

KHWARZIMIC SCIENCE SOCIETY

BRAIN-LIKE COMPUTING

A TUTORIAL ON NEURAL NETS

PRESENTER: MUHAMMAD ABUBAKAR

PLACE: SEMINAR ROOM, THE CENTRE FOR SOLID STATE PHYSICS,
THE PUNJAB UNIVERSITY

DATE: JULY 14, 1997

This was a seminar and demonstration on the principles, algorithms and applications of the neural networks which are a branch of electronics, electrical engineering and computer sciences. Muhammad AbuBakar is himself a student of the electrical engineering department at the University of Engineering and Technology, Lahore. His work on neural networks primarily focuses on image and pattern recognition.

OUTLINE:

[What are Neural Networks.](#) ~ [The need for Neural Networks.](#) ~ [Applications of Neural Networks.](#)

[Modelling artificial Neural Nets.](#) ~ [The Concept of Training.](#) ~ [Some basic algorithms of training Neural Nets.](#)

[How to make artificial Neurons ... The engineer's problem.](#) ~ [Simulating Neural Nets on Digital Computers.](#)

Demonstration of a [C++ program](#) by Muhammad AbuBakar, that remembers and recognises visual patterns.

Introduction to Neural Nets.

Neural Nets is a fast growing science, that is being studied by not only computer and electrical engineers, but also by a large community of scientists from disciplines such as psychology, physics, mathematics, medical sciences, sociology etc. It is this interdisciplinary flavour, that has made this subject very exciting. The field of neural computing, as its name suggests is a way of computing, that nature has gifted to the living things. However this should be kept in mind that the aim of neural nets, (specially in our case) is not to copy biological computing. You can say that the Computer Scientists have come up with a wonderful way of making machines "think" which imitates the way "we" think. *They are, simply, mathematical models of information processing.*

The need for neural networks.

Aside from modelling biological systems, the real prospect for using neural nets lies in doing tasks where traditional computers fail miserably. The computer is an excellent number cruncher. You give it a list of 1000 numbers. It will add them up in less than a second. You give this task to a skilled mathematician. He may be counting them when you end reading this tutorial. So number crunching is perhaps not in the domain of neural computing. But computers are absolutely dumb at doing

tasks such as remembering faces. You would instantly remember your friend even if he has shaved his head or you are seeing him upside down. But a computer requires exact information. If the picture of the person presented to it differs slightly from the picture it "remembers" it will fail to recognise him. This is where neural computing rules: Answering right when all is vague and unclear. The most exciting thing about neural nets is that they learn, they recall and they forget, the way we do.

Applications of Neural Networks:

Neural Nets are being used in:

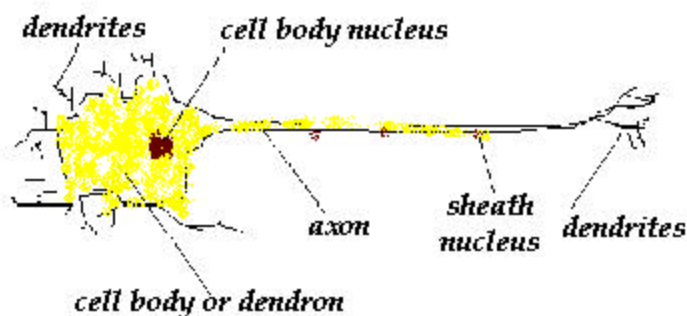
- Pattern Recognition.
- Speech recognition and production.
- Medical Diagnosis
- Business applications such as mortgage assessment, evaluation of risk of default on loans Control Systems.
- Signal Processing such as echo suppression, noise reduction, filters and Artificial Vision and speech units for the handicapped *to mention just a few*

Modelling Artificial Neural Nets:

The Biological Neuron:

We remember from our school-level biology lessons, the basic structure of a neuron, which is given below.

From the figure, we see that neuron has a head like structure at the top called Soma which has many dendrites and a long Axon. The axon is connected to the dendrites of other neurons. Similarly its dendrites are connected to axons of other neighbouring neurons. The neuron receives information in the form of electric currents from other neurons on the dendrites. The information is "processed" and the neuron "fires" to pass its result in the form of current, to other neuron through its axon.

