What are Neural Networks?

- Neural Networks are networks of neurons, for example, as found in real (i.e. biological) brains.
- Artificial neurons are crude approximations of the neurons found in real brains. They may be physical devices, or purely mathematical constructs.
- Artificial Neural Networks (ANNs) are networks of Artificial Neurons and hence constitute crude approximations to parts of real brains. They maybe physical devices, or simulated on conventional
- Computers point of view, an ANN is just a parallel computational system consisting of many simple processing elements connected together in a specific way in order to perform a particular task.

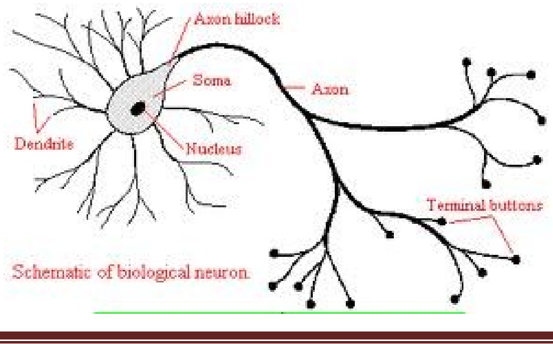
Why are Artificial Neural Networks?

- They are extremely powerful computational devices.
- Massive parallelism makes them very efficient
- They can learn and generalize from training data, so there is no need for enormous feats of programming
- They are particularly fault tolerant.
- They are very noise tolerant, so they can cope with situations where normal symbolic systems would have difficulty
- In principle, they can do anything a symbolic/logic system can do, and more

Brains vs. Computers

- ***Processing elements***: There are 10 synapses in the brain, compared with 10 transistors in the computer
- ***Processing speed***: 100 Hz for the brain compared to 10 Hz for the computer
- ***Style of computation***: The brain computes in parallel and distributed mode, whereas the computer mostly serially and centralized.
- ***Fault tolerant****: The brain is fault tolerant, whereas the computer is not*
- ***Adaptive:*** *T*he brain learns fast, whereas the computer doesn't even compare with an infant's learning capabilities
- ***Intelligence and consciousness***: The brain is highly intelligent and conscious, whereas the computer shows lack of intelligence
- ***Evolution***: The brains have been evolving for tens of millions of years; computers have been evolving for decades.

Basic Components of Biological Neurons



• The majority of neurons encode their activation or outputs as a series of brief electrical pulses (i.e. spikes or action potentials)
• The neuron's cell body (soma) processes the incoming activations and converts them into output activations.
• The neuron's nucleus contains the genetic material (DNA)
•Dendrites are fibers which emanate from the cell body and provide the receptive zone that receive activation from other neurons
•Axons are fibers acting as transmission lines that send action potentials to other neurons
• The junctions that allow signal transmission between the axons and the dendrites are called synapses.
The process of transmission is by diffusion of chemicals called neurotransmitters across the synaptic cleft.

Biological NN	**Artificial NN**
Soma	Neuron
Dendrite	Input
Axon	Output
Synapse	weight

What are Neural Networks used for?

There are two basic goals for neural network research:

Brain modeling:

- The biological goal of constructing models is of how real brains work.
- It can potentially help to understand the nature of perception, actions, learning and memory, thought and intelligence and/or formulate medical solutions to brain damaged patients.

Artificial System Construction:

- It may make machines more powerful and intelligent, relieve humans of tedious tasks, and may even improve upon human performance.

Weighting Factors (W)

The values w_1,w_2,w_3…. W_n are weights to determine the strength of input x_1,x_2,x_3….x_n

- The neuron computes the weighted sum of the input signals and compares the result with a threshold value, ω.

- $X = x_1w_1+x_2w_2+x_3w_3+\ldots..+x_nw_n = \sum_{i=1}^{n} xiwi$

- If the net input is less than the threshold (ω), the neuron output is -1.
- If the net input is greater than or equal to the threshold(ω) , the neuron becomes activated and its output attains value +1.

