YAPAY SİNİR AĞLARINA GİRİŞ

Doç. Dr. Sırma Yavuz

D 110

Çarşamba 9:00 – 11:50

Ders Planı

1	17.Şub.16	Introduction to Neural Networks and their History
2	24.Şub.16	Networks of Artificial Neurons, Single Layer Perceptrons
3	02.Mar.16	No Class
4	09.Mar.16	Hebbian Learning, Gradient Descent Learning, Delta Rule
5	16.Mar.16	Learning in Multi-Layer Perceptrons - Back-Propagation
6	23.Mar.16	Bias and Variance - Under-Fitting and Over-Fitting
7	30.Mar.16	Applications of Multi-Layer Perceptrons - Lab Session
8	06.Nis.16	Midterm week 1
9	13.Nis.16	Recurrent Neural Networks
10	20.Nis.16	Radial Basis Function Networks: Algorithms
11	27.Nis.16	Self Organizing Maps
12	04.May.16	Learning Vector Quantization
13	11.May.16	Midterm week 2
14	18.May.16	Model Selection and Evolutionary Optimization
15	25.May.16	Exercise session

Değerlendirme

- 1 adet ara sınav (%30)
- Haftalık quizler (%10)
- 1 hafta Matlab ile Laboratuarda Uygulama (%10)
- 1 yıliçi programlama ödevi : Verilen data set üzerinde Matlab ile farklı algoritmaların denenmesi - Elde edilen sonuçların kıyaslanması ve “bilimsel” olarak yorumlanması (%10)
- Final Sınavı (%40)

Kitaplar

- Fundamentals of Neural Networks: Architectures, Algorithms And Applications - Laurene V. Fausett

- Neural Network Design (2nd Edition) –
  Martin T Hagan

What is Neural Computation ?

1. ***Neural Computation*** is a general ***Machine Learning*** approach that involves processing information in a similar manner to the networks of neurons (i.e. ***Neural Networks***) found in human/animal brains.

2. ***Artificial Neurons*** are crude approximations of the neurons found in biological brains. They may be physical devices, or purely mathematical constructs.

3. ***Artificial Neural Networks*** (ANNs) are networks of Artificial Neurons, and hence constitute crude approximations to parts of real brains. They may be physical devices, or simulated on conventional computers.

4. From a practical point of view, any 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.

5. One should never lose sight of how crude the approximations are, and how over-simplified ANNs are compared to the neural networks in real brains.

Why are Artificial Neural Networks worth studying?

1. Even though individual artificial neurons are very simple, networks of them can be shown to be extremely powerful computational devices (Turing equivalent, universal computers).

2. Very simple ANNs can be set up to learn and generalize well – so they can perform difficult tasks without the need for enormous feats of programming.

3. Their massive parallelism can make them very efficient.

4. They are particularly fault tolerant – this is equivalent to the "graceful degradation" found in biological brains.

5. They are very noise tolerant – so they can cope with situations where normal symbolic (rule-based) systems would have difficulty.

6. In principle, they can do anything a symbolic/logic system can do, and a lot more. Though, in practice, getting them to do it can be rather difficult…

What are Artificial Neural Networks used for?

As with the field of AI in general, there are two basic goals for neural network research:

Brain modelling : The *scientific* goal of building models of how real brains work. This can potentially help us understand the nature of human intelligence, formulate better teaching strategies, or better remedial actions for brain damaged patients.

Artificial System Building : The *engineering* goal of building efficient systems for real world applications. This may make machines more powerful, relieve humans of tedious tasks, and may even improve upon human performance.

These should not be thought of as competing goals. We often use exactly the same neural networks and techniques for both. Frequently progress is made when the two approaches are allowed to feed into each other. There are fundamental differences though, e.g. the need for biological plausibility in brain modelling, and the need for computational efficiency in artificial system building.

Learning in Neural Networks

There are many different types of neural networks. Most operate by passing neural "activations" through a network of connected neurons.

One of the most useful and powerful features of neural networks is their ability to ***learn*** and ***generalize*** from a set of training data. They adapt the strengths/weights of the connections between neurons so that their final output activations are optimized.

There are three broad types of learning:

1. Supervised Learning (i.e. learning with a teacher)
2. Reinforcement learning (i.e. learning with limited feedback)
3. Unsupervised learning (i.e. learning with no help)

This module will study in some detail the most common learning algorithms for the most common types of neural network.

A Brief History of the Field

1943	McCulloch and Pitts proposed the McCulloch-Pitts neuron model
1949	Hebb published his book *The Organization of Behavior*, in which the Hebbian learning rule was proposed.
1958	Rosenblatt introduced the simple single layer networks now called Perceptrons.
1969	Minsky and Papert's book *Perceptrons* demonstrated the limitation of single layer perceptrons, and almost the whole field went into hibernation.
1982	Hopfield published a series of papers on Hopfield networks.
1982	Kohonen developed the Self-Organising Maps that now bear his name.
1986	The Back-Propagation learning algorithm for Multi-Layer Perceptrons was re-discovered and the whole field took off again.
1990s	The sub-field of Radial Basis Function Networks was developed.
2000s	The power of Neural Network Ensembles, Support Vector Machines, Bayesian Techniques, Simulated Evolution, and Deep Learning becomes apparent.

ANNs compared with Classical Symbolic AI

The distinctions can be categorized under three broad headings:

1. Level of Explanation
2. Processing Style
3. Representational Structure

These lead to a traditional set of dichotomies:

1. Sub-symbolic vs. Symbolic
2. Non-modular vs. Modular
3. Distributed representation vs. Localist representation
4. Bottom up vs. Top Down
5. Parallel processing vs. Sequential processing

In practice, however, the distinctions are becoming increasingly blurred.

Some Current Artificial Neural Network Applications

Brain modelling

Models of human development – to help children with developmental problems

Simulations of adult performance – aiding our understanding of how the brain works

Neuropsychological models – suggesting remedial actions for brain damaged patients

Artificial Life simulations – clarifying brain evolution, structure, growth, etc.

