ARTIFICIAL NEURAL NETWORKS MODULE 3

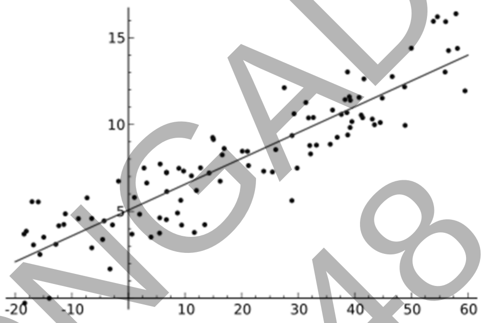
Dr. R.B.Ghongade,
VIIT, Pune-411048

Topics covered

- Linear Regression
- Gradient Descent Algorithm
- More Activation Functions
- Learning Processes
  - Error correction Learning
  - Memory based Learning
  - Hebbian Learning
  - Competitive Learning
- Biases and Thresholds
- Linear Separability
- Perceptron

Linear Regression

- In statistics, *linear regression* is an approach to modeling the relationship between a scalar variable y and one or more explanatory variables denoted X
- Once we know the equation of this fitted line we can use it to find y given X
- Say , we have conducted an experiment that gives us some output value for a set of inputs



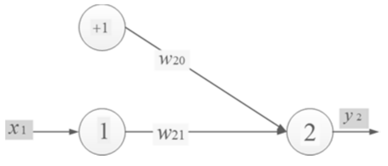
- There are various conventional methods to do this ,e.g. Ordinary least squares
- The best fit line has equation : $y = mx + c$
- We have a choice of two parameters: m and c

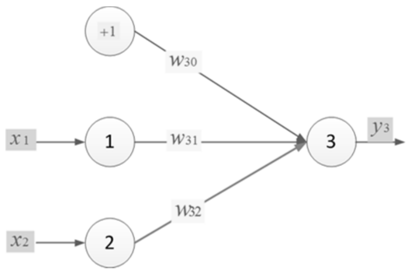
- A linear neuron can be used for this task since output of a linear neuron is:

$$y = x_1 w_{21} + x_2 w_{20}$$

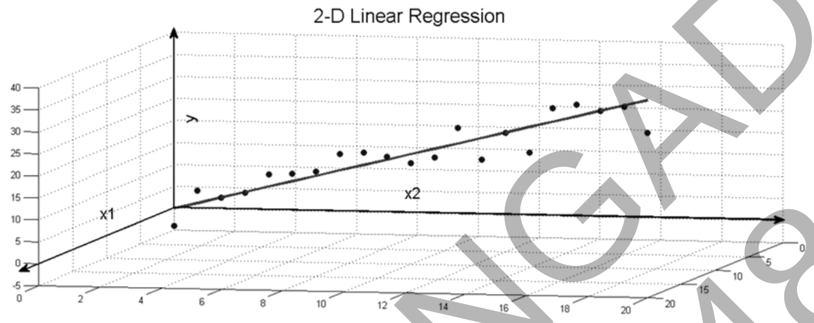
- If we keep x_2 =1, then w_{21} is slope andw_{20} is the intercept, hence we have to modify w_{21} and w_{20} to obtain the best fit line

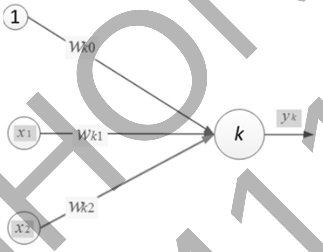
- If we make a good choice of w_{21} and w_{20} our job is over!
- Many real world problems involve more than one independent variables then the regression is called *multiple regression*
- For example if we have two independent variables x_1 and x_2, then it becomes a 2-D problem, the new network would different
- Now we have to adjust two slopes and one intercept
- This can be extended to solve n-dimensional problems



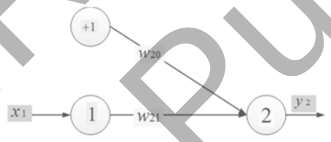


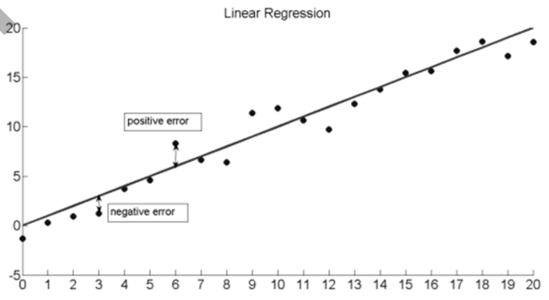
- For a function like $y = f(x_1, x_2)$ we have a 2-D problem



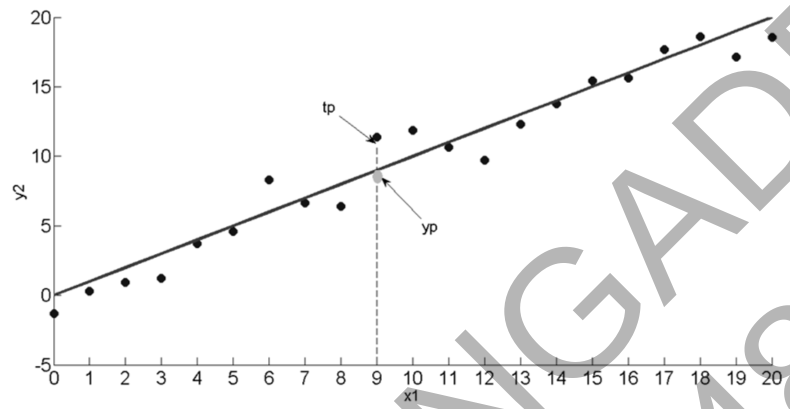


The Concept of Error Energy





- We can have a combined error measure by squaring and adding all the errors and dividing it by number of observations giving MEAN SQUARE ERROR (mse), this in fact is the ERROR ENERGY
- This *mse* shows how good or bad the line is fitted!