Threshold (ω)

Threshold is the magnitude offset value of the node. It affects the activation of the node output Y as

$$\mathbf{Y= f(X) = f\,\{} \sum_{i=1}^{n} xiwi \ \mathbf{- \omega\}}$$

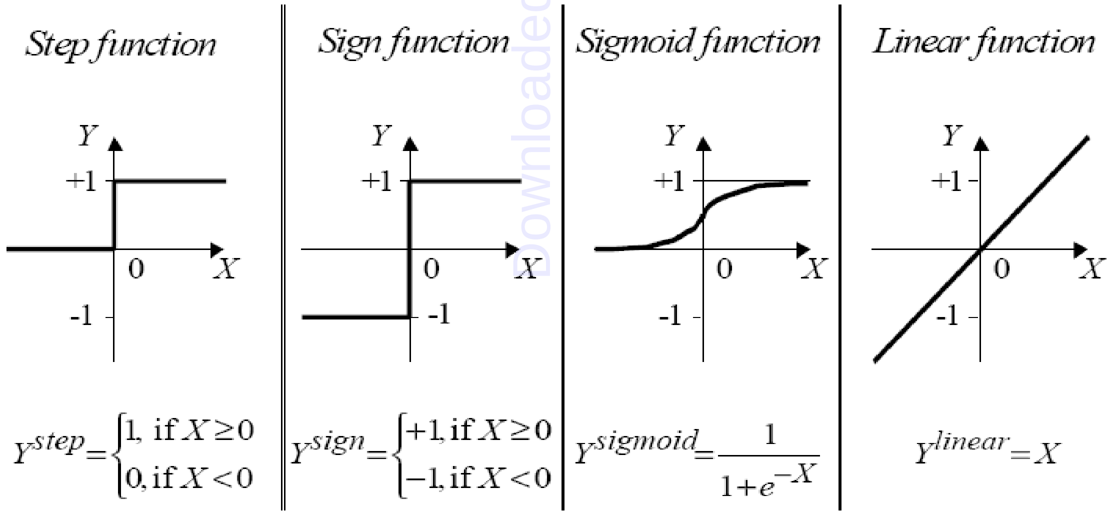
In practice neurons generally do not fire unless their total input goes above a threshold value.

Activation function

An activation function f performs a mathematical operation on the signal output. Activation function in example below is called sign function.

$$X = \sum_{i=1}^{n} x_i w_i \qquad Y = \begin{cases} +1, & \text{if } X \geq \theta \\ -1, & \text{if } X < \theta \end{cases}$$

Different types of Activation Function are given as figure below



$$Y^{step} = \begin{cases} 1, \text{if } X \geq 0 \\ 0, \text{if } X < 0 \end{cases} \quad Y^{sign} = \begin{cases} +1, \text{if } X \geq 0 \\ -1, \text{if } X < 0 \end{cases} \quad Y^{sigmoid} = \frac{1}{1+e^{-X}} \quad Y^{linear} = X$$

