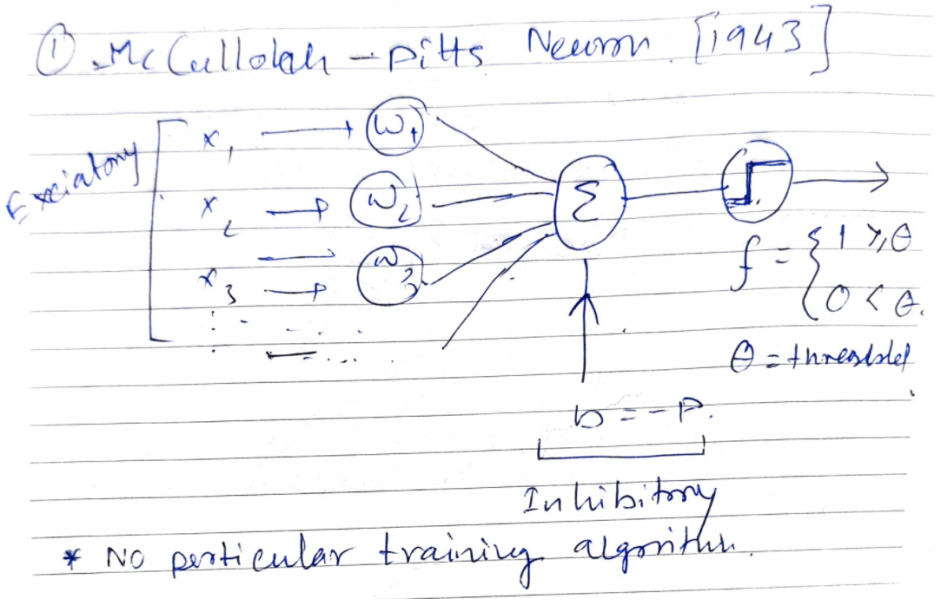
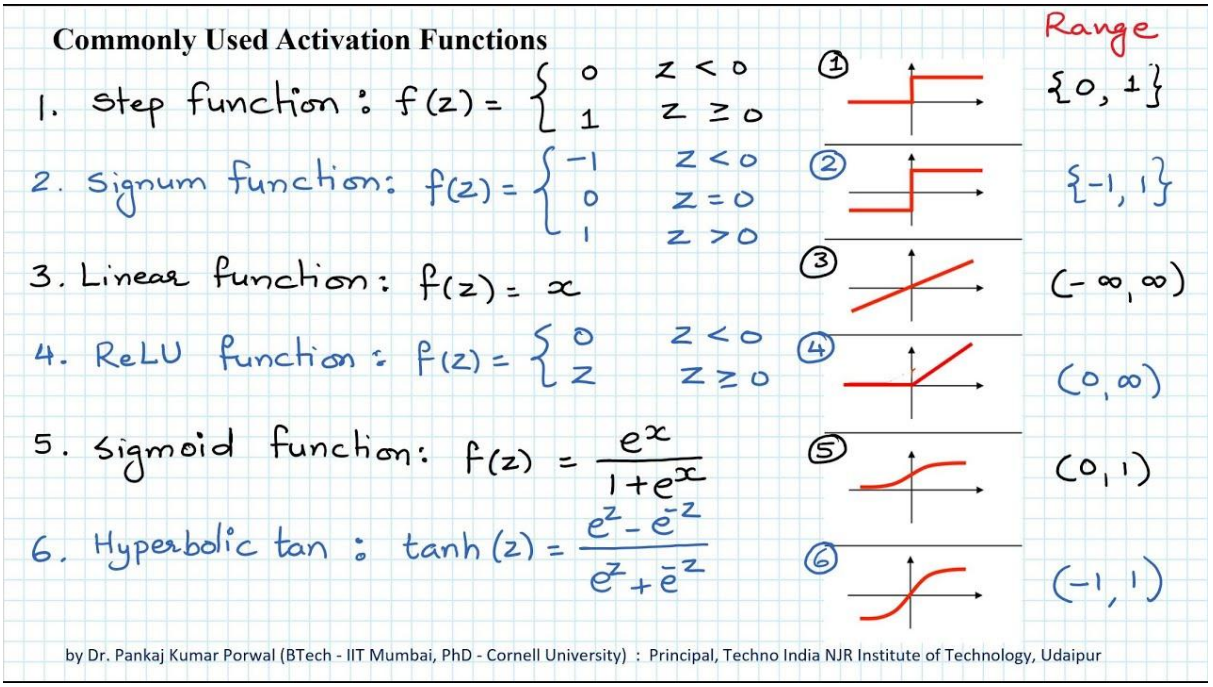
Deep Learning Prep

Q1. Explain different parts of a single-layer neural network with a clear diagram.
Or
Explain McCulloch-Pitts model.



