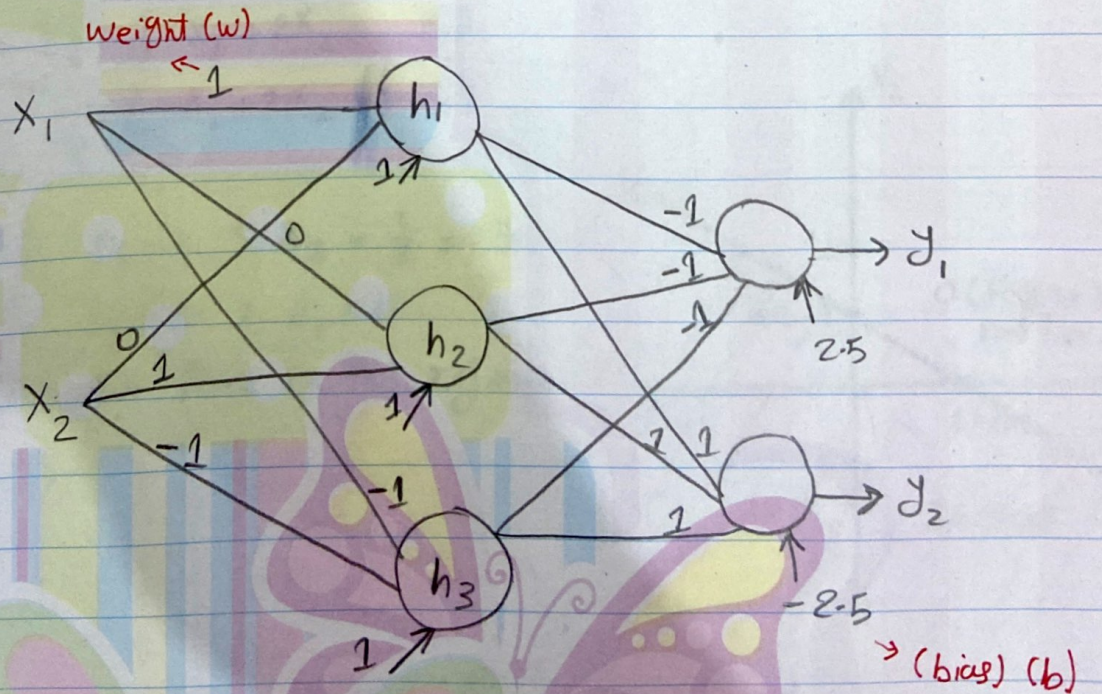
$$y_1 = \sigma(2.5 - h_1 - h_2 - h_3) \quad (4)$$

$$y_2 = \sigma(h_1 + h_2 + h_3 - 2.5) \quad (5)$$

where the activation function is step Function,

$\sigma(x) = 1$ if $x \geq 0$ and else $\sigma(x) = 0$ otherwise.

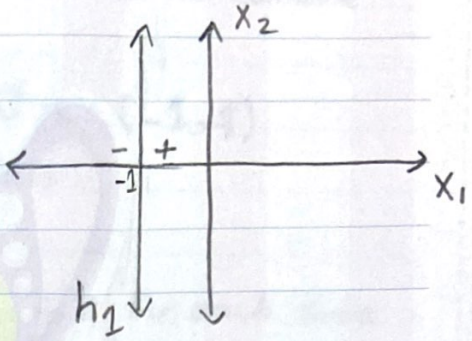
first, lets draw the Network neurons as shown →



[1]

* first solve for h_1 → $h_1 = \sigma(x_1 + 1)$ and we use step function so

$$h_1 = \begin{cases} 1 & x_1 > -1 \\ 0 & x_1 \leq -1 \end{cases}$$



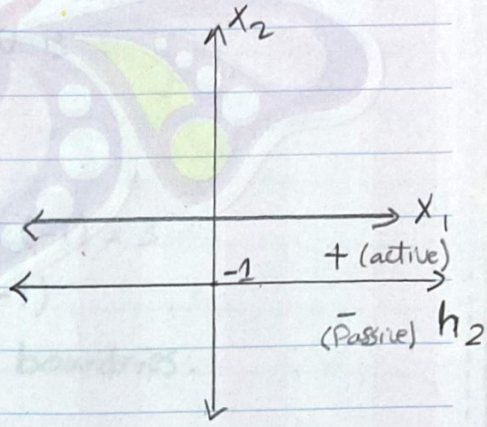
So, this neuron fires when x_1 is greater than -1

