

neural networks
up to and including backprop

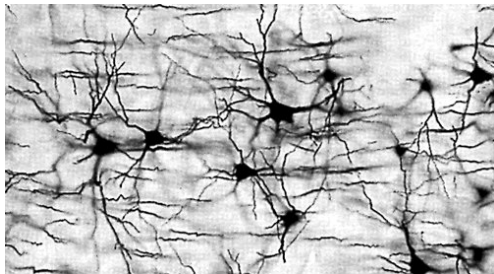
Patrick van der Smagt

December 2011

Biological Neural Networks

A biological neural network consist of a collection of **neurons** linked together in a certain way, often (but not always) in **layers**.

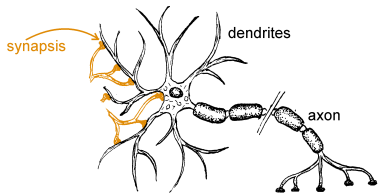
Biological neural network: Stained brain cells under the microscope



biological neurons

Each cell collects incoming electrical signals through its **dendrites**, **processes** their accumulated effect in a fairly simple manner, and forwards another electrical signal through its **axon** to a number of follow-on neurons, to which it is connected via **synapses** of varying conduction efficiency.

biological neuron



encoding

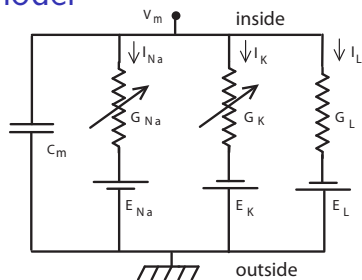
spike encoding:

an action potential sequence, or spike train, can be characterised simply by a series of all-or-none point events in time. The lengths of interspike intervals (ISIs) between two successive spikes in a spike train often vary, apparently randomly. The study of neural coding involves measuring and characterizing how stimulus attributes, such as light or sound intensity, or motor actions, such as the direction of an arm movement, are represented by neuron action potentials or spikes. In order to describe and analyze neuronal firing, statistical methods and methods of probability theory and stochastic point processes have been widely applied.

neuronal coding can be contained in:

- ▶ rate coding: the firing rate of the neuron
- ▶ temporal coding: spike timing or high-frequency firing-rate fluctuations
- ▶ population coding: joint activities of a number of neurons

Hodgkin-Huxley model



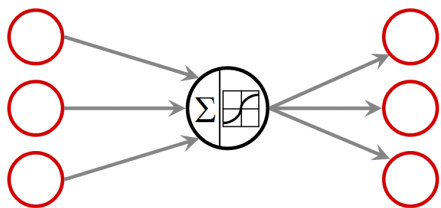
$$C_m \frac{dV_m}{dt} + \sum I_{\text{ion}} = I_{\text{ext}} \quad (1)$$

where C_m is the membrane capacitance, V_m is the membrane potential, I_{ion} is the net ionic current flowing across the membrane, and I_{ext} is an externally applied current, and

$$I_{\text{ion}} = \sum_k I_k = \sum_k G_k (V_m - E_k) \quad (2)$$

Simulated Neural Networks

A simulated neural network contains the same elements:



Neurons sum up inputs and put them through a **transfer function**.

Synapses are represented by real number weights, which modify the signals.

Axons/Dendrites are wired together such that a structured system arises (often layered).

Did you notice... we slowly moved away from spiking *neuronal* networks to real-valued *neural* networks? The former are often used to explain biological phenomena; the latter are used to compute results. However, there are, of course, “computing” spiking networks!

1949: Hebb learning rule

basic biological synapse learning rule, formulated by Hebb in 1949:

$$\Delta w_{ij} = \alpha a_i a_j$$

in words, “cells that fire together, wire together.”

There is ample biological evidence that various (low life form) learning methods are based on Hebbian learning. . . but there are other effects (LTP; LTD; . . .)

It explains *associative learning*, and has been implemented in the *Hopfield network* (later!)

1957: Rosenblatt's perceptron

in Rosenblatt's perceptron, the network output is

$$\mathcal{F}(\mathbf{x}) = \phi \left(\sum_i w_i x_i \right) \quad (3)$$

with

$$\phi(x) = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{otherwise} \end{cases} \quad (4)$$

perceptron learning rule:

1. choose a random \mathbf{w}_0
2. select an input vector (\mathbf{x}_i, z_i) with $z_i \in \{-1, 1\}$
3. if $\mathcal{F}(\mathbf{x}_i) \neq z_i$ then $\mathbf{w}_{i+1} = \mathbf{w}_i + z_i \mathbf{x}_i$
4. go back to 2.