The McCulloch-Pitts neuron was the earliest neural network discovered in 1943. It is usually called as M-P neuron. The M-P neurons are connected by directed weighted paths. It should be noted that the activation of a M-P neuron is binary, that is, at any time step the neuron. The weights associated wilh the communication links may be excitatocy (weight is positive) or inhibioocy (weight is negative). All the excitatory connected weights entering into a particular neuron will have same weights. The threshold plays a major role in M-P neuron: There is a fiXed threshold for each neuron, and if ilie net input to the neuron is greater than the.threshold then ilie neuron fires. Also, it should be noted that any nonzero inhibitory input would prevent the neuro,n from firing. The M-P neurons are most widely used in the case of logic function.

Q2. Explain different types of transfer functions used in neural networks.



Q3. Explain the importance of transfer function, weight and bias.

Importance of TF:

Transfer Function used for introducing non linearity. In neural network. Without TF the Neural Network will contain only Linear Combination of input. Which can't handle Non- Linear Decision Boundary.

The activation function is central to the idea of neural networks for two reasons:

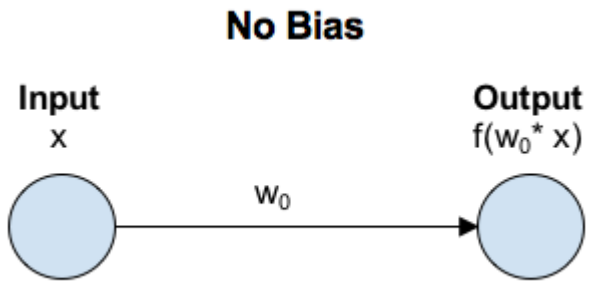
- · First, if there were no activation functions, the whole neural network could be reduced to a group of linear function of the network input - one linear function for each output neuron. So, without activation functions, a neural network could not learn non-linear relationships.
- · And second, each neuron can be seen as recognising a certain feature, with an activation of 0 indicating the absence of that feature. A negative value can't be interpreted in this framework (e.g. if the feature is, say, a round shape, then a positive value indicates the strength with which the network believes that there is a round shape and a value of 0 means, that there is no round shape. Obviously, there can't be less than no round shape.)

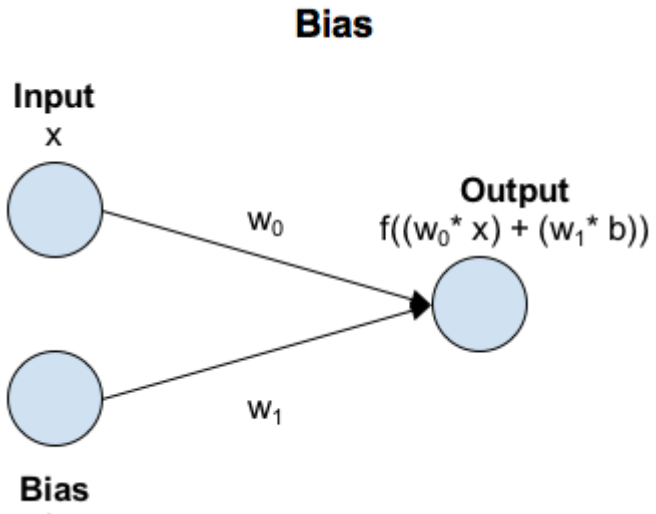
Importance of Weight:

Weights are used for arranging the input space in an importance hierarchy. Without weights all the variables of input space will have same importance, which will definitely not results a good prediction.

Importance of Bias:

Bias allows you to shift the activation function by adding a constant (i.e. the given bias) to the input. Bias in Neural Networks can be thought of as analogous to the role of a constant in a linear function, whereby the line is effectively transposed by the constant value.





Q4. The input to a single input neuron is 2.0, its weight 2.3 and its bias -3.

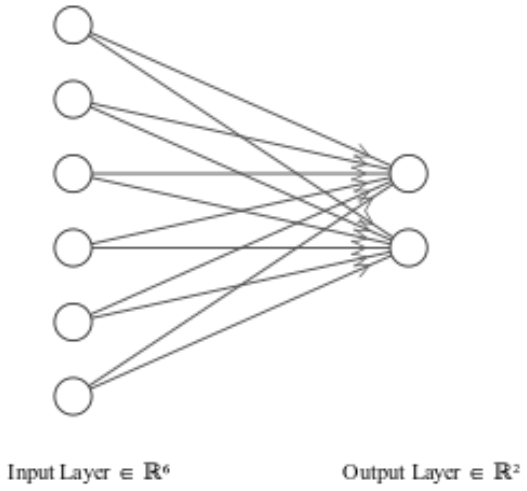
A. What is the net input to the transfer function?
B. What is the output of the above neuron if it has the following TF?
   a. Hardlim
   b. Linear / purelin
   c. Sigmoid
   d. Symmetric hardlim
   e. Hyperbolic tan

A) Net input = xw + b = 2.0 * 2.3 + -3 = 1.6
B) The outputs are-
   a) **Hardlim = 1**

b) **Linear / purelin = 1.6**
c) **Sigmoid = 0.83**
d) **Symmetric hardlim = 1**
e) **Hyperbolic tan = 0.92**

Q5. A single layer NN is to have 6 inputs and 2 outputs. The outputs r to be limited to and continuous over the range (0,1). What can u tell about this architecture?

