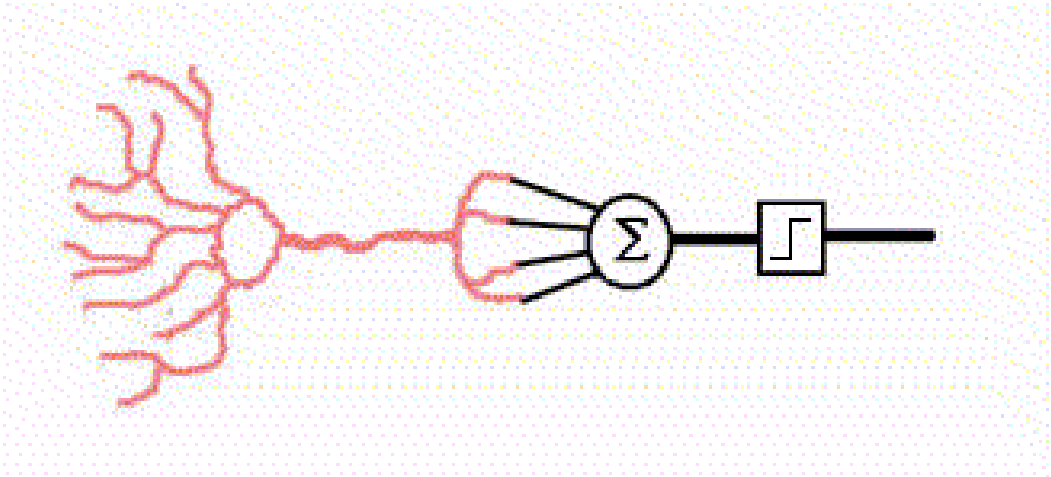


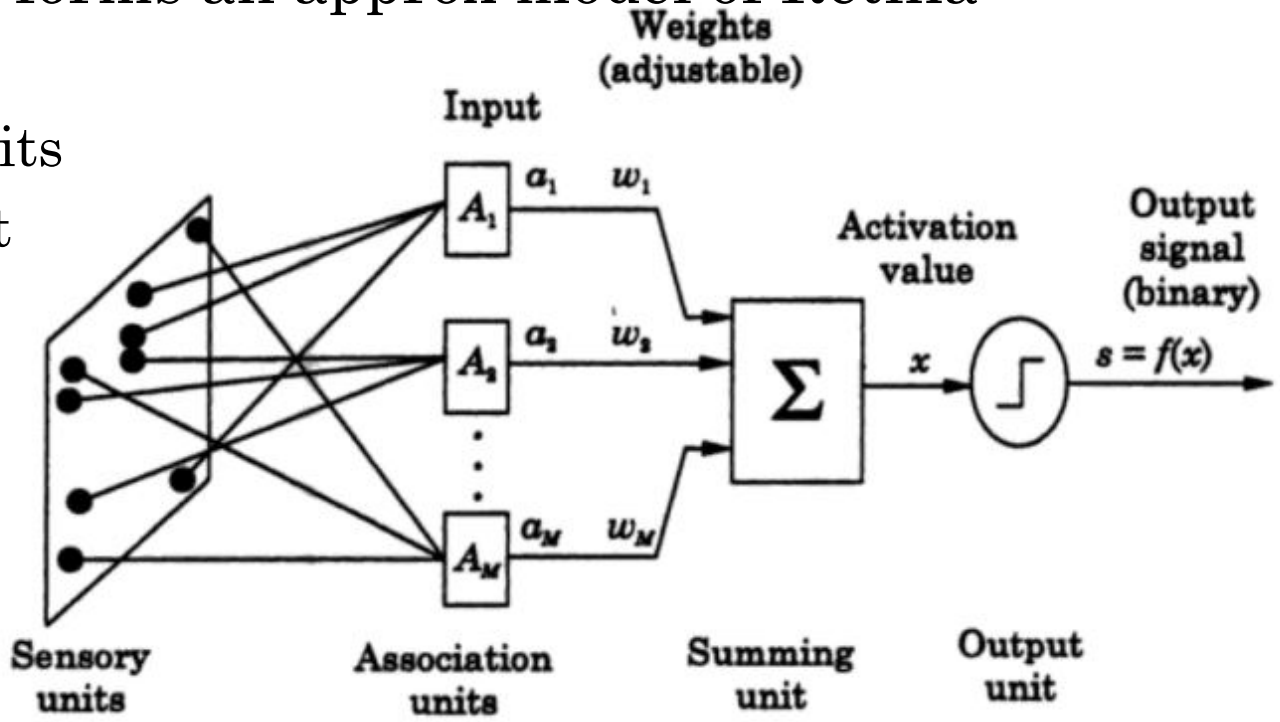
Neural Networks (CS010 805G02)

