Unit I

Contents

1 Neural Networks-1(Introduction & Architecture)

1.1 Computation in the brain (Neuron, Nerve structure and synapse)

The brain - that's my second most favourite organ! - Woody Allen

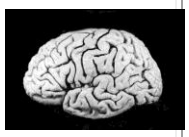
1.2 The Brain as an Information Processing System

The human brain contains about 100 billion nerve cells, or neurons. On average, each neuron is connected to other neurons through about 10,000 synapses. (The actual figures vary greatly, depending on the local neuroanatomy.) The brain's network of neurons forms a massively parallel information processing system. This contrasts with conventional computers, in which a single processor executes a single series of instructions.

Against this, consider the time taken for each elementary operation: neurons typically operate at a maximum rate of about 100 Hz, while a conventional CPU carries out several hundred million machine level operations per second. Despite of being built with very slow hardware, the brain has quite remarkable capabilities:

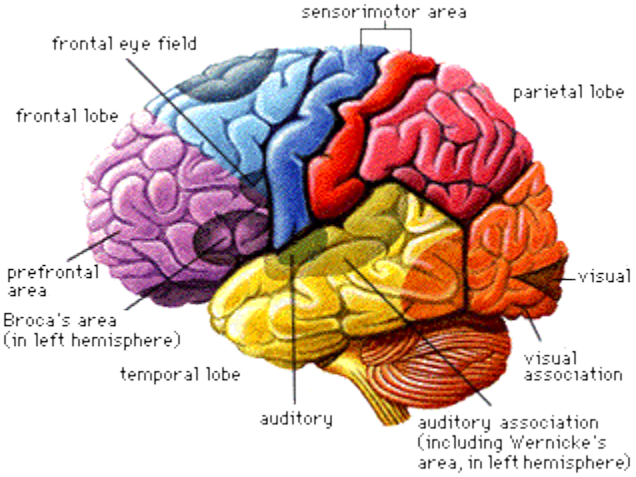
- Its performance tends to degrade gracefully under partial damage. In contrast, most programs and engineered systems are brittle: if you remove some arbitrary parts, very likely the whole will cease to function.
- It can learn (reorganize itself) from experience.
- This means that partial recovery from damage is possible if healthy units can learn to take over the functions previously carried out by the damaged areas.
- It performs massively parallel computations extremely efficiently. For example, complex visual perception occurs within less than 100 ms, that is, 10 processing steps!

- It supports our intelligence and self-awareness. (Nobody knows yet how this occurs.)

	processing elements	element size	energy use	processing speed	style of computation	fault tolerant	learns	intelligent, conscious
	10^{14} synapses	10^{-6} m	30 W	100 Hz	parallel, distributed	yes	yes	usually
	10^{8} transistors	10^{-6} m	30 W (CPU)	10^{9} Hz	serial, centralized	no	a little	not (yet)

As a discipline of Artificial Intelligence, Neural Networks attempt to bring computers a little closer to the brain's capabilities by imitating certain aspects of information processing in the brain, in a highly simplified way.

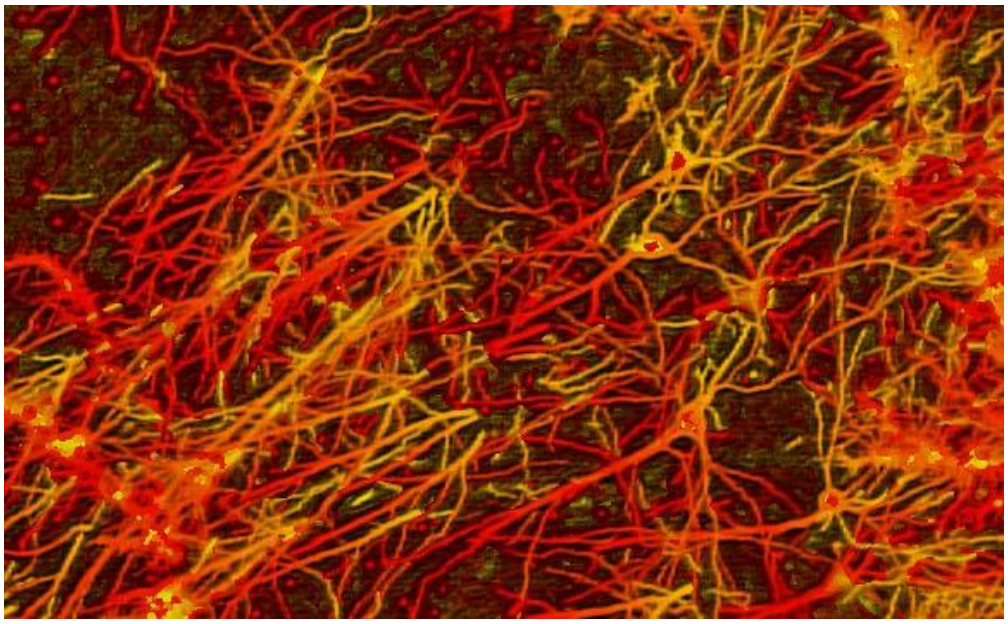
1.3 Neural Networks in the Brain



The brain is not homogeneous. At the largest anatomical scale, we distinguish **cortex**, **midbrain**, **brainstem**, and **cerebellum**. Each of these can be hierarchically subdivided into many **regions**, and **areas** within each region, either according to the anatomical structure of the neural networks within it, or according to the function performed by them.

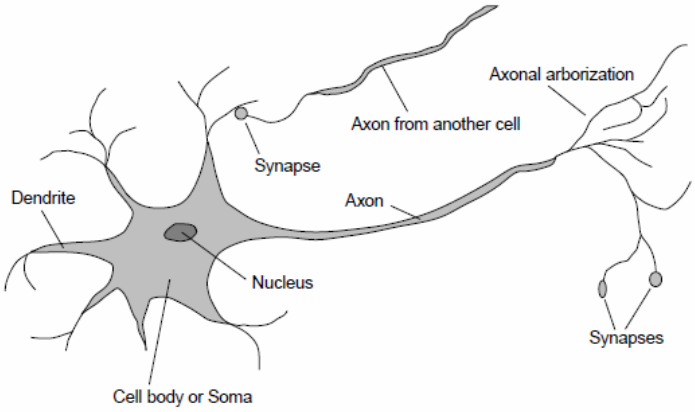
The overall pattern of **projections** (bundles of neural connections) between areas is extremely complex, and only partially known. The best mapped (and largest) system in the human brain is the visual system, where the first 10 or 11 processing stages have been identified. We distinguish **feedforward** projections that go from earlier processing stages (near the sensory input) to later ones (near the motor output), from **feedback** connections that go in the opposite direction.

In addition to these long-range connections, neurons also link up with many thousands of their neighbours. In this way they form very dense, complex local networks:

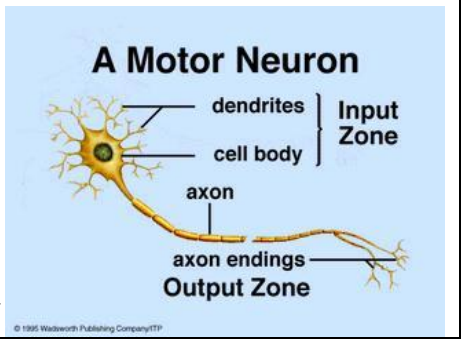


Brains

10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time
Signals are noisy "spike trains" of electrical potential



1.4 Neurons and Synapses



The basic computational unit in the nervous system is the nerve cell, or **neuron**. A neuron has 3 parts:

i) Dendrites (inputs) ii) Cell body iii) Axon (output)

A neuron receives input from other neurons (typically many

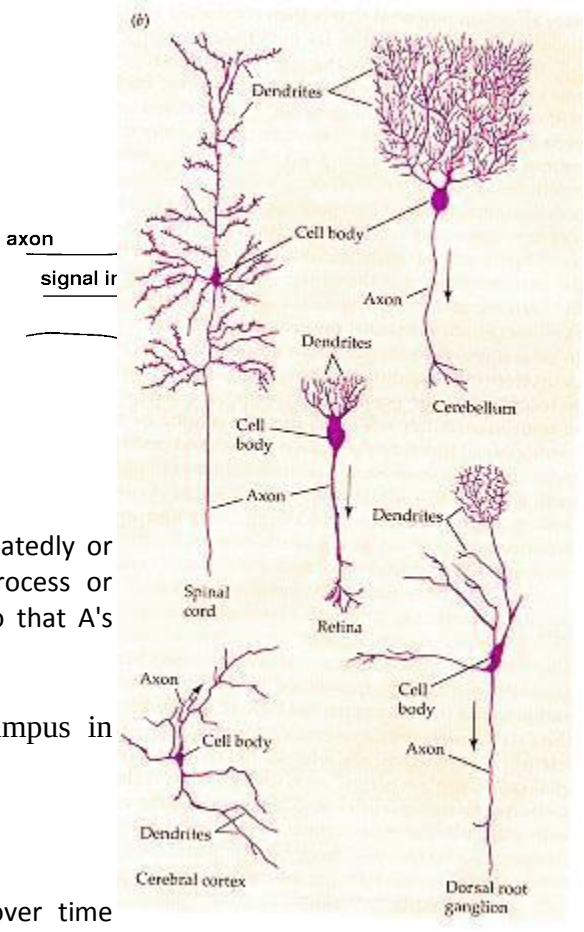
thousands). Inputs sum (approximately). Once input exceeds a critical level, the neuron discharges a **spike** - an electrical pulse that travels from the body, down the axon, to the next neuron(s) (or other receptors). This spiking event is also called **depolarization**, and is followed by a **refractory period**, during which the neuron is unable to fire.