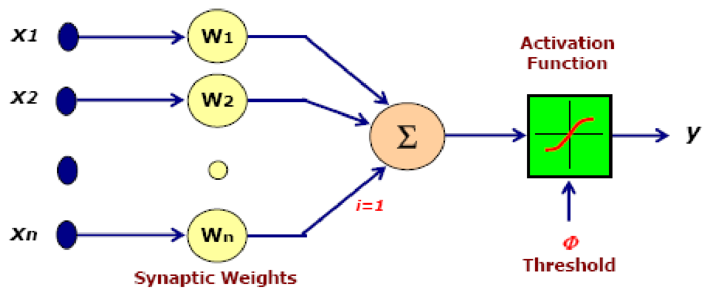


Fig Basic Elements of an Artificial Linear Neuron

Types of Neural Network

1. **Single layer feed forward network**

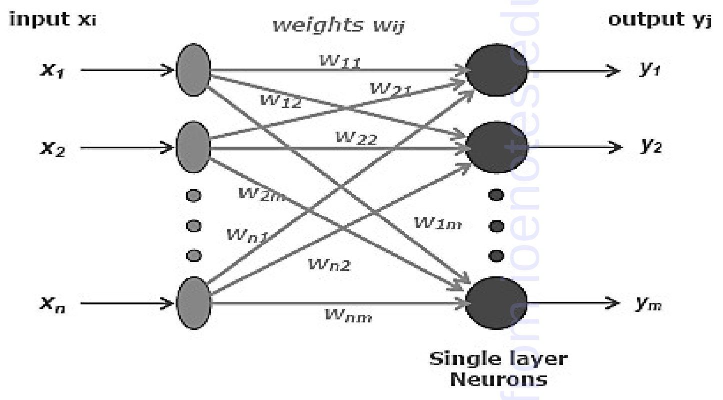


Fig. Single Layer Feed-forward Network

2. **Multilayer feed forward network**

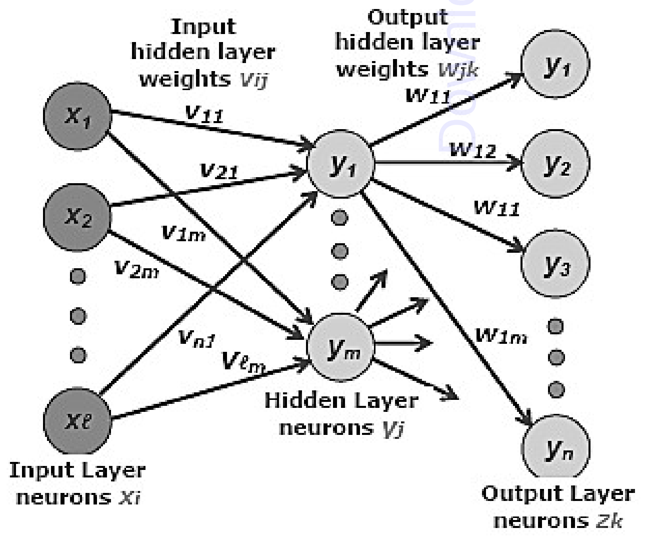


Fig. Multilayer feed-forward network in $(\ell - m - n)$ configuration.

3. **Recurrent network (feedback)**

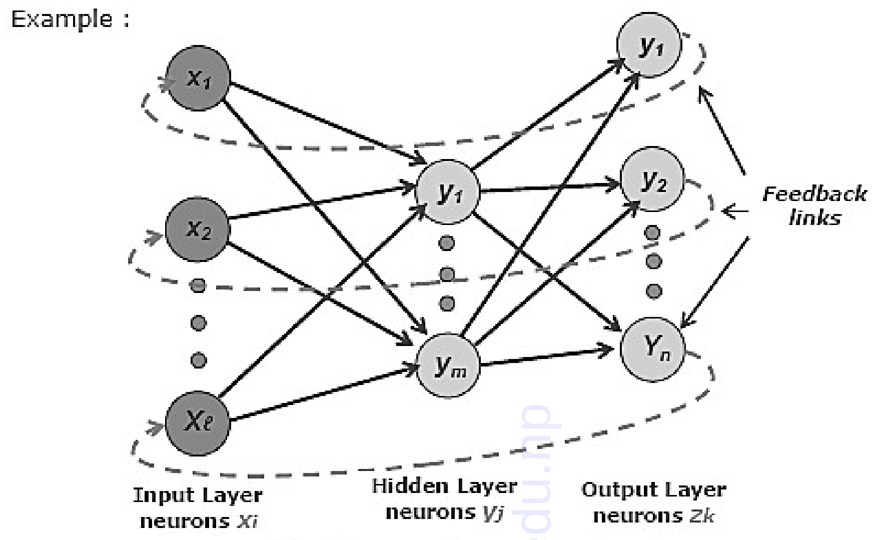


Fig. Recurrent neural network

Advantage of NN

- NN can perform tasks that a linear program cannot perform
- NN help where we can not formulate an algorithmic solution or where we get lots of examples of the behavior we require.
- When an element of the NN fails, it can continue without any problem by their parallel nature because of their distributive information nature.
- NN learns and does not need to be reprogrammed.
- Has massive parallel processing.
- Widely applied in data classification, clustering, pattern recognition,etc

Neural Network Applications

- ***Brain modeling:*** Aid our understanding of how the brain works, how behavior emerges from the interaction of networks of neurons, what needs to "get fixed" in brain damaged patients

- ***Real world applications***
  - Financial modeling – predicting the stock market
  - Time series prediction – climate, weather, seizures
  - Computer games – intelligent agents, chess, backgammon
  - Robotics – autonomous adaptable robots
  - Pattern recognition – speech recognition, seismic activity, sonar signals
  - Data analysis – data compression, data mining

- Bioinformatics – DNA sequencing, alignment

Learning Processes in Neural Networks

- Neural network has the ability to learn from its environment, and to improve its performance through learning.
- The improvement in performance takes place over time in accordance with some prescribed measure.
- A neural network learns about its environment through an iterative process of adjustments applied to its synaptic weights and thresholds.
- The network becomes more knowledgeable about its environment after each iteration of the learning process.

