Lecture 4:

Feed–Forward Neural Networks

Dr. Roman V Belavkin

Middlesex University

BIS4435

HISTORICAL BACKGROUND

1943	McCulloch and Pitts proposed the first computational model of a neuron
1949	Hebb proposed the first learning rule
1958	Rosenblatt's work on perceptrons
1969	Minsky and Papert's paper exposed limitations of the theory
1970s	Decade of dormancy for neural networks
1980–90s	Neural network return (self–organisation, back–propagation algorithms, etc)

SOME FACTS

- Human brain contains $\approx 10^{11}$ neurons

SOME FACTS

- Human brain contains $\approx 10^{11}$ neurons
- Each neuron is connected to $\approx 10^{4}$ others

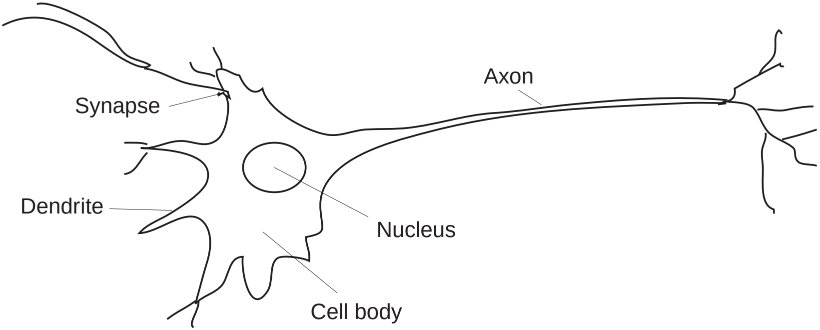
SOME FACTS

- Human brain contains $\approx 10^{11}$ neurons
- Each neuron is connected to $\approx 10^{4}$ others
- Some scientists compared the brain with a 'complex, nonlinear, parallel computer'.

SOME FACTS

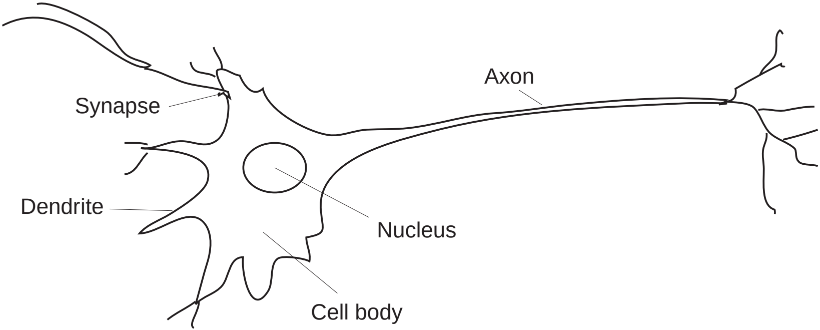
- Human brain contains $\approx 10^{11}$ neurons
- Each neuron is connected to $\approx 10^4$ others
- Some scientists compared the brain with a 'complex, nonlinear, parallel computer'.
- The largest modern neural networks achieve the complexity comparable to a nervous system of a fly.

NEURONS



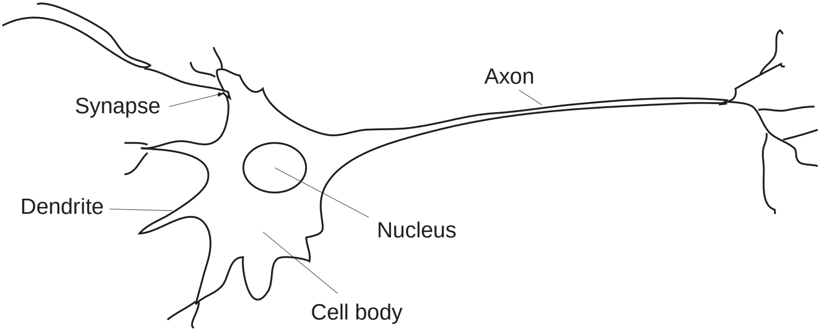
- Evidence suggests that neurons **receive**, **analyse** and **transmit** information.