- Considering a 1-D problem ,error energy at a point *p* is :

$$e^p = (t^p - y^p)^2$$

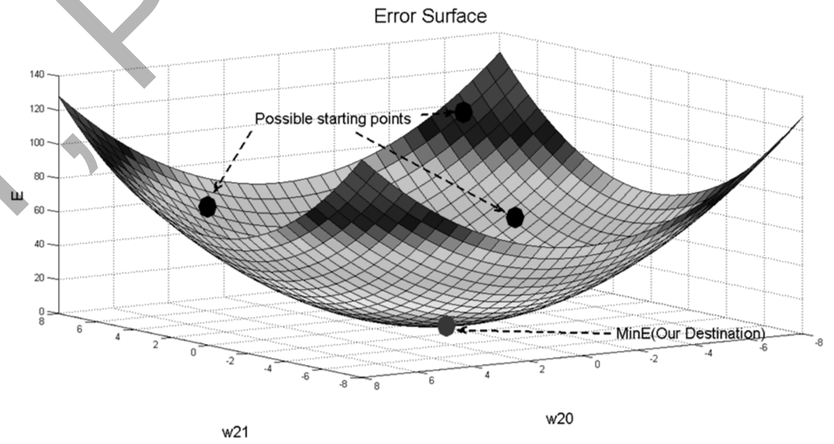
- The total error energy is given by:

$$E = \sum_p e^p = \sum_p (t^p - y^p)^2$$

- Why do we have to measure this error energy?
- In order to get the best fit the error energy should be minimized!

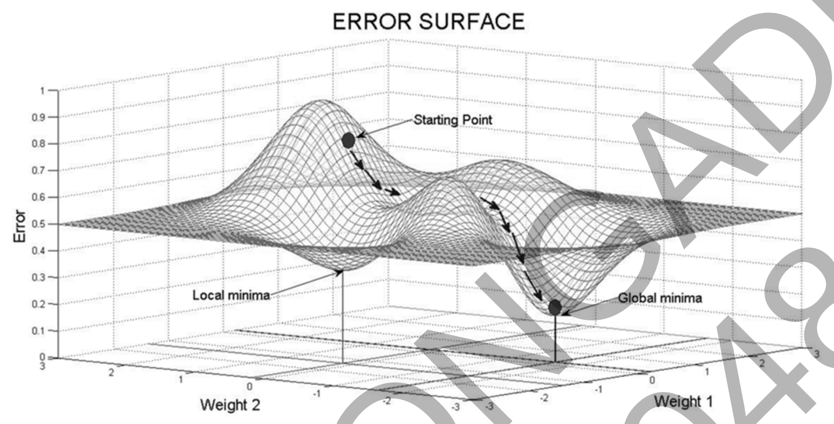
Gradient Descent Algorithm

- If we plot the error energy versus weight w_{21} and w_{20} , we get a error graph showing combined error over the set of all input points for various values of w_{21} and w_{20}



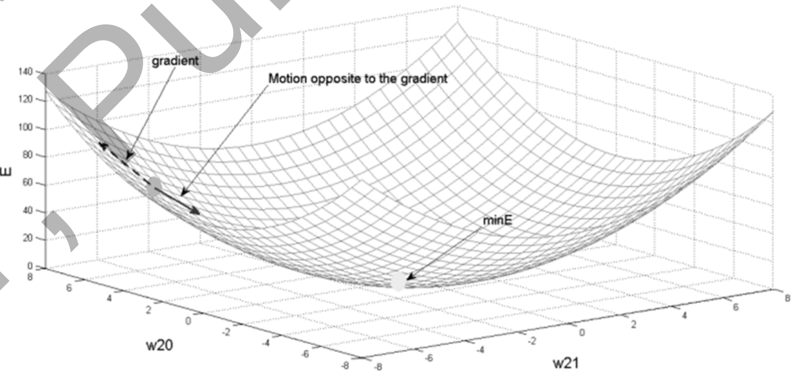
- Objective is to reach the minimum error energy point by adjusting w_{21} and w_{20}, where we say that the ANN has converged

Minima



- We may have two or more minimas, our goal being to reach the global minina
- We cannot guarantee global minima

How do we reach the(hopefully global) minima?



- We can find the gradient(slope) at the starting point and slide down in the opposite direction of the gradient to reach the minima
- This technique is called the steepest descent approach
- Even though reaching global minima is not guaranteed by this approach we may find a good combination of w_{21} and w_{20} which gives lowest fitting error!

The Algorithm

- Let there be *p* observations available for training then

$$E = \sum_p e^p$$

where
$$e^p = \sum \left(t_o^p - y_o^p\right)^2$$

since there may be several outputs such that:

$$y_0 = f_0\ (x_1,\ x_2, \cdots, x_n)$$
$$y_1 = f_1\ (x_1,\ x_2, \cdots, x_n)$$
$$\vdots$$
$$y_m = f_m\ (x_1,\ x_2, \cdots, x_n)$$

For computational convenience we define e^p as:

$$e^p = \frac{1}{2}\sum \left(t_o^p - y_o^p\right)^2 \ldots\ldots\ldots\ldots(1)$$

- Gradient G of error E with respect to any weight w_{ij} is

$$G = \frac{\partial E}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_p e^p$$
$$= \sum_p \frac{\partial e^p}{\partial w_{ij}}$$

we wish to find the gradient with respect to a certain weight w_{oi} and a particular pattern

Using chain rule

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial x}$$

we get

$$\frac{\partial e}{\partial w_{oi}} = \frac{\partial e}{\partial y_o} \cdot \frac{\partial y_o}{\partial w_{oi}}$$

From (1) we have

$$\frac{\partial e}{\partial y_o} = -(t_o - y_o) \ldots\ldots\ldots\ldots (2)$$

- But

$$y_o = \sum_j w_{oj} \cdot x_j$$

$$\therefore \frac{\partial y_o}{\partial w_{oi}} = \frac{\partial}{\partial w_{oi}} \sum_j w_{oj} \cdot x_j$$

$$= x_i \ldots \ldots (3)$$

Combining (2) and (3) we have

$$\frac{\boldsymbol{\partial e}}{\boldsymbol{\partial w_{oi}}} = -(\boldsymbol{t_o} - \boldsymbol{y_o}) \cdot \boldsymbol{x_i}$$

- This is the gradient with respect to one particular weight considering $\boldsymbol{o}$ as the output and $\boldsymbol{i}$ as the input unit
- Now we have to move in the opposite direction of the derivative hence correction to be applied to weights is $-G$