Real world applications

Pattern recognition – speech recognition, hand-writing recognition, sonar signals

Data analysis – data compression, data mining, PCA, GTM

Noise reduction – function approximation, ECG noise reduction

Control systems – autonomous adaptable robots, microwave controllers

Computer games – intelligent agents, backgammon, first person shooters

Financial modelling – predicting stocks, shares, currency exchange rates

Other time series prediction – climate, weather, airline marketing tactician

Overview and Reading

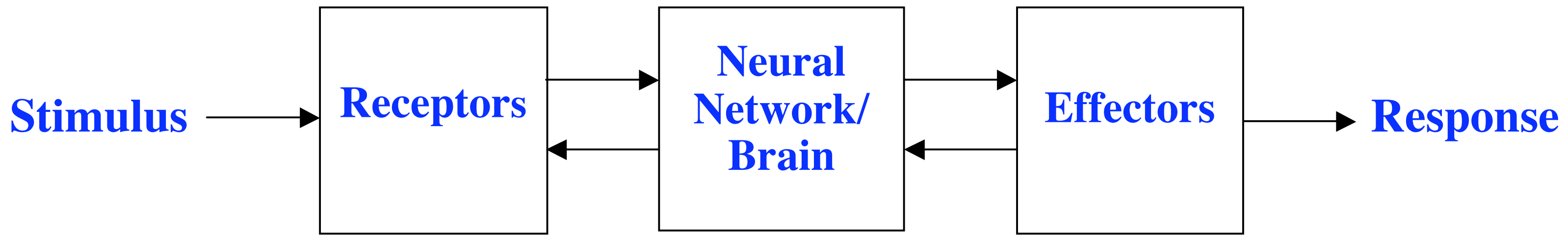
1. Artificial Neural Networks are extremely powerful (Turing equivalent) computational systems consisting of many simple processing elements connected together to perform tasks analogously to biological brains.
2. They are massively parallel, which makes them efficient, robust, fault tolerant and noise tolerant.
3. They can learn from training data and generalize to new situations.
4. They are useful for brain modelling and real world applications involving pattern recognition, function approximation, prediction, …

Reading

1. Haykin-1999: Sections 1.1, 1.8, 1.9
2. Gurney: Sections 1.1, 1.2, 1.3
3. Beale & Jackson: Sections 1.1, 1.2, 1.3, 1.4
4. Ham & Kostanic: Sections 1.1, 1.2

The Nervous System

The human nervous system can be broken down into three stages that may be represented in block diagram form as:



The receptors collect information from the environment – e.g. photons on the retina.

The effectors generate interactions with the environment – e.g. activate muscles.

The flow of information/activation is represented by arrows – feedforward and feedback.

Naturally, this module will be primarily concerned with how the neural network in the middle works, but understanding its inputs and outputs is also important.

Levels of Brain Organization

The brain contains both large scale and small scale anatomical structures and different functions take place at the higher and lower levels.

There is a hierarchy of interwoven levels of organization:

1. Molecules and Ions
2. Synapses
3. Neuronal microcircuits
4. Dendritic trees
5. **Neurons**
6. **Local circuits**
7. Inter-regional circuits
8. Central nervous system

The ANNs studied in this module are mostly approximations of levels 5 and 6.

Brains versus Computers : Some numbers

1. There are approximately 10 billion neurons in the human cortex, compared with tens of thousands of processors in the most powerful parallel computers.

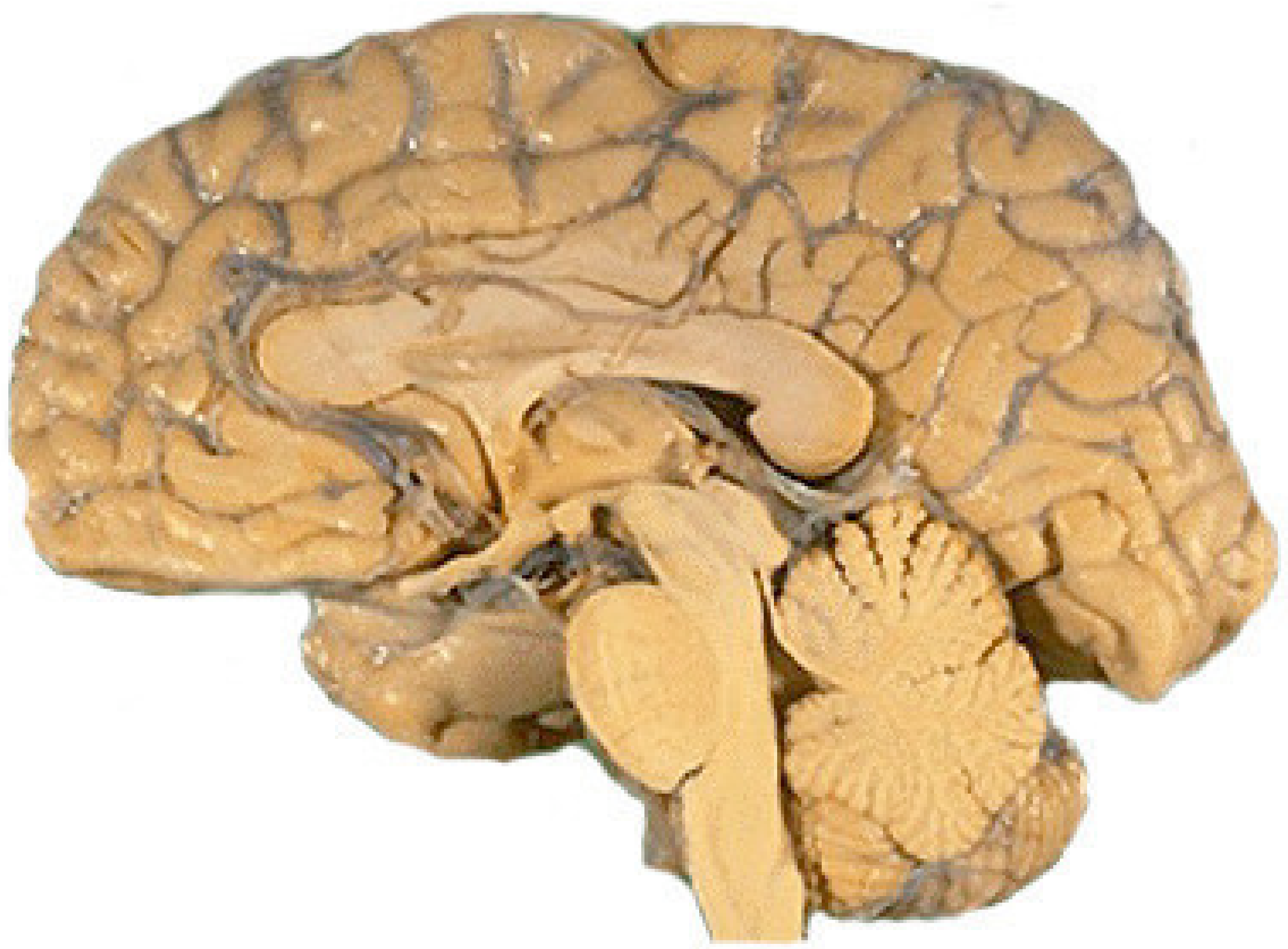
2. Lack of processing units can be compensated by speed. The typical operating speeds of biological neurons is measured in milliseconds (10^{-3} s), while current silicon chips can usually operate in nanoseconds (10^{-9} s).

3. Each biological neuron is connected to several thousands of other neurons, similar to the connectivity in powerful parallel computers.

4. The human brain is extremely energy efficient, using approximately 10^{-16} joules per operation per second, whereas the best computers today use around 10^{-6} joules per operation per second.