NEURONS



- Evidence suggests that neurons **receive**, **analyse** and **transmit** information.
- The information in transmitted in a form of electro-chemical signals (pulses).

NEURONS



- Evidence suggests that neurons **receive**, **analyse** and **transmit** information.
- The information in transmitted in a form of electro-chemical signals (pulses).
- When a neuron sends the information we say that a neuron '**fires**'.

EXCITATION AND INHIBITION

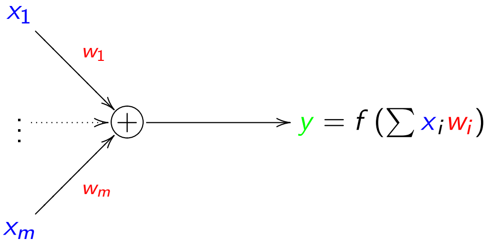
- The receptors of a neuron are called **synapses**, and they are located on many branches, called **dendrites**.
- There are many types of synapses, but roughly they can be divided into two classes:

  Excitatory: a signal received at this synapse ‘encourages’ the neuron to fire

  Inhibitory: a signal received at this synapse inhibits the neuron (as if asking to ‘shut up’)
- The neuron analyses all the signals received at its synapses. If most of them are ‘encouraging’, then the neuron gets ‘excited’ and fires its own message along a single wire, called **axon**.
- The axon may have branches to reach many other neurons.

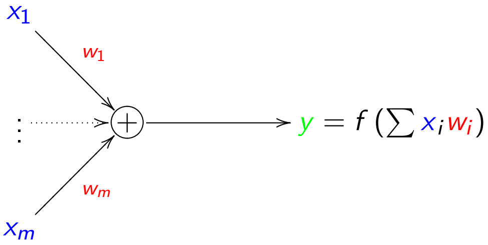
A MODEL OF A SINGLE NEURON (UNIT)

McCulloch and Pitts (1943) proposed the 'integrate and fire' model:



A MODEL OF A SINGLE NEURON (UNIT)

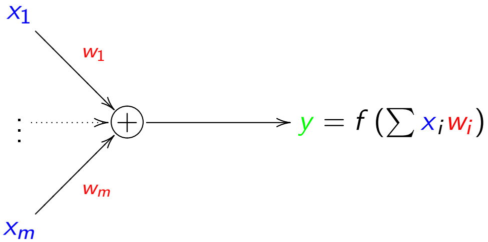
McCulloch and Pitts (1943) proposed the 'integrate and fire' model:



- Denote the m input values by x_1, x_2, . . . , x_m.

A MODEL OF A SINGLE NEURON (UNIT)

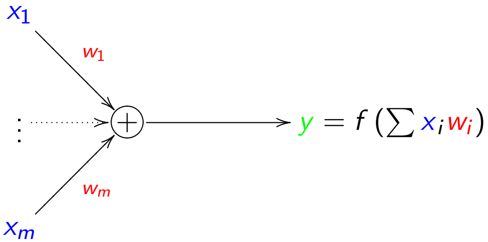
McCulloch and Pitts (1943) proposed the 'integrate and fire' model:



- Denote the m input values by $x_1, x_2, \ldots, x_m$.
- Each of the m inputs (synapses) has a weight $w_1, w_2, \ldots, w_m$.

A MODEL OF A SINGLE NEURON (UNIT)

McCulloch and Pitts (1943) proposed the ‘integrate and fire’ model:

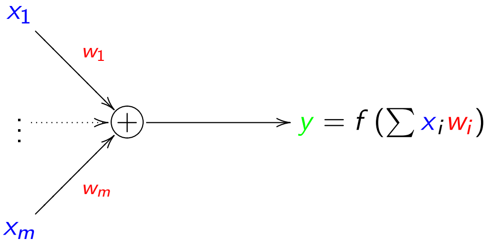


- Denote the m input values by x_1, x_2, $\ldots$, x_m.
- Each of the m inputs (synapses) has a weight w_1, w_2, $\ldots$, w_m.
- The input values are multiplied by their weights and summed

$$v = w_1 x_1 + w_2 x_2 + \cdots + w_m x_m = \sum_{i=1}^{m} w_i x_i$$

A MODEL OF A SINGLE NEURON (UNIT)