Perceptrons – Mod 1

Shiney Thomas
AP,CSE,AJCE

Introduction

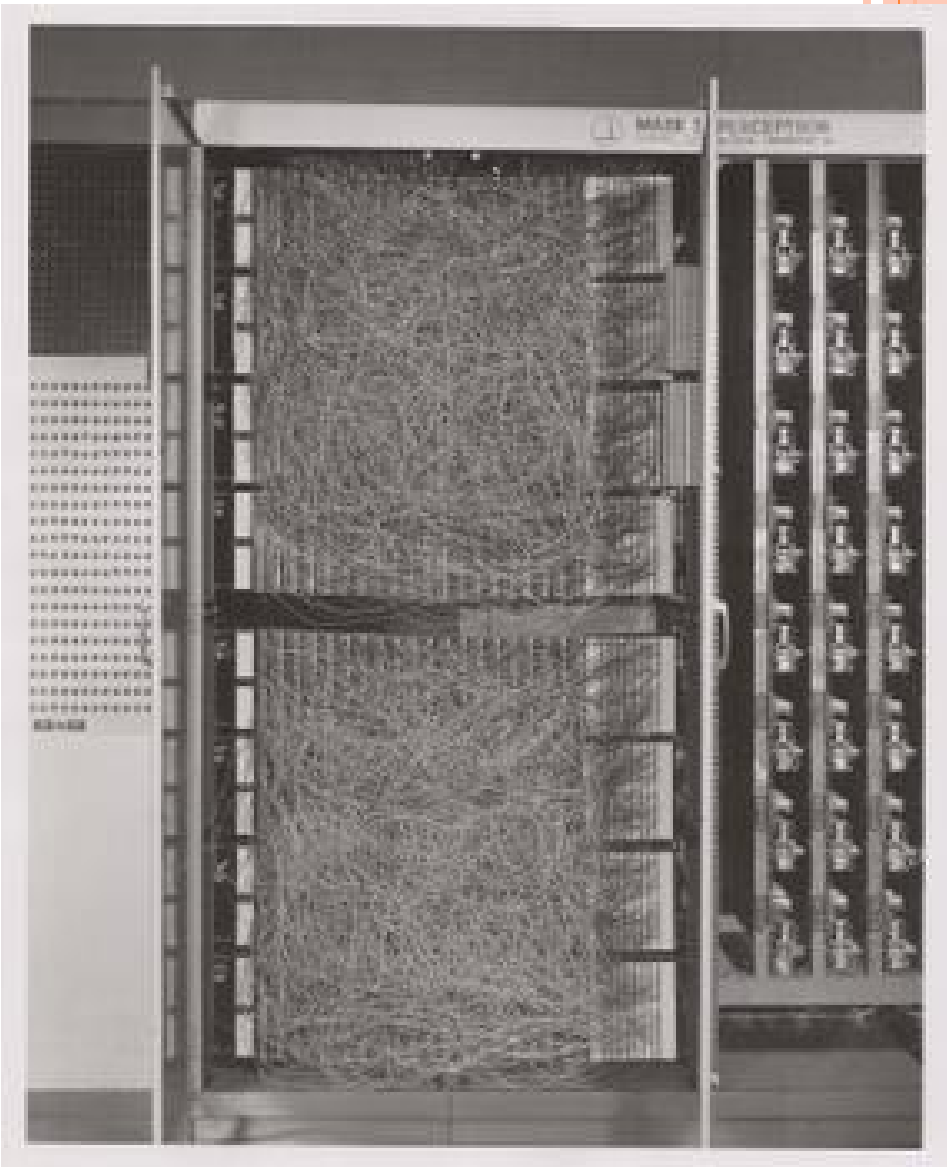
- Also known as the Rosenblatt's Perceptron
- Frank Rosenblatt(1962); Others Minsky and Papert(1969,1988), Block(1962)
- Three layers - forms an approx model of Retina
  - Sensory units
  - Associator units
  - Response unit



Perceptron History

- The perceptron algorithm was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt, funded by the United States Office of Naval Research.
- The perceptron was intended to be a machine, rather than a program, and while its first implementation was in software for the IBM 704, it was subsequently implemented in custom-built hardware as the "Mark 1 perceptron".
- This machine was designed for image recognition: it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors

- Fig -The Mark I Perceptron machine was the first implementation of the perceptron algorithm. The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image. The main visible feature is a patchboard that allowed experimentation with different combinations of input features. To the right of that are arrays of potentiometers that implemented the adaptive weights.



