Perceptron

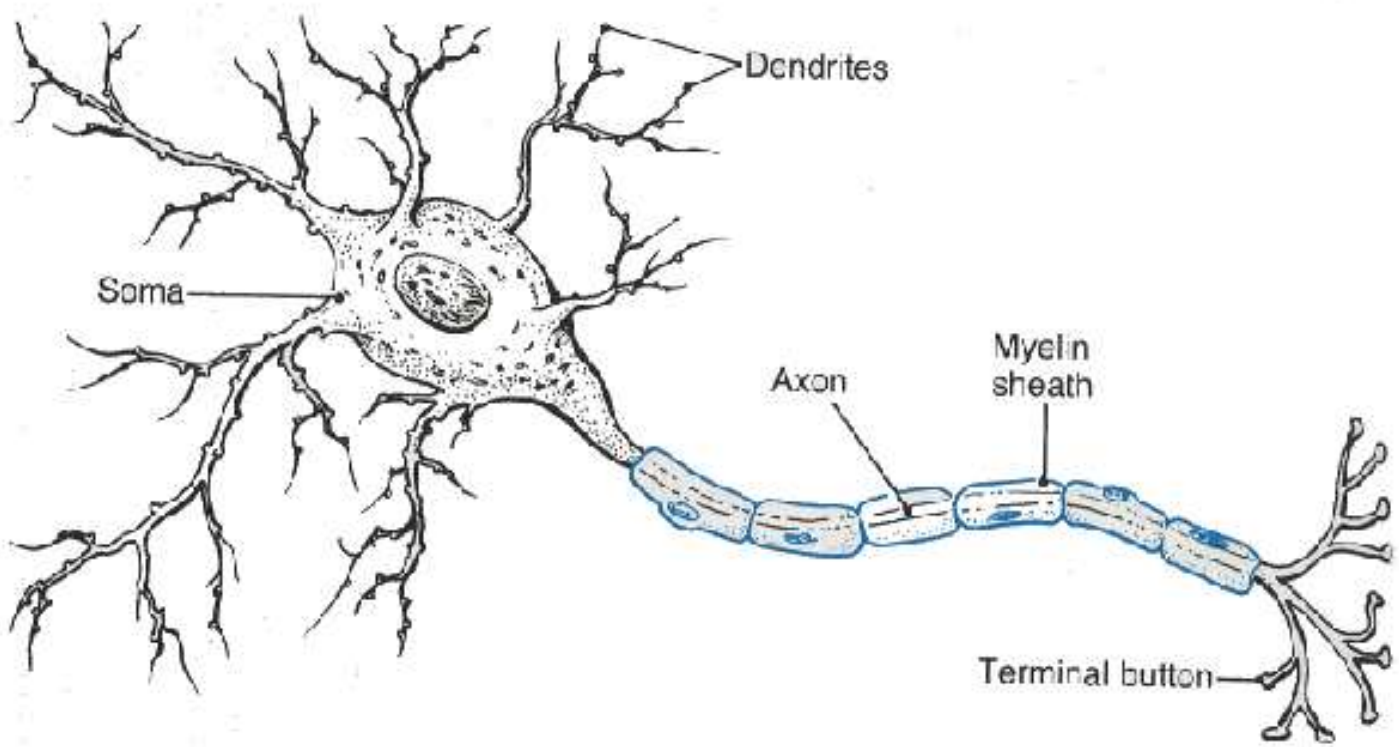
(c) Marcin Sydow

Topics covered by this lecture:

- Neuron and its properties
- Mathematical model of neuron: Perceptron
- Perceptron as a classifier
- Perceptron' Learning Rule (Delta Rule)

Neuron

Human neural system has been a natural source of inspiration for artificial intelligence researchers. Hence the interest in a neuron – the fundamental unit of the neural system.



Behaviour of a Neuron

- Transmit the neural signal from the **"inputs"** (dendrites: the endings of a neural cell contacting with other cells) to the **"output"** (neurite, which is often a very long ending, transmitting the signal to further neurons)
- Non-linear signal processing: the output state is not a simple sum of input signals
- Dynamically modify the connections with other neurons via synapses (connecting elements), which makes it possible to strengthen or weaken the signal received from other neurons according to the current task

Perceptron: an Artificial Neuron

Perceptron is a simple mathematical model of a neuron.