McCulloch and Pitts (1943) proposed the 'integrate and fire' model:



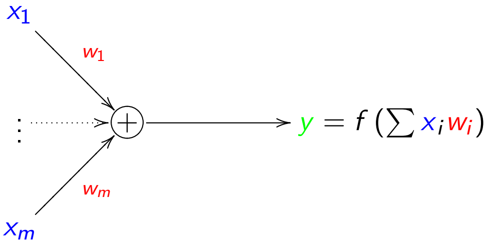
- Denote the m input values by $x_1, x_2, \ldots, x_m$.
- Each of the m inputs (synapses) has a weight $w_1, w_2, \ldots, w_m$.
- The input values are multiplied by their weights and summed

$$v = w_1 x_1 + w_2 x_2 + \cdots + w_m x_m = \sum_{i=1}^{m} w_i x_i$$

- The output is some function $y = f(v)$ of the weighted sum

A MODEL OF A SINGLE NEURON (UNIT)

McCulloch and Pitts (1943) proposed the 'integrate and fire' model:



Example

Let $x = (0, 1, 1)$ and $w = (1, -2, 4)$. Then

$$v = 1 \cdot 0 - 2 \cdot 1 + 4 \cdot 1 = 2$$

ACTIVATION FUNCTION

- The output of a neuron (y) is a function of the weighted sum

$$y = f(v)$$

- This function is often called the **activation function**.
- What function is it and how is it computed?

ACTIVATION FUNCTION

- The output of a neuron (y) is a function of the weighted sum

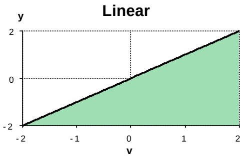
$$y = f(v)$$

- This function is often called the **activation function**.
- What function is it and how is it computed?

Linear function:

$$f(v) = a + v = a + \sum w_i x_i$$

where parameter a is called **bias**. Noice that in this case, a neuron becomes a linear model with bias a being the *intercept* and the weights, $w_1, \ldots, w_m$, being the *slopes*.



ACTIVATION FUNCTION

- The output of a neuron (y) is a function of the weighted sum

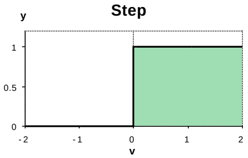
$$y = f(v)$$

- This function is often called the **activation function**.
- What function is it and how is it computed?

Heviside **step** function:

$$f(v) = \begin{cases} 1 & \text{if } v \geq a \\ 0 & \text{otherwise} \end{cases}$$

Here a is called the **threshold**



Example

If $a = 0$ and $v = 2 > 0$, then $f(2) = 1$, the neuron fires

ACTIVATION FUNCTION

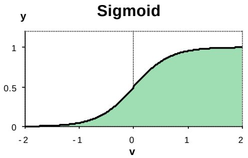
- The output of a neuron (y) is a function of the weighted sum

$$y = f(v)$$

- This function is often called the **activation function**.
- What function is it and how is it computed?

Sigmoid function:

$$f(v) = \frac{1}{1 + e^{-v}}$$



NEURONS AS DATA-DRIVEN MODELS

- We use data to create models representing the relation between the input x and the output y variables (e.g. between income and credit score)

$$y = f(x) + \text{Error}$$

NEURONS AS DATA-DRIVEN MODELS

- We use data to create models representing the relation between the input x and the output y variables (e.g. between income and credit score)

$$y = f(x) + \text{Error}$$

- If we use data to adjust parameters (the weights) to reduce the error, then a neuron becomes a data-driven model

NEURONS AS DATA-DRIVEN MODELS

- We use data to create models representing the relation between the input x and the output y variables (e.g. between income and credit score)

$$y = f(x) + \text{Error}$$

- If we use data to adjust parameters (the weights) to reduce the error, then a neuron becomes a data-driven model
- If we use only linear activation functions, then a neuron is just a linear model with weights corresponding to slopes (i.e. related to correlations)

$$f(x_1, \ldots, x_m) = {\color{red}a} + {\color{blue}w_1} x_1 + \cdots + {\color{blue}w_m} x_m$$

SO, WHAT IS DIFFERENT FROM LINEAR MODELS?