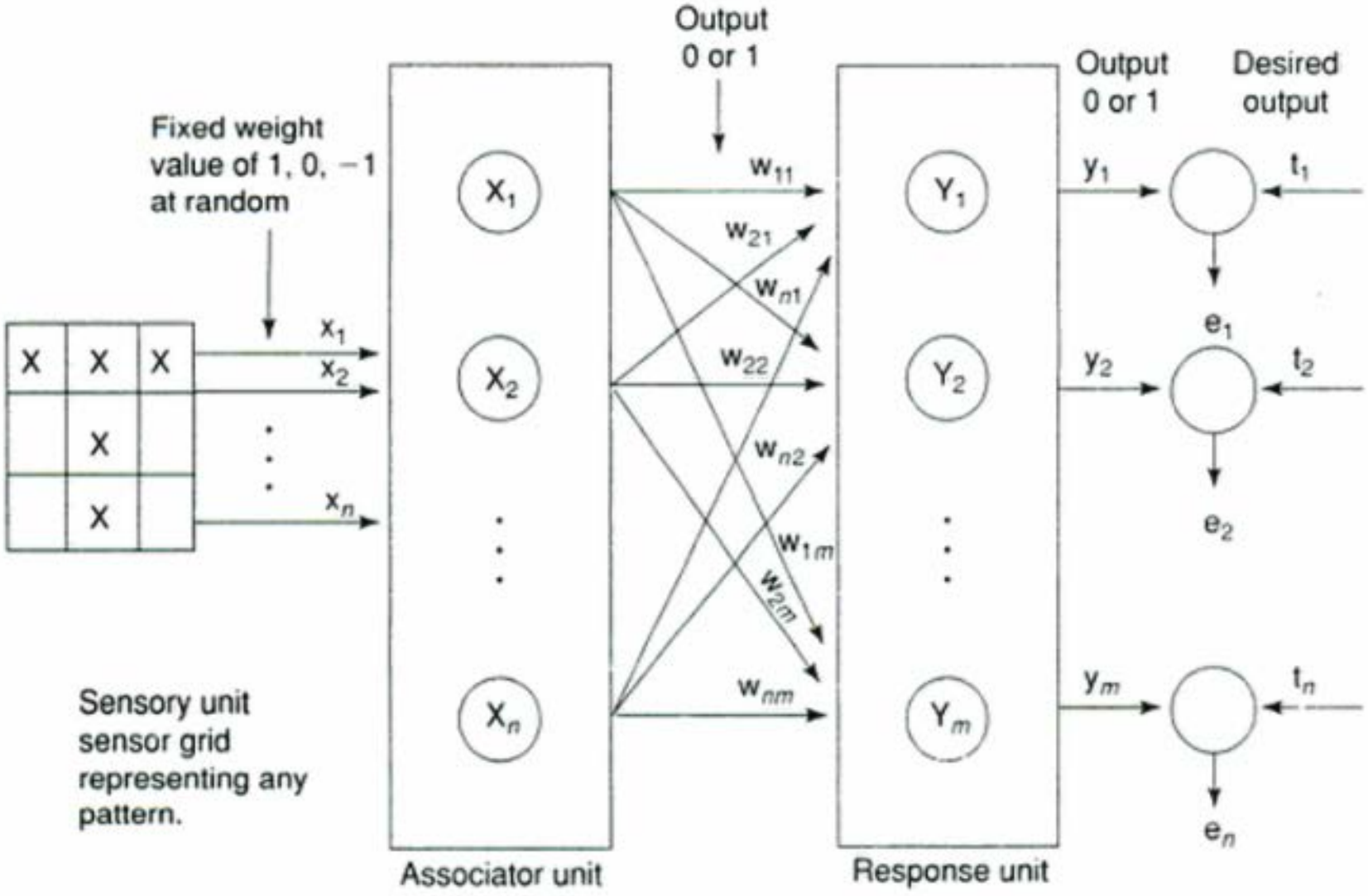
Perceptron History

- Although the perceptron initially seemed promising, it was quickly proved that perceptrons could not be trained to recognise many classes of patterns.
- This caused the field of neural network research to stagnate for many years, before it was recognised that a feedforward neural network with two or more layers (also called a multilayer perceptron) had far greater processing power than perceptrons with one layer (single layer perceptron)

Perceptron

- Simple perceptron used [Block ,1962]
  - Binary activation function for sensory and associator units
  - Activation of +1,0,-1 for response units
- Sensory unit connected to Associator units with fixed weights having values of +1,0,-1,assigned at random.
- The activation function for each associator unit was the binary step function with an arbitrary, but fixed, threshold.
- Thus, the signal sent from the associator units to the output unit was a binary (0 or 1) signal.

Original Perceptron Network



Perceptron

- The output of the perceptron is $y = f(y_{in})$ where the activation function is

$$f(y_{in}) = \begin{cases} 1 & if\ y_{in} > \theta \\ 0 & if\ -\theta \leq y_{in} \leq \theta \\ -1 & if\ y_{in} < -\theta \end{cases}$$

- The weights from the associator units to the response (or output) unit were adjusted by the **perceptron learning rule**.
- For each training input, the net would calculate the response of the output unit.
- Then the net would determine whether an error occurred for this pattern (by comparing the calculated output with the target value).

Perceptron

- If an error occurred for a particular training input pattern, the weights would be changed according as

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i,$$

$$b(\text{new}) = b(\text{old}) + \alpha t.$$

- where **t** is the target value ,+1 or -1
- **α** is the **learning rate**
- If an error did not occur, the weights would not be changed
- Continue training until no error

Perceptron

- 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).
- Moreover, the net will find these weights in a finite number of training steps.
- The perceptron learning algorithm does not terminate if the learning set is not linearly separable.

Linear Separability

- 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

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

Linear Separability

- The AND function (for bipolar inputs and target) is defined as follows:

INPUT (x_1, x_2)	OUTPUT (t)
(1, 1)	+1
(1, −1)	−1
(−1, 1)	−1
(−1, −1)	−1

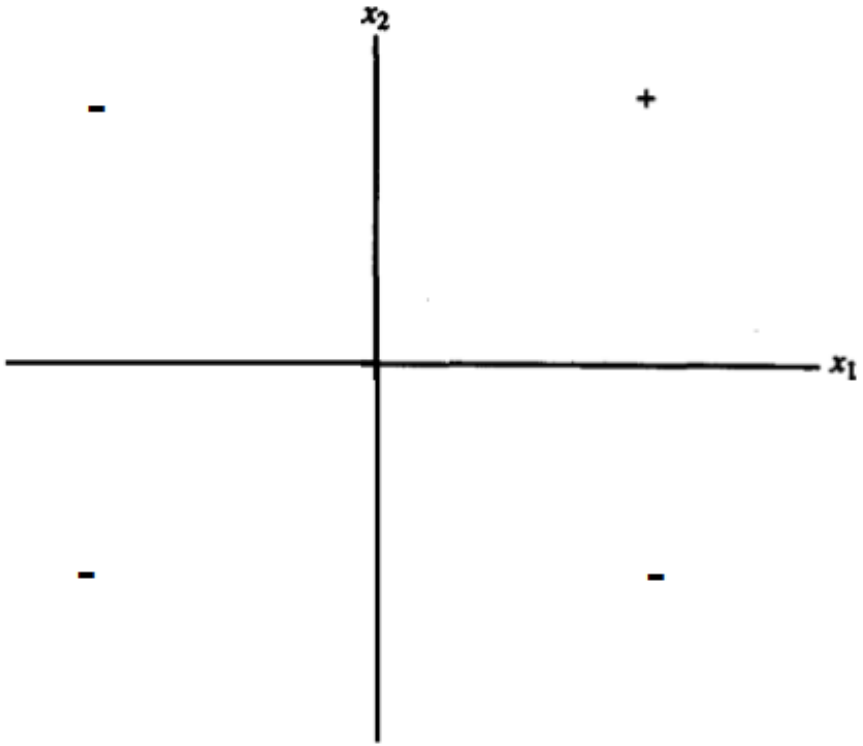
- An example of weights that would give the decision boundary illustrated in the figure, namely, the separating line $x_2 = -x_1 + 1,$

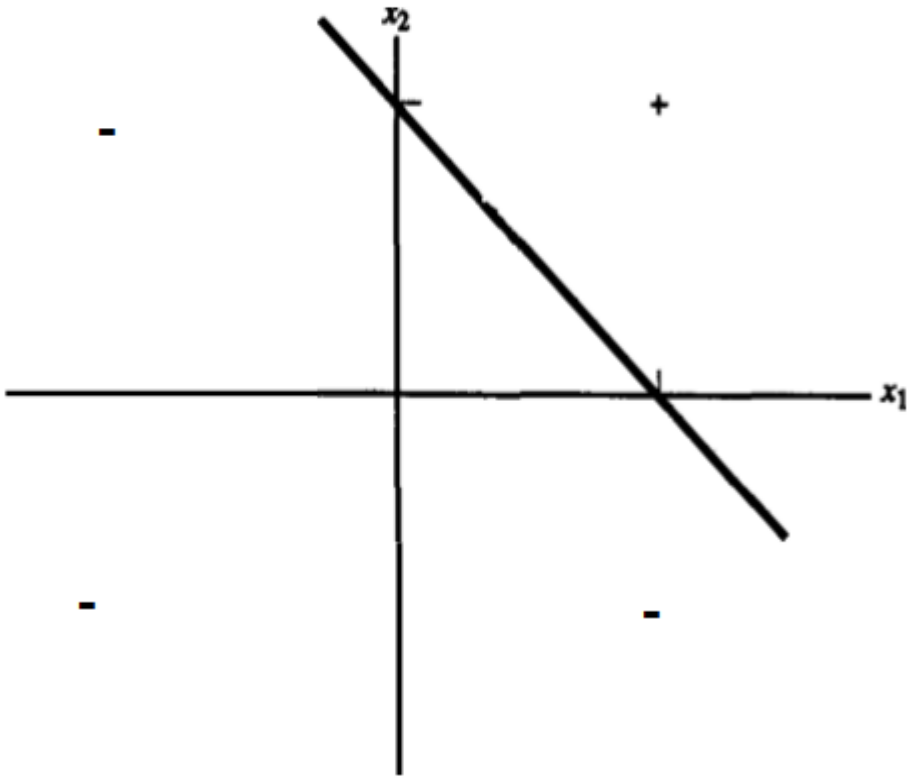
- is

$$b = -1,$$
$$w_1 = 1,$$
$$w_2 = 1.$$

Linear Separability

- The AND function (for bipolar inputs and target) is defined as follows:





Linear Separability

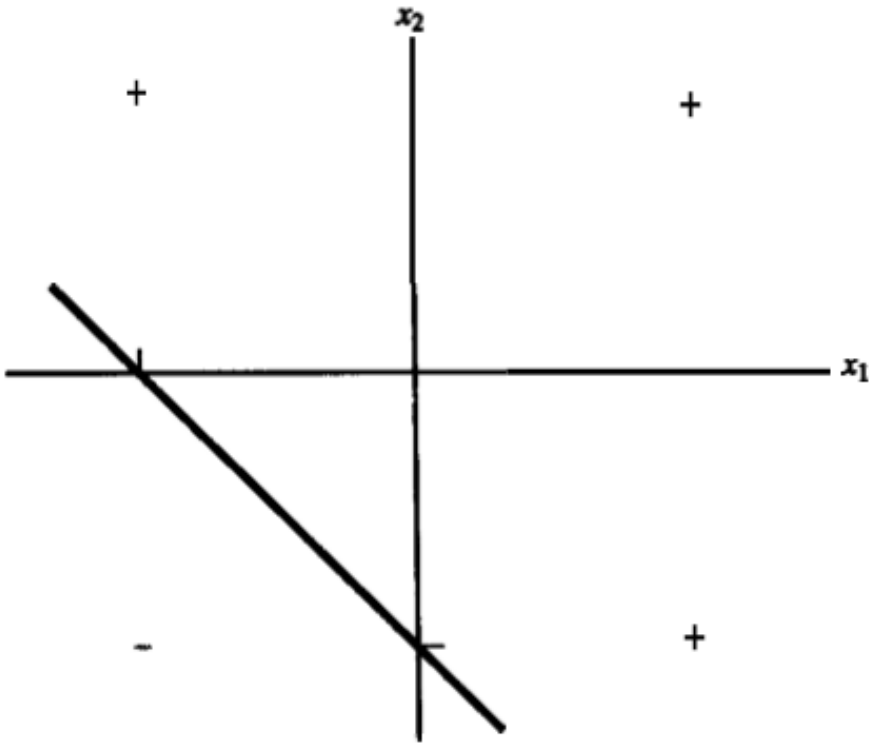
- The OR function (for bipolar inputs and target) is defined as follows:

INPUT (x_1, x_2)	OUTPUT (t)
(1, 1)	+1
(1, −1)	+1
(−1, 1)	+1
(−1, −1)	−1

- One example of suitable weights is
  - b=1, w_1=1,w_2=1
  - giving the separating line $x_2 = -x_1 - 1$

Linear Separability

- The OR function (for bipolar inputs and target) is defined as follows:



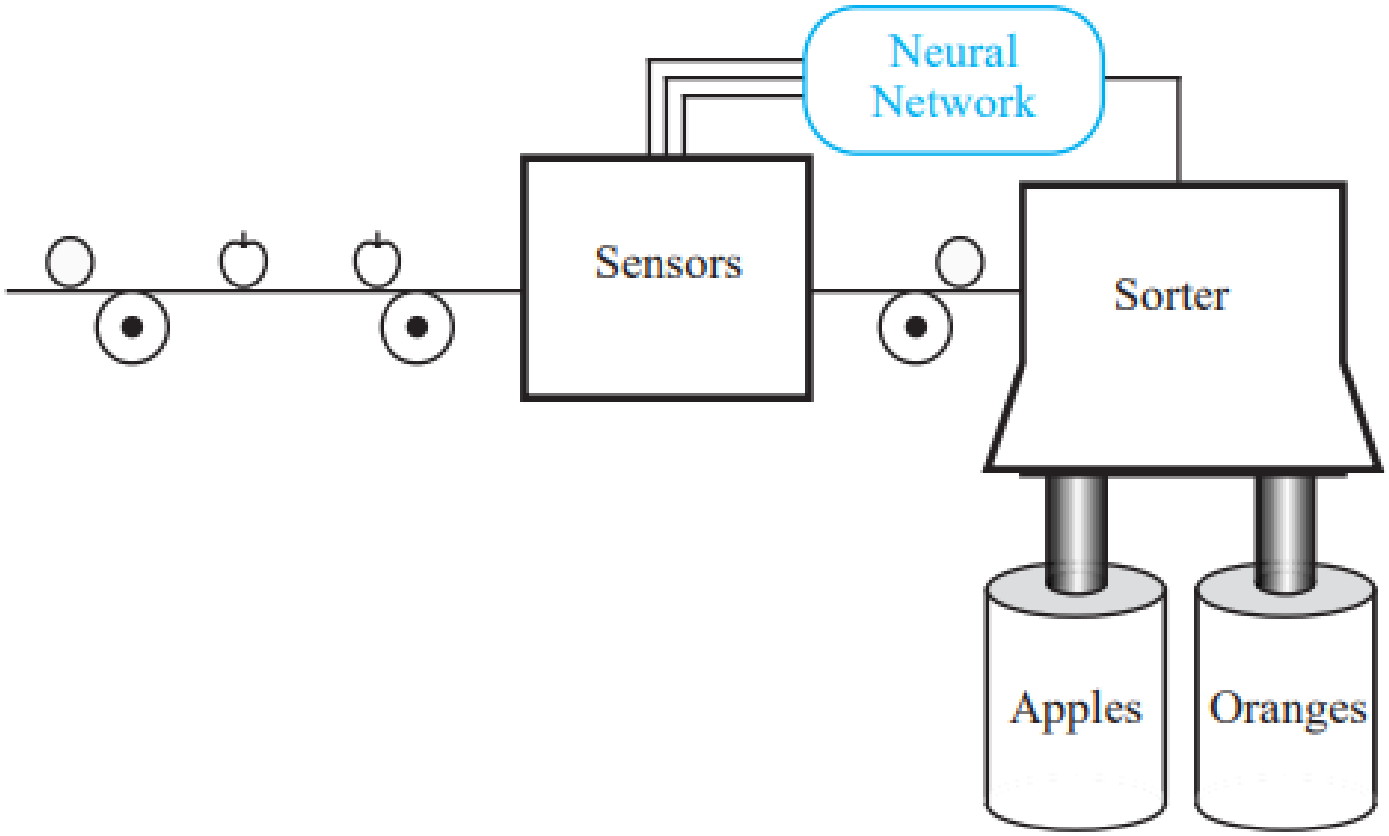
Linear Separability

- The input points to be classified positive can be separated from the input points to be classified negative by a straight line.
- The equations of the decision boundaries are not unique.
- Note that if a bias weight were not included in these examples, the decision boundary would be forced to go through the origin.
- Not all simple two-input, single-output mappings can be solved by a single- layer net (even with a bias included), eg. XOR function

Simple Perceptron as Pattern Classifier

- 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.