Historically, the goal for the work on neural networks was to gain the ability of generalisation (approximation) and learning, specific for the human brain (according to one of the definitions of artificial intelligence).

Currently, artificial neural networks focus on less "ambitiuous", but more realistic tasks.

A single perceptron can serve as a classifier or regressor

Perceptron is a building block in more complex artificial neural network structures that can solve practical problems:

- supervised or unsupervised learning
- controlling complex mechanical devices (e.g. robotics)

Perceptron - a simple model of natural neuron

A perceptron consists of:

- n inputs $x_1, \ldots, x_n$ corresponding to dendrites
- n weights $w_1, \ldots, w_n$, corresponding to synapses
  Each weight w_i is attached to the $i-th$ input x_i
- a threshold Θ
- one output y

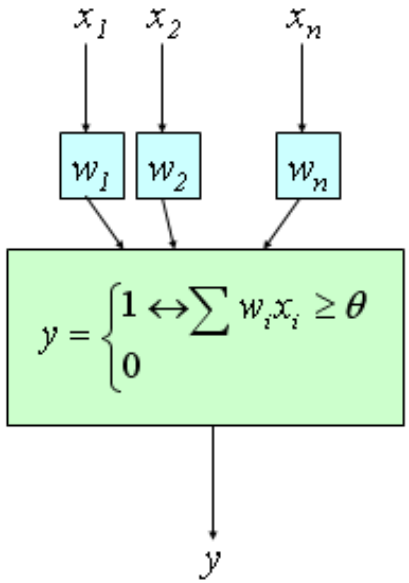
(All the variables are real numbers)

The output y is computed as follows:

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i \cdot x_i = W^T X \geq \Theta \quad \text{(perceptron "activated")} \\ 0 & \text{else} \quad \text{(not "activated")} \end{cases}$$

$W, X \in R^n$ denote the vector of weights and inputs, respectively

The perceptron is activated (y=1) only when the dot product $W^T X$ (sometimes called as "net") reaches the specified threshold Θ



We call the perceptron **discrete** if $y \in \{0, 1\}$ (or $\{-1, 1\}$) **continuous** if $y \in [0, 1]$ (or $[-1, 1]$)

Perceptron: Geometrical Interpretation

The computation of the output of the perceptron has simple geometrical interpretation.

Consider the n-dimensional input space (each point here is a potential input vector $X \in R^n$).

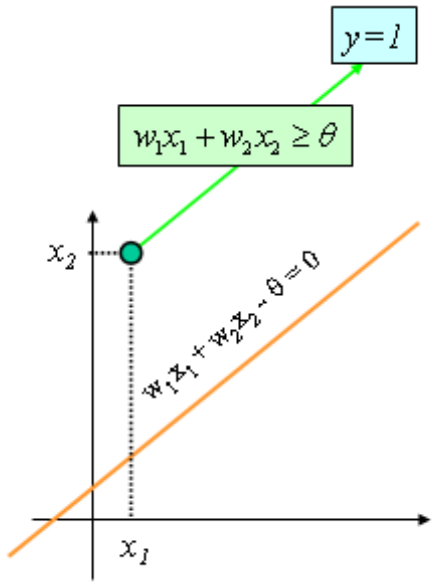
The weight vector $W \in R^n$ is the **normal** vector of the **decision hyperplane**.

The perceptron is activated (outputs 1) only if the input vector X is on the same side (of the decision hyperplane) as the weight vector W.

Moreover, the maximum net value ($W^T X$) is achieved for X being close to W, is 0 if they are orthogonal and minimum (negative) if they are opposite.

Geometrical Interpretation of Perceptron, cont.

The required perceptron's behavior can be obtained by adjusting appropriate weights and threshold.