There are three broad types of learning:
1. Supervised learning (i.e. learning with an external teacher)
2. Unsupervised learning (i.e. learning with no help)
3. Reinforcement learning (i.e. learning with limited feedback)

Supervised learning

- Supervised learning is the machine learning task of inferring a function from training data and the training data consist of a set of training examples i.e. a supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples.
- In supervised learning, each example is a pair consisting of an input object and a desired output value. In other words, the inputs are assumed to be at the beginning and outputs at the end of the causal chain.
- In supervised the variables under investigation can be split into two groups: explanatory variables and one (or more) dependent variables.
- The target of the analysis is to specify a relationship between the explanatory variables and the dependent variable as it is done in regression analysis.
- Approaches to unsupervised learning include: Classification

Unsupervised Learning

- Unsupervised learning is the task of finding hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution.
- In unsupervised learning situations all variables are treated in the same way, there is no distinction between explanatory and dependent variables.
- For unsupervised learning typically either the target variable is unknown or has only been recorded for too small a number of cases.
- Unsupervised learning is closely related to the problem of density estimation in statistics.
- Approaches to unsupervised learning include: Clustering

Learning Rate (α)

- Learning rate is a constant in the algorithm of a neural network that affects the speed of learning**.**
- It is a control parameter of some training algorithms, which controls the step size when weights are iteratively adjusted.
- The performance of the algorithm is very sensitive to the proper setting of the learning rate.
- If the learning rate is set too high, the algorithm may oscillate and become unstable whereas if the learning rate is too small, the algorithm will take too long to converge.
- It is not practically significant to determine the optimal setting for the learning rate before training. In most of the cases the optimal learning rate changes during the training process.

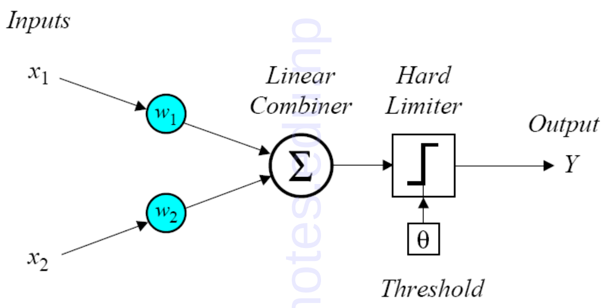
Backpropagation Algorithm

1. **Initialization**: Set all the weights and threshold levels of the n/w to random numbers uniformly distributed inside a small range[-2.4/Ti,2.4/Ti] where Ti is the total number of inputs of neuron i in the network
2. **Activation**: Activate the back-propagation neural network by applying inputs $x_i(p)$ and desired outputs $y_i(p)$
   - Calculate the actual outputs of the neuron in the hidden layer.
     - $Y_j(p)$ = sigmoid [$\sum_{i=0}^{n} x\,i(p).\,wij(p) - \omega\, j$]
   - Calculate the actual output of the neurons in the output layer
     - $Y_k(p)$=sigmoid [$\sum_{j=0}^{m} x\,jk(p).\,wjk(p) - \omega\, k$]
     - Where m is the number of inputs of neuron k in the output layer
3. **Weight training**: Update weights in the back-propagation network by propagating backward the errors associated with output neurons
   - Calculate the error gradient of the neurons in the output layer
     - $^{\delta}{}_k = y_k(p)\,[1-y_k(p)].\, e_k(p)$ where $e_k(p) = y_{d,k}(p) - y_k(p)$
   - Calculate the weight corrections
     - $\Delta\, w_{jk} = \alpha\, .\, y_j(p),{}^{\delta}{}_k(p)$
   - Update the weight at the output neurons
     - $w_{jk}(p+1) = w_{jk} + \Delta w_{jk}(p)$
   - Calculate the error gradient for the neurons in the hidden layer
     - $^{\delta}{}_j(p) = y_j(p)[1-y_j(p)].$ [$\sum_{k=1}^{l} {}^{\delta}k(p)\,w\,jk(p)$]
     - Calculate the weight corrections $\Delta wij = \alpha.\, x_i(p).\, {}^{\delta}{}_j(p)$
     - Update the weight at the hidden neuron
     - $W_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$

4. **Iteration**: Increase iteration by one, go back to step 2 and repeat the process until the selected error criterion is satisfied.