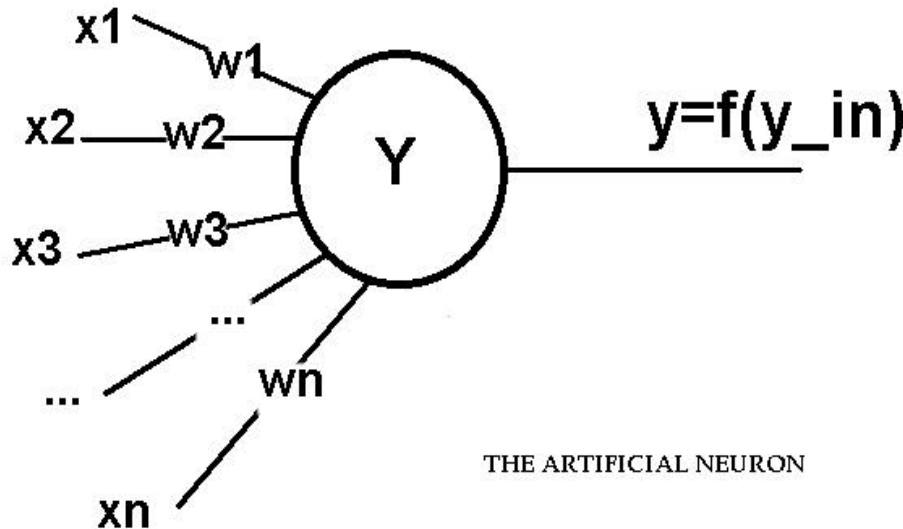


THE BIOLOGICAL NEURON

The Artificial Neuron:

The artificial neuron has almost the same structure. The processing element,

denoted by Y_j (i.e. the j -th neuron of a system) has a lot of inputs indicated by $x_1, x_2, x_3, \dots, x_n$. These inputs can be binary or bipolar or continuous. Binary means they can be 1 or 0 to indicate presence or absence of signal. Bipolar inputs are 1 or -1. In neural nets bipolar inputs are found to be more useful than binary inputs. The inputs are also accompanied by "weights" indicated by $w_1, w_2, w_3, \dots, w_n$. They can be assigned any value. Weights represent the strength of connection between two processing elements. The net input to the neuron is given by:



The output of the neuron is some function of y_{in} . The simple words, the neuron will collect data from other neurons through inputs, calculate the total input, and then would decide to "fire" or not, based on some rule indicated here by $f(y_{in})$. A simple example of such a function is given below.

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

i.e the neuron fires if the net input is greater than zero. Other firing rules are also used and are selected by considering the nature of our problem. The firing rule is called "**Activation function**".

The design of neural nets involves an algorithm for selecting the weights and activation functions, and the way to connect different neurons, in order to achieve a task.

Example:

A three input neuron has the weights [1 -2 2], and the activation function given above. The inputs are given as bipolar (1,-1). The different outputs of the neuron are given below:

Weights: (Predefined)

$$w_1 = 1 \quad w_2 = -2 \quad w_3 = 2$$

x_1	x_2	x_3	$y_{in} = x_1 w_1 + x_2 w_2 + x_3 w_3$	Output = $f(y_{in})$
1	-1	1	$1(1) + (-1)(-2) + 1(2) = 5$	$f(5) = 1$ i.e neuron fires
1	1	-1	$1(1) + 1(-2) + (-1)(2) = -3$	$f(-3) = -1$ i.e. does not fire
1	1	1	$1(1) + 1(-2) + 1(2) = 1$	$f(1) = 1$ i.e. neuron fires

The concept of training:

Like in biological systems, the artificial neural systems are first trained for a particular task. A child does not know by itself the difference between an A or a B. The teacher trains the child by presenting him the alphabets. The more he is trained the quicker and better will be his response. Likewise the neural net is first "trained". It is given many training inputs and is instructed about the desired output. e.g. A neural net designed to recognize a letter A from other alphabets will be presented A, and told that this pattern is A. Then it would be given B and instructed that this is not A. Similarly A and other letters will be presented in different fonts and sizes. The more it is trained the better will be its output. The incredible thing about these nets is that if an input is not what it is trained to recognise, but is somewhat close to the training inputs, it still recognises the input. This is where traditional computers are a failure.