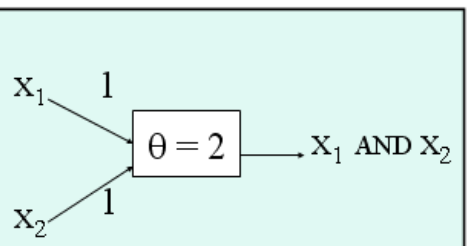


The weight vector W determines the "direction" of the decision hyperplane. The threshold Θ determines how much decision hyperplane is moved from the origin (0 point)

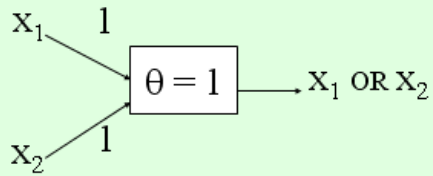
Example: Perceptrons can simulate logical circuits

A single perceptron with appropriately set weights and threshold can easily simulate basic logical gates:

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1



x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1



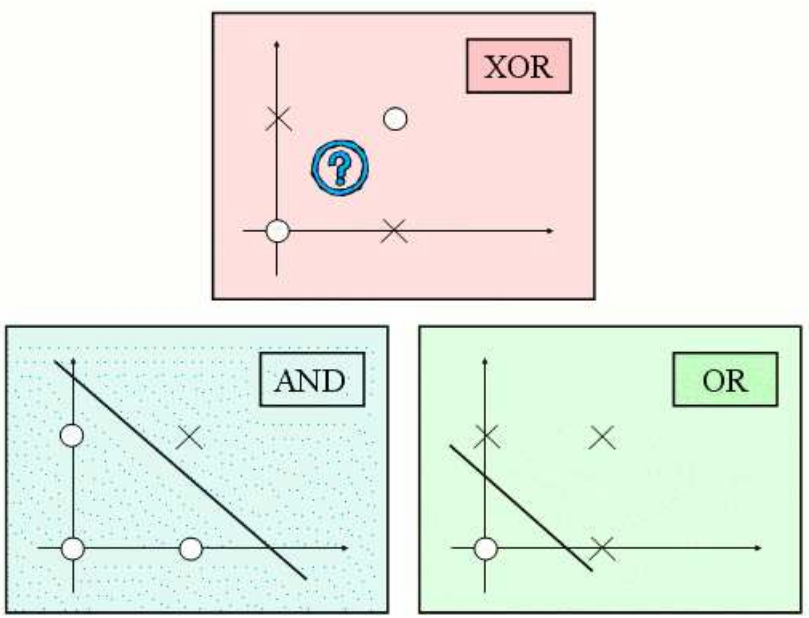
Limitations of single perceptron

A single perceptron can "distinguish" (by the value of its output) only the sets of inputs which are **linearly separable** in the input space (i.e. there exists a n-1-dimensional hyperplane separating the positive and negative cases)

One of the simplest examples of **linearly non-separable** sets is logical function XOR (excluding alternative).

Limitations of a single perceptron

One can see on the pictures below that functions AND and OR correspond to the linearly separable sets, so we can model each of them using a single perceptron (as shown in the previous section), while XOR cannot be modeled by any single perceptron.



Network of Perceptrons

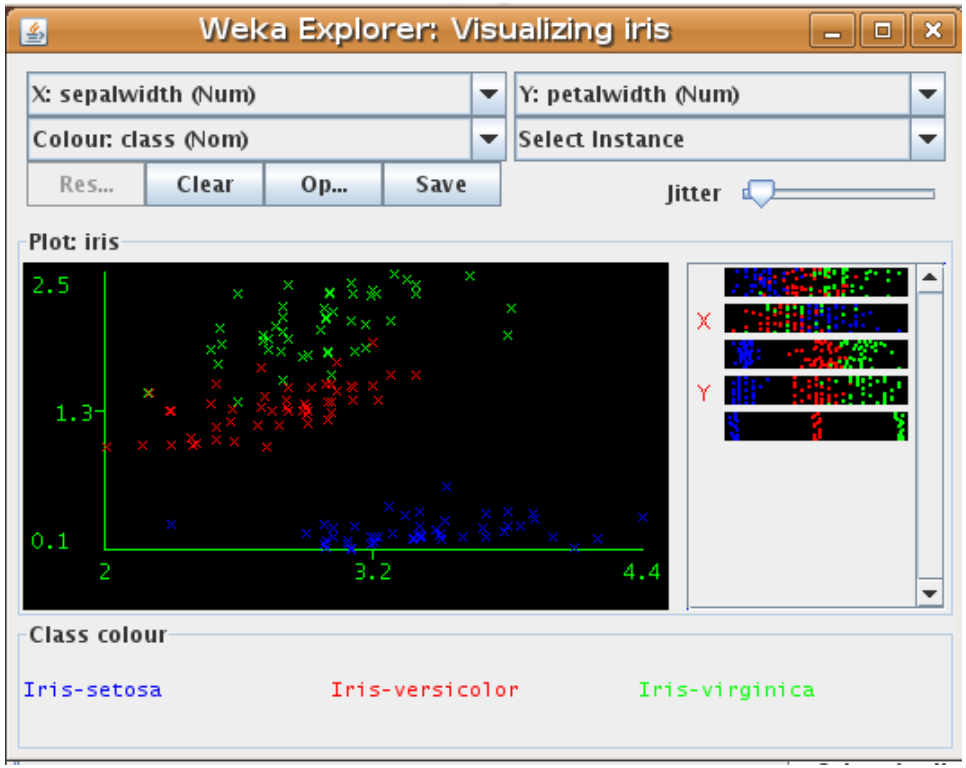
The output of one perceptron can be connected to input of other perceptron(s) (as in neural system). This makes it possible to extend the computational possibilities of a single perceptron.