* Solve for h_2

$h_2 = \sigma(x_2 + 1)$

$$h_2 = \begin{cases} x_2 > -1 & 1 \\ x_2 \leq -1 & 0 \end{cases}$$

This neuron fires when x_2 is greater than -1



* Solve for h_3

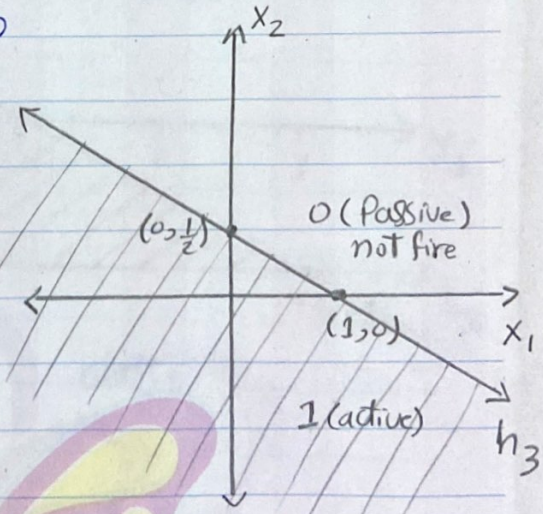
$\therefore h_3 = \sigma(1 - x_1 - 2x_2)$ → $1 = x_1 + 2x_2 \rightarrow 1 > x_1 + 2x_2 \rightarrow x_1 + 2x_2 < 1$

$$\therefore h_3 = \begin{cases} x_1 + 2x_2 < 1 & 1 \\ x_1 + 2x_2 \geq 1 & 0 \end{cases}$$

Put $x_1 = 0$ ⇒ $x_2 = \frac{1}{2} = 0.5$

Put $x_2 = 0$ ⇒ $x_1 = 1$

The Muron fires in this region →



Then we need to find the intersection Points to Combine the three decision boundries,