A. How many neurons are required?
B. Dimension of weight matrix.
C. Type of transfer function can be used?
D. Is bias required?



Minimum Neurons Required: 8

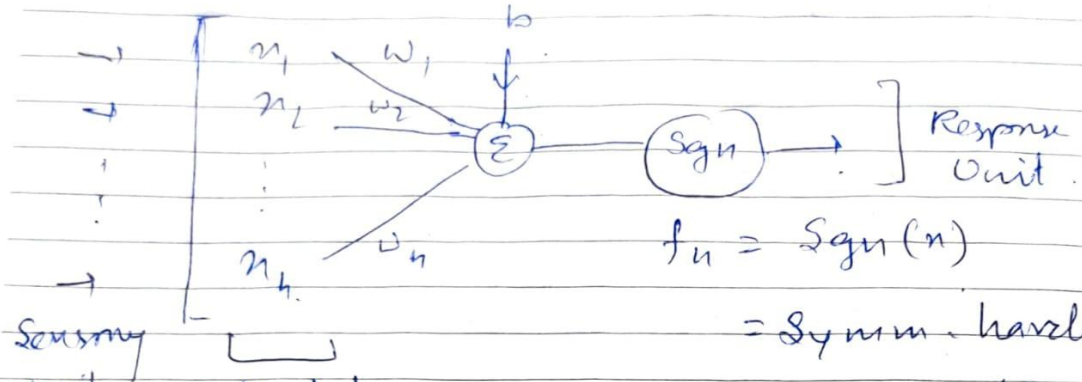
Dimention of Weight matrices: [2,6]

Type of Transfer Function: Sigmoid

Bias: As per the question there is nothing mention about Input and Output type, so that we can draw any conclusion from. The model specifically mentioned in the question probably not required bias, though we can use it for in the architecture and if not required we can make b = 0 anytime.

Q6. Perceptron Learning Algorithm

(3). Rosenblatt's Perceptron: [1958]



$f_n = Sgn(n)$

$= Symm.\ hardlim$

$$= \begin{cases} -1 & n < 0 \\ 0 & n = 0 \\ +1 & n > 0 \end{cases}$$

Perceptron Learning Theorem:

$$w_i(new) = w_i(old) + \alpha [t - y] x_i$$

$$t = \begin{cases} +1 & x \in C_1 \\ -1 & x \in C_2 \end{cases}$$

* C_1 & C_2 are two class for binary classification

If soln exist this learning will generate correct response for all training pattern within finite number of steps.

Q7. Different NN Architecture

The arrangement of neuron to form layers and the connection pattern formed within and between layers is called the network architecture. There exist five basic types of neuron connection architectures. They are:

1. single-layer feed-forward network;
2. multilayer feed-forward network;
3. single node with itS own feedback;
4. single-layer recurrent network;
5. multilayer recurrent network.

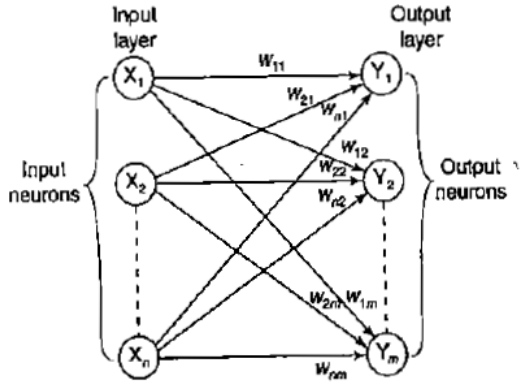


Figure 2-6 Single-layer feed-forward network.