- We apply correction by simply multiplying the error by input thus

$$\Delta \boldsymbol{w_{oi}} = (\boldsymbol{t_o} - \boldsymbol{y_o}) \cdot \boldsymbol{x_i}$$

- Hence the new weight is

$$\boldsymbol{w_{oi}(new)} = \boldsymbol{w_{oi}(old)} + \Delta \boldsymbol{w_{oi}}$$

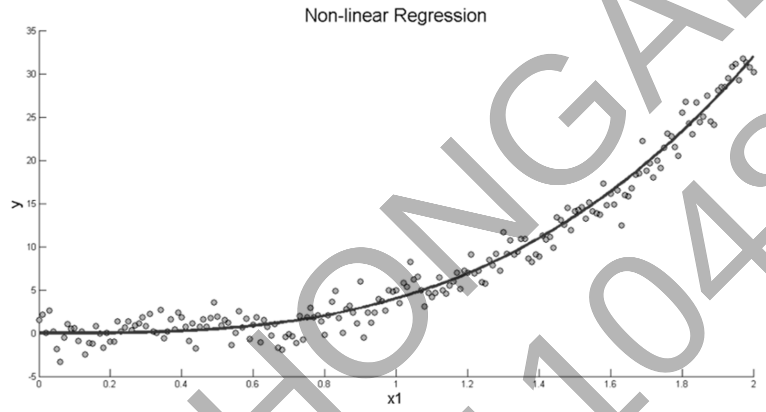
- Generally we apply a constant η as a controlling parameter called as the learning rate

$$\therefore \Delta \boldsymbol{w_{oi}} = \eta(\boldsymbol{t_o} - \boldsymbol{y_o}) \cdot \boldsymbol{x_i}$$

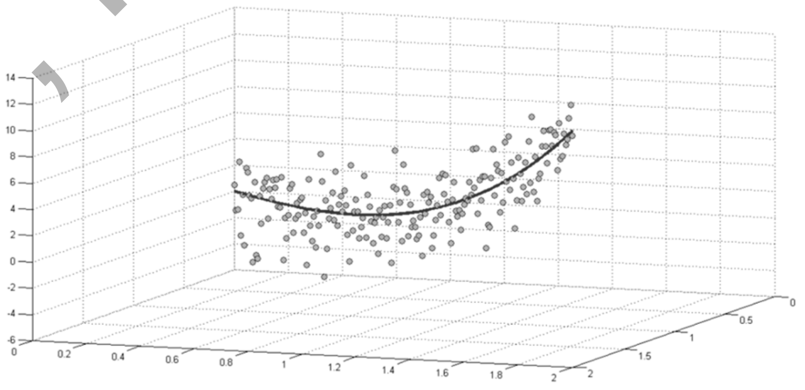
- If learning rate is high the network learns fast and vice-versa
- But a higher learning rate may lead to unstable operation and no learning at all
- If learning is slower, it is at least guaranteed that there is progress
- We have to thus look out for an optimal learning rate

Non-linear Regression

- Real world problems are mostly non-linear hence we have to go for non-linear regression
- This can be done with a non-linear neuron i.e. a neuron with non-linear activation function



2-D Non-linear Regression

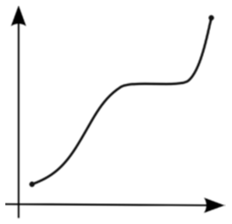


Activation Functions

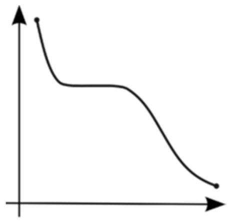
- To map non-linear output functions we require non-linear activation functions
- The activation functions should be
  - Continuous
  - Monotonically increasing
  - Differentiable

Monotonicity

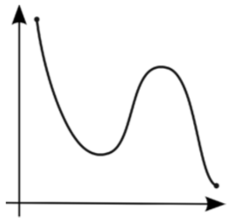
- Monotonically increasing function



- Monotonically decreasing function

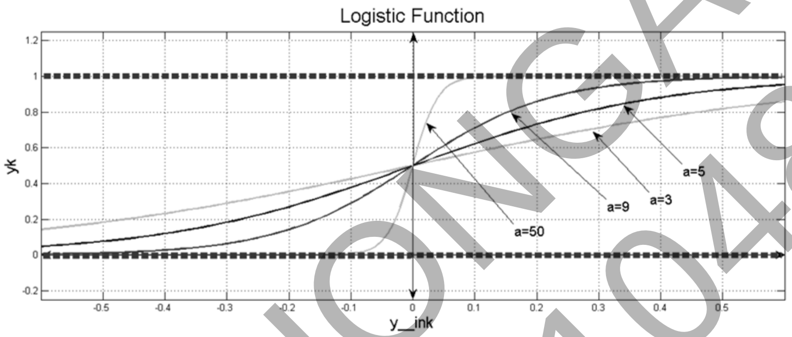


- Non Monotonic function



Activation Functions

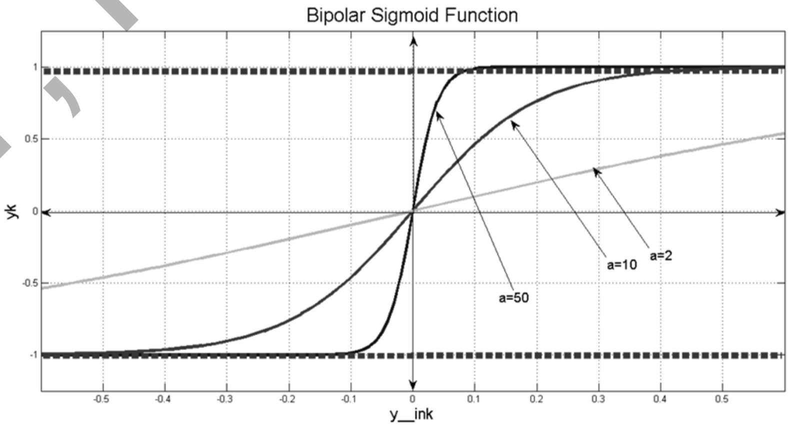
Log Sigmoid $y_k = \dfrac{1}{1+\exp(-ay\_in_k)}$ Output limits:[0,1]