PERCEPTRON

- A Perceptron is the simplest kind of feed forward neural network invented by Frank Rosenblatt
- A perceptron can learn any linearly separable functions, given enough training.
- The model consists of a linear combiner followed by a activation function.
- The weighted sum of the inputs is applied to the activation function, which produces an output equal to +1 if its input is positive and -1 if it is negative.



Single layer 2 input Perceptron model

Perceptron algorithm

1. **Initialization**
   a. Set the initial weights w_i and threshold ɷ to random numbers in the range [-0.5, +0.5]
   b. If the error, e(p) is positive, we need to increase perceptron output Y(p), but if it is negative, we need to decrease Y(p)
2. **Activation**
   a. Activate the perceptron by applying inputs $x_i(p)$ and desired output $Y_d(p)$. Calculate the actual output at iteration p=1

   Where n is the number of the perceptron inputs, and step is activation function

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

step

3. **Weight Training**
   a. Update the weights of the perceptron
   $W_i(p+1) = w_i(p) + \Delta w_i(p)$, Where $\Delta w_i(p)$ is the weight correction at iteration p. the weight correction is computed by the delta rule.
   $\Delta w_i(p) = \alpha.\ X_i(p).\ e(p)$, α is learning rate
4. **Iteration**
   a. Increase iteration *p* by one, go back to Step 2 and repeat the process until convergence.

Perceptron example: construct Perceptron neural network model that performs like AND gate, with learning rate *$\alpha = 0.1$ and threshold $\omega = 0.2$*

HEBBIAN LEARNING

The oldest and most famous of all learning rules is Hebb’s postulate of learning:

“*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*”

It is applied to artificial neurons as “If two interconnected neurons are both “on” at the same time, then the weight between them should be increased”

The operation of a NN is determined by the values of the interconnection weights

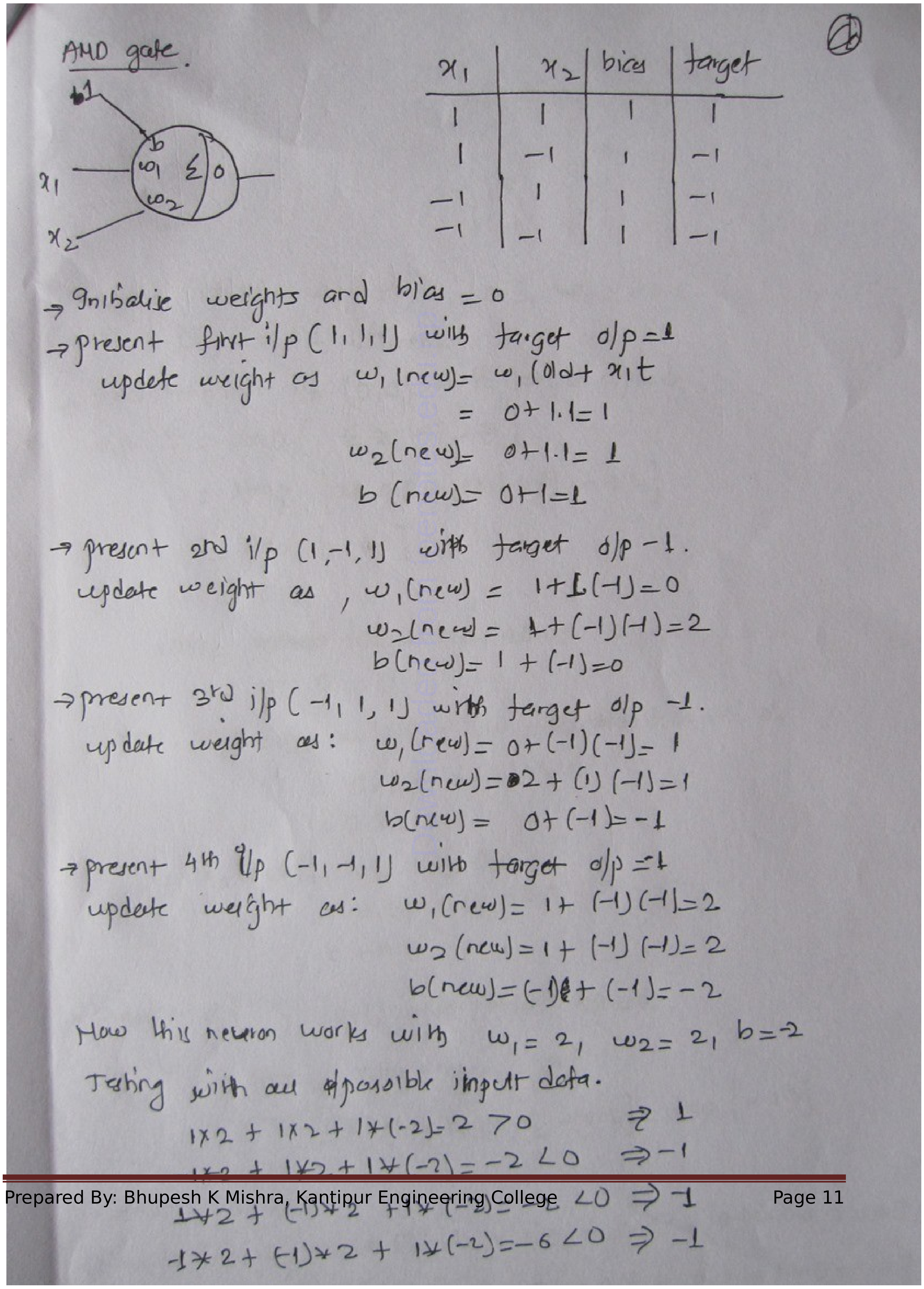
There is no algorithm that determines how the weight should be assigned in order to solve a specific problem. Hence weights are determined by a learning process