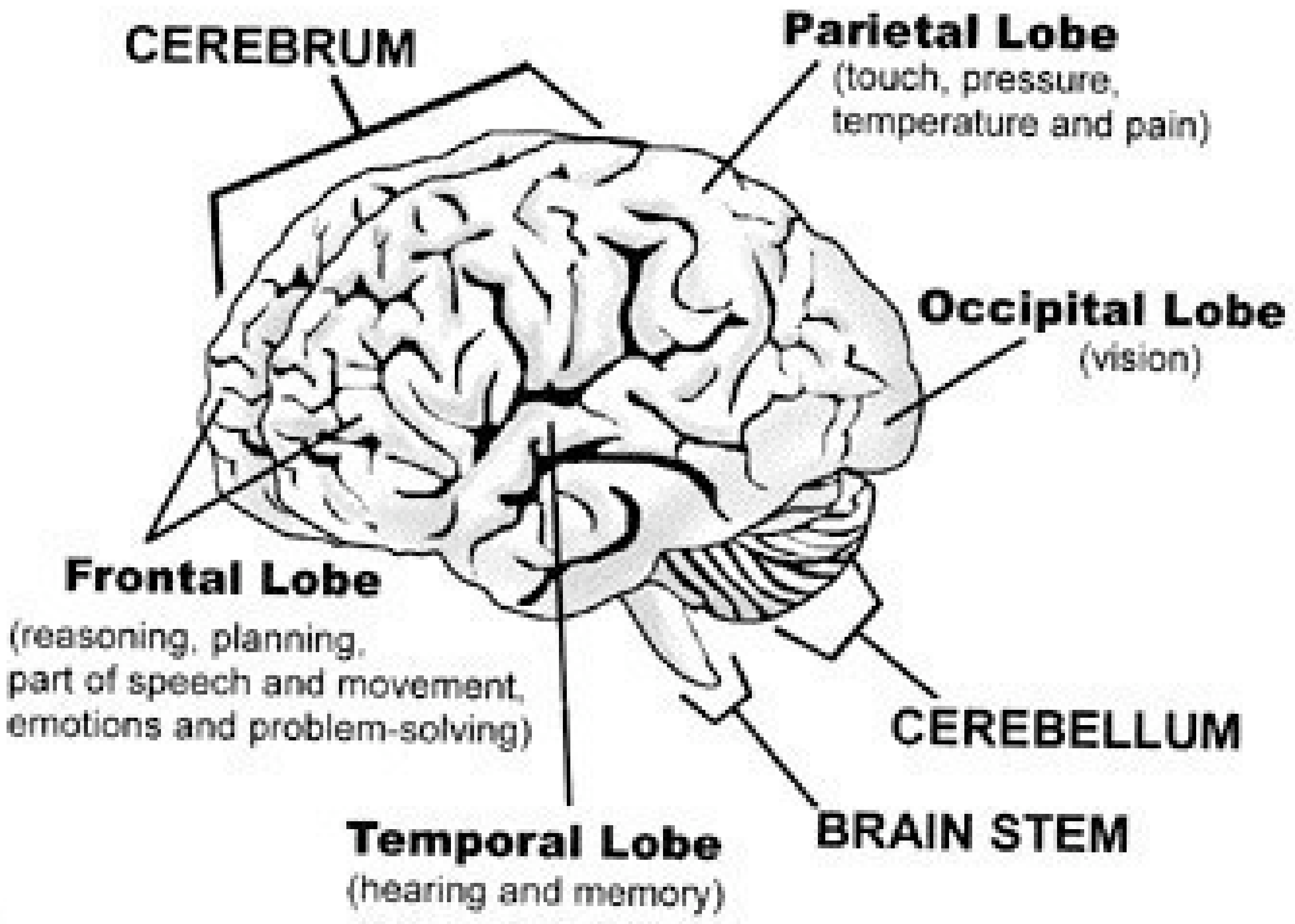
5. Brains have been evolving for tens of millions of years, but computers have only been evolving for tens of decades, though different mechanisms are involved.

Slice Through a Real Brain



http://library.med.utah.edu/WebPath/HISTHTML/NEURANAT/NEURANCA.html

Structure of a Human Brain


CEREBRUM
Parietal Lobe
(touch, pressure, temperature and pain)
Occipital Lobe
(vision)
Frontal Lobe
(reasoning, planning, part of speech and movement, emotions and problem-solving)
Temporal Lobe
(hearing and memory)
CEREBELLUM
BRAIN STEM

Basic Components of Biological Neurons

1. The majority of ***neurons*** encode their activations or outputs as a series of brief electrical pulses (i.e. spikes or action potentials).

2. The neuron's ***cell body (soma)*** processes the incoming activations and converts them into output activations.

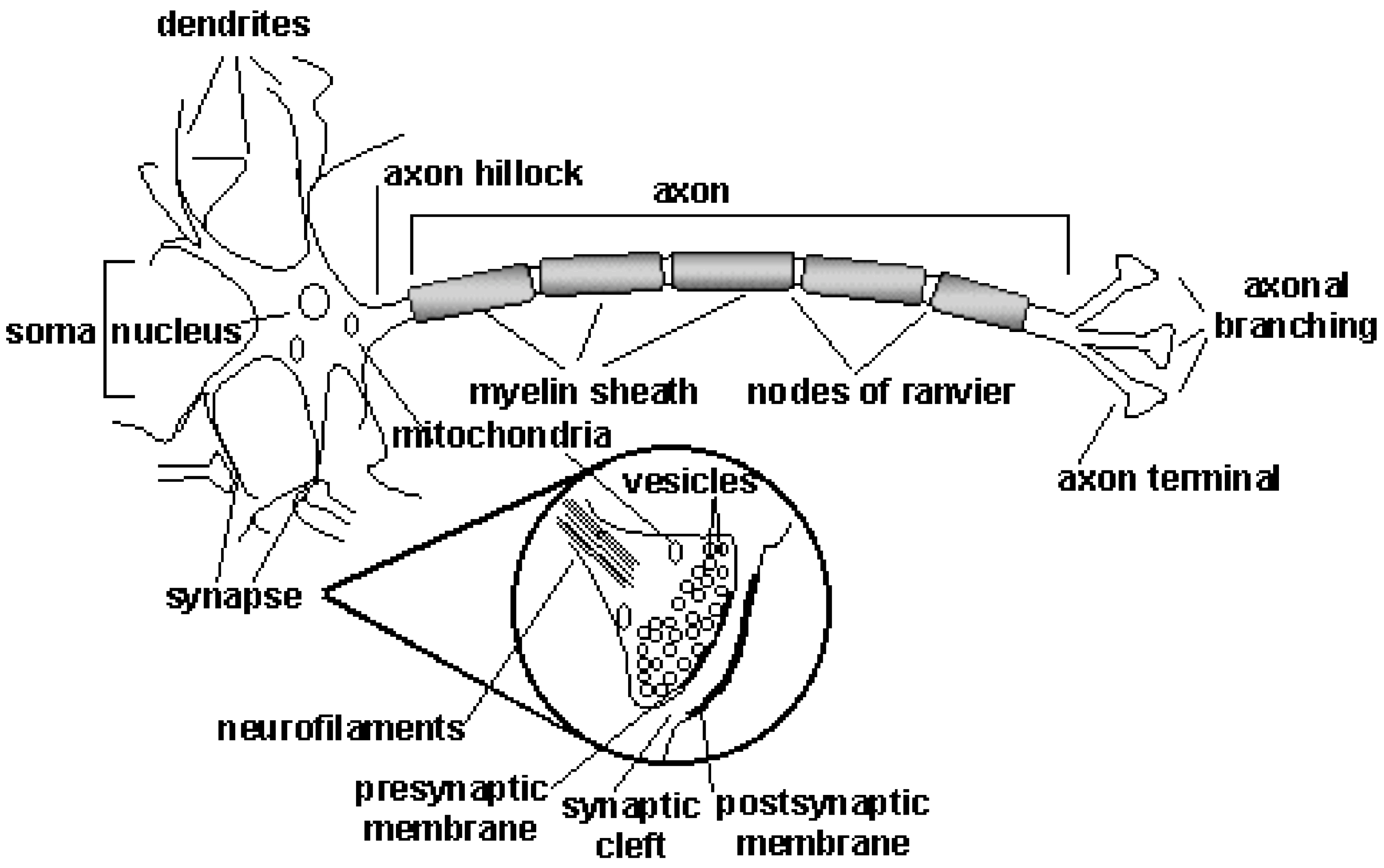
3. The neuron's ***nucleus*** contains the genetic material in the form of DNA. This exists in most types of cells, not just neurons.