The axon endings (Output Zone) almost touch the dendrites or cell body of the next neuron. Transmission of an electrical signal from one neuron to the next is effected by**neurotransmittors**, chemicals which are released from the first neuron and which bind to receptors in the second. This link is called a **synapse**. The extent to which the signal from one neuron is passed on to the next depends on many factors, e.g. the amount of neurotransmittor available, the number and arrangement of receptors, amount of neurotransmittor reabsorbed, etc.

1.5 Synaptic Learning

Brains learn. Of course. From what we know of neuronal structures, one way brains learn is by altering the strengths of connections between neurons, and by adding or deleting connections between neurons. Furthermore, they learn "on-line", based on experience, and typically without the benefit of a benevolent teacher.

The efficacy of a synapse can change as a result of experience, providing both memory and learning through **long-term potentiation**. One way this happens is through release of more neurotransmitter. Many other changes may also be involved.



axon

signal ir

Long-term Potentiation:

An enduring (>1 hour) increase in synaptic efficacy that results from high-frequency stimulation of an afferent (input) pathway.

Hebbs Postulate:

"When an axon of cell A… excites[s] cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells so that A's efficiency as one of the cells firing B is increased."

Bliss and Lomo discovered LTP in the hippocampus in 1973

Points to note about LTP:

Synapses become more or less important over time

(plasticity).
LTP is based on experience
LTP is based only on *local* information (Hebb's postulate)

1.6 Summary

The following properties of nervous systems will be of particular interest in our neurally-inspired models:

parallel, distributed information processing
high degree of connectivity among basic units
connections are modifiable based on experience
learning is a constant process, and usually unsupervised
learning is based only on local information
performance degrades gracefully if some units are removed
etc.....

1.7 Definition: NN

A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. Knowledge is acquired by the network from its environment through a learning process.
2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

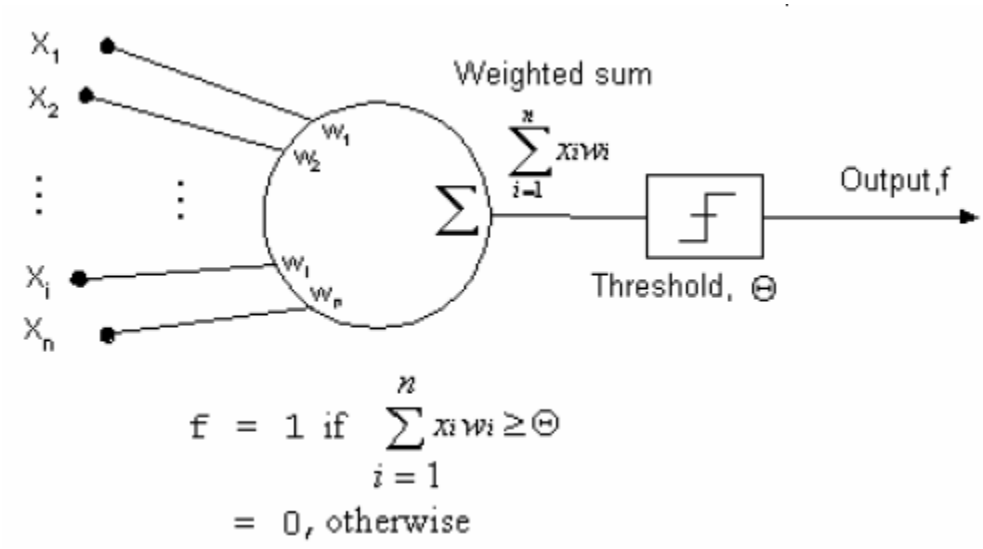
2 Artificial Neuron and its model

Alternative names are Artificial Neural Network (ANN), Parallel Distributed Processing System (PDPs), and Connectionist Systems.

Computational neurobiologists have constructed very elaborate computer models of neurons in order to run detailed simulations of particular circuits in the brain. As Computer Scientists, we are more interested in the general properties of neural networks, independent of how they are actually "implemented" in the brain. This means that we can use much simpler, abstract "neurons", which (hopefully) capture the essence of neural computation even if they leave out much of the details of how biological neurons work.

People have implemented model neurons in hardware as electronic circuits, often integrated on VLSI chips. Remember though that computers run much faster than brains - we can therefore run fairly large networks of simple model neurons as software simulations in reasonable time. This has obvious advantages over having to use special "neural" computer hardware.

2.1 A Simple Artificial Neuron / The Mathematical Model



$$f = 1 \text{ if } \sum_{i=1}^{n} x_i w_i \geq \Theta$$
$$= 0, \text{ otherwise}$$

Early, simple model of neuron called a Threshold Logic Unit (TLU) describe in 1943 by McCulloch & Pitts with the formula:

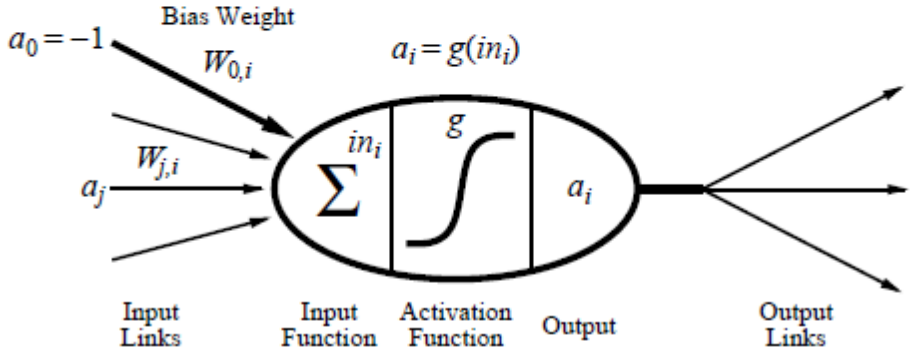
$$x_i = \sum_{i=1}^{n} w_{ij}\, y_j - \Theta_i$$

OR we can say neuron as:

McCulloch–Pitts "unit"

Output is a "squashed" linear function of the inputs:

$$a_i \leftarrow g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$

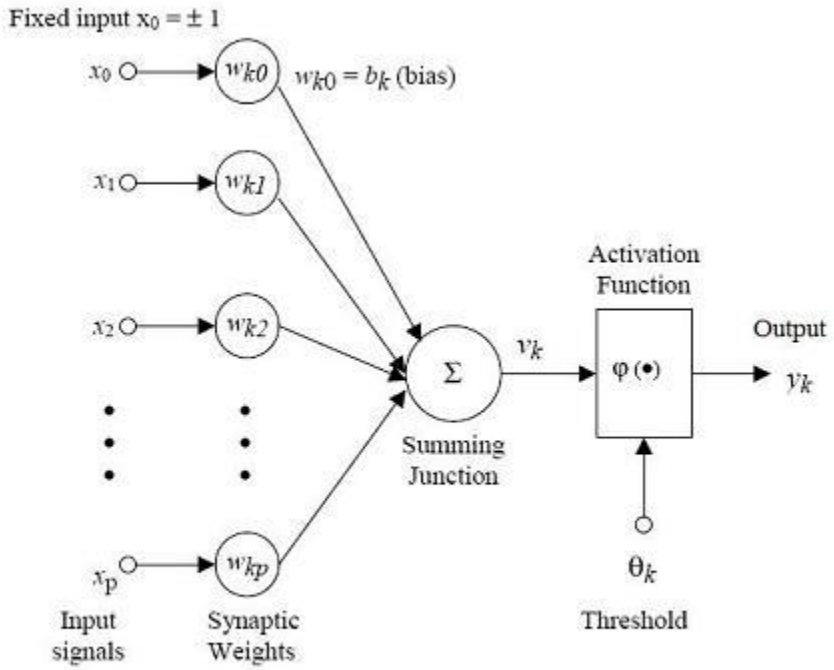


Another Description:

Once modeling an artificial functional model from the biological neuron, we must take into account

three basic components. First off, the synapses of the biological neuron are modeled as weights. Let's remember that the synapse of the biological neuron is the one which interconnects the neural network and gives the strength of the connection. For an artificial neuron, the weight is a number, and represents the synapse. A negative weight reflects an inhibitory connection, while positive values designate excitatory connections. The following components of the model represent the actual activity of the neuron cell. All inputs are summed altogether and modified by the weights. This activity is referred as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be -1 and 1.