Activation Functions

Bipolar Sigmoid Output limits:[-1,1]

$$y_k = \frac{2}{1+\exp(-ay\_in_k)} - 1$$

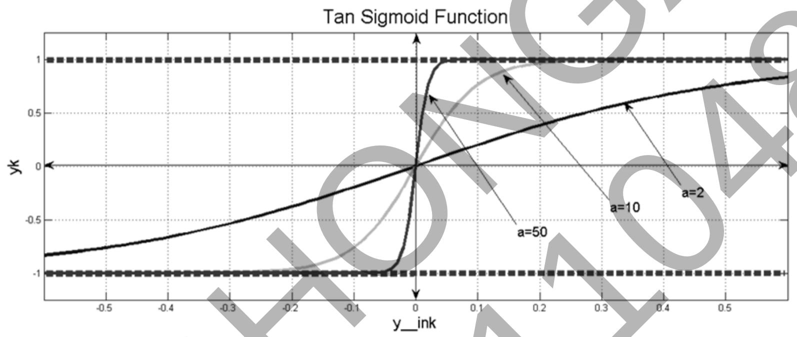


Activation Functions

Tanh Sigmoid

Output limits:[-1,1]

$$y_k = \tanh(ay\_in_k) = \frac{\exp(ay_{in_k}) - \exp(-ay_{in_k})}{\exp(ay_{in_k}) + \exp(-ay_{in_k})}$$



Linear regression MATLAB Demo

Learning Processes

- Ability of the network to *learn* from its environment, and to *improve* its performance through learning
- Neural network learns about its environment through an interactive process of adjustments applied to its synaptic weights and bias levels
- Definition:

Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the ***manner in which the parameter changes take place***

- Learning process implies the following sequence of events"
  1. The neural network is ***stimulated*** by an environment
  2. The neural network ***undergoes*** *changes* in its free parameters as a result of this stimulation
  3. The neural network ***responds*** *in a new way* to the environment because of the changes that have occurred in its internal structure
- A prescribed set of well-defined rules for the solution of a learning problem is called a ***learning algorithm***

Learning Mechanisms

- Five Types
  - Error correction Learning
  - Memory based Learning
  - Hebbian Learning
  - Competitive Learning
  - Boltzmann Learning

Error correction Learning

- We compute the error at time step n and k^{th} output as:

$$e_k(n) = t_k(n) - y_k(n)$$

- Then we minimize the term

$$E(n) = \frac{1}{2}\sum_k \left(e_k(n)\right)^2$$

- The weight correction rule we get is

$$\Delta \boldsymbol{w_{kj}(n)} = \boldsymbol{\eta e_k(n) \cdot x_j}$$

- This learning rule is called as the DELTA rule or WIDROW-HOFF rule
- The updated synaptic weight for the $(n+1)^{th}$ time step is then

$$\boldsymbol{w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)}$$

- ***The adjustment made to a synaptic weight of a neuron is proportional to the product of the error signal and the input signal of the synapse in question***

Memory based Learning

- In *memory-based learning,* all (or most) of the past experiences are explicitly stored in a large memory of correctly classified input-output examples

$$\left\{\mathbf{x_i}, \mathbf{d_i}\right\}_{i=1}^{N}$$

where $\mathbf{x_i}$= input vector and $\mathbf{d_i}$=desired response

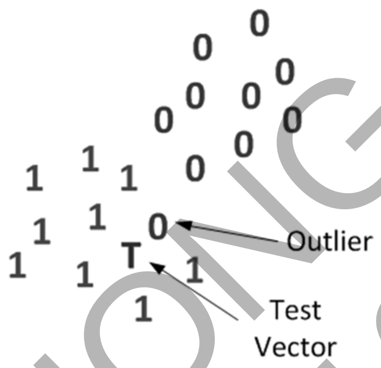
- If this is a binary classification problem, there are two classes/hypotheses denoted by C_1 and C_2 , then $\mathbf{d_i}$ takes the value 0 (-1) for C_1 and 1 for C_2
- When classification of a test vector $\mathbf{x_{test}}$ (not seen before) is required, the algorithm responds by retrieving and analyzing the training data in a "local neighborhood" of $\mathbf{x_{test}}$

- Memory-based learning algorithms involve two essential ingredients
  1. Criterion used for defining the local neighborhood of the test vector $\mathbf{x_{test}}$
  2. Learning rule applied to the training examples in the local neighborhood of $\mathbf{x_{test}}$
- The algorithms differ from each other in the way in which these two ingredients are defined
- The simple type of memory-based learning known as the *nearest neighbor rule,* the local neighborhood is defined as the training example that lies in the immediate neighborhood of the test vector $\mathbf{x_{test}}$
- The vector $\mathbf{x_N}' \in \{\mathbf{x_1}, \mathbf{x_2}, \cdots \mathbf{x_N}\}$ is said to be the nearest neighbor of $\mathbf{x_{test}}$ if

$$\min_i d\left(\mathbf{x_i}, \mathbf{x_{test}}\right) = d\left(\mathbf{x'_N}, \mathbf{x_{test}}\right)$$

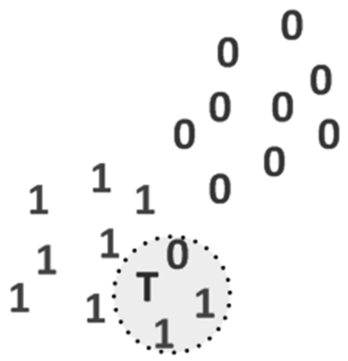
where $d(\mathbf{x_i}, \mathbf{x_{test}})$ is the **Euclidean distance** between the vectors $\mathbf{x_i}$ and $\mathbf{x_{test}}$

- The class associated with the minimum distance, that is, vector $\mathbf{x_N}'$ is reported as the classification of $\mathbf{x_{test}}$
- But this simple rule poses a problem:



- The test vector seems to have the least Euclidean distance with an outlier from Class 0 and is classified as belonging to Class 0
- This is wrong!