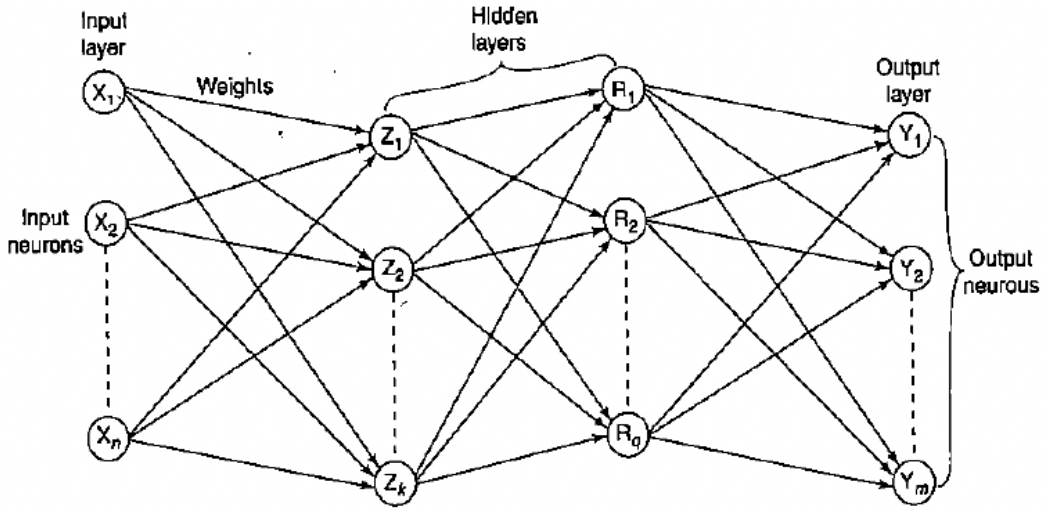


Figure 2-7 Multilayer feed-forward network.

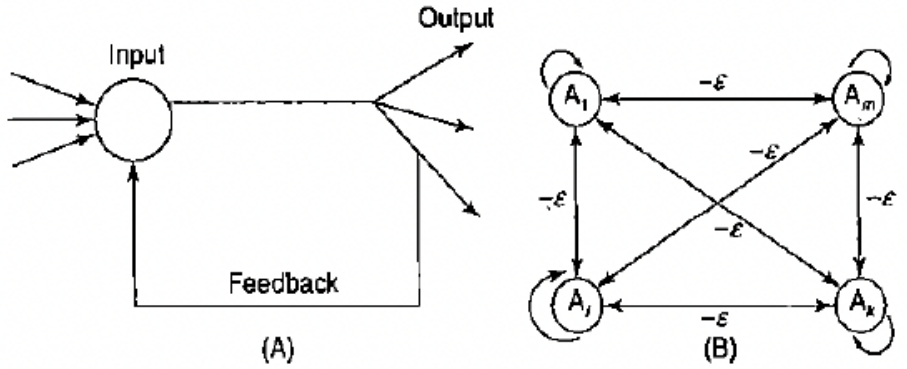


Figure 2-8 (A) Single node with own feedback. (B) Competitive nets.

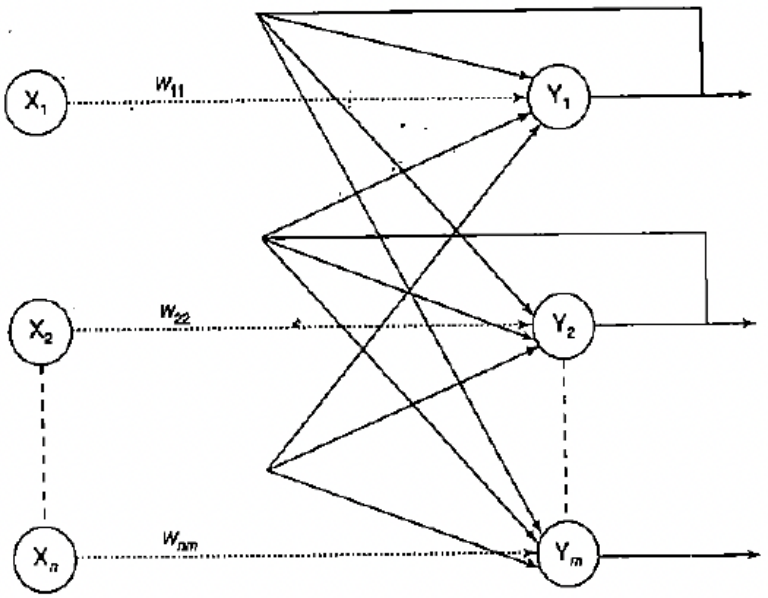


Figure 2-9 Single-layer recurrent network.

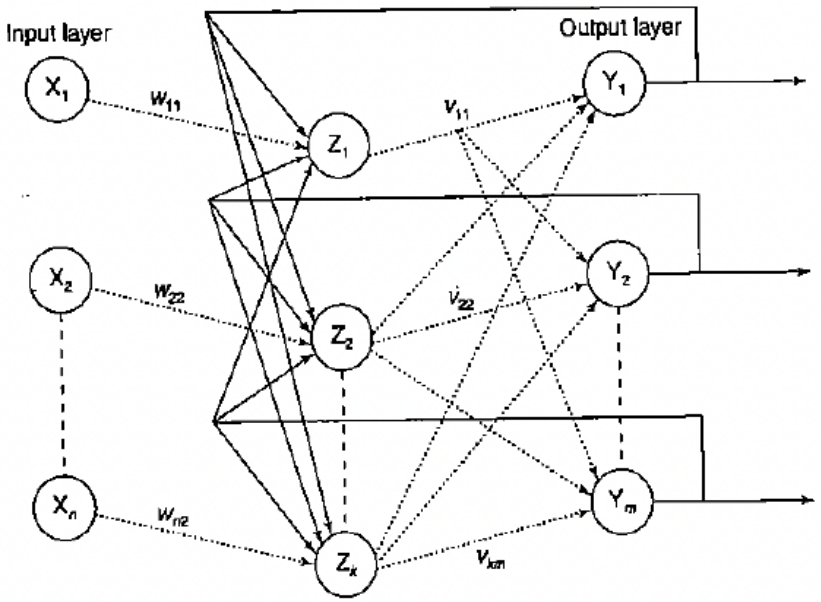


Figure 2-10 Multilayer recurrent network.