This is modelled after the Hebb rule, but here no weights are changed when the classification is correct. Convergence can be proven, but there are better methods.

1960: Widrow and Hoff's delta rule

they introduced the *adaline* (ADAPtive LInear NEuron; later: ADAPtive LInear Element), very similar to the perceptron, but:

- ▶ the nonlinear transfer function ϕ is the identity function:

$$\mathcal{F}(\mathbf{x}) = \sum_i w_i x_i \quad (5)$$

- ▶ learning is done with the delta rule

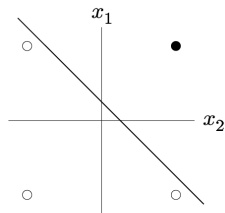
delta rule: (aka Widrow-Hoff rule or Least-Mean Square)

1. choose a random \mathbf{w}_0
2. select an input vector (\mathbf{x}_i, z_i)
3. $\mathbf{w}_{i+1} = \mathbf{w}_i + \alpha[z_i - F(\mathbf{x}_i)]\mathbf{x}_i$
4. go back to 2.

we know α as the learning rate

the 1969 crisis: Minsky & Papert

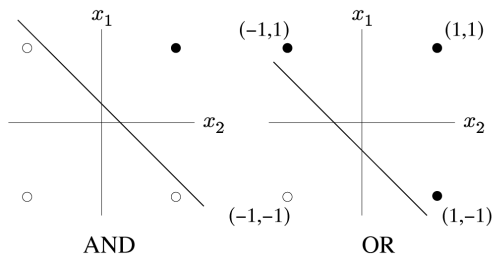
in 1969, Minsky and Papert showed that the adaline could only separate linearly. As an example, the XOR problem was given: it cannot be solved with linear classification.



AND

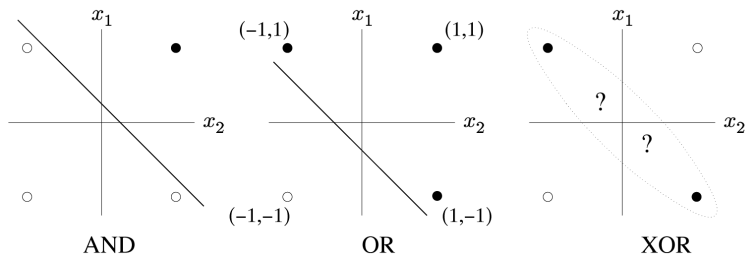
the 1969 crisis: Minsky & Papert

in 1969, Minsky and Papert showed that the adaline could only separate linearly. As an example, the XOR problem was given: it cannot be solved with linear classification.



the 1969 crisis: Minsky & Papert

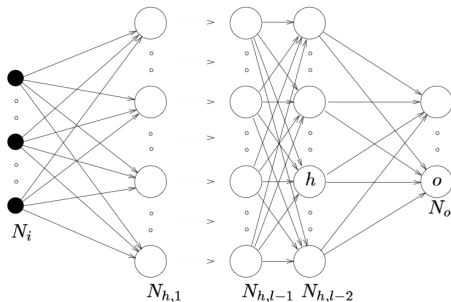
in 1969, Minsky and Papert showed that the adaline could only separate linearly. As an example, the XOR problem was given: it cannot be solved with linear classification.



the 1969 crisis: Minsky & Papert

in 1969, Minsky and Papert showed that the adaline could only separate linearly. As an example, the XOR problem was given: it cannot be solved with linear classification.

they also showed a solution to the problem: multi-layered perceptrons can solve that. However, a general rule to optimally find the weights \mathbf{w} was not discovered until 1974 (Paul Werbos) or 1985 (LeCun) and 1986 (Rumelhart *et al.*): **back propagation**.

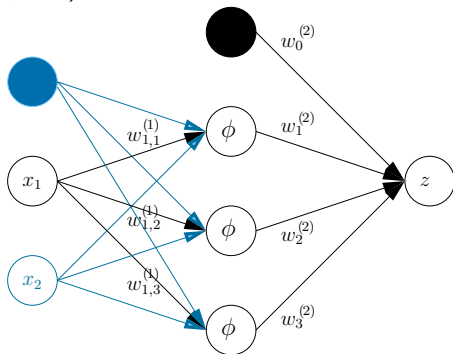


the multi-layered perceptron

we can compactly write the above network with *one hidden layer* as

$$\mathcal{M}(\mathbf{w}, \mathbf{x}) = \mathbf{w}^{(2)} \cdot \phi(\mathbf{w}^{(1)} \mathbf{x}) \quad (6)$$