Mathematically, this process is described in the figure



From this model the interval activity of the neuron can be shown to be:

$$v_k = \sum_{j=1}^{p} w_{kj} x_j$$

The output of the neuron, yk, would therefore be the outcome of some activation function on the value of vk.

2.2 ACTIVATION FUNCTIONS

As mentioned previously, the activation function acts as a squashing function, such that the output of a neuron in a neural network is between certain values (usually 0 and 1, or -1 and 1). In general, there are three types of activation functions, denoted by Φ(.) .

First, there is the

2.2.1 Threshold Function/ Step Function-

which takes on a value of 0 if the summed input is less than a certain threshold value (v), and the value 1 if the summed input is greater than or equal to the threshold value.

$$\varphi(v)=\begin{cases}1 & \text{if } v \geq 0\\ 0 & \text{if } v < 0\end{cases}$$

2.2.2 Piecewise-Linear function-

This function again can take on the values of 0 or 1, but can also take on values between that depending on the amplification factor in a certain region of linear operation.

$$\varphi(v)=\begin{cases}1 & v \geq \frac{1}{2}\\ v & -\frac{1}{2} > v > \frac{1}{2}\\ 0 & v \leq -\frac{1}{2}\end{cases}$$

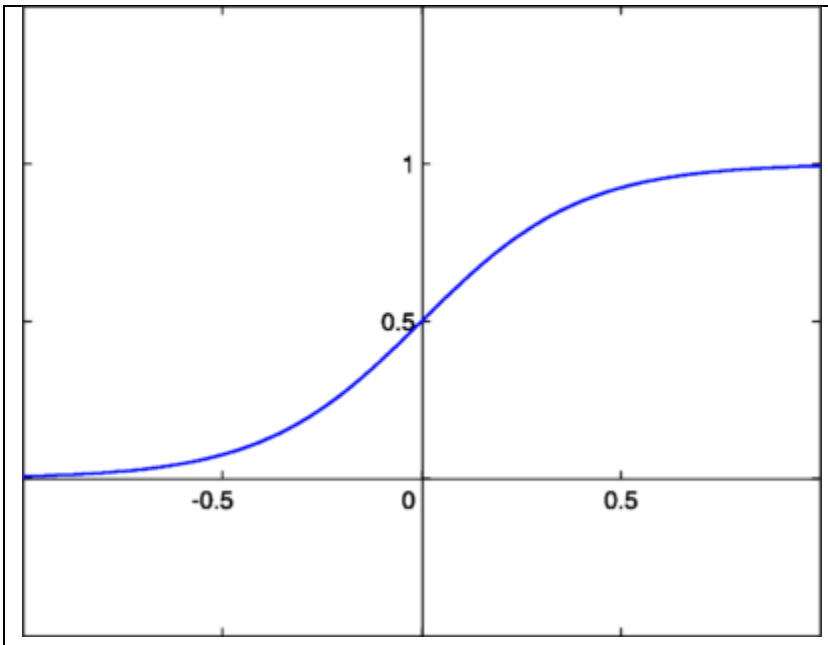
2.2.3 Sigmoid function-

This function can range between 0 and 1, but it is also sometimes useful to use the -1 to 1 range.

Continuous Log-Sigmoid Function, also known as a logistic function, is given by the relationship:

$$\sigma(t) = \frac{1}{1+e^{-\beta t}}$$

Where β is a slope parameter. This is called the log-sigmoid because a sigmoid can also be constructed using the hyperbolic tangent function instead of this relation, in which case it would be called a tan-sigmoid. Here, we will refer to the log-sigmoid as simply "sigmoid". The sigmoid has the property of being similar to the step function, but with the addition of a region of uncertainty. Sigmoid functions in this respect are very similar to the input-output relationships of biological neurons, although not exactly the same. Below is the graph of a sigmoid function.



Sigmoid functions are also prized because their derivatives are easy to calculate, which is helpful for calculating the weight updates in certain training algorithms. The derivative when $\beta = 1$ is given by:

$$\frac{d\sigma(t)}{dt} = \sigma(t)[1 - \sigma(t)]$$

When $\beta \neq 1$, using $\sigma(\beta, t) = \frac{1}{1 + e^{-\beta t}}$, the derivative is given by:

$$\frac{d\sigma(\beta, t)}{dt} = \beta[\sigma(\beta, t)[1 - \sigma(\beta, t)]]$$

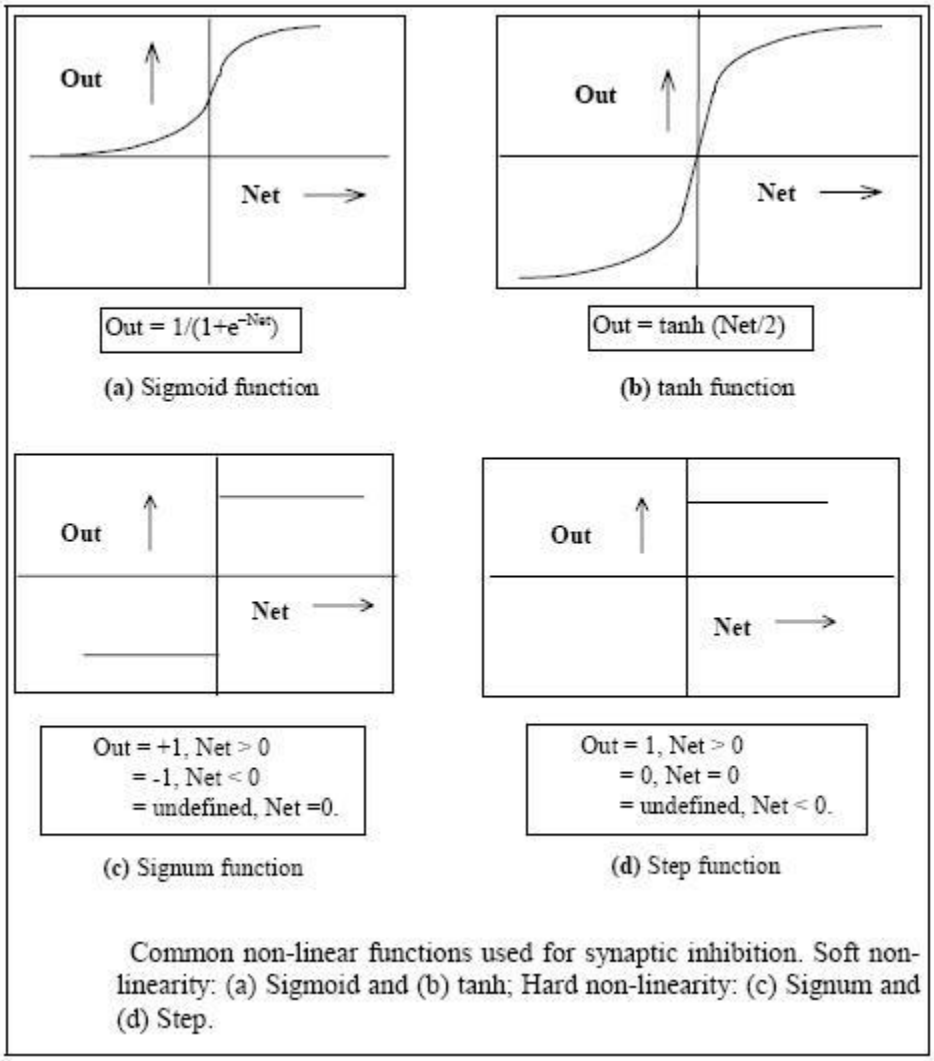
Another example of the sigmoid function is the **hyperbolic tangent function**.or

Continuous Tan-Sigmoid Function

$$\sigma(t) = tanh(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$$

Its derivative is:

$$\frac{d\sigma(t)}{dt} = 1 - tanh^2(t) = sech^2(t) = 1 - \frac{(e^t - e^{-t})^2}{(e^t + e^{-t})^2}$$



Common non-linear functions used for synaptic inhibition. Soft non-linearity: (a) Sigmoid and (b) tanh; Hard non-linearity: (c) Signum and (d) Step.