An Algorithm of training: The Hebb's Net:

The Hebb's Network named after its designer is one the simplest neural network around. We will apply the Hebb's Learning algorithm to a single neuron net. Keeping the rest of the notation as above, t is the "desired" target for an input pattern. (1 = desired, -1 = not desired)

The algorithm is given below:

Step 1: Initialize all weights to zero:

$$w_i = 0 \quad i = 1, 2, 3, \dots, n$$

Step 2: For Each training input

Set inputs x_i

Step 3: Set activation for desired output

$$y = t$$

Step 4: Adjust weights as:

$$w_i(\text{new}) = w_i(\text{old}) + x_i y$$

In other words,

$$\Delta w_i = x_i y$$

The basic idea behind this algorithm is that the weights are considered as the strength of connection between two units. If an input belongs to an input pattern, that is desired to fire the neuron, then the weight corresponding to the input is strengthened. If not it is weakened.

Example of Hebb's Training: The Logical AND Function.

We shall now design a net that has a response like a conventional AND function. The truth table is given below.

x_1	x_2	$x_1 \text{ AND } x_2$
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

We have $i = 2$

Initialising weights to zero.

$$w_1 = 0$$

$$w_2 = 0$$

For training input 1,

$$x_1 = -1$$

$$x_2 = -1$$

$$t = -1$$

$$\Delta w_1 = x_1 t = (-1)(-1) = 1$$

$$\Delta w_2 = x_2 t = (-1)(-1) = 1$$

New weights,

$$w_1 = 0 + \Delta w_1 = 1$$

$$w_2 = 0 + \Delta w_2 = 1$$

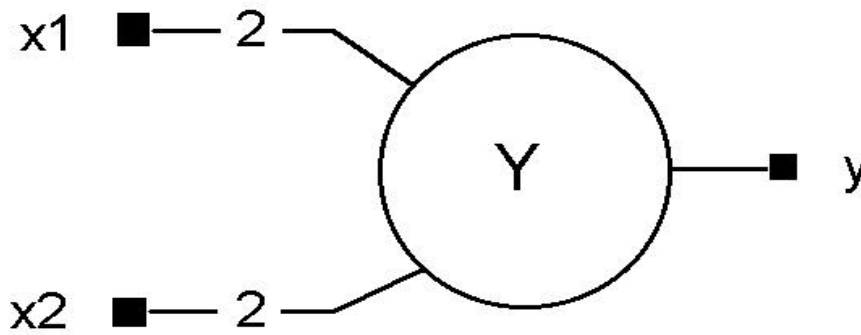
Similarly other three input patterns are presented. The details are given in the following table.

INPUTS		TARGET	WEIGHT CHANGE		WEIGHTS	
x_1	x_2	t	Dw_1	Dw_2	w_1	w_2
					0	0
-1	-1	-1	1	1	1	1
-1	1	-1	1	-1	2	0
1	-1	-1	-1	1	1	1
1	1	1	1	1	2	2

Finally we have $w_1 = 2, w_2 = 2$.

We now chose an appropriate activation function. I propose,

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



Neural Net of AND function

Let us test this net.

x_1	x_2	$y_{in} = x_1w_1 + x_2w_2$	Output = $f(y_{in})$
1	1	$1(2) + (1)(2) = 4$	$f(4) = 1$ Correct
1	-1	$1(2) + (-1)(2) = 0$	$f(0) = -1$ Correct
-1	1	$(-1)(2) + 1(2) = 0$	$f(0) = -1$ Correct
-1	-1	$(-1)(2) + (-1)(2) = -4$	$f(-4) = -1$ Correct

It can be seen that the net has produced excellent results.

Similarly using Hebb's Algorithm, we can have nets that do a variety of tasks.

Making Artificial Neurons:

(This section may be skipped on first reading)

On paper, Artificial neural networks look amazingly simple and easy but the task of making artificial neurons is not an easy problem. The task for an engineer is to make a device that has many inputs. Each input is associated with a weight which should be programmable i.e. whose value can be changed. Then there is the problem of connecting together neurons. Practical Nets have dozens or even hundreds of neurons each having many inputs, each connected to many. In this section we address the problem of the hardware implementation of neurons.