4. ***Dendrites*** are fibres which emanate from the cell body and provide the receptive zones that receive activation from other neurons.

5. ***Axons*** are fibres acting as transmission lines that send activation to other neurons.

6. The junctions that allow signal transmission between the axons and dendrites are called ***synapses***. The process of transmission is by diffusion of chemicals called ***neurotransmitters*** across the synaptic cleft.

Schematic Diagram of a Biological Neuron

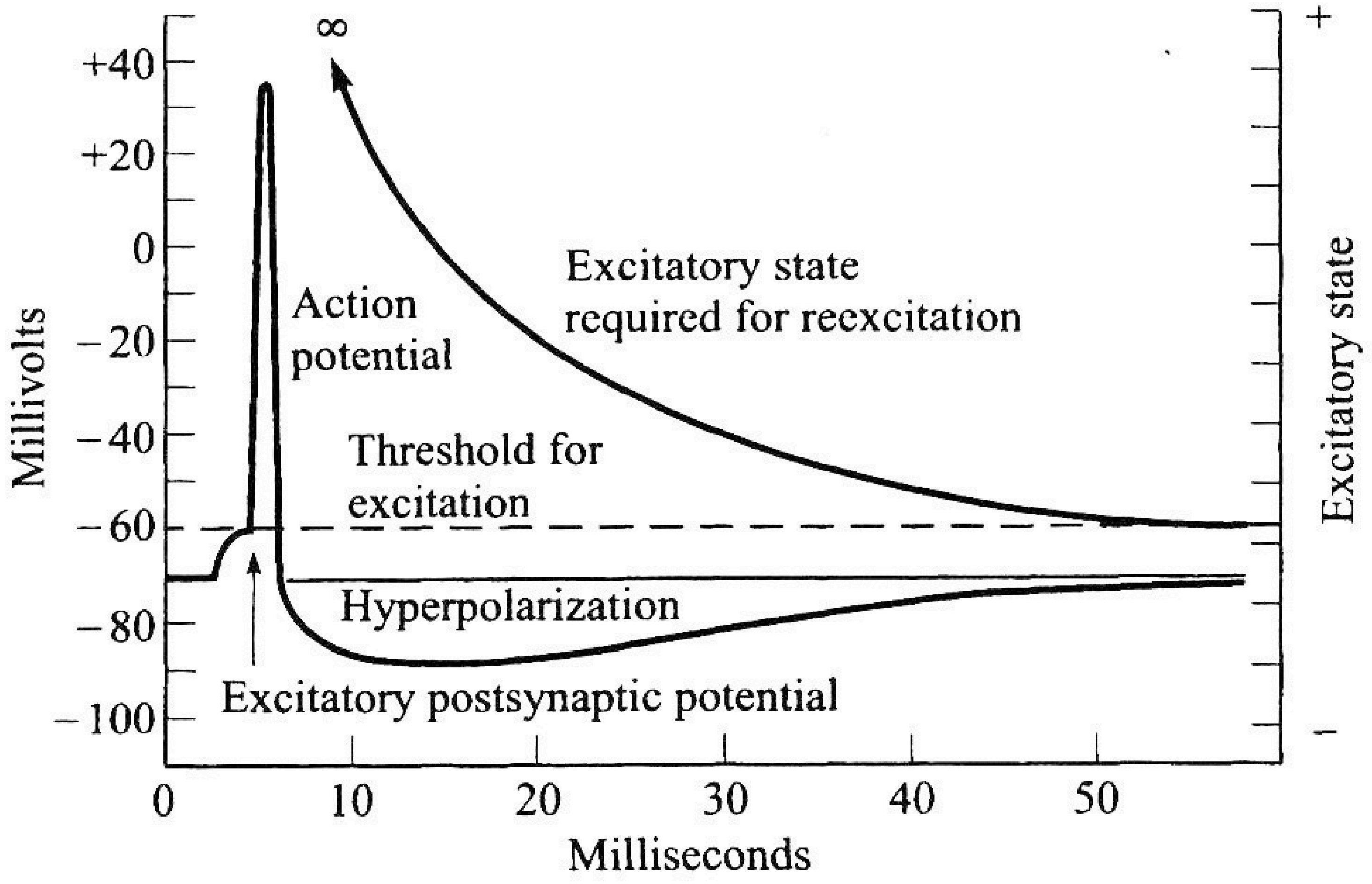


Neural Signal Processing

The key components of neural signal processing are:

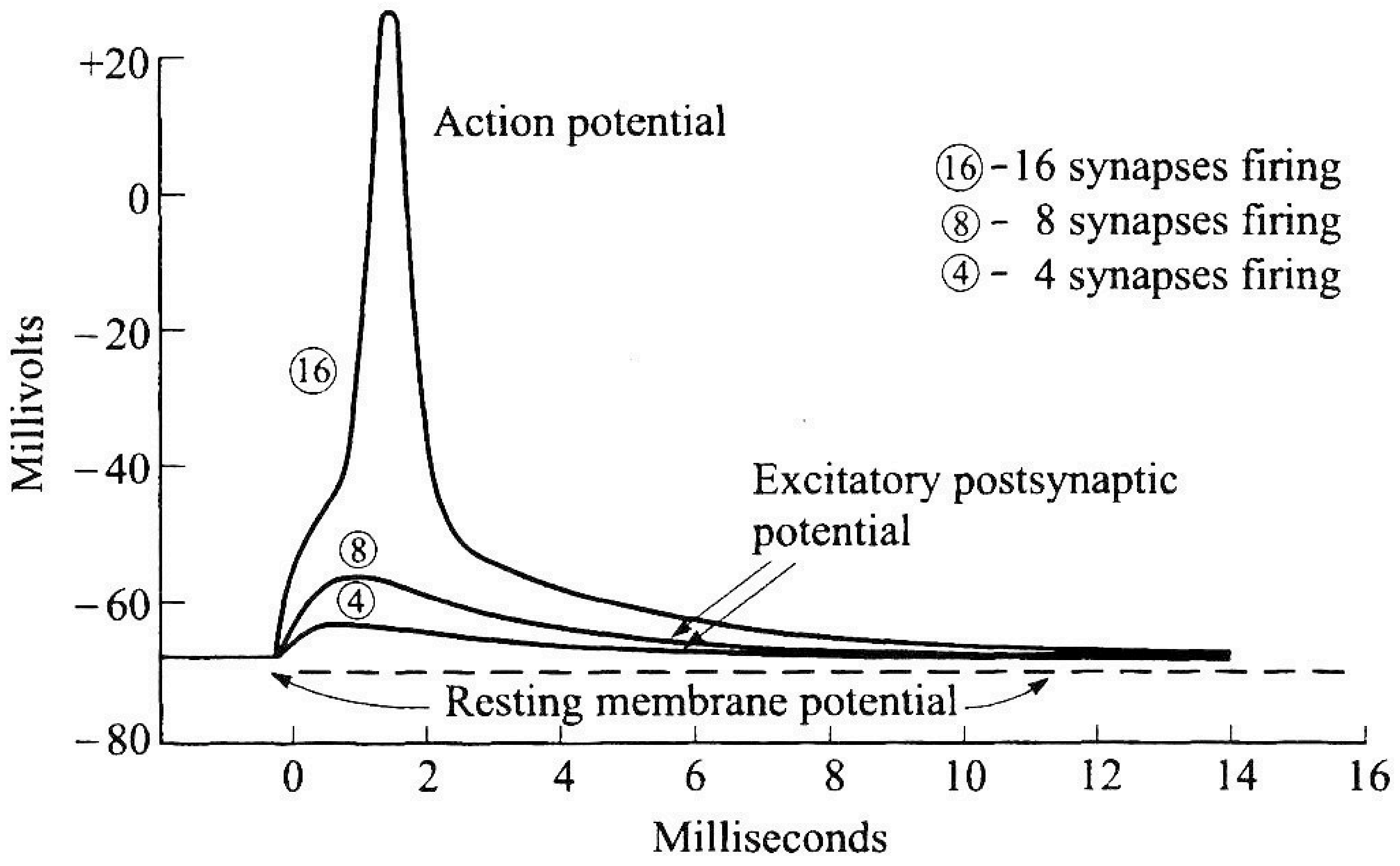
1. Signals from connected neurons are collected by the dendrites.
2. The cells body (soma) sums the incoming signals (spatially and temporally).
3. When sufficient input is received (i.e., a threshold is exceeded), the neuron generates an action potential or ‘spike’ (i.e., it ‘fires’).
4. That action potential is transmitted along the axon to other neurons, or to structures outside the nervous systems (e.g., muscles).
5. If sufficient input is not received (i.e. the threshold is not exceeded), the inputs quickly decay and no action potential is generated.
6. Timing is clearly important – input signals must arrive together, strong inputs will generate more action potentials per unit time.

Neuron Action Potential



From: Principles of Neurocomputing for Science & Engineering, Ham & Kostanic, McGraw-Hill, 2001.

Excitatory Postsynaptic Potential



From: Principles of Neurocomputing for Science & Engineering, Ham & Kostanic, McGraw-Hill, 2001.

Rate Coding versus Spike Time Coding

In biological neural networks, the individual spike timings are often important. So "***spike time coding***" is the most realistic representation for artificial neural networks.

However, averages of spike rates across time or populations of neurons carry a lot of the useful information, and so "***rate coding***" is a useful approximation.

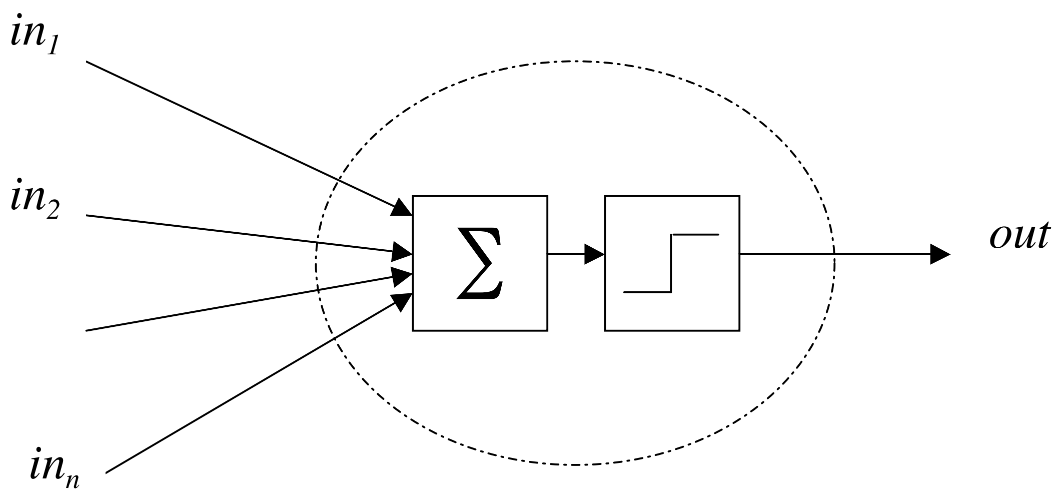
Spike coding is more powerful, but the computer models are much more complicated and more difficult to train.

Rate coding blurs the information coded in individual neurons, but usually leads to simpler models with differentiable outputs, which we will see later is important for generating efficient learning algorithms.

Sigmoid shaped activation functions in the rate coding approach follow from the cumulative effect of Gaussian distributed spikes.

The McCulloch-Pitts Neuron

A simple rate coding model of real neurons is also known as a ***Threshold Logic Unit*** :



1. A set of synapses (i.e. connections) brings in activations from other neurons.
2. A processing unit sums the inputs, and then applies a non-linear activation function (which is also often called a threshold or transfer or squashing function).
3. An output line transmits the result to other neurons.