Simple Perceptron as Pattern Classifier



Simple Perceptron as Pattern Classifier

- The dealer wants a machine that will sort the fruit according to type
- Friuts loaded on conveyer belt,passes through a set of sensors, which measure three properties of the fruit: shape, texture and weight.
- The shape sensor will output
  - a 1 if the fruit is approximately round and
  - a -1 if it is more elliptical.
- The texture sensor will output
  - a 1 if the surface of the fruit is smooth and
  - a -1 if it is rough.
- The weight sensor will output
  - a 1 if the fruit is more than one pound and
  - a -1 if it is less than one pound.
- apples and **oranges**.

Simple Perceptron as Pattern Classifier

- Weight vector $\mathbf{p} = \begin{bmatrix} shape \\ texture \\ weight \end{bmatrix}$.
- a prototype orange would be represented by $\mathbf{p}_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$,
- and apple by $\mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$.

ALGORITHM

- Step 0. Initialize weights and bias.

  (For simplicity, set weights and bias to zero.)

  Set learning rate α ($0 < \alpha < 1$).

  (For simplicity, α can be set to 1.)
- Step 1. While stopping condition is false, do Steps 2-6.
  - Step 2. For each training pair s:t, do Steps 3-5.
    - Step 3. Set activations of input units: $x_i = s_i$.
    - Step 4. Compute response of output unit:

$$y\_in = b + \sum_i x_i w_i;$$

$$y = \begin{cases} 1 & \text{if } y\_in > \theta \\ 0 & \text{if } -\theta \le y\_in \le \theta \\ -1 & \text{if } y\_in < -\theta \end{cases}$$

ALGORITHM

- Step 5. Update weights and bias if an error occurred for this pattern.

  If y != t,

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i,$$

$$b(\text{new}) = b(\text{old}) + \alpha t.$$

  else

$$w_i(\text{new}) = w_i(\text{old}),$$

$$b(\text{new}) = b(\text{old}).$$

- Step 6. Test stopping condition:

  If no weights changed in Step 2, stop; else, continue.

Learning and Generalization

- The main property of a neural network is the ability to learn from its environment, and to improve its performance through learning.
- Learning can be viewed as the problem of updating network weights.
- Learning algorithms–rules for adjustment of weights.
  - Supervised learning : Provide the network with a series of sample inputs and compare the response with the desired output.
  - Un-supervised learning : In contrast to supervised learning, unsupervised learning does not require an external teacher. During the training session, the neural network receives a number of different input patterns, discovers significant features in these patterns and organize them to form clusters.
  - Reinforcement Learning : Right answer is not provided but indication of whether 'right' or 'wrong' is provided.

Learning and Generalization

- Generalization
  - Of a concept is an extension of that concept into a situation that goes beyond the context in which it was learnt
  - Ability to produce correct output (or nearly so) for a input data point which it has never seen before.

Hebbian Learning

- Hebb's *postulate of learning* (or simply Hebb's rule) (1949), is the following:
- "*When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth processes or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B, is increased*".

Hebbian Learning

- In 1949, Donald Hebb proposed one of the key ideas in biological learning, commonly known as Hebb's Law.
- Hebb's Law states that if neuron **i** is near enough to excite neuron **j** and repeatedly participates in its activation, the synaptic connection between these two neurons is strengthened and neuron **j** becomes more sensitive to stimuli from neuron **i**.

Hebbian Learning

- Hebb's Law can be represented in the form of two rules:
  - If two neurons on either side of a connection are activated synchronously, then the weight of that connection is increased.
  - If two neurons on either side of a connection are activated asynchronously, then the weight of that connection is decreased.
- Hebb's Law provides the basis for learning without a teacher. Learning here is a local phenomenon occurring without feedback from the environment.

HEBBIAN NET

- We shall refer to a single-layer (feedforward) neural net trained using the (extended) Hebb rule as a **Hebb net.**
- Using Hebb's Law we can express the adjustment applied to the **weight w_{ij}** at iteration **p** in the following form:

$$\Delta w_{ij}(p) = F[y_j(p), x_i(p)]$$

- As a special case, we can represent **Hebb's Law** as follows:

$$\Delta w_{ij}(p) = \alpha\, y_j(p)\; x_i(p)$$

where α is the learning rate parameter.
This equation is referred to as the **activity product rule.**

Hebbian Learning Algorithm

Step 0. Initialize all weights:

$wi = 0 \quad (i = 1 \text{ to } n).$

Step 1. For each input training vector and target output pair, s : *t*, do steps 2–4.