- The linear mean–square regression is a good technique, but it relies heavily on the use of quadratic cost function

$$c(y, f(x)) = |y - f(x)|^2$$

SO, WHAT IS DIFFERENT FROM LINEAR MODELS?

- The linear mean–square regression is a good technique, but it relies heavily on the use of quadratic cost function

$$c(y, f(x)) = |y - f(x)|^2$$

- Neurons can be 'trained' to using other cost functions, such as the absolute deviation:

$$c(y, f(x)) = |y - f(x)|$$

SO, WHAT IS DIFFERENT FROM LINEAR MODELS?

- The linear mean–square regression is a good technique, but it relies heavily on the use of quadratic cost function

$$c(y, f(x)) = |y - f(x)|^2$$

- Neurons can be 'trained' to using other cost functions, such as the absolute deviation:

$$c(y, f(x)) = |y - f(x)|$$

- Networks of many neurons can be seen as sets of multiple and competing models.

SO, WHAT IS DIFFERENT FROM LINEAR MODELS?

- The linear mean–square regression is a good technique, but it relies heavily on the use of quadratic cost function

$$c(y, f(x)) = |y - f(x)|^2$$

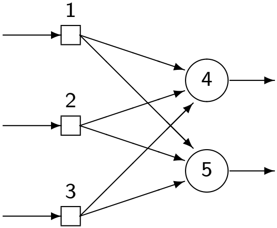
- Neurons can be 'trained' to using other cost functions, such as the absolute deviation:

$$c(y, f(x)) = |y - f(x)|$$

- Networks of many neurons can be seen as sets of multiple and competing models.
- Neural networks can be used to model non-linear relations in data.

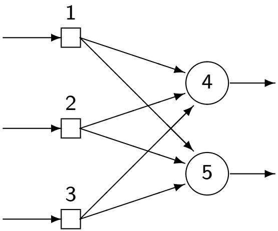
FEED-FORWARD NEURAL NETWORKS

A collection of neurons connected together in a network can be represented by a directed graph:



FEED-FORWARD NEURAL NETWORKS

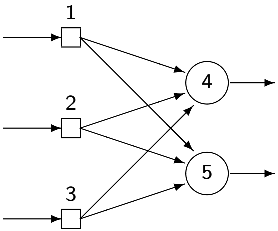
A collection of neurons connected together in a network can be represented by a directed graph:



- **Nodes** represent the neurons, and **arrows** represent the links between them.

FEED-FORWARD NEURAL NETWORKS

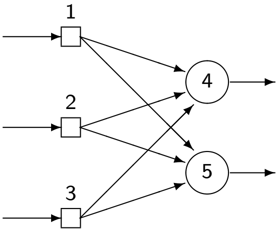
A collection of neurons connected together in a network can be represented by a directed graph:



- **Nodes** represent the neurons, and **arrows** represent the links between them.
- Each node has its number, and a link connecting two nodes will have a pair of numbers (e.g. $(1,4)$ connecting nodes 1 and 4).

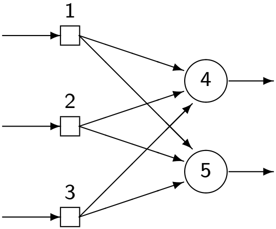
FEED-FORWARD NEURAL NETWORKS

A collection of neurons connected together in a network can be represented by a directed graph:



- **Nodes** represent the neurons, and **arrows** represent the links between them.
- Each node has its number, and a link connecting two nodes will have a pair of numbers (e.g. $(1, 4)$ connecting nodes 1 and 4).
- Networks without cycles (feedback loops) are called a feed-forward networks (or **perceptron**).