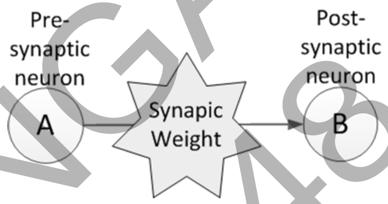
- To solve this problem we modify the nearest neighbor rule to ***k-nearest neighbor classifier***
- Identify the k classified patterns that lie nearest to the test vector $\mathbf{x_{test}}$ for some integer k
- Assign $\mathbf{x_{test}}$ to the class (hypothesis) that is most frequently represented in the k-nearest neighbors to $\mathbf{x_{test}}$ (i.e., use a majority vote to make the classification)
- Here $k = 3$, and T is now classified As belonging to Class 1 (out of three nearest neighbors , two belong to Class 1)



Hebbian Learning

- Hebbian learning is considered to be more closer to the learning mechanism of a biological neuron
- Hebb , a neurophysiologist , in his book " Organization of Behaviour", in 1949 postulated the "Hebb Rule"

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B, is increased"



- Thus if cell A fires consistently the cell B then the synaptic weight increases such that the next time cell A has got a better probability of firing cell B
- Also if cell A does not take part in firing cell B, the synaptic weight weakens

- If pre-synaptic neuron and the post-synaptic neurons show similar activation we shall increase the synaptic weight and vice-versa
- Hebbian synapse is a synapse that uses a *time dependent, highly local, and strongly interactive mechanism to increase synaptic efficiency as a function of the correlation between the presynaptic and postsynaptic activities*
- Four key mechanisms (properties) that characterize a Hebbian synapse
  1. ***Time-dependent mechanism**:* modifications in a Hebbian synapse depend on the exact time of occurrence of the presynaptic and postsynaptic signals
  2. ***Local mechanism:*** locally available information is used by a Hebbian synapse to produce a local synaptic modification that is input specific
  3. ***Interactive mechanism:*** Hebbian form of learning depends on a "true interaction" between presynaptic and postsynaptic signals in the sense that we cannot make a prediction from either one of these two activities by itself
  4. ***Correlational mechanism:*** the correlation over time between presynaptic and postsynaptic signals is viewed as being responsible for a synaptic change

Mathematical Models of Hebbian Modifications

- Consider a synaptic weight w_{kj} of neuron k with presynaptic and postsynaptic signals denoted by x_j and y_k respectively.
- The adjustment applied to the synaptic weight w_{kj} at time step *n* is expressed in the general form:

$$\Delta w_{kj}(n) = F\big(y_k(n), x_j(n)\big)$$

where $F(\cdot,\cdot)$is a function of both postsynaptic and presynaptic signals

- **Hebbs hypothesis** or Hebbian learning rule is then given as

$$\boldsymbol{\Delta w_{kj}(n) = \eta y_k(n) x_j(n)}$$

where η is a positive constant that determines the *rate of learning*

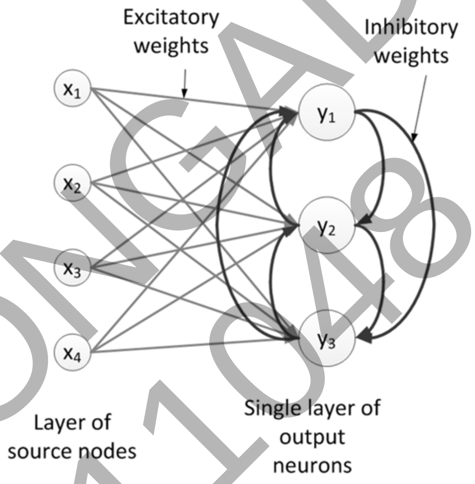
- This rule is also called as *activity product rule*

- But this rule has a basic flaw:
- The repeated application of the input signal (presynaptic activity) x_j leads to an increase in y_k, and therefore *exponential growth* that finally drives the synaptic connection into saturation
- At that point no information will be stored in the synapse and selectivity is lost
- This limitation is overcome by **Covariance hypothesis**
- Presynaptic and postsynaptic signals are replaced by the departure of presynaptic and postsynaptic signals from their **respective average values** over a certain time interval
- Let $\bar{x}$ and $\bar{y}$ denote the *time-averaged values* of the presynaptic signal x_j and postsynaptic signal y_k, respectively
- According to the covariance hypothesis, the adjustment applied to the synaptic weight w_{kj} is defined by:

$$\boldsymbol{\Delta w_{kj} = \eta(y_k - \bar{y})(x_j - \bar{x})}$$

Competitive Learning

- Output neurons of a neural network compete among themselves to become active (fired)
- In competitive learning only a single output neuron is active at any one time
- Highly suited to discover statistically salient features that may be used to classify a set of input patterns
- Accordingly the individual neurons of the network **learn to specialize on ensembles of** similar patterns; in so doing they become ***feature detectors*** for different classes of input patterns



- There are three basic elements to a competitive learning rule
  - A set of neurons that are all the same except for some randomly distributed synaptic weights, and which therefore *respond differently* to a given set of input patterns
  - A *limit* imposed on the "strength" of each neuron
  - A mechanism that permits the neurons to *compete* for the right to respond to a given subset of inputs, such that only *one* output neuron, or only one neuron per group, is active (i.e., "on") at a time. The neuron that wins the competition is called a *winner-takes-all neuron*

- For a neuron k to be the winning neuron, its induced local field v_k, for a specified input pattern **x** must be the largest among all the neurons in the network
- The output signal y_k of winning neuron k is set equal to one; the output signals of all the neurons that lose the competition are set equal to zero
- Thus

$$y_k = \begin{cases} 1, & if\ v_k > v_j\ for\ all\ j, j \neq k \\ 0, & otherwise \end{cases}$$

where the induced local field v_k, represents the combined action of all the forward and feedback inputs to neuron k

- Let w_{kj} denote the synaptic weight connecting input node *j* to neuron *k*
- Suppose that each neuron is allotted a *fixed* amount of synaptic weight (i.e., all synaptic weights are positive), which is distributed among its input nodes; that is

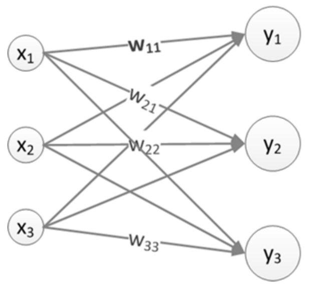
$$\sum_j w_{kj} = 1 \text{ for all } j$$