for h_1, h_2 the intersection Point is $(-1, -1)$

for h_1, h_3 the intersection Point is

$$X_1 + X_2 = 1 - X_1 - 2X_2$$

$$2X_1 = -2X_2 \qquad \div 2 \text{ for each side}$$

$$X_1 = -X_2$$

$\therefore$ The intersection Point is $(-1, 1)$

for h_2, h_3 the intersection Point is

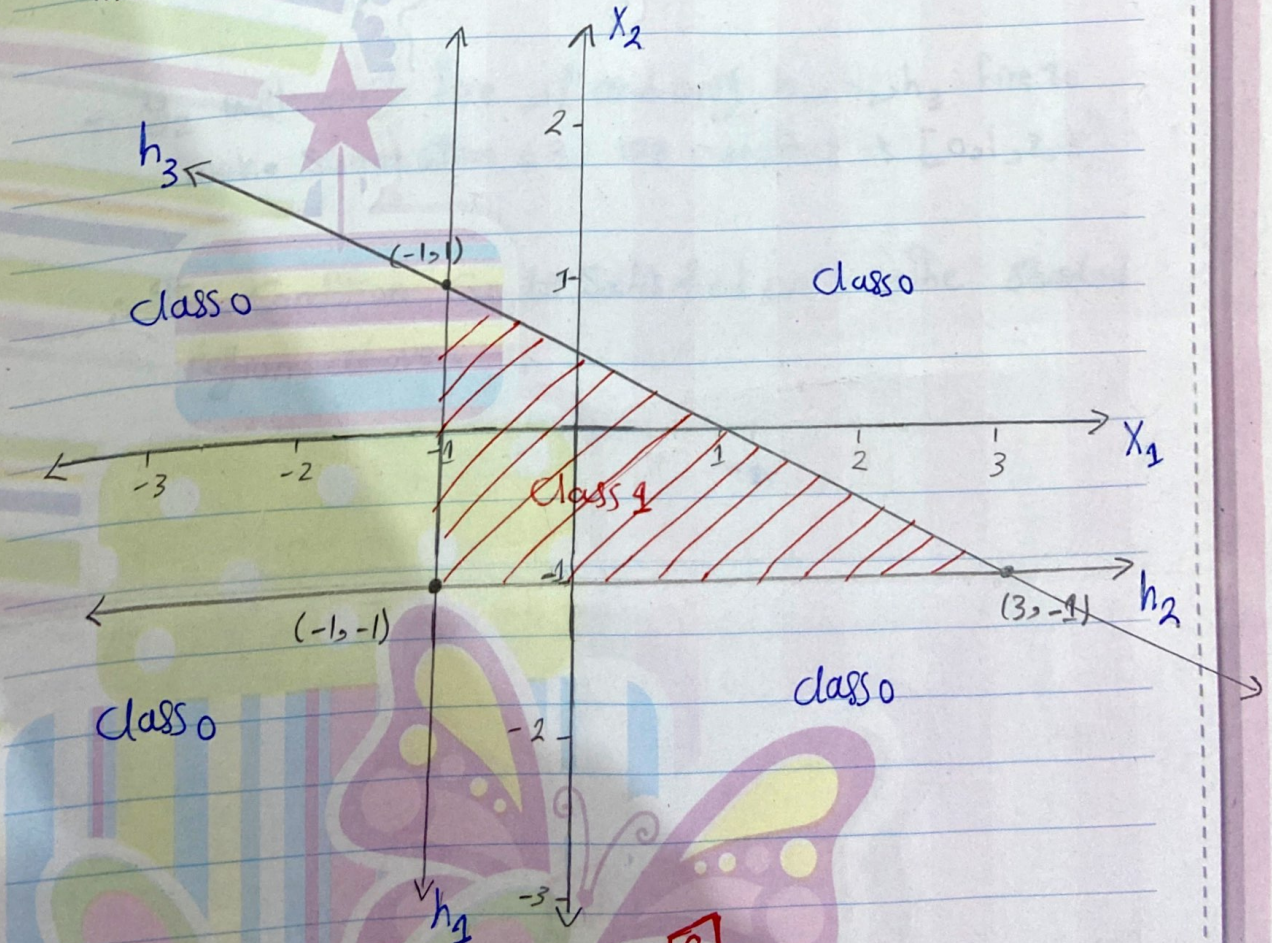
$$X_2 + X_2 = 1 - X_1 - 2X_2$$

$$3X_2 = -X_1$$

$$X_1 = -3X_2 = -3(-1) = 3$$

$\therefore$ The intersection Point is $(3, -1)$

Then Combining the three decision boundries.



* Solving for y_1

$$y_1 = \sigma(2.5 - h_1 - h_2 - h_3) = \begin{cases} h_1+h_2+h_3 < 2.5 & 1 \\ h_1+h_2+h_3 > 2.5 & 0 \end{cases}$$

∵ for each hidden neuron, it can only have 0 or 1 as a value

∵ our space for the summation is $[0, 1, 2, 3]$

∵ our threshold for y_1 to fire is 2.5

∴ y_1 will fire in any region except the shaded region shown above.

alternatively,

$$y_2 = \sigma(h_1+h_2+h_3 - 2.5) = \begin{cases} 1 & h_1+h_2+h_3 > 2.5 \\ 0 & h_1+h_2+h_3 \leq 2.5 \end{cases}$$

∵ y_2 will only fire, if and only h_1, h_2, h_3 fire to make summation = 3 as our space is $[0, 1, 2, 3]$

∴ This condition can be satisfied only in the shaded region above.

2 There is a three layer network (neural network),
Sensitivity $\delta_K = -\frac{\partial j}{\partial net_K}$ at the output node K,

(a) J is chosen as the squared error $J(w) = \frac{1}{2}|t-z|^2$
Sigmoid $f(net_K) = \frac{1}{(1+e^{-net_K})}$ is chosen as activation Function.
where in J(w), t is the target value and z is the Predicted value at the output node K, the net activation of the output node K is given as:

$$net_K = W_K \times h_{K-1}$$

Where W_K is the weight matrix connecting the K-th output node to the (K-1)-Th hidden layer node and h_{K-1} is the output of the (K-1)-Th hidden layer node.