INPUT AND OUTPUT NODES

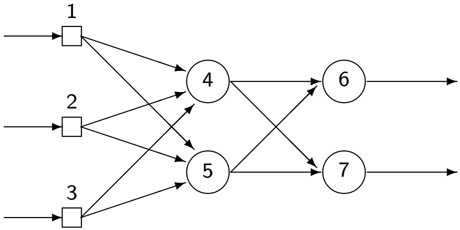


Input nodes of the network (nodes 1, 2 and 3) are associated with the input variables ($x_1, \ldots, x_m$). They do not compute anything, but simply pass the values to the processing nodes.

Output nodes (4 and 5) are associated with the output variables ($y_1, \ldots, y_n$).

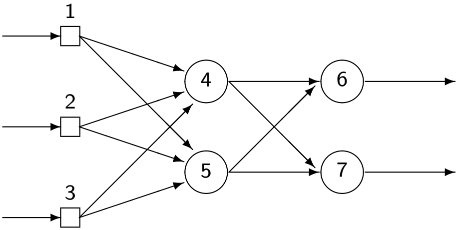
HIDDEN NODES AND LAYERS

- A neural network may have hidden nodes — they are not connected directly to the environment ('hidden' inside the network):



HIDDEN NODES AND LAYERS

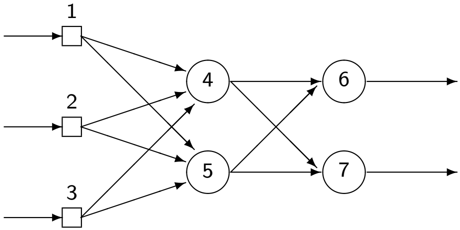
- A neural network may have hidden nodes — they are not connected directly to the environment ('hidden' inside the network):



- We may organise nodes in layers: input (nodes 1,2 and 3), hidden (4 and 5) and output (6 and 7) layers.

HIDDEN NODES AND LAYERS

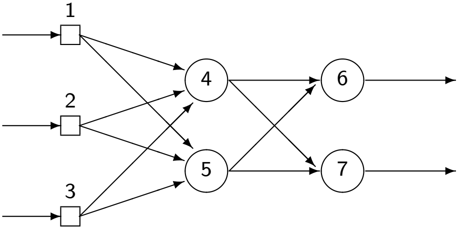
- A neural network may have hidden nodes — they are not connected directly to the environment ('hidden' inside the network):



- We may organise nodes in layers: input (nodes 1,2 and 3), hidden (4 and 5) and output (6 and 7) layers.
- Neural networks can have several hidden layers.

NUMBERING THE WEIGHTS

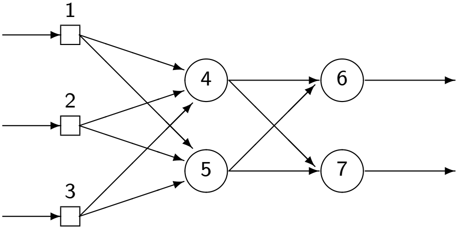
- Each jth node in a network has a set of weights w_{ij}.



- For example, node 4 has weights w_{14}, w_{24} and w_{34}.

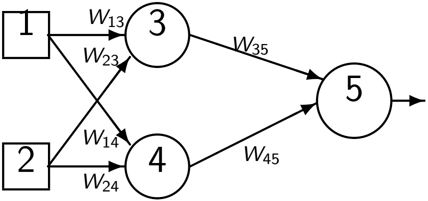
NUMBERING THE WEIGHTS

- Each jth node in a network has a set of weights w_{ij}.



- For example, node 4 has weights w_{14}, w_{24} and w_{34}.
- A network is completely defined if we know its **topology** (its graph), the set of all weights w_{ij} and the activation functions, f, of all the nodes.

Example



$w_{13} = 2$ $w_{23} = -3$	$w_{35} = 2$ $w_{45} = -1$
$w_{14} = 1$ $w_{24} = 4$	

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

What is the network output, if the inputs are $x_1 = 1$ and $x_2 = 0$?