Some Useful Notation

We often need to deal with ordered sets of numbers, which we write as ***vectors***, e.g.

$$\boldsymbol{x} = (x_1, x_2, x_3, \ldots, x_n) \quad , \quad \boldsymbol{y} = (y_1, y_2, y_3, \ldots, y_m)$$

The components x_i can be added up to give a ***scalar*** (number), e.g.

$$s = x_1 + x_2 + x_3 + \ldots + x_n = \sum_{i=1}^{n} x_i$$

Two vectors of the same length may be ***added*** to give another vector, e.g.

$$\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y} = (x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n)$$

Two vectors of the same length may be ***multiplied*** to give a scalar, e.g.

$$p = \boldsymbol{x}.\boldsymbol{y} = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n = \sum_{i=1}^{n} x_i y_i$$

To avoid any ambiguity or confusion, we will mostly be using the component notation (i.e. explicit indices and summation signs) throughout this module.

The Power of the Notation : Matrices

We can use the same vector component notation to represent more complex things with many dimensions/indices. For two indices we have matrices, e.g. an $m \times n$ ***matrix***

$$\mathbf{w} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{1n} \\ \vdots & \vdots & & \vdots \\ w_{m1} & w_{m1} & \cdots & w_{mn} \end{pmatrix}$$

Matrices of the same size can be ***added*** or ***subtracted*** component by component.

An $m \times n$ matrix **a** can be ***multiplied*** with an $n \times p$ matrix **b** to give an $m \times p$ matrix **c**. This becomes straightforward if we write it in terms of components:

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}$$

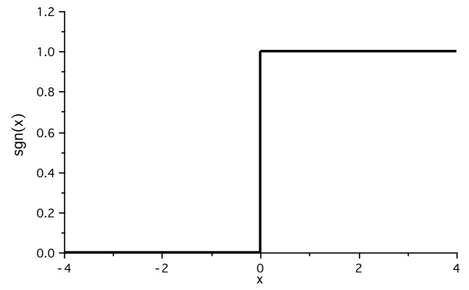
An n component vector can be regarded as a $1 \times n$ or $n \times 1$ matrix.

Some Useful Functions

A function $y = f(x)$ describes a relationship (input-output mapping) from x to y.

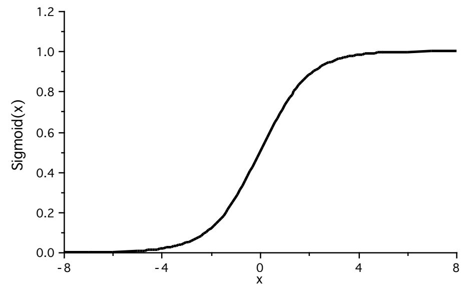
Example 1 The threshold or step function $\mathrm{step}(x)$ is defined as

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



Example 2 The logistic sigmoid function $\mathrm{Sigmoid}(x)$ is defined as

$$\mathrm{Sigmoid}(x) = \frac{1}{1+e^{-x}}$$



This is a smoothed (differentiable) form of the threshold function.

The McCulloch-Pitts Neuron Equation

Using the above notation, it is possible to write down a simple equation for the ***output*** *out* of a McCulloch-Pitts neuron as a function of its n ***inputs*** in_i :

$$out = step(\sum_{i=1}^{n} in_i - \theta)$$

where θ is the neuron's activation ***threshold***. We can easily see that:

$$out = 1 \quad \text{if} \quad \sum_{k=1}^{n} in_k \geq \theta \qquad\qquad out = 0 \quad \text{if} \quad \sum_{k=1}^{n} in_k < \theta$$

Note that the McCulloch-Pitts neuron is an extremely simplified model of real biological neurons. Some of its missing features include: non-binary inputs and outputs, non-linear summation, smooth thresholding, stochasticity, and temporal information processing.

Nevertheless, McCulloch-Pitts neurons are computationally very powerful. One can show that networks of such neurons are capable of universal computation.

Overview and Reading

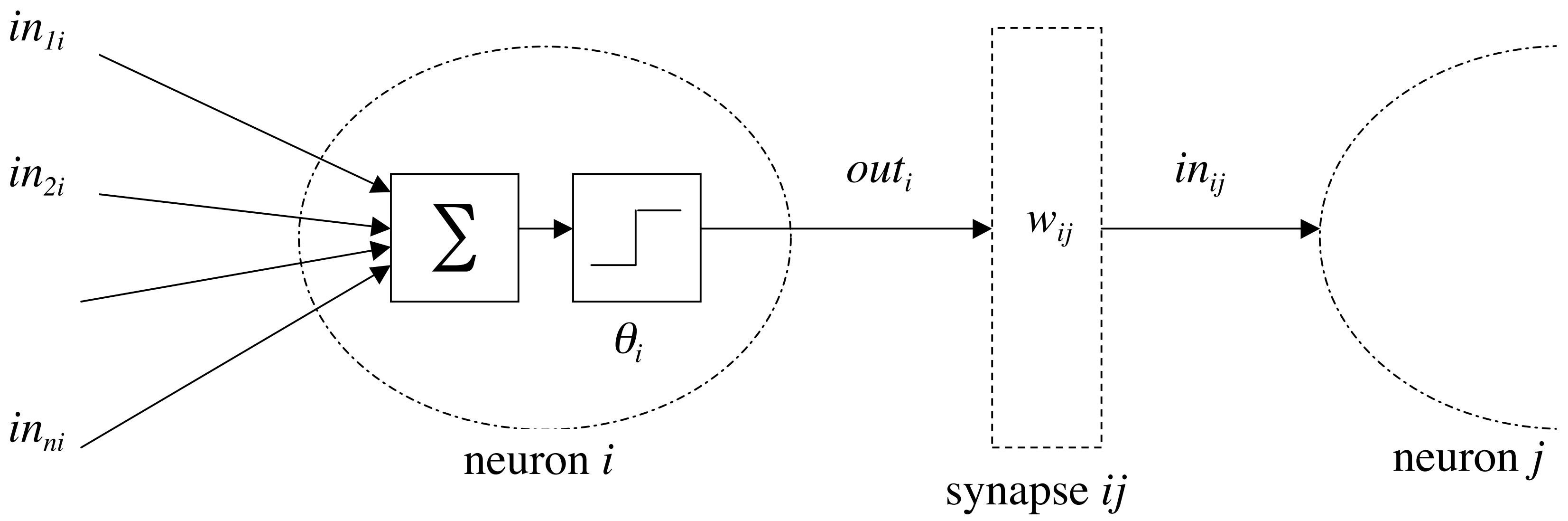
1. Biological neurons, consisting of a cell body, axons, dendrites and synapses, are able to process and transmit neural activation.
2. The McCulloch-Pitts neuron model (Threshold Logic Unit) is a crude rate-coding approximation to real neurons, that performs a simple summation and thresholding function on activation levels.
3. Appropriate mathematical notation facilitates the specification and programming of artificial neurons and networks of artificial neurons.