→ Now, we need to Calculate the Sensitivity δK, which is defined as shown above → $\delta_K = -\frac{\partial j}{\partial net_K}$

to find the Partial derivative of j with respect to net_K, we first need to find the Partial derivative of j with respect to z:

$$\frac{\partial j}{\partial z} = (z-t)$$

Next, we need to find the Partial derivative of z with respect to net_K:

$$\frac{\partial z}{\partial net_K} = f'(net_K)$$

where $f'(net_k)$ is the derivative of the sigmoid function with respect to net_k and is given as:

$$f'(net_k) = f(net_k)(1 - f(net_k))$$

Therefore, the sensitivity δk is given as :-

$$\delta k = \frac{-\partial j}{\partial net_k} = \frac{-\partial j}{\partial z} \times \frac{\partial z}{\partial net_k}$$

$$\delta_k = +(z - t) \times f'(net_k)$$

$$\boxed{\delta_k = -(t - z) \times f(net_k) \times (1 - f(net_k))}$$

and also we can write it as

$$\delta_k = (z_k - t_k) * z_k * (1 - z_k)$$

** when z_k is near to 0 or 1, δ_k will be close to 0 even if $(z_k - t_k)$ is large. The network will not update parameters if such situation happens and the prediction will remain far away from the target.

→ also we can say if we analyze the expression for squared error case (δ_k), we can see that δ_k depend on the prediction error $-(t-z)$, the derivative of sigmoid and the value of sigmoid function. if the prediction error is large, then δ_k will also be large in magnitude. However, if the sigmoid function output $f(net_k)$ is close to 0.5 (i.e, net_k is close to 0), then δk will be close to zero even if the prediction error is large.

And, This is bad because it means that the sensitivity of the output node k to changes in the input is very low, which can lead to slow convergence and poor performance of the neural network.

(b) Cross Entropy Case :-

in this case, the objective function is chosen as cross-entropy, which is defined as:

$$J(w) = -\sum_{k=1}^{c} t_k \log(z_k)$$

where t_k is the target value for the k-th output node (i.e $t_k = 1$ if the input belongs to the k-th class, and $t_k = 0$ otherwise) and z_k is the predicted output value for th k-th output node (i.e, the output of the softmax activation function). The softmax activation function is defined as:-

$$f(net_k) = \frac{e^{net_k}}{\sum_{j=1}^{c} e^{net_j}}$$

where net_k is the net input to the k-th output node (i.e, the weighted sum of the inputs to the output node) and the sum in the denominator is taken over all output nodes.

→ we want to calculate the sensitivity $\delta_k = -\frac{\partial J}{\partial net_k}$ at the output node k. To do this, we first need to calculate the derivative of the objective function J with respect to the net input net_k. we can use the chain rule to do this:-

$$\because \frac{\partial z_k}{\partial net_k} = \frac{e^{net_k} * \sum_{k'} e^{net_{k'}} - e^{net_k} * e^{net_k}}{\left(\sum_{k'} e^{net_{k'}}\right)^2} = z_k(1-z_k)$$

$$\because \frac{\partial z_{k''}}{\partial net_k} = \frac{-e^{net_k} * e^{net_k}}{\left(\sum_{k'} e^{net_{k'}}\right)^2} = -z_k * z_{k''}$$

$$\therefore \delta_k = -\frac{\partial J}{\partial net_k} = -\sum_{k'} \frac{\partial J}{\partial z_{k'}} * \frac{\partial z_{k'}}{\partial net_k}$$

$$= \sum_{k'} \frac{t_{k'}}{z_{k'}} * \frac{\partial z_{k'}}{\partial net_k} = t_k(1-z_k) - \sum_{k'' \neq k} t_{k''} * z_k$$

$$= t_k - t_k * z_k - z_k * (1-t_k) = \boxed{t_k - z_k}$$

⇒ Because the non linear activation function is Softmax, the Prediction Sum is to be 1. if the Prediction error is large, Then for the element with target value to be 1, the Prediction value will be far away from 1. That δ_k will be large.

[3] Graph A → represents True gradient descent as the use of gradient descent to a machine learning network to minimize an error function is known as True gradient descent. The error function's gradient is then used to alter the network's design in order to lower the overall error across all training samples. That makes the curve looks smooth.

Graph B → represent Stochastic gradient descent in that the overall objective function are measured as gradient steps in relation to mini-batches. That makes the curves looks noisy.

[4] Answer is → (b) underfitting

because the model can't learn well as the ~~the~~ Model is too simple so it is unable to capture the relationship between input and output variables accurately, generating high error rate on both the training set and unseen data.

as the solution for that is increasing the Model complexity.