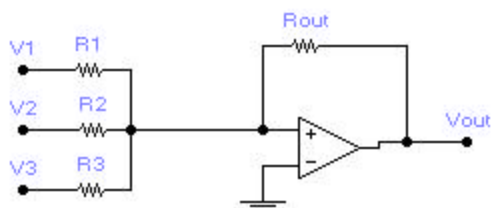
(a). Operational Amplifiers:

The simplest way to make a neuron is to use the operational amplifier which fortunately is a perfect summing device. Consider the Circuit shown in figure. We have the output voltage as:

$$V_{out} = -R_{out} \left(\frac{V_1}{R_1} + \frac{V_2}{R_2} + \frac{V_3}{R_3} \right)$$

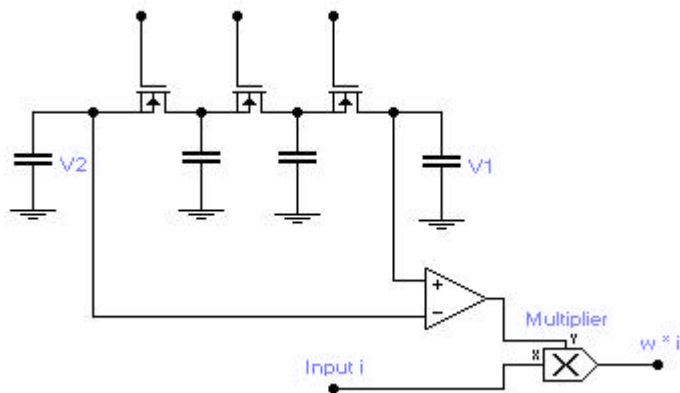
If we replace $1/R$ by the conductivity w and take $R_{out} = \text{unity}$

$$V_{out} = V_1 w_1 + V_2 w_2 + V_3 w_3 + \dots$$



Thus if we have a variable resistance it can act as a weight. The conductivity of this resistance is the value of the weight. Similarly the activation level of the neuron is adjustable by changing the reference voltage. The real problem is to make the programmable variable resistance. Some early researchers used actual variable resistors and coupled their knobs to servo-motors so that the resistance can be changed electrically by driving the motor. This is a rather absurd approach. The neurons will be extremely bulky and complex to control.

Another approach is shown in the figure which is the Bell Lab's Capacitive weight. The amplifier senses the difference of voltage on both capacitors. The MOSFETs transfer charge between the two capacitors. The rest of the operation is self explanatory.



(b). Binary Neural Chips: The Digital Approach.

Although analog storage has a lot of promise, as shown above, but the cheap VLSI technology now available is a close competitor. Most Neural Nets implemented digitally consist of registers instead of weights that are processed by special purpose microprocessors or multiplier units. The main hurdle in this kind of technology is making the enormous connections required between neurons, due to limitation of space. However this approach is the most promising due to cheap VLSI technology.

(c). Optical and holographic Devices:

The problem of limitation of space can be overcome by using optical devices where signals can cross over. Neural systems based on photo-diode arrays have been implemented.

Simulating Neural Nets on Computer: The easiest way.

Neural Networks can be simulated on traditional computers. Software implementation of Neural Nets is practical where speed is not the main concern. In actual Neural Nets the computing is done in parallel. No sequential steps are taken. The decision is made in a collective manner. By implementing neural nets on sequential machines we take away the essence of Neural Computing but it is the most economical and easiest approach to making Neural Nets. A C++ implementation of Hebb's Algorithm is given in [this link](#).

[Home](#) |

The Khwarzimid Science Society

Centre of Excellence in Solid State Physics
University of the Punjab | Quaid-e-Azam Campus
Lahore 54590 | PAKISTAN

Tel: (0) 92 - 42 - 5864534, (0) 92 - 42 - 7565474 | Fax: (0) 92 - 42 - 5864534

info@khwarzimid.org

Since 23 Jan 1997
you are Visitor No:

011320