Reading

1. Haykin-1999: Sections 1.1, 1.2, 1.3
2. Ham & Kostanic: Sections 1.2, 1.3
3. Beale & Jackson: Sections 1.2, 3.1, 3.2
4. Gurney: Sections 2.1, 2.2.

Networks of McCulloch-Pitts Neurons

One neuron can't do much on its own. Usually we will have many neurons labelled by indices *k*, *i*, *j* and activation flows between them via synapses with strengths w_{ki}, w_{ij}:



$$out_k w_{ki} = in_{ki} \qquad out_i = step(\sum_{k=1}^{n} in_{ki} - \theta_i) \qquad out_i w_{ij} = in_{ij}$$

The Need for a Systematic Notation

It requires some care to keep track of all the neural activations and connection weights in a network without introducing confusion or ambiguity.

There are numerous complications that need to be dealt with, for example:

Each neuron has input and output activations, but

the outputs from one neuron provide the inputs to others.

The network has input and output neurons that need special treatment.

Most networks will be built up of layers of neurons, and

it makes sense to number the neurons in each layer separately, but

the weights and activations for different layers must be distinguished.

The letters used for labels/subscripts are arbitrary, and limited in number.

Frequently, the most convenient notation for a new neural network doesn't match that of other networks which have been studied previously.

A Convenient Approach to Notation

To start with, we shall adopt the following conventions for our notation:

The labels "in_i" and "out_i" will now be reserved to indicate the network input and output activations, with other notation used for the activations of other neurons, e.g. "hid_i".

The network input neurons don't process information – they simply supply the input activations to the network. Therefore, when counting the layers of neurons within a network, the input layer is not counted, only the layers of processing neurons are.

The labels used to distinguish neurons within a layer (e.g., i = 1, 2, 3, …) are arbitrary, but it helps avoid confusion if they are consistently different for different layers.

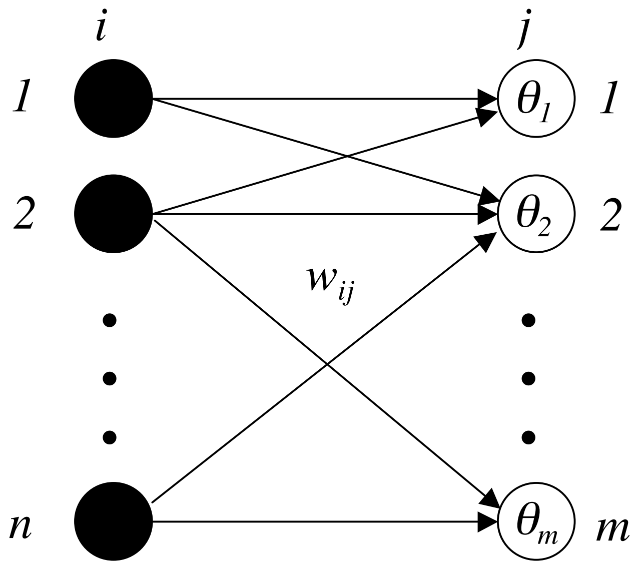
Weights between different layers need to be distinguished, and this can be done most systematically by using some kind of subscript or superscript (e.g., $w_{ij}^{(1)}$, $w_{ij}^{(2)}$, $w_{ij}^{(3)}$).

The term "artificial neuron" is often replaced by "processing unit", or just "unit".

The Perceptron

Any number of McCulloch-Pitts neurons can be connected together in any way we like.

The arrangement that has one layer of input neurons feeding forward to one output layer of McCulloch-Pitts neurons, with full connectivity, is known as a ***Perceptron***:



$$out_j = step\left(\sum_{i=1}^{n} in_i w_{ij} - \theta_j\right)$$

This is a very simple network, but it is already a powerful computational device. Later we shall see variations of it that make it even more powerful.

Implementing Logic Gates with M-P Neurons

It is possible to implement any form of logic gate using McCulloch-Pitts neurons.

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

The first step to prove this is to show explicitly that it is possible to construct simple networks that perform NOT, AND, and OR. It is then a well known result from logic that one can construct any logical function from these three basic operations. So it follows that a network of McCulloch-Pitts neurons can compute any logical function. This constitutes an ***existence proof*** of the computational power of neural networks.

However, we generally want to avoid decomposing complex problems into basic logic gates, by finding weights and thresholds that work directly in a simple neural network.

The resulting networks will usually have a more complex architectures than simple Perceptrons though, because they require more than a single layer of neurons.

Implementation of Logical NOT, AND, and OR

In each case, we have inputs in_i and outputs out, and need to determine appropriate weights and thresholds. It is easy to find solutions by inspection:

NOT

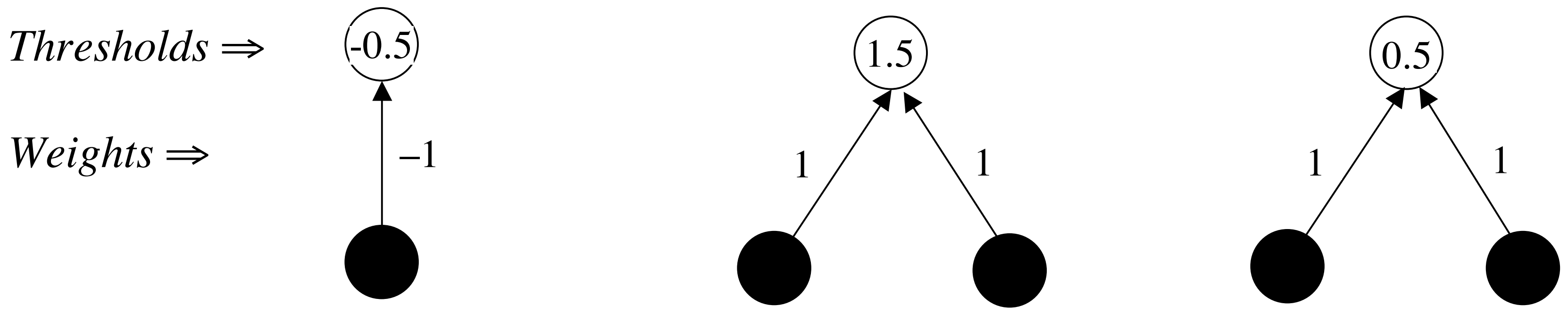
in	out
0	1
1	0

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

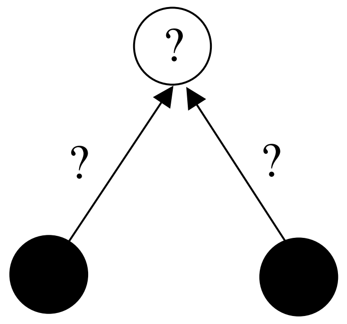


The Need to Find Weights Analytically

Constructing simple networks by hand (e.g., by trial and error) is one thing. But what about harder problems? For example, what about:

XOR

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



How long should we keep looking for a solution? We need to be able to calculate appropriate parameter values rather than searching for solutions by trial and error.