SOLUTION

1. Calculate weighted sums in the first hidden layer:

$$
\begin{aligned}
v_3 &= w_{13}x_1 + w_{23}x_2 = 2 \cdot 1 - 3 \cdot 0 = 2 \\
v_4 &= w_{14}x_1 + w_{24}x_2 = 1 \cdot 1 + 4 \cdot 0 = 1
\end{aligned}
$$

SOLUTION

1. Calculate weighted sums in the first hidden layer:

$$\begin{aligned} v_3 &= w_{13}x_1 + w_{23}x_2 = 2 \cdot 1 - 3 \cdot 0 = 2 \\ v_4 &= w_{14}x_1 + w_{24}x_2 = 1 \cdot 1 + 4 \cdot 0 = 1 \end{aligned}$$

2. Apply the activation function:

$$y_3 = f(2) = 1 \ , \qquad y_4 = f(1) = 1$$

SOLUTION

1. Calculate weighted sums in the first hidden layer:

$$\begin{aligned} v_3 &= w_{13}x_1 + w_{23}x_2 = 2 \cdot 1 - 3 \cdot 0 = 2 \\ v_4 &= w_{14}x_1 + w_{24}x_2 = 1 \cdot 1 + 4 \cdot 0 = 1 \end{aligned}$$

2. Apply the activation function:

$$y_3 = f(2) = 1 \ , \qquad y_4 = f(1) = 1$$

3. Calculate the weighted sum of node 5:

$$v_5 = w_{35}y_3 + w_{45}y_4 = 2 \cdot 1 - 1 \cdot 1 = 1$$

SOLUTION

1. Calculate weighted sums in the first hidden layer:

$$\begin{aligned} v_3 &= w_{13}x_1 + w_{23}x_2 = 2 \cdot 1 - 3 \cdot 0 = 2 \\ v_4 &= w_{14}x_1 + w_{24}x_2 = 1 \cdot 1 + 4 \cdot 0 = 1 \end{aligned}$$

2. Apply the activation function:

$$y_3 = f(2) = 1 \;, \qquad y_4 = f(1) = 1$$

3. Calculate the weighted sum of node 5:

$$v_5 = w_{35}y_3 + w_{45}y_4 = 2 \cdot 1 - 1 \cdot 1 = 1$$

4. The output is $y_5 = f(1) = 1$

TRAINING NEURAL NETWORKS

- Let us invert the previous problem:
  - Suppose that the inputs to the network are $x_1 = 1$ and $x_2 = 0$, and f is a step function.
  - Find values of the weights, w_{ij}, such that the output of the network $y_5 = 0$?

TRAINING NEURAL NETWORKS

- Let us invert the previous problem:
  - Suppose that the inputs to the network are $x_1 = 1$ and $x_2 = 0$, and f is a step function.
  - Find values of the weights, w_{ij}, such that the output of the network $y_5 = 0$?
- This problem is more difficult, because there are more unknowns (weights) than knowns (input and output). In general, there is an infinite number of solutions.

TRAINING NEURAL NETWORKS

- Let us invert the previous problem:
  - Suppose that the inputs to the network are $x_1 = 1$ and $x_2 = 0$, and f is a step function.
  - Find values of the weights, w_{ij}, such that the output of the network $y_5 = 0$?
- This problem is more difficult, because there are more unknowns (weights) than knowns (input and output). In general, there is an infinite number of solutions.
- The process of finding a set of weights such that for a given input the network produces the desired output is called training.

SUPERVISED LEARNING

- Algorithms for training neural networks can be **supervised** (i.e. with a 'teacher') and **unsupervised** (self–organising).

SUPERVISED LEARNING

- Algorithms for training neural networks can be **supervised** (i.e. with a ‘teacher’) and **unsupervised** (self–organising).
- Supervised algorithms use a training set — a set of pairs (x, y) of inputs with their corresponding desired outputs.
- We may think of a training set as a set of examples.

SUPERVISED LEARNING

- Algorithms for training neural networks can be **supervised** (i.e. with a 'teacher') and **unsupervised** (self–organising).
- Supervised algorithms use a training set — a set of pairs (x, y) of inputs with their corresponding desired outputs.
- We may think of a training set as a set of examples.
- An outline of a supervised learning algorithm:
  1. Initially, set all the weights w_{ij} to some random values
  2. Repeat
     1. Feed the network with an input x from one of the examples in the training set
     2. Compute the network's output $f(x)$
     3. Change the weights w_{ij} of the nodes
  3. Until the error $c(y, f(x))$ is small

DISTRIBUTED MEMORY

- After training, the weights represent properties of the training data (similar to the covariance matrix, slopes of a linear model, etc)

DISTRIBUTED MEMORY

- After training, the weights represent properties of the training data (similar to the covariance matrix, slopes of a linear model, etc)
- Thus, the weights form the memory of a neural network.