Q8. What is ADALINE? Different types of training algorithms used for ADALINE network.

What is Adaline? what different types of training algorithms used for Adaline network?

Adaline (Adaptive Linear Neuron):

→ Adaline is a single layer neural network which uses linear activation function (bipolar activation function) for its input and target output ↓(1, -1)

→ Learning rate is between (0.1 to 1) & Bias & weight ≠ 0

→ Adaline use delta rule to update the weights between the connections to minimize the difference between ou value & target value.

Delta rule for change in weights

$$\Delta w_i = \alpha (t - y_{in}) x_i$$

where;

Δw_i = change in i^{th} weight.

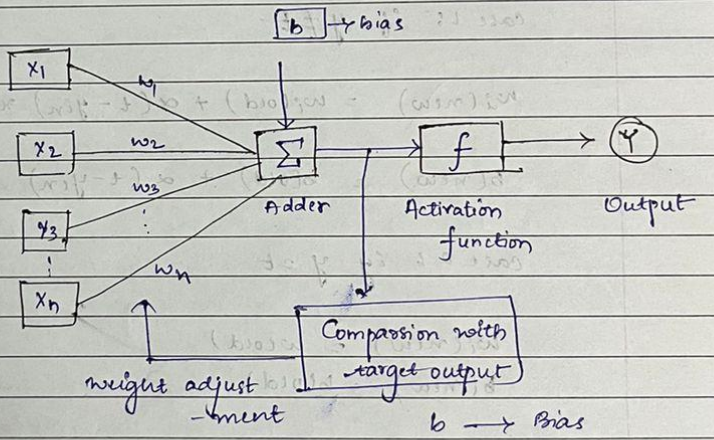
α = learning rate (0.1 to 1)

x_i = input vector to (-1, +1)

y_{in} = net input to output unit.

Architecture :-

The basic structure of Adaline is similar to perceptron having an extra feedback loop with the help of which the actual output is compared with the desired/target. After Comparsion on the basis of training Algorithm, the weights & bias are updated.



Training Algorithm :-

Step 1 :- Initialize the following to start the training.

- → weights
- → Bias
- → learning rate α.

for easy calculation and simplicity, weights & bias must be set equal to 0 and the learning rate must be set to 1

Step 2 : Continue step 3-8 when stopping cal is not true.

Step 3 : Continue step 4-6 for every bipolar training set.

Step 4 : Activate each input unit as follows

$$x_i = s_i \; (i = 1 \text{ to } n)$$

Step 5: obtain the net input with the following relation -

$$y_{in} = \sum_{i}^{n} x_i w_i + b$$

where b is bias and n is total number of input neurons

Step 6: Apply the following activation function to obtain the final output

$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

Step 7:- Adjust the weights & bias as follows if

case 1: if $y \neq t$

$$w_i(new) = w_i(old) + \alpha (t - y_{in}) x_i$$

$$b(new) = b(old) + \alpha (t - y_{in})$$

case 2: if $y = t$

$$w_i(new) = w(old)$$
$$b(new) = b(old)$$

Here 'y' is the actual output & 't' is the desired / target output

$(t - y_{in})$ is the computed error.

Step 8:- Test the stopping condition, which will happen when there is no change in weight or the highest weight change occurred during training is smaller than the specified tolerance.

Q9. What is Hamming network?

Hamming Net

Hamming Distance → dissimilarity of 2 vector.