Hebb’s Algorithm

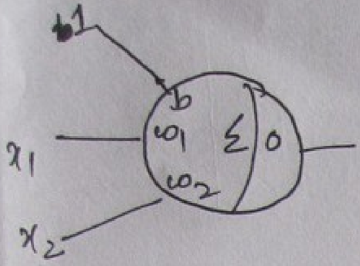
- **Step 0**: initialize all weights to 0
- **Step 1**: Given a training input, s, with its target output, t, set the activations of the input units: $x_i = s_i$
- **Step 2**: Set the activation of the output unit to the target value: $y = t$
- **Step 3**: Adjust the weights: $w_i(new) = w_i(old) + x_i y$
- **Step 4**: Adjust the bias (just like the weights): $b(new) = b(old) + y$

Example: Construct a Hebb Net which performs like an AND function, that is, only when both features are “active” will the data be in the target class.

Hence this neuron is functional



AND gate.



x_1	x_2	bias	target
1	1	1	1
1	−1	1	−1
−1	1	1	−1
−1	−1	1	−1

→ Initialise weights and bias = 0

→ present first i/p (1, 1, 1) with target o/p = 1

update weight as $w_1(new) = w_1(old) + x_1 t$

$= 0 + 1.1 = 1$

$w_2(new) = 0 + 1.1 = 1$

$b(new) = 0 + 1 = 1$

→ present 2nd i/p (1, −1, 1) with target o/p −1.

update weight as, $w_1(new) = 1 + 1(-1) = 0$

$w_2(new) = 1 + (-1)(-1) = 2$

$b(new) = 1 + (-1) = 0$

→ present 3rd i/p (−1, 1, 1) with target o/p −1.

update weight as: $w_1(new) = 0 + (-1)(-1) = 1$

$w_2(new) = 2 + (1)(-1) = 1$

$b(new) = 0 + (-1) = -1$

→ present 4th i/p (−1, −1, 1) with target o/p = −1

update weight as: $w_1(new) = 1 + (-1)(-1) = 2$

$w_2(new) = 1 + (-1)(-1) = 2$

$b(new) = (-1) + (-1) = -2$

Now this neuron works with $w_1 = 2$, $w_2 = 2$, $b = -2$

Testing with all possible input data.