Activation Function	$g(a)$	$g'(a)$
Linear	$g(a) = a$	$g'(a) = 1$ (a constant)
Logistic	$g(a) = \frac{1}{1+e^{-a}}$	$g'(a) = g(a)(1-g(a))$
Hyperbolic-tangent	$g(a) = \tanh(a)$	$g'(a) = sech^2(a) = 1 - tanh^2(a)$
Squash	$g(a) = \frac{a}{1+\lvert a\rvert}$	$g'(a) = \frac{1}{(1+\lvert a\rvert)^2} = (1-\lvert g(a)\rvert)^2$

2.3 Perceptrons: Linear Threshold Units (LTU) or Threshold Logic Unit (TLU)

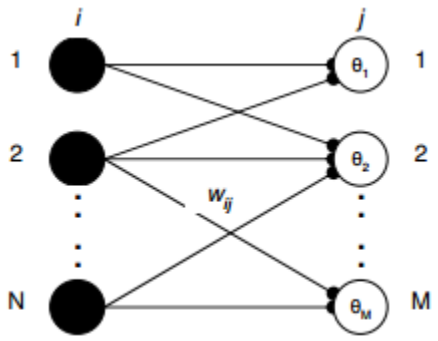
Defined as a feedforward network that has one layer of fixed inputs and trainable TLU's capable of computing any output (within physical limits of the structure) given a set of inputs by adjusting the weights and thresholds of the TLUs.

- can be single or multilayer
- known number of connections and weights, and first layer weights known

OR

The Perceptron

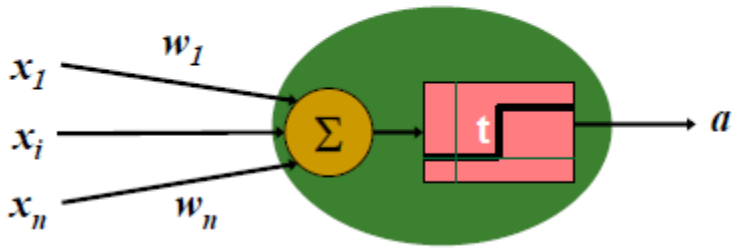
- We can connect any number of McCulloch-Pitts (MP) neurons together in any way we like.
- An arrangement of one input layer of McCulloch-Pitts neurons feeding forward to one output layer of McCulloch-Pitts neurons is known as a Perceptron.



$$Y_j = \text{sgn}(\sum_{i=1}^{n} Y_i \cdot w_{ij} - \theta_j)$$

Summary:

- **LTUs where studied in the 1950s!**
  - Mainly as single-layered nets
  - Since an effective learning algorithm was known
- **Perceptrons:**
  - Simple 1-layer network, units act independently
  - Composed of linear threshold units (LTU)
    - A unit's inputs, x_i, are weighted, w_i, and combined
    - Step function computes activation level, ***a***

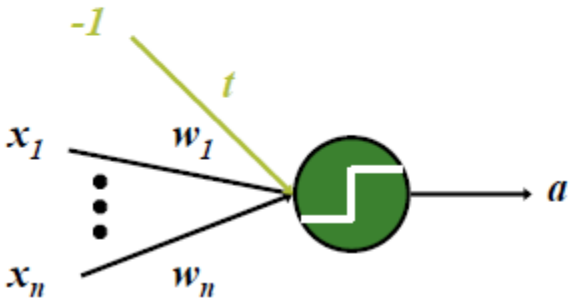


- **Threshold is just another weight (called the bias):**

$(w_1 * x_1) + (w_2 * x_2) + \ldots + (w_n * x_n) >= t$

is equivalent to

$(w_1 * x_1) + (w_2 * x_2) + \ldots + (w_n * x_n) + (t * -1) >= 0$



2.3.1 Implementing Logic Gates with MP Neurons

We can use McCulloch-Pitts neurons to implement the basic logic gates (e.g. AND, OR, NOT).

It is well known from logic that we can construct any logical function from these three basic logic gates.

All we need to do is find the appropriate connection weights and neuron thresholds to produce the right outputs for each set of inputs.

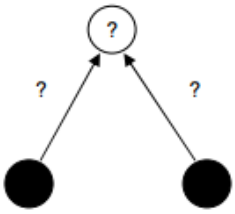
We shall see explicitly how one can construct simple networks that perform NOT, AND, and OR.

Implementation of Logical NOT, AND, and OR

NOT	
in	out
0	1
1	0

AND		
in_1	in_2	out
0	0	0
0	1	0
1	0	0
1	1	1

OR		
in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	1

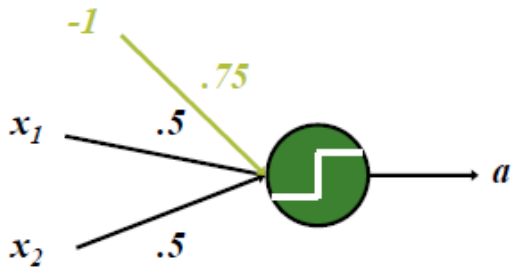


Problem:

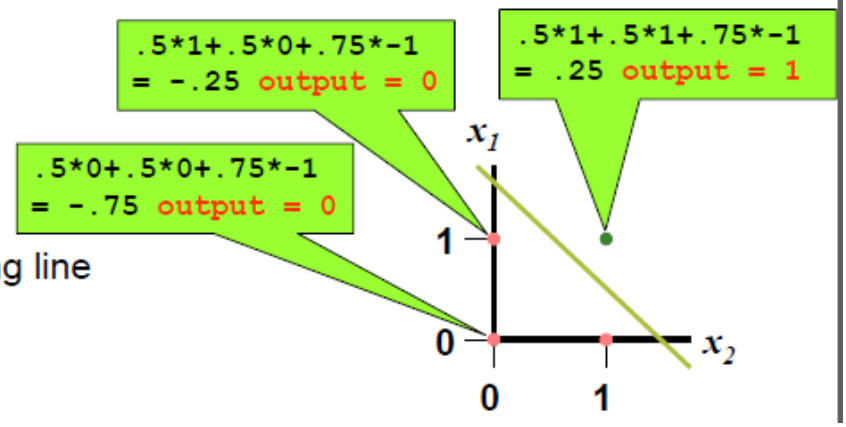
Train network to calculate the appropriate weights and thresholds in order to classify correctly the different classes (i.e. form decision boundaries between classes).

Perceptrons: AND Example

- **AND Perceptron:**
  - Inputs are 0 or 1
  - Output is 1 when both x_1 and x_2 are 1

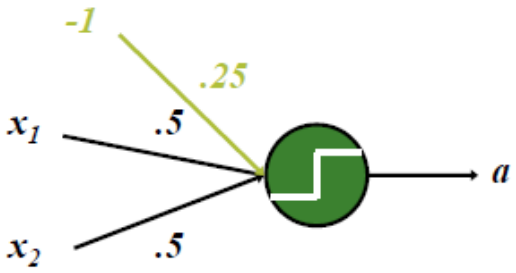


- **2-D input space**
  - 4 possible data points
  - Threshold is like a separating line

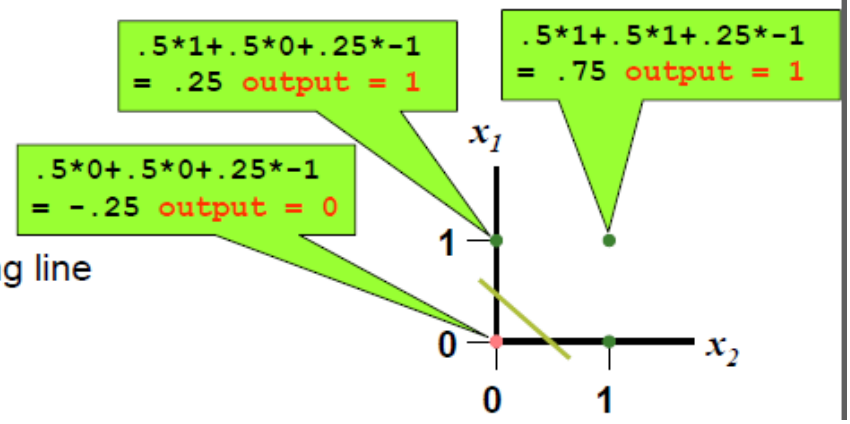


Perceptrons: OR Example

- **OR Perceptron:**
  - Inputs are 0 or 1
  - Output is 1 when either x_1 and/or x_2 are 1



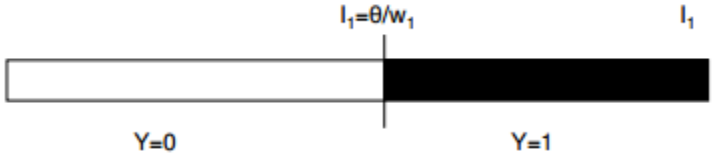
- **2-D input space**
  - 4 possible data points
  - Threshold is like a separating line



2.3.2 Decision Surfaces

Decision surface is the surface at which the output of the unit is precisely equal to the threshold, i.e. $\sum w_i I_i = \theta$

In 1-D the surface is just a point:



In 2-D, the surface is

$$I_1 \cdot w_1 + I_2 \cdot w_2 - \theta = 0$$

which we can re-write as

$$I_2 = \frac{\theta}{w_2} - \frac{w_1}{w_2} I_1$$

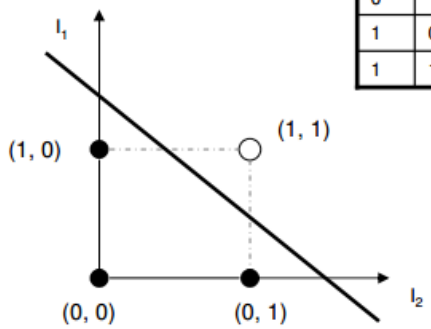
So, in 2-D the decision boundaries are **always** straight lines.