$x_i^T x$ = No of sim. − No of dissimilarity

= No of sim − Hamming Distance

No of similarity of HD = Total no. of elements

$x_i^T x = n - HD - HD = n - 2HD$

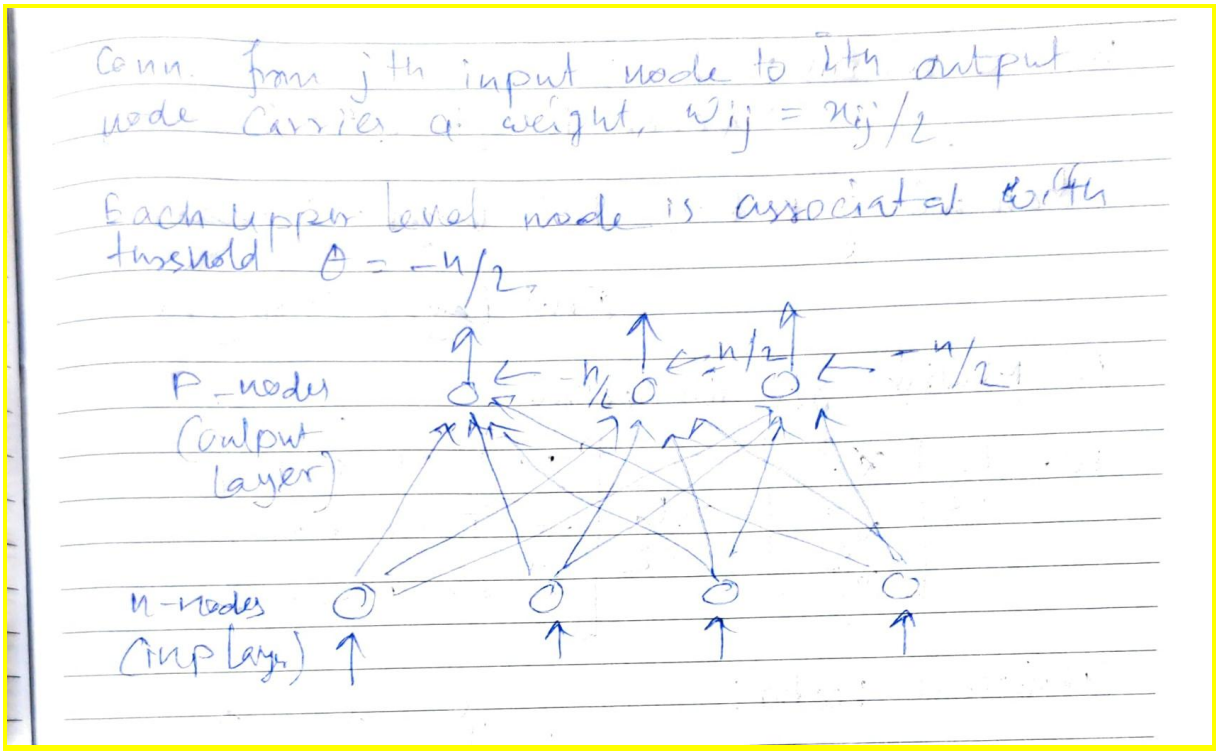
no. of similarity

$-HD = \frac{1}{2}[x_i^T x] - n/2$

$O = WX + \theta$ → bias.

output

$W = \frac{1}{2} x_i^T = \frac{1}{2} \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_p^T \end{bmatrix}$ $\theta = \frac{1}{2} \begin{bmatrix} -n \\ -n \\ \vdots \\ -n \end{bmatrix}$



Q10. Explain linear separability or decision boundary.

An ANN does not give an exact solution for a nonlinear problem. However, it provides an approximate solution to nonlinear problems. Linear separability is the concept wherein the separation of input space into regions is based on whether the network response is positive or negative.

A decision line is drawn to separate positive and negative responses. The decision line may also be called as the decision-making Line or decision-support Line or linear-separable line. The necessity of the linear separability concept was felt to clarify and classify the patterns based upon their output responses.

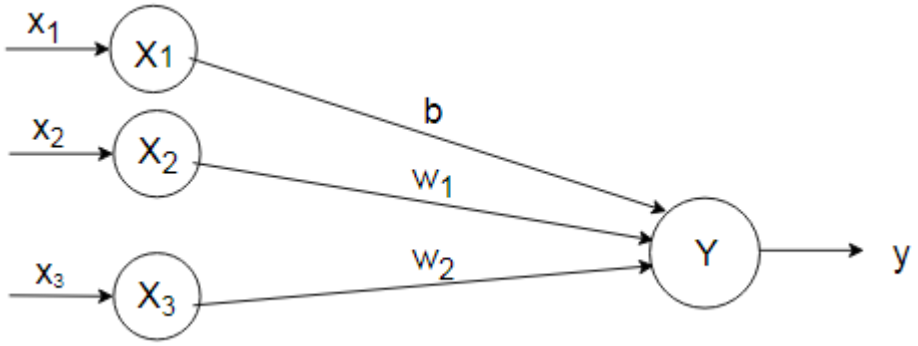
Generally, the net input calculated to the output unit is given as -

$$y_{in} = b + \sum_{i=1}^{n}(x_i w_i)$$

⇒

The linear separability of the network is based on the decision-boundary line. If there exist weight for which the training input vectors having a positive (correct) response, or lie on one side of the decision boundary and all the other vectors having negative, −1
, response lies on the other side of the decision boundary then we can conclude the problem is "Linearly Separable".

Consider, a single layer network as shown in the figure.