- As per the standard *competitive learning rule,* the change $\Delta\boldsymbol{w}_{kj}$ applied to synaptic weight $\boldsymbol{w}_{kj}$ is defined by:

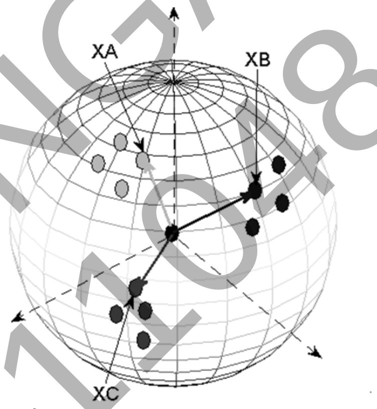
$$\Delta\boldsymbol{w}_{kj} = \begin{cases} \eta(x_j - w_{kj}), & if\ neuron\ k\ wins \\ 0, & if\ neuron\ k\ loses \end{cases}$$

- This rule has the overall effect of moving the synaptic weight vector $\boldsymbol{w}$, of winning neuron *k* toward the input pattern $\mathbf{x}$
- Thus if we have $\mathrm{W}_k = [w_{k1}\ \ w_{k2} \cdots w_{km}]$ and input vector as $\mathbf{X} = [x_1\ x_2 \cdots x_m]$ then we are effectively moving W_k towards the input pattern **X**
- **What ultimately we do is that we align the weight towards the input vector for a specific input and weight combination**

Geometric Interpretation



- Consider vector $\mathbf{X} = [x_1\ x_2\ x_3]$
- The constraint we lay down is that $\|X\| = 1$, this means
$$x_1{}^2 + x_2{}^2 + x_3{}^2 = 1$$

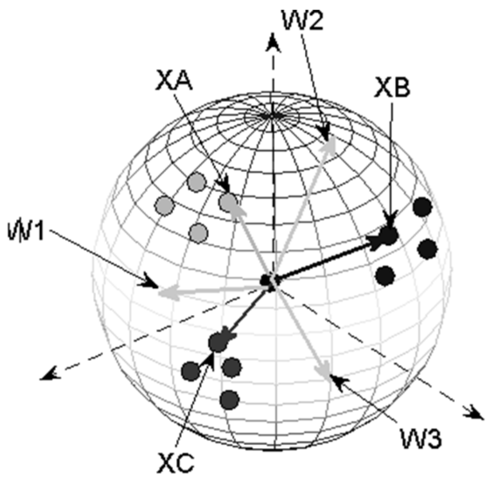


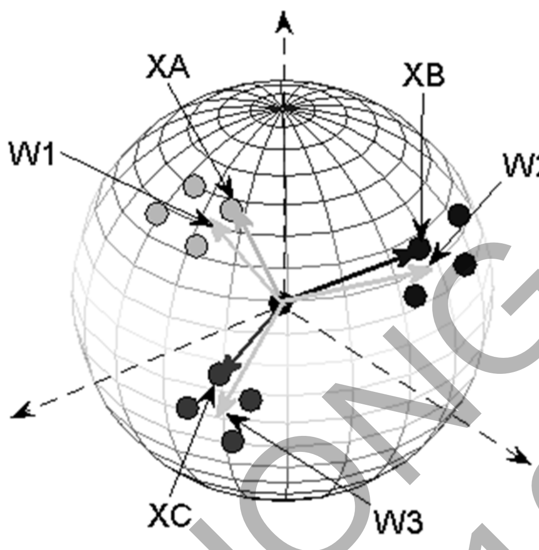
- Suppose we have three patterns with same unit length X_A, X_B, X_C
- These vectors will lie on the surface of a sphere(there can be a large number of vectors)
- We expect that the output neurons represent these clusters

- Similar to input vectors we can assume (constrain) that sum of squares of weights per neuron is equal to 1:

$$\sum_{i=1}^{3}(w_{ki})^2 = 1$$

- Now when we present say X_A, y_1 wins and its weights are adjusted while for other neurons the weights stay unchanged
- On presenting inputs belonging to cluster 1, W_1 moves closer to cluster 1
- Similarly on presenting other vectors X_B and X_C , W_2 and W_3 get aligned to the respective clusters





- But what will happen if there are more than four clusters?

Biases and Thresholds

- A bias acts exactly as a weight on a connection from a unit whose activation is always 1
- Increasing the bias increases the net input to the unit
- If a bias is included, the activation function is typically taken to be

$$y_k = f(y\_in_k) = \begin{cases} 1, & y\_in_k \geq 0; \\ -1, & y\_in_k < 0; \end{cases}$$

Where

$$y\_in_k = b + \sum_{j=1}^{N} x_j \cdot w_{kj} = \sum_{j=0}^{N} x_j \cdot w_{kj}$$

Here $x_0 = 1$ and $w_{k0} = b$

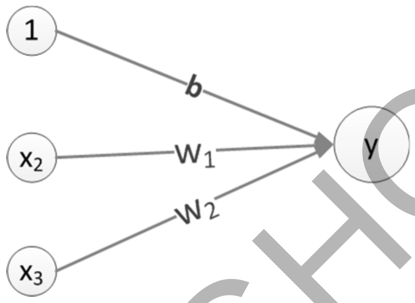
- If bias is not used the same network can be described as

$$y_k = f(y\_in_k) = \begin{cases} 1, & y\_in_k \geq \theta; \\ -1, & y\_in_k < \theta; \end{cases}$$

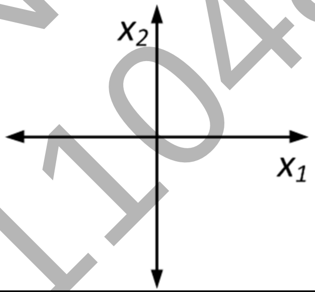
Where

$$y\_in_k = \sum_{j=1}^{N} x_j \cdot w_{kj}$$

- Consider the following neural net:



- We consider the separation of the input space into regions where the response of the net is positive and regions where the response is negative