Each training pattern produces a linear inequality for the output in terms of the inputs and the network parameters. These can be used to compute the weights and thresholds.

Finding Weights Analytically for the AND Network

We have two weights w_1 and w_2 and the threshold θ, and for each training pattern we need to satisfy:

$$out = step(w_1 in_1 + w_2 in_2 - \theta)$$

So the training data lead to four inequalities:

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

$$\Rightarrow \quad \begin{array}{l} w_1\,0 + w_2\,0 - \theta < 0 \\ w_1\,0 + w_2\,1 - \theta < 0 \\ w_1\,1 + w_2\,0 - \theta < 0 \\ w_1\,1 + w_2\,1 - \theta \geq 0 \end{array} \quad \Rightarrow \quad \begin{array}{c} \theta > 0 \\ w_2 < \theta \\ w_1 < \theta \\ w_1 + w_2 \geq \theta \end{array}$$

It is easy to see that there are an infinite number of solutions. Similarly, there are an infinite number of solutions for the NOT and OR networks.

Limitations of Simple Perceptrons

We can follow the same procedure for the XOR network:

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

$$\Rightarrow \quad \boxed{\begin{array}{l} w_1\,0 + w_2\,0 - \theta < 0 \\ w_1\,0 + w_2\,1 - \theta \geq 0 \\ w_1\,1 + w_2\,0 - \theta \geq 0 \\ w_1\,1 + w_2\,1 - \theta < 0 \end{array}} \quad \Rightarrow \quad \boxed{\begin{array}{c} \theta > 0 \\ w_2 \geq \theta \\ w_1 \geq \theta \\ w_1 + w_2 < \theta \end{array}}$$

Clearly the first, second and third inequalities are incompatible with the fourth, so there is in fact no solution. We need more complex networks, e.g. that combine together many simple networks, or use different activation/thresholding/transfer functions.

It then becomes much more difficult to determine all the weights and thresholds by hand. Next lecture we shall see how a neural network can ***learn*** these parameters.

First, let us consider what these more complex networks might involve.

ANN Architectures/Structures/Topologies

Mathematically, ANNs can be represented as ***weighted directed graphs***. For our purposes, we can simply think in terms of activation flowing between processing units via one-way connections. The three most common ANN architectures are:

Single-Layer Feed-forward NNs One input layer and one output layer of processing units. No feed-back connections. (For example, a simple Perceptron.)

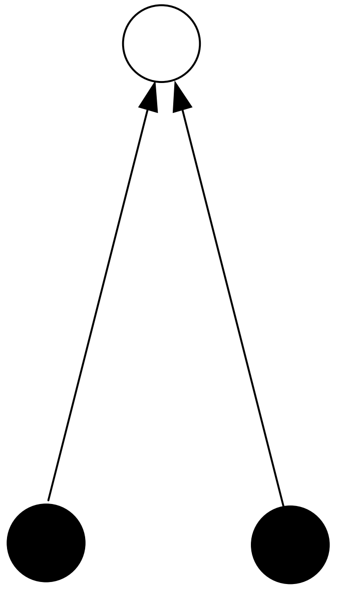
Multi-Layer Feed-forward NNs One input layer, one output layer, and one or more hidden layers of processing units. No feed-back connections. The hidden layers sit in between the input and output layers, and are thus *hidden* from the outside world. (For example, a Multi-Layer Perceptron.)

Recurrent NNs Any network with at least one feed-back connection. It may, or may not, have hidden units. (For example, a Simple Recurrent Network.)

Further interesting variations include: short-cut connections, partial connectivity, time-delayed connections, Elman networks, Jordan networks, moving windows, …

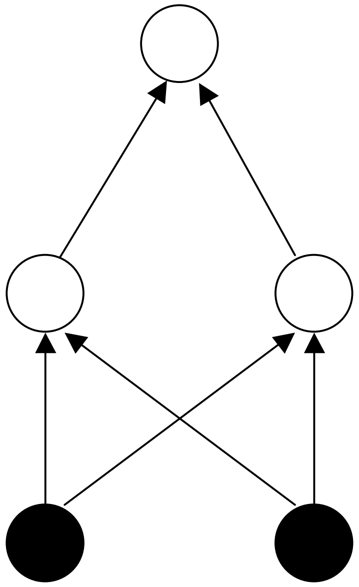
Examples of Network Architectures

Single Layer Feed-forward



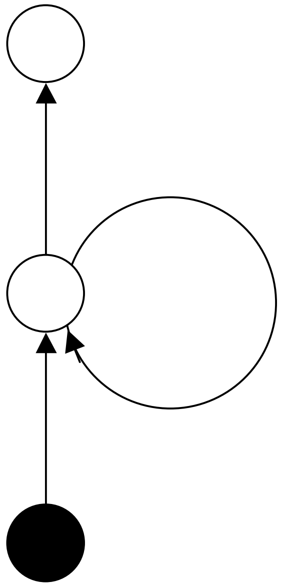
Single-Layer Perceptron

Multi-Layer Feed-forward



Multi-Layer Perceptron

Recurrent Network



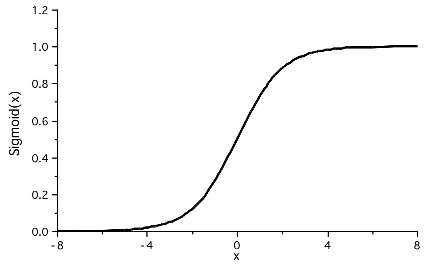
Simple Recurrent Network

Other Types of Activation/Transfer Function

Sigmoid Functions These are smooth (differentiable) and monotonically increasing.

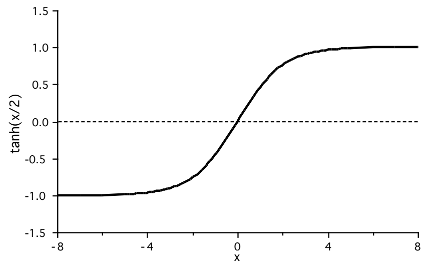
The logistic function

$$\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}$$



Hyperbolic tangent

$$\tanh\left(\tfrac{x}{2}\right) = \frac{1-e^{-x}}{1+e^{-x}}$$



Piecewise-Linear Functions These are approximations of a sigmoid functions.

$$f(x) = \begin{cases} 1 & \text{if x} \geq 0.5 \\ x+0.5 & \text{if -0.5} \leq \text{x} \leq 0.5 \\ 0 & \text{if x} \leq 0.5 \end{cases}$$