The net input for the network shown in the figure is given as-

$$y_{in} = b + x_1 w_1 + x_2 w_2$$

The separating line for which the boundary lies between the values x_1 and x_2, so that the net gives a positive response on one side and negative response on the other side, is given as,

$$b + x_1 w_1 + x_2 w_2 = 0$$

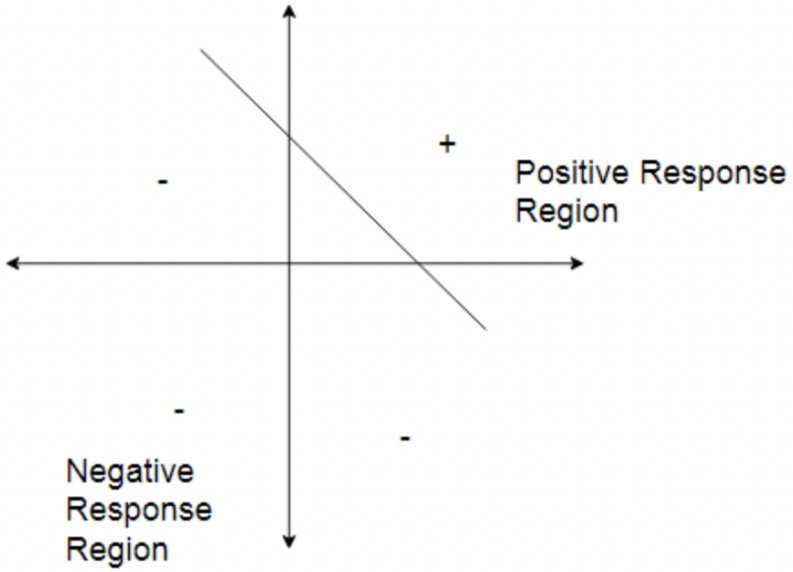
If weight w_2 is not equal to 0 then we get,

$$x_2 = \frac{-w_1}{w_2}, x_1 = \frac{-b}{w_2}$$

Thus, the requirement for the positive response of the net is

$$b + x_1 w_1 + x_2 w_2 > 0$$

During the training process, the values of w_1, w_2 and b are determined so that the net will produce a positive (correct) response for the training data. If on the other hand, the threshold value is being used, then the condition for obtaining the positive response from the output unit is, Net input received $> \theta$ (threshold) $y_{in} > \theta$ $x_1 w_1 + x_2 w_2 > \theta$ The separating line equation will then be, $x_1 w_1 + x_2 w_2 = \theta$ $x_2 = \frac{-w_1}{w_2} x_1 + \frac{\theta}{w_2}$ (with $w_2 \neq 0$) During the training process, the values of w_1 and w_2 have to be determined, so that the net will have a correct response to the training data. For this correct response, the line passes close through the origin. In certain situations, even for a correct response, the separating line does not pass through the origin.



Over and Underfitting

overfitting is defined as the situation where the accuracy on your training data is greater than the accuracy on your testing data. Underfitting is generally defined as poor performance on both the training and testing side.

https://medium.com/mlearning-ai/underfitting-and-overfitting-in-deep-learning-687b1b7eb738

Q.11 What is Hebb Learning rule

Hebb Learning rule is used for pattern classification. It is a single layer neural network, i.e., it has one input layer and one output layer. The input layer can have many units, say n. The

output layer only has one unit. Hebbian rule works by updating the weights between neurons in the neural network for each training sample.

We use hebbian learning rule to identify how to improve the weights of a newtork.

This assumes that:

1. If two neighbour neurons are activated and deactivated at the same time, then the weight connecting these neurons should increase.

2. For neurons operating in the opposite phase, the weight between them should decrease.

3. If there is no single connection, the weight shouldn't change

Hebbian Learning Rule Algorithm :

1. Set all weights to zero, wi = 0 for i=1 to n, and bias to zero.
2. For each input vector, S(input vector) : t(target output pair), repeat steps 3-5.
3. Set activations for input units with the input vector i.e., Xi = Si for i = 1 to n.
4. Set the corresponding output value to the output neuron, i.e., y = t.
5. Update weight and bias by applying Hebb rule for all i = 1 to n:

U

(2) Hebbian Learning [1949]

"Neurons fire together, wire together".

x	y	Δw.
+	+	+
−	−	+
+	−	−
−	+	−

Weight & bias updation:

$$w_i(new) = w_i(old) + x_i \cdot y$$

$$b(new) = b(old) + y.$$

Q. 12. Explain LMS:

(5) Least Mean Square:

Loss (L) = Mean Sq Err.

ie $L(w,b) = E[t(k) - y(k)]^2$

$= E\,e(k)^2 = E[e(k)^T \cdot e(k)]$

$\nabla \hat{L}(w,b) = \nabla e(k)^2$

* Approx LMS without expectation of e.