$1\times2 + 1\times2 + 1\times(-2) = 2 > 0 \Rightarrow 1$

$1\times2 + 1\times2 + 1\times(-2) = -2 < 0 \Rightarrow -1$

$1\times2 + (-1)\times2 + 1\times(-2) = -2 < 0 \Rightarrow -1$

$-1\times2 + (-1)\times2 + 1\times(-2) = -6 < 0 \Rightarrow -1$

ADALINE NETWORK

Adaline network is a variation on the Perceptron Network

- inputs are +1 or -1
- outputs are +1 or -1
- uses a bias input

It is trained using the Delta Rule which is also known as the least mean squares (LMS) or Widrow-Hoff rule. The activation function, during training is the **identity function.** After training the activation is a threshold function

Adaline Algorithm

- **Step 0**: initialize the weights to small random values and select a learning rate, a
- **Step 1**: for each input vector s, with target output, t set the inputs to s
- **Step 2**: compute the neuron inputs

Neuron input

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

Delta rule

$$b(new) = b(old) + \alpha(t - y\_in)$$
$$w_i(new) = w_i(old) + \alpha(t - y\_in)x_i$$

- **Step 3**: use the delta rule to update the bias and weights
- **Step 4**: stop if the largest weight change across all the training samples is less than a specified tolerance, otherwise cycle through the training set again

Learning Rate for adaline network

- The performance of an ADALINE neuron depends heavily on the choice of the learning rate
    - o if it is too large the system will not converge
    - o if it is too small the convergence will take to long
- Typically, a is selected by trial and error
    - o typical range: $0.01 < \alpha < 10.0$
    - o often start at 0.1
    - o sometimes it is suggested that: $0.1 < n \; \alpha < 1.0$ where n is the number of inputs

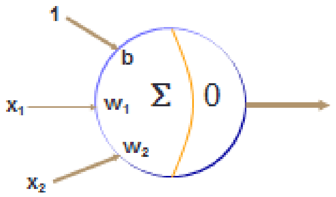
Example: Construct an AND function for a ADALINE neuron. let$\alpha = 0.1$

$$y\_in = b + \quad xiwi$$

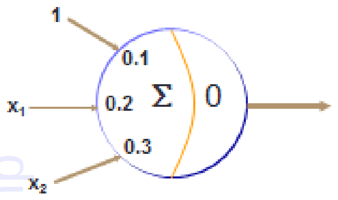
Activation Function

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

x1	x2	bias	Target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

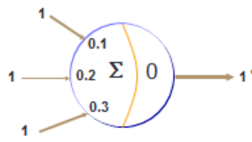


Initial Conditions: Set the weights to small random values:



First Training Run

- Apply the input (1,1) with output 1



Neuron input

$y\_in = b + \Sigma\ x_i w_i$

The net input is:

$y\_in = 0.1 + 0.2*1 + 0.3*1 = 0.6$

Delta rule

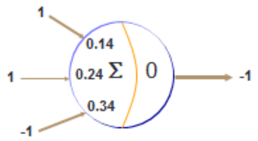
$b(new) = b(old) + \alpha(t - y\_in)$
$w_i(new) = w_i(old) + \alpha(t - y\_in)x_i$

The new weights are:
$b = 0.1 + 0.1(1-0.6) = 0.14$
$w_1 = 0.2 + 0.1(1-0.6)1 = 0.24$
$w_2 = 0.3 + 0.1(1-0.6)1 = 0.34$

The largest weight change is 0.04

Second Training Run

- Apply the second training set (1 -1) with output -1



The net input is:

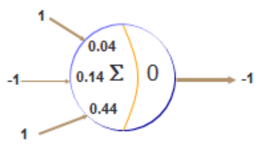
$y\_in = 0.14 + 0.24*1 + 0.34*(-1) = 0.04$

The new weights are:
$b = 0.14 - 0.1(1+0.04) = 0.04$
$w_1 = 0.24 - 0.1(1+0.04)1 = 0.14$
$w_2 = 0.34 + 0.1(1+0.04)1 = 0.44$

The largest weight change is 0.1

Third Training Run

- Apply the third training set (-1 1) with output -1



The net input is:

$y\_in = 0.04 - 0.14*1 + 0.44*1 = 0.34$

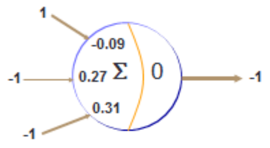
The new weights are:

$b = 0.04 - 0.1(1+0.34) = -0.09$
$w_1 = 0.14 + 0.1(1+0.34)1 = 0.27$
$w_2 = 0.44 - 0.1(1+0.34)1 = 0.31$

The largest weight change is 0.13

Fourth Training Run

- Apply the fourth training set (-1 -1) with output -1



The net input is:

$y\_in = -0.09 - 0.27*1 - 0.31*1 = -0.67$

The new weights are:

$b = -0.09 - 0.1(1+0.67) = -0.27$
$w_1 = 0.27 + 0.1(1+0.67)1 = 0.43$
$w_2 = 0.31 + 0.1(1+0.67)1 = 0.47$

The largest weight change is 0.16

Continue to cycle through the four training inputs until the largest change in the weights over a complete cycle is less than some small number (say 0.01)

In this case, the solution becomes

- $b = -0.5$
- $w_1 = 0.5$
- $w_2 = 0.5$

Kohnen Nwtwork

- The Kohonen neural network differs from the feedforward back propagation neural network.
- Kohonen nueral networks are used because they are a relatively simple network to construct that can be trained very rapidly.
- The Kohonen neural network contains only an input and output layer of neurons. There is no hidden layer in a Kohonen neural network
- It differs both in how it is trained and how it recalls a pattern. The Kohohen neural network does not use any sort of activation function and a bias weight.
- Output from the Kohonen neural network does not consist of the output of several neurons.
- When a pattern is presented to a Kohonen network one of the output neurons is selected as a "winner". This "winning" neuron is the output from the Kohonen network.
- Often these "winning" neurons represent groups in the data that is presented to the Kohonen network.

- The most significant difference between the Kohonen neural network and the feed forward back propagation neural network is that the Kohonen network trained in an unsupervised mode.

Difference between the Kohonen neural network and the feed forward back propagation neural network:

- Kohonen network trained in an unsupervised mode.
- This means that the Kohonen network is presented with data, but the correct output that corresponds to that data is not specified.
- Using the Kohonen network this data can be classified into groups.

Limitations of the Kohonen neural network:

- The neural networks with only two layers can only be applied to linearly separable problems and this is the case with the Kohonen neural network.
- Kohonennueral networks are used because they are a relatively simple network to construct that can be trained very rapidly.

Hopfield Neural Network

- A Hopfield network is a form of recurrent artificial neural network invented by John Hopfield.
- They are guaranteed to converge to a local minimum, but convergence to a false pattern (wrong local minimum) rather than the stored pattern (expected local minimum) can occur. Hopfield networks also provide a model for understanding human memory.
- The Hopfield neural network is perhaps the simplest of neural networks. The Hopfield neural network is a fully connected single layer auto associative network. This means it has one single layer, with each neuron connected to every other neuron.
- The units in Hopfield nets are binary threshold units, i.e. the units only take on two different values for their states and the value is determined by whether or not the units' input exceeds their threshold. Hopfield nets normally have units that take on values of 1 or -1.
- Hopfield network consists of a set of N interconnected neurons which update their activation values asynchronously and independent of other neurons.

Single layer n neuron Hopfield network

Primary application of the hopfield network is an associative memory. Weights of the connections between the neurons have to be set such that it corresponds to some pattern.

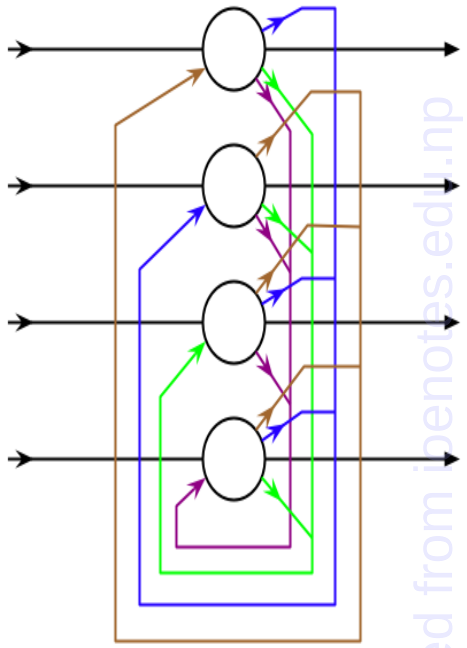
- Hopfield chose activation values of 1 and 0, but now we use values +1 and -1
- The net input $s_k(t+1)$ of a neuron k at cycle t+1 is a weighted sum

$$s_k(t+1) = \sum_{j \neq k} y_j(t)\, w_{jk} + \theta_k.$$

A simple threshold function is applied to the net input to obtain the new activation value $y_k(t + 1)$ at time t + 1:

$$y_k(t+1) = \begin{cases} +1 & \text{if } s_k(t+1) > U_k \\ -1 & \text{if } s_k(t+1) < U_k \\ y_k(t) & \text{otherwise,} \end{cases}$$

-



7