2.3.3 Decision Boundaries for AND and OR

We can now plot the decision boundaries of our logic gates:

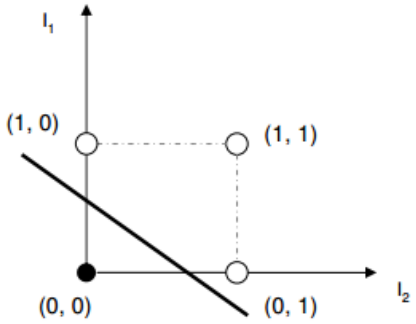
AND
w1=1, w2=1, θ=1.5

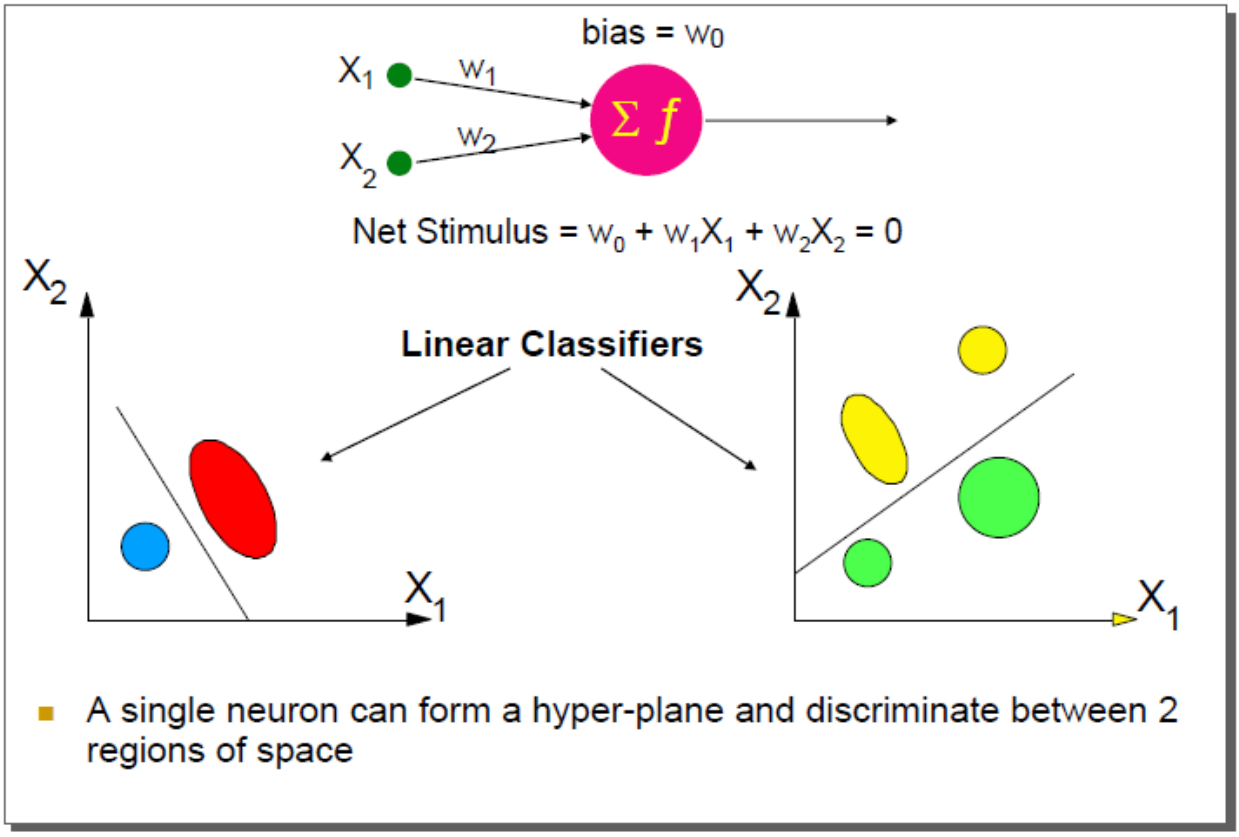
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1

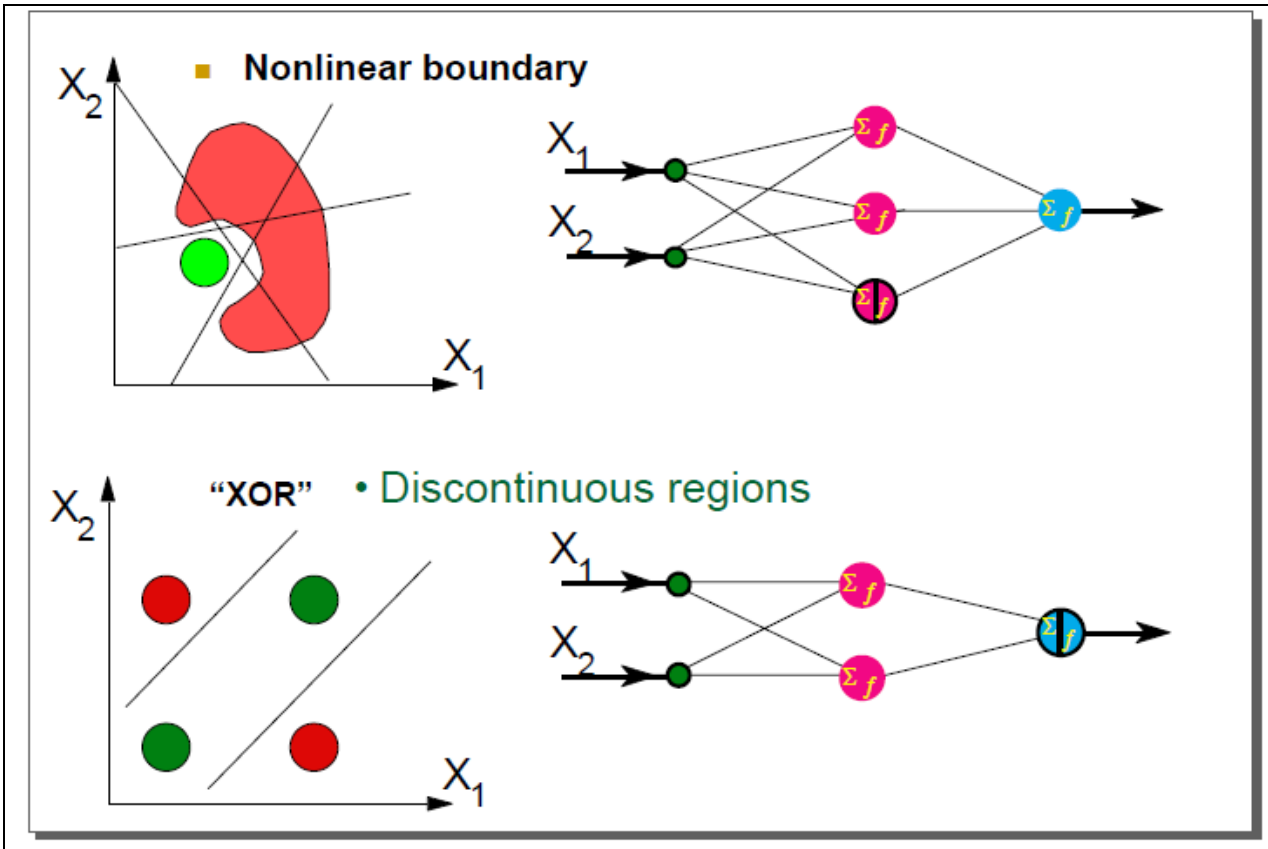


OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1

OR
w1=1, w2=1, θ=0.5



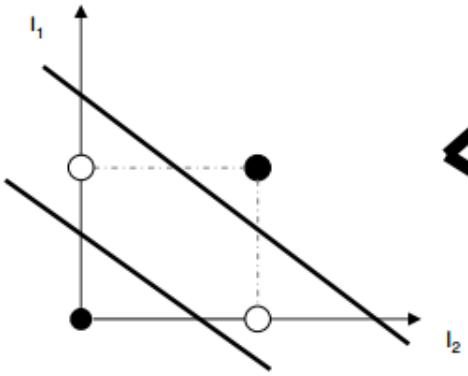

bias = w_0
X_1
w_1
X_2
w_2
Σ f
Net Stimulus = $w_0 + w_1X_1 + w_2X_2 = 0$
X_2
Linear Classifiers
X_1
X_2
X_1
A single neuron can form a hyper-plane and discriminate between 2 regions of space