For example, XOR can be simulated by joining 2 perceptrons and appropriately setting their weights and thresholds.

Remark: by "perceptron" or "multilayer perceptron" one can also mean a network of connected entities (neurons).

Example: Iris classification

Single perceptron can distinguish Iris-setosa from 2 other sub-species



However, it cannot exactly recognise any of the other 2 sub-species

Perceptron: Overcoming the limitations

Discovery of the above limitations (1969) blocked further development of neural networks for years.

An obvious method to cope with this problem is to join perceptrons together (e.g. two perceptrons are enough to model XOR) to form artificial neural networks.

However, it is mathematically far more difficult to adjust more complex networks to our needs than in case of a single perceptron.

Fortunately, the development of efficient techniques of learning the perceptron networks (80s of XX century), i.e. automatic tuning of their weights on the basis of positive and negative examples, caused the "renaissance" of artificial neural networks.

Perceptron as a Classifier that can Learn

A single perceptron can be used as a tool in supervised machine learning, as a binary classifier (output equal to 0 or 1)

To achieve this, we present a perceptron with a training set of pairs:

- input vector
- correct answer (0 or 1)

The perceptron can "learn" the correct answers by appropriately setting it's weight vector.

Perceptron's Learning: "Delta Rule"

We apply the "training examples" one by one. If the current output is the same as the desired, we pass to the next example.

If it is incorrect we apply the following "perceptron learning rule" to the vector of its weights:

$$W' = W + (d - y)\alpha X$$

d - desired (correct) output
y - actual output
$0 < \alpha < 1$ - a parameter, tuned experimentally

Interpretation of Perceptron Learning Rule

To "force" the perceptron to give the desired ouputs, its weight vector should be maximally "close" to the positive (y=1) cases.

Hence the formula:

$$W' = W + (d - y)\alpha X$$

- move W towards "positive" X if it outputs 0 instead to 1 ("too weak activation")
- move W away from "negative" X (if it outputs 1 instead of 0) ("too strong activation")

Usually, the whole training set should be passed several times to obtain the desired weights of perceptron.

The Role of Threshold

If the activation threshold Θ is set to 0, the perceptron can "distinguish" only the classes which are separable by a decision hyperplane containing the origin of the input space (0 vector)

To "move" the decision hyperplane away from the origin, the threshold has to be set to some non-zero value.

Incorporating Threshold into Learning

$W^T X \geq \Theta$

$W^T X - \Theta \geq 0$

Hence, X can be extended by appending -1 (fake n+1-th input) and W can be extended by appending Θ. Denote by W' and X' the extended (n+1)-th dimensional vectors. Now we have:

$W'^T X' \geq 0$ - the same form as "without" the threshold

Now, the learning rule can be applied to the extended vectors W' and X'

Questions/Problems:

- desired properties of artificial neuron
- mathematical formulation of perceptron
- how perceptron computes it's output
- geometric interpretation of perceptron's computation
- mathematical limitations of perceptron
- learning rule for perceptron
- geometric interpretation of perceptron's learning rule

Thank you for attention