DISTRIBUTED MEMORY

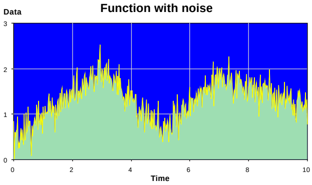
- After training, the weights represent properties of the training data (similar to the covariance matrix, slopes of a linear model, etc)
- Thus, the weights form the memory of a neural network.
- The knowledge in this case is said to be **distributed** across the network. Large number of nodes not only increases the storage 'capacity' of a network, but also ensures that the knowledge is robust.

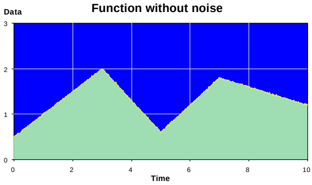
DISTRIBUTED MEMORY

- After training, the weights represent properties of the training data (similar to the covariance matrix, slopes of a linear model, etc)
- Thus, the weights form the memory of a neural network.
- The knowledge in this case is said to be **distributed** across the network. Large number of nodes not only increases the storage ‘capacity’ of a network, but also ensures that the knowledge is robust.
- By changing the weights in the network we may store new information.

GENERALISATION

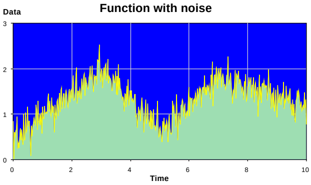
- By memorising patterns in the data during training, neural networks may produce reasonable answers for input patterns not seen during training (generalisation).

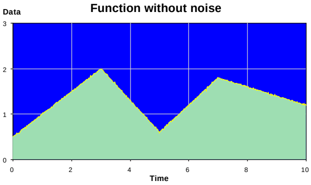




GENERALISATION

- By memorising patterns in the data during training, neural networks may produce reasonable answers for input patterns not seen during training (generalisation).
- Generalisation is particularly useful for classification of noisy data, the 'what-if' analysis and prediction (e.g. time-series forecast)





APPLICATION OF ANN

Include:

- Function approximation (modelling)
- Pattern classification (analysis of time–series, customer databases, etc).
- Object recognition (e.g. character recognition)
- Data compression
- Security (credit card fraud)

PATTERN CLASSIFICATION

- In some literature, the set of all input values is called the input pattern, and the set of output values the output pattern

$$x = (x_1, \ldots, x_m) \quad \longrightarrow \quad \mathbf{y} = (y_1, \ldots, y_n)$$

PATTERN CLASSIFICATION

- In some literature, the set of all input values is called the input pattern, and the set of output values the output pattern

$$x = (x_1, \ldots, x_m) \quad \longrightarrow \quad \mathbf{y} = (y_1, \ldots, y_n)$$

- A neural network 'learns' the relation between different input and output patterns.

PATTERN CLASSIFICATION

- In some literature, the set of all input values is called the input pattern, and the set of output values the output pattern

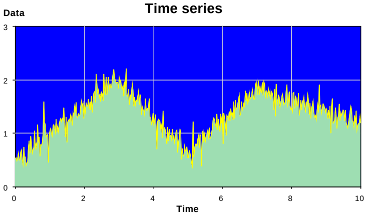
$$x = (x_1, \dots, x_m) \quad \longrightarrow \quad \mathbf{y} = (y_1, \dots, y_n)$$

- A neural network 'learns' the relation between different input and output patterns.
- Thus, a neural network performs pattern **classification** or pattern **recognition** (i.e. classifies inputs into output categories).

TIME SERIES ANALYSIS

A time series is a recording of some variable (e.g. a share price, temperature) at different time moments:

$$x(t_1), x(t_2), \ldots, x(t_m)$$



The aim of the analysis is to learn to predict the future values.