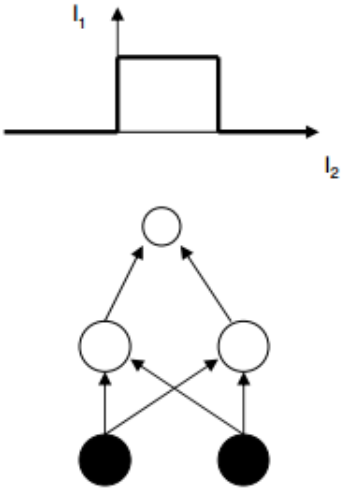


2.3.4 Decision Boundary for XOR

The difficulty in dealing with XOR is rather obvious. We need two straight lines to separate the different outputs/decisions:

XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0





Solution: either change the transfer function so that it has more than one decision boundary, or use a more complex network that is able to generate more complex decision boundaries.

2.4 ANN Architectures

Mathematically, ANNs can be represented as weighted directed graphs. The most common ANN architectures are:

2.4.1 Single-Layer Feed-Forward NNs:

One input layer and one output layer of processing units. No feedback connections (e.g. a P

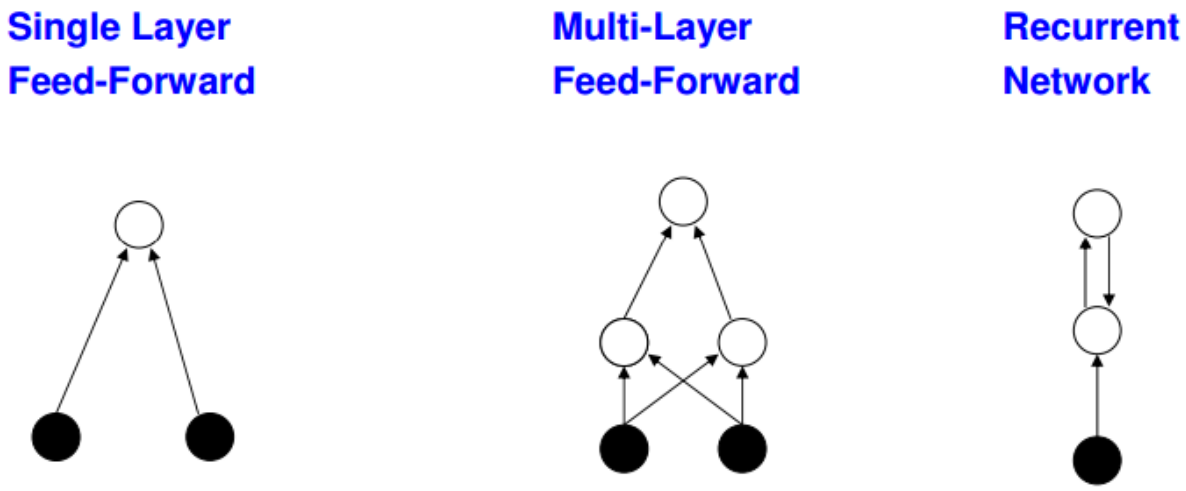
erceptron)

2.4.2 Multi-Layer Feed-Forward NNs:

One input layer, one output layer, and one or more hidden layers of processing units. No feedback connections (e.g. a Multi-Layer Perceptron)

2.4.3 Recurrent NNs:

Any network with at least one feedback connection. It may, or may not, have hidden units



Further interesting variations include: sparse connections, time-delayed connections, moving windows, …

2.4.4 NOTE: The Threshold as a Special Kind of Weight

The basic Perceptron equation can be simplified if we consider that the threshold is another connection weight:

$$\sum_{i=1}^{n} I_i \cdot w_{ij} - \theta_j = I_1 \cdot w_{1j} + I_2 \cdot w_{2j} + \ldots + I_n \cdot w_{nj} - \theta_j$$

If we define $w_{0j}=-\theta_j$ and $I_0=1$ then

$$\sum_{i=1}^{n} I_i \cdot w_{ij} - \theta_j = I_1 \cdot w_{1j} + I_2 \cdot w_{2j} + \ldots + I_n \cdot w_{nj} + I_0 \cdot w_{0j} = \sum_{i=0}^{n} I_i \cdot w_{ij}$$

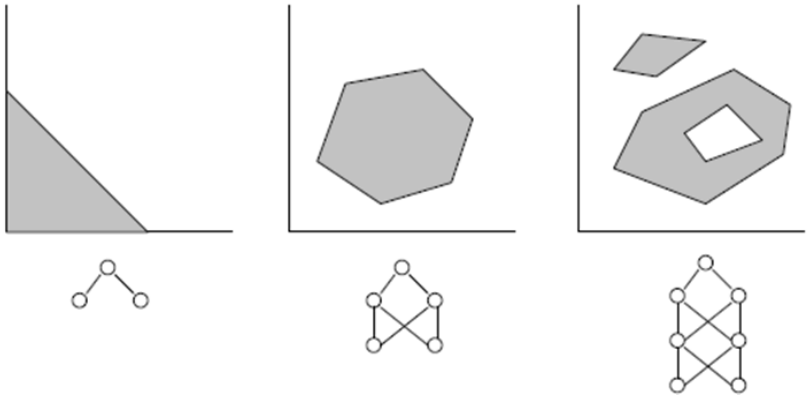
The Perceptron equation then becomes

$$Y_j = \text{sgn}(\sum_{i=1}^{n} I_i \cdot w_{ij} - \theta_j) = \text{sgn}(\sum_{i=0}^{n} I_i \cdot w_{ij})$$

So, we only have to compute the weights.

2.5 Recommendations for Designing NN:

A network with three layers (numLayers=3: one input, one hidden and one output) is enough for a vast majority of cases. According to the Cybenko Theorem (1989), a network with one hidden layer is capable of approximating any continuous, multivariate function to any desired degree of accuracy; a network with two hidden layers is capable of approximating any discontinuous, multivariate function:

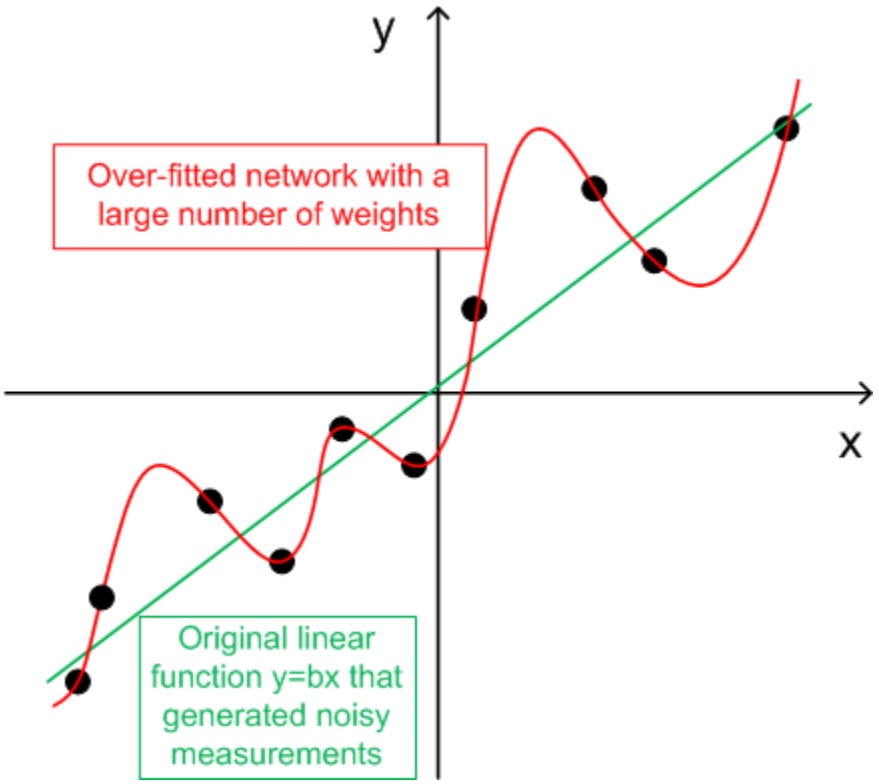


The optimum number of neurons in the hidden layer can be found through trial and error. The following "rules of thumb" can be found in the literature: # of hidden neurons = (# of inputs + # of outputs)/2, or SQRT(# of inputs * # of outputs). Keep track of the training error, reported by the indicator in the experts window of metatrader.