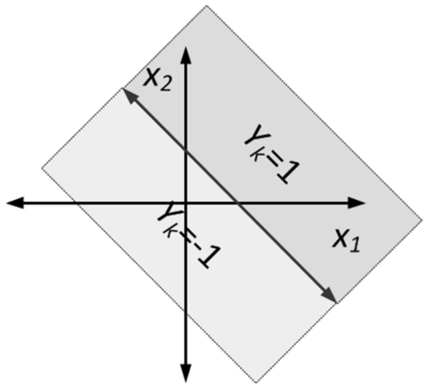
- The boundary between the values of $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ for which the net gives a positive response and the values for which it gives a negative response is the separating line:

$$b + w_1x_1 + w_2x_2 = 0$$

or assuming $w_2 \neq 0$

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

- The requirement for a positive response from the output unit is that the net input , y_in_k it receives, namely,
  $$b + w_1x_1 + w_2x_2$$
  be greater than 0
- During training, values of w_1, w_2 and b are determined so that the net will have the correct response for the training data



- If we think in terms of the threshold θ, the requirement for a positive response from the output unit is that the net input it receives, namely, $w_1x_1 + w_2x_2$, be greater than the threshold
- This gives the equation of the line separating positive from negative output as:

$$w_1x_1 + w_2x_2 = \theta$$

or assuming $w_2 \neq 0$

$$x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_2}$$

- During training, values of w_1 and w_2 are determined so that the net will have the correct response for the training data
- In this case, the separating line ***cannot*** pass through the origin, but a line can be found that passes arbitrarily close to the origin

- Thus there is no advantage to including both a bias and a nonzero threshold for a neuron that uses the step function as its activation function
- Also including neither a bias nor a threshold is equivalent to requiring the separating line (or plane or hyperplane for inputs with more components) to pass through the origin
- This may or may not be appropriate for a particular problem
- **Example: Going to watch a cricket match!**
- Conclusion: Since it is the relative values of the weights, rather than their actual magnitudes, that determine the response of the neuron, the model can cover all possibilities using either the fixed threshold or the adjustable bias
- So we can use either adjustable bias or fixed threshold NOT both!

Advantages of bias over threshold

- Using bias can remove the dependency on threshold
- We can modify the bias (more adaptable) but threshold is fixed
- Bias helps in simplifying the separation boundaries
- Computational flexibility and ease is more if we use bias as thresholds may be different for different output neurons
- Hence use of bias is more preferable

Linear Separability

- The intent is to train the net (i.e., adaptively determine its weights) so that it will respond with the desired classification when presented with an input pattern that it was trained on or when presented with one that is sufficiently similar to one of the training patterns
- For a particular output unit, the desired response is a "*yes*" if the input pattern is a member of its class and a "*no*" if it is not
- A "*yes*" response is represented by an output signal of **1**, a "*no*" by an output signal of **-1** (for bipolar signals)
- Since we want one of two responses, the activation (or transfer or output) function is taken to be a step function
- The value of the function is 1 if the net input is positive and -1 if the net input is negative

- Since the net input to the output unit is

$$y_{in} = b + \sum_i x_i w_i$$

- It is easy to see that the boundary between the region where $y\_in > 0$ and the region where $y\_in < 0$, which we call the ***decision boundary,*** is determined by the relation

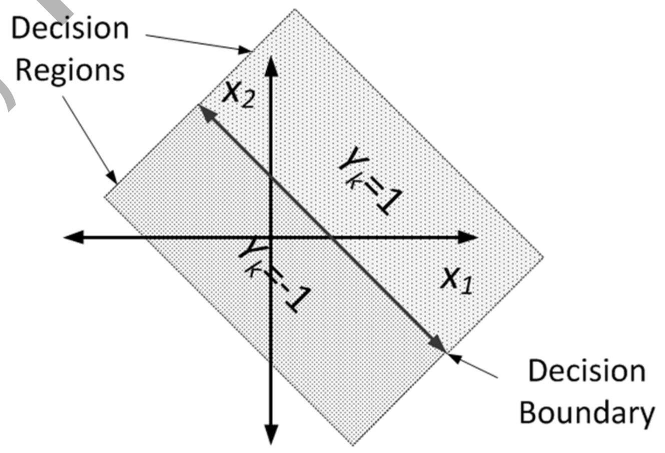
$$b + \sum_i x_i w_i = 0$$

- If there are weights (and a bias) so that all of the training input vectors for which the correct response is + 1 lie on one side of the decision boundary and all of the training input vectors for which the correct response is - 1 lie on the other side of the decision boundary, we say that the problem is "**linearly separable**"

- The region where *y* is positive is separated from the region where it is negative by the line

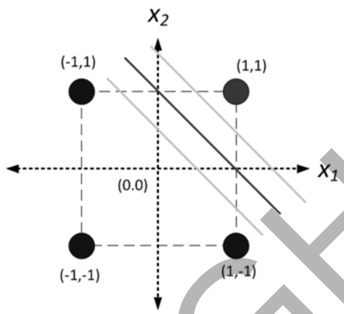
$$b + w_1x_1 + w_2x_2 = 0$$

- These two regions are often called ***decision regions*** for the net



Response regions for the AND function

x1	x2	Y
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1



- Solving this graphically, we find that the red line can be a good decision boundary
- Points(0,1) and (1,0) lie on the line hence the using equation

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

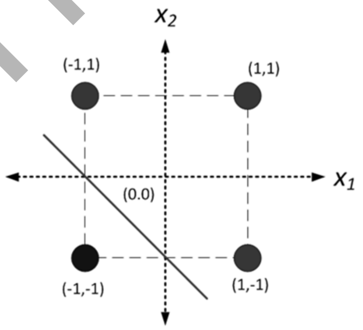
assuming $w_2 = 1$, gives

$b = -1$ using point(0,1) and $w_1 = 1$, using point (1,0) and $b = -1$

- Actually the choice of sign for *b* is determined by the requirement that
$$b + w_1x_1 + w_2x_2 < 0$$
- We can then set x_1 and x_2=0 and compute b knowing to which side of the line the point(0,0) should lie