$$\frac{\partial \hat{L}(w,b)}{\partial w} = \frac{\partial e(k)^2}{\partial w}$$

$$= 2e(k)\frac{\partial e(k)}{\partial w}$$

$$\frac{\partial \hat{L}(w,b)}{\partial b} = \frac{\partial e(k)^2}{\partial b} = 2e(k)\frac{\partial e(k)}{\partial b}$$

$$\frac{\partial e(k)}{\partial w} = \frac{\partial}{\partial w}[t(k) - y(k)]$$

03 Sunday

$$y(k) = \sigma\left[\underbrace{w^T x(k) + b}_{z}\right]$$

$$\frac{\partial}{\partial w}\left[t(k) - \sigma\left[w^T x(k) + b\right]\right]$$

$$= -\frac{\partial}{\partial w}\sigma\left[w^T x(k) + b\right]$$

$$= -\frac{\partial \sigma(z)}{\partial z} \cdot \frac{\partial}{\partial w}\left[w^T x(k) + b\right]$$

$$= -\frac{\partial \sigma(z)}{\partial z} x(k) = -\sigma'(z) x(k)$$

$$\frac{\partial e(k)}{\partial b} = \frac{\partial}{\partial b}\left[t(k) - \sigma\left(w^T x(k) + b\right)\right]$$

$$= -\frac{\partial \sigma(z)}{\partial z} \cdot \frac{\partial}{\partial b}\left[w^T x(k) + b\right]$$

$$= -\frac{\partial \sigma(z)}{\partial z} = -\sigma'(z)$$

A/c Steepest Decent,

$$x_{k+1} = x_k - \alpha \nabla L(w, b)$$

$$w(k+1) = w(k) + 2\alpha\, e(k) \cdot \sigma(z) \cdot x(k)$$

$$= w(k) + \eta\, e(k) \cdot \sigma'(z) \cdot x(k)$$

$$b(k+1) = b(k) + 2\alpha\, e(k) \cdot \sigma'(z)$$

$$= b(k) + \eta\, e(k) \cdot \sigma'(z)$$

Q. 13 What is Steepest Descent. Exp with Diagram.

Algorithm 9.4 *Steepest descent method.*

given a starting point $x \in \mathbf{dom}\, f$.

repeat

1. Compute steepest descent direction Δx_{sd}.
2. *Line search.* Choose t via backtracking or exact line search.
3. *Update.* $x := x + t\Delta x_{\text{sd}}$.

until stopping criterion is satisfied.

Q. 14 What is Gradiant Descent.

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates. Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error. Once machine learning models are optimized for accuracy, they can be powerful tools for artificial intelligence (AI) and computer science applications.

Algorithm 9.1 *General descent method.*

given a starting point $x \in \mathbf{dom}\, f$.

repeat

1. Determine a descent direction Δx.
2. *Line search.* Choose a step size $t > 0$.
3. *Update.* $x := x + t\Delta x$.

until stopping criterion is satisfied.

Q. 15 What is directional derivative?

The directional derivative is the rate at which any function changes at any particular point in a fixed direction. It is a vector form of any derivative. It characterizes the instantaneous rate of modification of the function.

15 b) $\vec{\nabla} f$ → Gradient

direction of the slope.

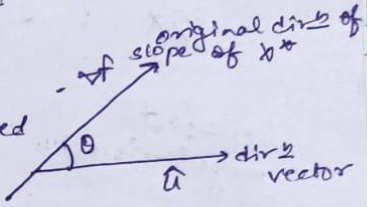
~~which dir~~

$\hat{u}$ → arbitrary dir^n ..

$$\vec{\nabla} f \cdot \hat{u} = \vec{\nabla} f |\vec{u}| \cos\theta = \vec{\nabla} f \cos\theta$$

$|\vec{u}| = 1$

If $\cos\theta$ value is 0, x^* slope dir^n is perpendicular to the considered dir^n $\hat{u}$.



If $\cos\theta = 1$, same dir^n & $\hat{u}$

15 c) $f(x) = x_1^2 + 2x_2^2$

$x^* = [0.5 \quad 0.5]^T$ $\quad p = [2 \quad -1]^T$

cal the grad → $\nabla f = \frac{\partial}{\partial x_1}(x_1^2 + 2x_2^2)\hat{i} + \frac{\partial}{\partial x_2}(x_1^2 + 2x_2^2)\hat{j}$

$= \frac{\partial}{\partial x_1} x_1^2 \hat{i} + \frac{\partial}{\partial x_2} 2x_2^2 \hat{j}$

$= 2x_1 \hat{i} + 4x_2 \hat{j}$

$x^* = [1/2, 1/2]$ $\quad \nabla f|_{x^*} = 1/2 \times 2\hat{i} + 4 \times \frac{1}{2}\hat{j}$

$= \hat{i} + 2\hat{j}$

dir^n derivative → $D_{\vec{p}} f = (\hat{i} + 2\hat{j}) \cdot (2\hat{i} - \hat{j})$

$= 2 - 2 = 0 \quad \because \cos\theta$.

$\boxed{\vec{p} \perp \vec{\nabla} f}$