TIME SERIES (CONT.)

- We may use a neural network to analyse time series:

  Input: consider m values in the past $x(t_1), x(t_2), \ldots, x(t_m)$ as m input variables.

  Output: consider n future values $y(t_{m+1}), y(t_{m+2}), \ldots, y(t_{m+n})$ as n output variables.

TIME SERIES (CONT.)

- We may use a neural network to analyse time series:

  Input: consider m values in the past $x(t_1), x(t_2), \ldots, x(t_m)$ as m input variables.

  Output: consider n future values $y(t_{m+1}), y(t_{m+2}), \ldots, y(t_{m+n})$ as n output variables.

- Our goal is to find the following model

$$(y(t_{m+1}), \ldots, y(t_{m+n})) \approx f(x(t), x(t_1), \ldots, x(t_m))$$

TIME SERIES (CONT.)

- We may use a neural network to analyse time series:

  Input: consider m values in the past $x(t_1), x(t_2), \ldots, x(t_m)$ as m input variables.

  Output: consider n future values $y(t_{m+1}), y(t_{m+2}), \ldots, y(t_{m+n})$ as n output variables.

- Our goal is to find the following model

$$(y(t_{m+1}), \ldots, y(t_{m+n})) \approx f(x(t), x(t_1), \ldots, x(t_m))$$

- By training a neural network with m inputs and n outputs on the time series data, we can create such a model.

BENEFITS OF NEURAL NETWORKS

- Can be applied to many problems, as long as there is some data.

BENEFITS OF NEURAL NETWORKS

- Can be applied to many problems, as long as there is some data.
- Can be applied to problems, for which analytical methods do not yet exist

BENEFITS OF NEURAL NETWORKS

- Can be applied to many problems, as long as there is some data.
- Can be applied to problems, for which analytical methods do not yet exist
- Can be used to model non-linear dependencies.

BENEFITS OF NEURAL NETWORKS

- Can be applied to many problems, as long as there is some data.
- Can be applied to problems, for which analytical methods do not yet exist
- Can be used to model non-linear dependencies.
- If there is a pattern, then neural networks should quickly work it out, even if the data is ‘noisy’.

BENEFITS OF NEURAL NETWORKS

- Can be applied to many problems, as long as there is some data.
- Can be applied to problems, for which analytical methods do not yet exist
- Can be used to model non-linear dependencies.
- If there is a pattern, then neural networks should quickly work it out, even if the data is ‘noisy’.
- Always gives some answer even when the input information is not complete.

BENEFITS OF NEURAL NETWORKS

- Can be applied to many problems, as long as there is some data.
- Can be applied to problems, for which analytical methods do not yet exist
- Can be used to model non-linear dependencies.
- If there is a pattern, then neural networks should quickly work it out, even if the data is ‘noisy’.
- Always gives some answer even when the input information is not complete.
- Networks are easy to maintain.

LIMITATIONS OF NEURAL NETWORKS

- Like with any data-driven models, they cannot be used if there is no or very little data available.

LIMITATIONS OF NEURAL NETWORKS

- Like with any data-driven models, they cannot be used if there is no or very little data available.
- There are many free parameters, such as the number of hidden nodes, the learning rate, minimal error, which may greatly influence the final result.

LIMITATIONS OF NEURAL NETWORKS

- Like with any data-driven models, they cannot be used if there is no or very little data available.
- There are many free parameters, such as the number of hidden nodes, the learning rate, minimal error, which may greatly influence the final result.
- Not good for arithmetics and precise calculations.

LIMITATIONS OF NEURAL NETWORKS

- Like with any data-driven models, they cannot be used if there is no or very little data available.
- There are many free parameters, such as the number of hidden nodes, the learning rate, minimal error, which may greatly influence the final result.
- Not good for arithmetics and precise calculations.
- Neural networks do not provide explanations. If there are many nodes, then there are too many weights that are difficult to interprete (unlike the slopes in linear models, which can be seen as correlations). In some tasks, explanations are crucial (e.g. air traffic control, medical diagnosis).