Response regions for the OR function

x1	x2	Y
-1	-1	-1
-1	1	1
1	-1	1
1	1	1



- Points(-1,0) and (0,-1) lie on the line hence the using equation

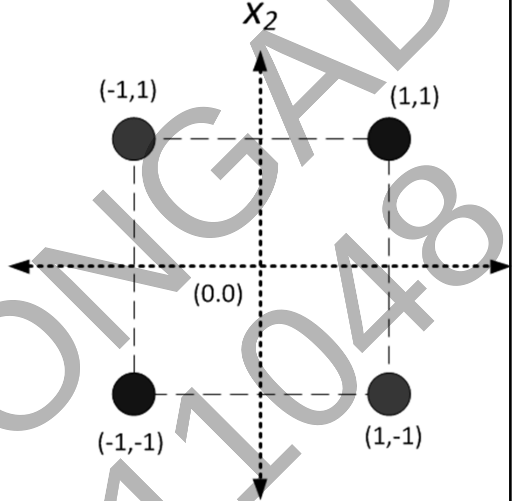
$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{b}{w_2}$$

assuming $w_2 = 1$, gives

$b = 1$ and $w_1 = 1$

XOR-Example of linearly non-separable problems

x1	x2	Y
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



What should the decision boundary be and where?

The Perceptron

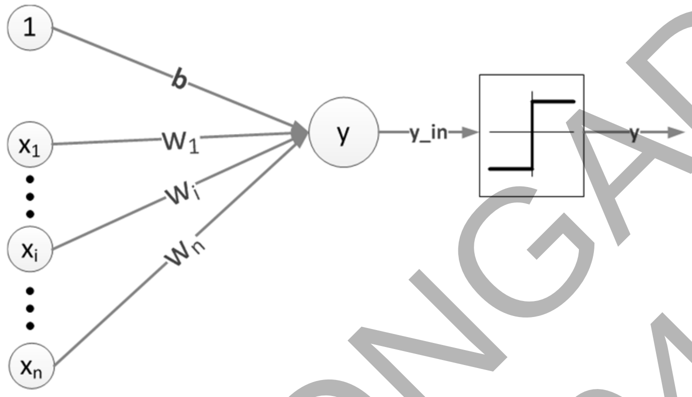
- The perceptron learning rule is a more powerful learning rule than the Hebb rule
- Under suitable assumptions, its iterative learning procedure can be proved to converge to the correct weights, i.e., the weights that allow the net to produce the correct output value for each of the training input patterns
- The weight update rule is

$$\boldsymbol{w_i(new) = w_i(old) + \eta t x_i}$$

Where $\boldsymbol{t}$ is the target output and $\boldsymbol{\eta}$ is the learning rate

- If an error did not occur, the weights would not be changed
- Training would continue until no error occurred
- The perceptron learning rule convergence theorem states that if **weights exist** to allow the net to respond correctly to all training patterns, then the rule's procedure for adjusting the weights will find values such that the net does respond correctly to all training
- patterns (i.e., the net solves the problem-or learns the classification
- Also the net will find these weights in a finite number of training steps

Perceptron Architecture



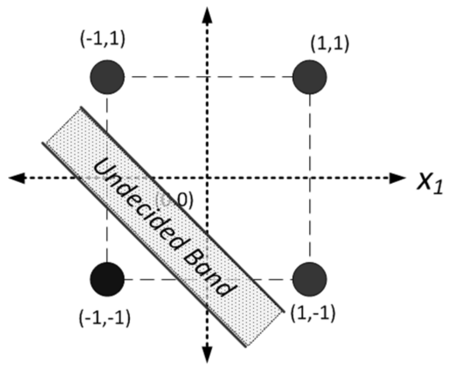
- The goal of the net is to classify each input pattern as belonging, or not belonging, to a particular class.
- Belonging is signified by the output unit giving a response of + 1; not belonging is indicated by a response of - 1

The perceptron algorithm

STEP 0	Initialize weight and biases. (Set to zero) Set learning rate η (0 < η < 1)		
STEP 1	While stopping condition is false, do Steps 2-6.		
	STEP 2	For each training pair, do Steps 3-5.	
		STEP 3	Set activations of input units $x_i = s_i$
		STEP 4	Compute response of the output unit $y\_in = b + \sum_i x_i \cdot w_i$ $y = \begin{cases} 1 & \text{if } y\_in > \theta \\ 0 & \text{if } -\theta \le y\_in \le \theta \\ -1 & \text{if } y\_in < -\theta \end{cases}$

STEP 1	STEP 2	STEP 5	Update weights and bias if an error occurred for this pattern
			If $y \neq t$ $$w_i(new) = w_i(old) + \eta t x_i$$ $$b(new) = b(old) + \eta t$$ else $$w_i(new) = w_i(old)$$ $$b(new) = b(old)$$
		STEP 6	Test stopping condition If no weights changed in Step 2, stop; else, continue

- Threshold here plays a different role
- The threshold on the activation function for the response unit is a fixed, non-negative value θ
- The form of the activation function for the output unit(response unit) is such that there is an "undecided" band (of fixed width determined by θ) separating the region of positive response from that of negative response.



A Perceptron for the AND function: bipolar inputs, bipolar targets

x1	x2	t
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

- The training process for bipolar input, η = 1, and threshold and initial weights =0,θ=0

$$\boldsymbol{y = f(y\_in) = \begin{cases} 1, y\_in \geq 0; \\ -1, y\_in < 0; \end{cases}}$$

$$\boldsymbol{w_i(new) = w_i(old) + tx_i}$$

First iteration

Input			y_in	y	t	Weight Changes			Weights		
x1	x2	1							w1	w2	b
									0	0	0
-1	-1	1	0	1	-1	1	1	-1	1	1	-1
-1	1	1	-1	-1	-1	0	0	0	1	1	-1
1	-1	1	-1	-1	-1	0	0	0	1	1	-1
1	1	1	1	1	1	0	0	0	1	1	-1

Second iteration

Input			y_in	y	t	Weight Changes			Weights		
x1	x2	1							w1	w2	b
									1	1	-1
-1	-1	1	-3	-1	-1	0	0	0	1	1	-1
-1	1	1	-1	-1	-1	0	0	0	1	1	-1
1	-1	1	-1	-1	-1	0	0	0	1	1	-1
1	1	1	1	1	1	0	0	0	1	1	-1

- We see that there is no weight change after second iteration hence we conclude that the net has converged