Step 2. Set activations for input units:

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

Step 3. Set activation for output unit:

$y = t.$

Step 4. Adjust the weights for

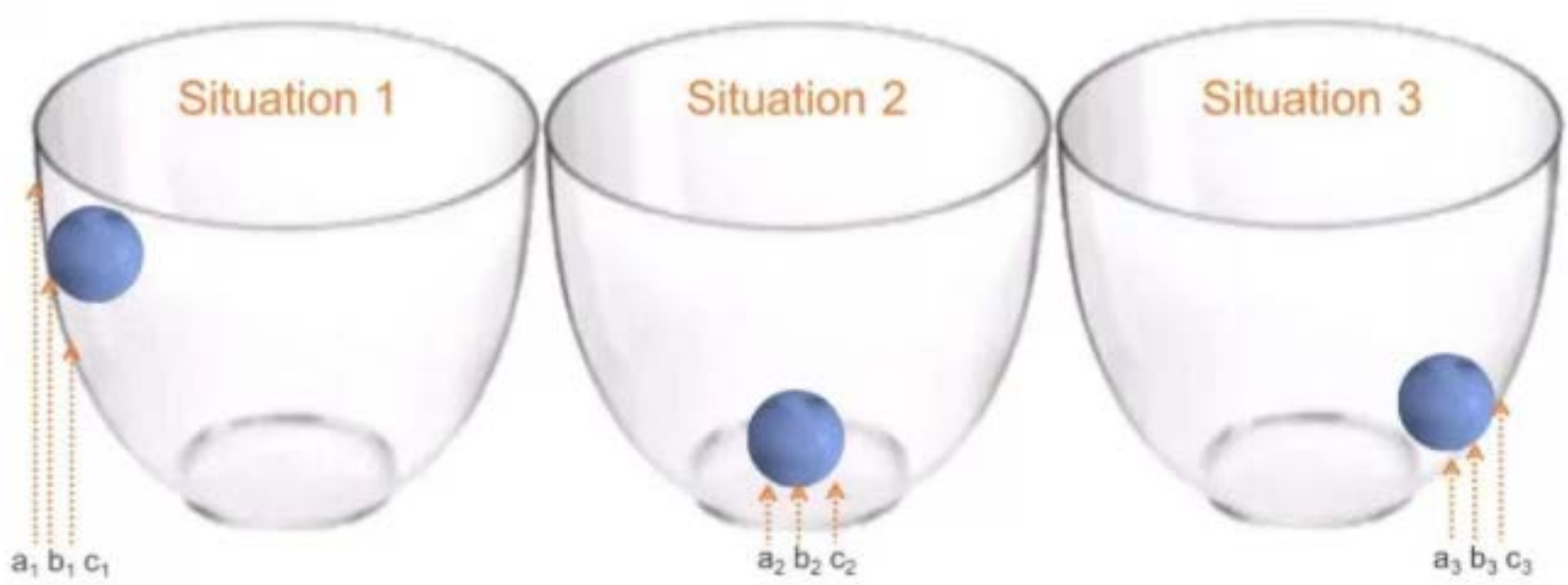
$$w_i(\text{new}) = w_i(\text{old}) + x_i y \quad (i = 1 \text{ to } n).$$

Adjust the bias:

$$b(\text{new}) = b(\text{old}) + y.$$

Gradient Descent Learning

- Many powerful machine learning algorithms use gradient descent optimization to identify patterns and learn from data.

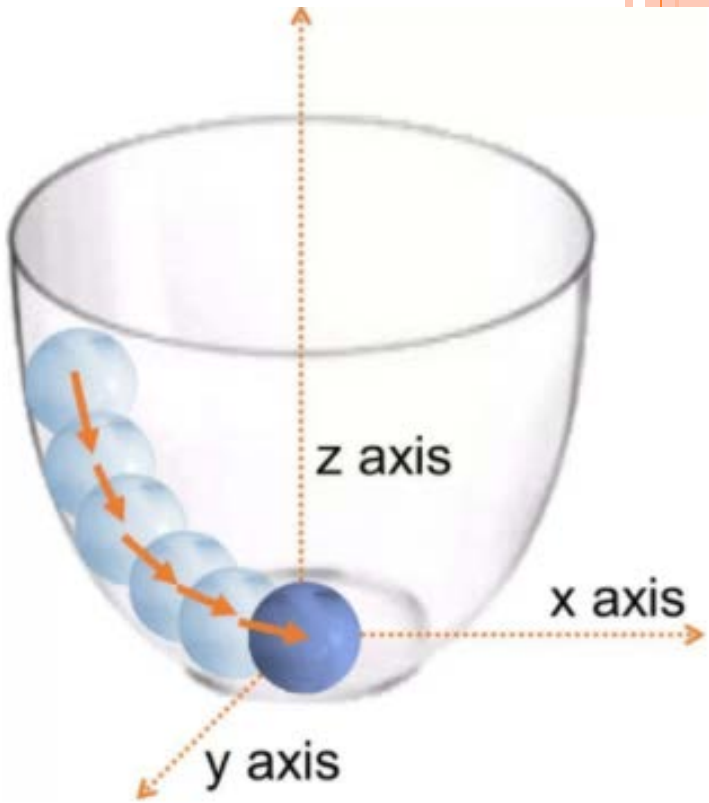


Gradient Descent Learning

- In situation 1, the ball will move from position b_1 to c_1 and not a_1. This ball will continue to travel till it reaches the bottom of the bowl.
- In situation 2, since the static ball is already at the bottom it will stay at b_2 and it won't move at all. The ball always tries to move from a higher potential energy state to a lower.
- This is similar to gradient descent optimization. Gradient descent optimization tries to minimize a loss function instead of potential energy.