For generalization, the number of training sets (ntr) should be chosen 2-5 times the total number of the weights in the network. For example, by default, BPNN Predictor.mq4 uses a 12-5-1 network. The total number of weights is (12+1)*5+6=71. Therefore, the number of training sets (ntr) should be at least 142. The concept of *generalization* and*memorization* (over-fitting) is explained on the graph below.

The input data to the network should be transformed to stationary. Forex prices are not stationary. It is also recommended to normalize the inputs to -1..+1 range.

The graph below shows a linear function y=b*x (x-input, y-output) whose outputs are corrupted by noise. This added noise causes the function measured outputs (black dots) to deviate from a straight line. Function y=f(x) can be modeled by a feed forward neural network. The network with a large number of weights can be fitted to the measured data with zero error. Its behavior is shown as the red curve passing through all black dots. However, this red curve has nothing to do with the original linear function y=b*x (green). When this over-fitted network is used to predict future values of function y(x), it will result in large errors due to randomness of the added noise.



2.5.1 History of Perceptrons

Although much of the artificial intelligence (AI) community focused on symbolic reasoning, a few researchers around the mid-twentieth century were investigating parallel distributed computing, and notably models inspired by the nervous activity in biological brains. In 1943, McCulloch and Pitts started experimenting with simple simulations of the nervous system [McCulloch43].

2.5.2 Rosenblatt's Perceptron

It wasn't until 1959 that a neural system caught the attention of the AI community. This was Rosenblatt's perceptron [Rosenblatt59], modeling the human visual system—hence the name.

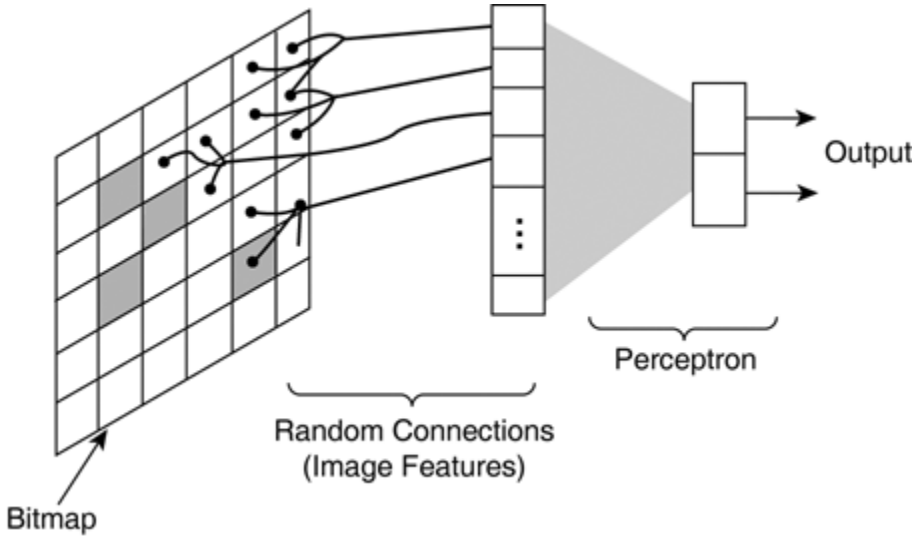
"In 1957 **Frank Rosenblatt** of the Cornell Aeronautical Laboratory at Cornell University in Ithaca, New York, began investigating artificial neural networks that he called **perceptrons**. He made major contributions to the field of AI, both through experimental investigations of the properties of neural networks (using computer simulations) and through detailed mathematical analysis. Rosenblatt was a charismatic communicator, and there were soon many research groups in the United States studying perceptrons. Rosenblatt and his followers called their approach connectionist to emphasize the

importance in learning of the creation and modification of connections between neurons. Modern researchers have adopted this term" (Britannica). Rosenblatt's perceptron was able to recognize patterns of similarity between new data and data it has already seen in a feed-forward model that demonstrated a primitive type of learning or trainability. His work was highly influential in the development of later multi-layered neural networks.

One of the earliest and most influential neural networks: Frank Rosenblatt's introduction of the perceptron, an important milestone in the field of artificial intelligence.

The perceptron is capable of extracting visual patterns from images (a popular problem at the time). From the bitmap, random weighted connections provide a set of features to the actual perceptron. In turn, these features are connected with weights to the output, which provide interpretations of the image. By training the perceptron on a collection of sample bitmaps with their corresponding outputs, the system could learn to classify the images (see Figure 17.1).

Figure 17.1. Rosenblatt's perceptron connected to a bitmap image, capable of recognizing some of its features.

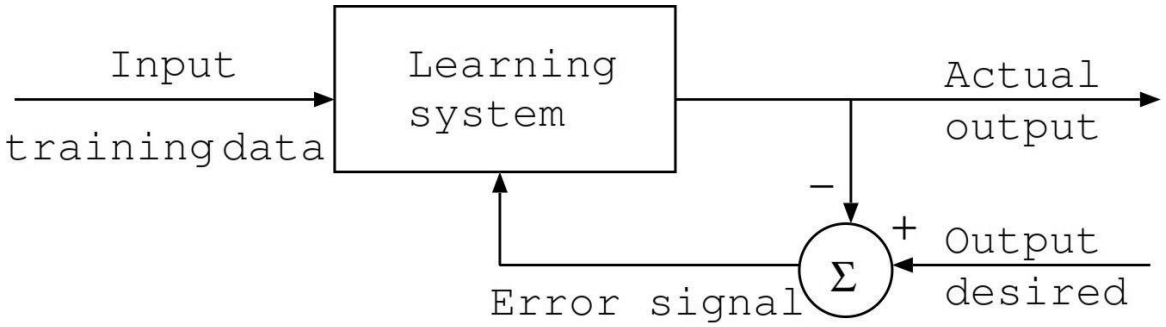


For the AI community, one of the most interesting aspects of this perceptron was the proof of convergence established by Rosenblatt. This proved that if a possible solution for the perceptron existed, the training algorithm would find it. This generated quite a bit of interest!

2.6 Learning

2.6.1 Supervised Learning —

Essentially, a strategy that involves a teacher that is smarter than the network itself. For example, let's take the facial recognition example. The teacher shows the network a bunch of faces, and the teacher already knows the name associated with each face. The network makes its guesses, then the teacher provides the network with the answers. The network can then compare its answers to the known "correct" ones and make adjustments according to its errors. Our first neural network in the next section will follow this model.



- Used in examples like Recognizing hand-written digits, pattern recognition, regression.
- Labeled examples are provided ie. (input , desired output)
- Used in Neural Network models: perceptron, feed-forward, radial basis function, support vector machine.

2.6.2 Unsupervised Learning —

Required when there isn't an example data set with known answers. Imagine searching for a hidden pattern in a data set. An application of this is clustering, i.e. dividing a set of elements into groups according to some unknown pattern.

- Find similar groups of documents in the web, content addressable memory, clustering.
- Unlabeled examples (different realizations of the input alone)
- Neural Network models: self organizing maps, Hopfield networks.

2.6.3 Reinforcement Learning —

A strategy built on observation. Think of a little mouse running through a maze. If it turns left, it gets a piece of cheese; if it turns right, it receives a little shock. (Don't worry, this is just a pretend mouse.) Presumably, the mouse will learn over time to turn left. Its neural network makes a decision with an outcome (turn left or right) and observes its environment (yum or ouch). If the observation is negative, the network can adjust its weights in order to make a different decision the next time. Reinforcement learning is common in robotics. At time ***t***, the robot performs a task and observes the results. Did it crash into a wall or fall off a table? Or is it unharmed?

This ability of a neural network to learn, to make adjustments to its structure over time, is what makes it so useful in the field of artificial intelligence.

Here are some standard uses of neural networks in software today.