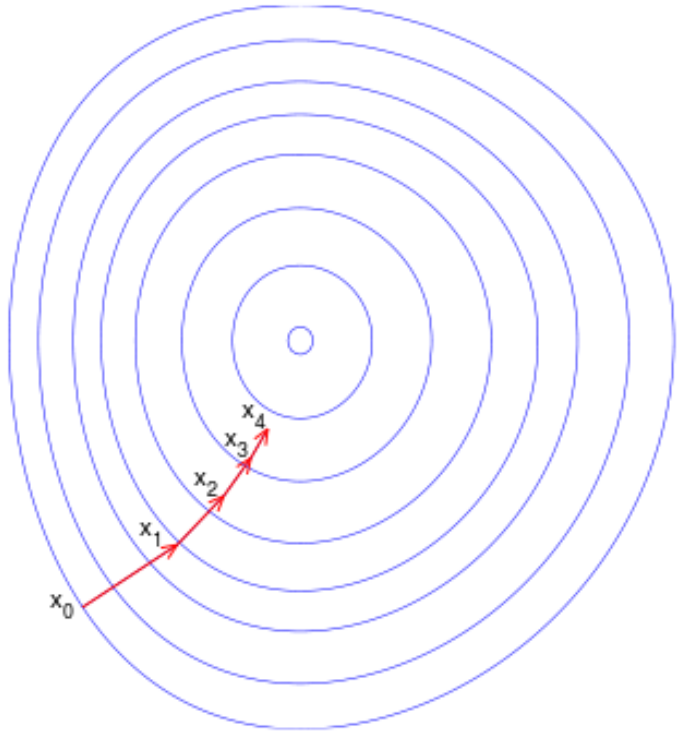
Gradient Descent Learning

- In terms of the coordinate system, the potential energy reduces as the ball rolls down the z-axis (vertical axis). The ball tried to modify it's position on the x and y-axes to determine the lowest possible potential energy or the minimum possible value on the z-axis.
- Also, notice that the ball experienced different pulls at different stages while it journeyed towards the bottom of the bowl. These pulls can be evaluated through the gradient or the slope of the orange arrows shown in the adjacent picture. The steeper the orange arrows larger is the force of gravity on the ball.
- For gradient descent optimization, the z-axis is the **loss function** and x & y-axes are the **coefficients** of the model. The loss function is equivalent to the potential energy of the ball in the bowl. The idea is to minimize loss function by adjusting x & y-axes



Gradient Descent Learning

- **Gradient descent** is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the *negative* of the gradient (or of the approximate gradient) of the function at the current point.(*Wiki)



- Here function **F** is assumed to be defined on the plane, and that its graph has a bowl shape. The blue curves are the contour lines, that is, the regions on which the value of **F** is constant. A red arrow originating at a point shows the direction of the negative gradient at that point. Note that the (negative) gradient at a point is orthogonal to the contour line going through that point. We see that gradient *descent* leads us to the bottom of the bowl, that is, to the point where the value of the function **F** is minimal.
- GD Simplified Example

Widrow and Hoff –LMS Learning Law

- The change vector is given by

  $\Delta w_i = \eta [b_i - \mathbf{w}^T_i \ \mathbf{a}] \ \mathbf{a}$ (b_i is desired o/p)

  Hence $\Delta w_{ij} = \eta [b_i - w^T_i \ a] \ a_j$ for j = 1,2...,M

- Supervised learning and a spl case of Delta Learning law,where output is assumed linear.(ADALINE) $f(x)=x_i$

Delta Learning Law

- The change in weight vector is given by

$$\Delta \mathbf{w}_i = \eta \, [b_i - f(\mathbf{w}_i^T \mathbf{a})] \; \dot{f}(\mathbf{w}_i^T \mathbf{a}) \, \mathbf{a}$$

- Where $\dot{f}(\mathbf{w}_i^T \mathbf{a})$ is derivative of activation fn.
- Continuous perceptron learning is also called Delta Learning,and it can be generalized for a network consisting of several layers of feedforward units→ Generalized Delta Rule

Table 1.2 Summary of Basic Learning Laws (Adapted from [Zurada, 1992])

Learning law	Weight adjustment Δw_{ij}	Initial weights	Learning
Hebbian	$\Delta w_{ij} = \eta f(\mathbf{w}_i^T \mathbf{a})\, a_j$ $= \eta s_i a_j$, for $j = 1, 2, \ldots, M$	Near zero	Unsupervised
Perceptron	$\Delta w_{ij} = \eta [b_i - \text{sgn}(\mathbf{w}_i^T \mathbf{a})]\, a_j$ $= \eta (b_i - s_i)\, a_j$, for $j = 1, 2, \ldots, M$	Random	Supervised
Delta	$\Delta w_{ij} = \eta [b_i - f(\mathbf{w}_i^T \mathbf{a})]\, \dot{f}(\mathbf{w}_i^T \mathbf{a})\, a_j$ $= \eta [b_i - s_i]\, \dot{f}(x_i)\, a_j$, for $j = 1, 2, \ldots, M$	Random	Supervised
Widrow-Hoff	$\Delta w_{ij} = \eta [b_i - \mathbf{w}_i^T \mathbf{a}]\, a_j$, for $j = 1, 2, \ldots, M$	Random	Supervised
Correlation	$\Delta w_{ij} = \eta b_i a_j$, for $j = 1, 2, \ldots, M$	Near zero	Supervised
Winner-take-all	$\Delta w_{kj} = \eta (a_j - w_{kj})$, k is the winning unit, for $j = 1, 2, \ldots, M$	Random but normalised	Unsupervised
Outstar	$\Delta w_{jk} = \eta (b_j - w_{jk})$, for $j = 1, 2, \ldots, M$	Zero	Supervised