- ***Pattern Recognition*** —We've mentioned this several times already and it's probably the most common application. Examples are facial recognition, optical character recognition, etc.
- ***Time Series Prediction*** —Neural networks can be used to make predictions. Will the

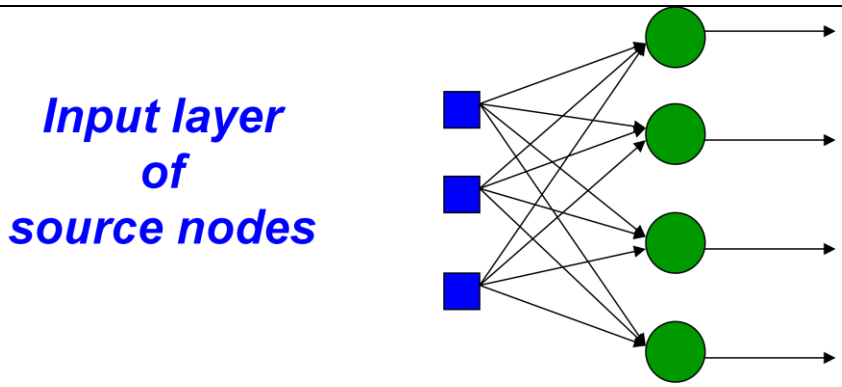
stock rise or fall tomorrow? Will it rain or be sunny?

- ***Signal Processing*** —Cochlear implants and hearing aids need to filter out unnecessary noise and amplify the important sounds. Neural networks can be trained to process an audio signal and filter it appropriately.
- ***Control*** —You may have read about recent research advances in self-driving cars. Neural networks are often used to manage steering decisions of physical vehicles (or simulated ones).
- ***Soft Sensors*** —A soft sensor refers to the process of analyzing a collection of many measurements. A thermometer can tell you the temperature of the air, but what if you also knew the humidity, barometric pressure, dewpoint, air quality, air density, etc.? Neural networks can be employed to process the input data from many individual sensors and evaluate them as a whole.
- ***Anomaly Detection*** —Because neural networks are so good at recognizing patterns, they can also be trained to generate an output when something occurs that doesn't fit the pattern. Think of a neural network monitoring your daily routine over a long period of time. After learning the patterns of your behavior, it could alert you when something is amiss.

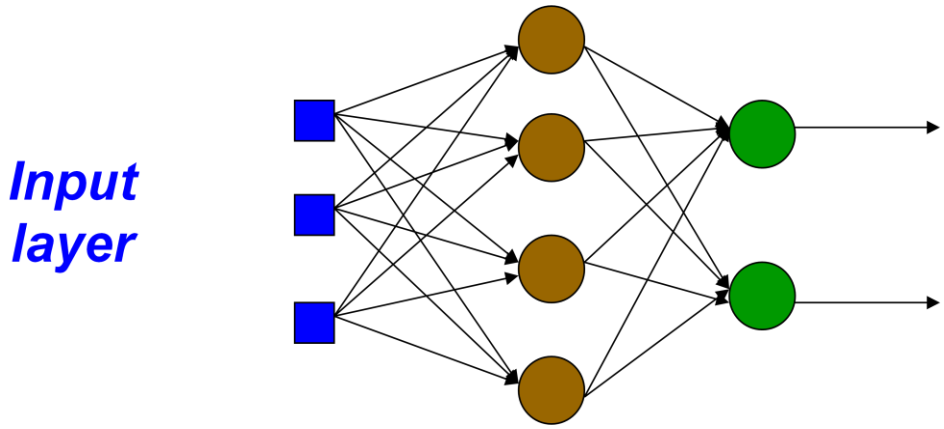
RECAP

Network architectures

Single Layer Feed-forward

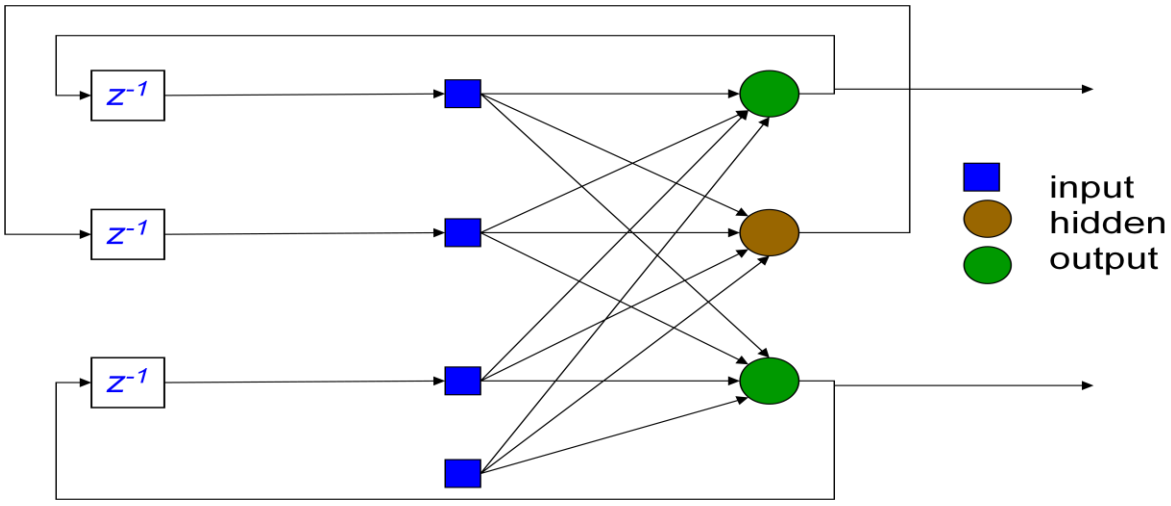


Multi layer feed-forward



Recurrent network

Recurrent Network with ***hidden neuron(s)***: unit delay operator z^{-1} implies dynamic system



Supervised Learning : An example application 1

- An emergency room in a hospital measures 17 variables (e.g., blood pressure, age, etc) of newly admitted patients.
- A decision is needed: whether to put a new patient in an intensive-care unit.
- Due to the high cost of ICU, those patients who may survive less than a month are given higher priority.
- Problem: to predict high-risk patients and discriminate them from low-risk patients.

Supervised Learning : An example application 2

- A credit card company receives thousands of applications for new cards. Each application contains information about an applicant,

- age
- Marital status
- annual salary
- outstanding debts
- credit rating
- etc.

Problem: to decide whether an application should approved, or to classify applications into two categories, approved and not approved.

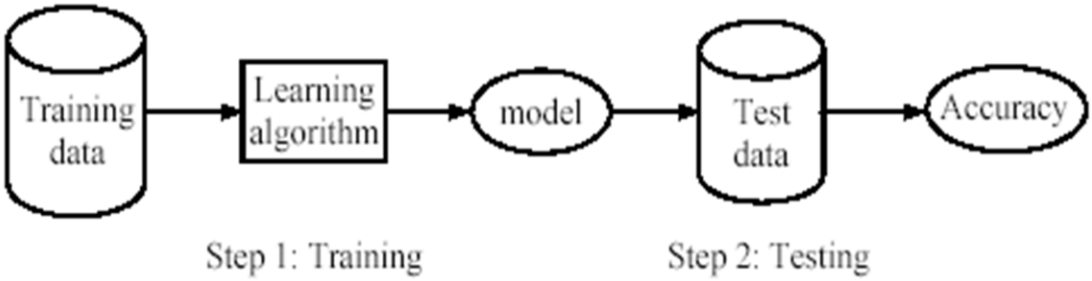
Summary: Supervised learning

- Like human learning from past experiences; A computer does not have "experiences".
- A computer system learns from data, which represent some "past experiences" of an application domain.
- Our focus: learn a target function that can be used to predict the values of a discrete class attribute, e.g., approve or not-approved, and high-risk or low risk.
- The task is commonly called: Supervised learning, classification, or inductive learning.

Supervised learning process: two steps

- Learning (training): Learn a model using the training data
- Testing: Test the model using unseen test data to assess the model accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}},$$



Step 1: Training Step 2: Testing

An example

- Data: Loan application data
- Task: Predict whether a loan should be approved or not.
- Performance measure: accuracy.

No learning: classify all future applications (test data) to the majority class (i.e., Yes):

Accuracy = 9/15 = 60%.

- We can do better than 60% with learning.

Fundamental assumption of learning

Assumption: The distribution of training examples is identical to the distribution of test examples (including future unseen examples).

- In practice, this assumption is often violated to certain degree.
- Strong violations will clearly result in poor classification accuracy.
- To achieve good accuracy on the test data, training examples must be sufficiently representative of the test data.

Useful Links:

http://www.intechopen.com/books/artificial-neural-networks-architectures-and-applications/applying-artificial-neural-network-hadron-hadron-collisions-at-lhc

http://www8.cs.umu.se/~peklund/utbildning/IntelligentComputing/Kompendium.pdf

Practical example: http://msdn.microsoft.com/en-us/magazine/hh975375.aspx

Neural Network Training Using Back-Propagation:
http://visualstudiomagazine.com/articles/2013/09/01/neural-network-training-using-back-propagation.aspx

Semi-supervised Learning of Feature Hierarchies for Object Detection in a Video:
http://www.cs.ucf.edu/~yyang/CVPR13/DAE.php