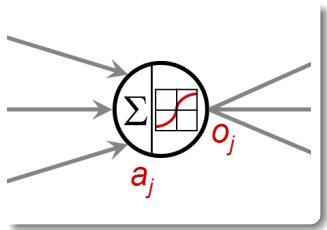
where $\mathbf{w} = (\mathbf{w}^{(1)}, \mathbf{w}^{(2)})$



the transfer function

The response of a simulated neuron to its activation a_j is given by a **transfer function**

$$o_j = \phi_j(a_j) \quad (7)$$

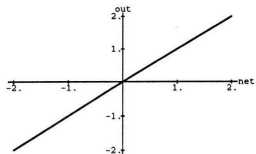


In the simplest case, the transfer function can be selected to be the identity function, or another linear function. However, in order for the network to exhibit nonlinear properties, $\phi(\cdot)$ also has to be nonlinear.

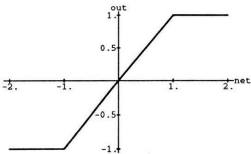
the transfer function

- ▶ historically, binary and piecewise linear transfer functions were chosen first, in reference to the threshold-like behaviour of biological neurons
- ▶ smooth, S-shaped (=sigmoid) functions behave like threshold or linear functions, depending on scaling, and can be differentiated
- ▶ another network class, radial basis function (RBF) networks, uses Gaussian-shaped transfer functions.

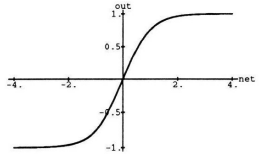
commonly used transfer functions



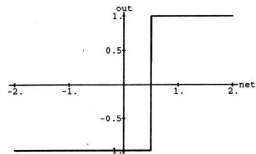
linear



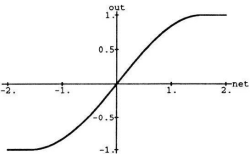
piecewise linear



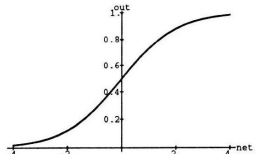
tanh(x)



threshold



sin(x) till saturation



logistic $1/(1+\exp(-x))$

this is, in fact, an optimisation problem

a feed-forward network is a special form of the more general model

$$\mathcal{M}^0(\mathbf{x}, \mathbf{w}) = \sum_j \phi_j(\mathbf{w}, \mathbf{x}) \quad (8)$$

of which we must minimise an error

$$E = \sum_{(\mathbf{x}_k, \mathbf{z}_k) \in \mathcal{D}} \|\mathbf{z}_k - \mathcal{M}^0(\mathbf{w}, \mathbf{x}_k)\|$$

How can this be minimised? Standard way uses...

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha \nabla E$$

How do we compute the gradient of E ? This is what back propagation is about.

Symbol juggling: we define $a_j := \sum_i w_{ji} x_i$ and write

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \quad (9)$$

now, when we define δ and can derive a_j towards x_i :

$$\delta_j := \frac{\partial E}{\partial a_j}, \quad \frac{\partial a_j}{\partial w_{ji}} = x_i \quad \implies \frac{\partial E}{\partial w_{ji}} = \delta_j x_i \quad (10)$$

(this is the delta rule we've seen above!)

backprop aka generalised delta rule

what is δ ? we apply the same trick as before. the output of a neuron is

$$o_k = \phi(a_k) \quad (11)$$

such that we can write

$$\delta_k = \frac{\partial E}{\partial a_k} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial a_k} \quad (12)$$

but of course we know that the latter term is

$$\frac{\partial o_k}{\partial a_k} = \phi'(a_k) \quad (13)$$

For a $w^{(2)}$ (an output neuron k) it's simple:

$$\delta_k = [z - \mathcal{M}(\mathbf{w}, \mathbf{x})] \phi'(a_k) \quad (14)$$

we often take the *output* activation function to be linear $\phi(x) = x$

δ for a hidden unit

now here's the "trick": we compute the delta's for the hidden units by *propagating back* the deltas of the layer l above:

$$\frac{\partial E}{\partial o_k} = \sum_l \frac{\partial E}{\partial a_l} \frac{\partial a_l}{\partial o_k} \quad (15)$$

now, that first term is the δ_l of the "layer above". Furthermore

$$a_l = \sum_k w_{lk} o_k \quad (16)$$

so that the second term of (15) is just w_{lk} ! intuitively, we *scale* the higher deltas over the weights and propagate them back

algorithm for backprop (“on-line” aka “stochastic” learning)

back-propagation algorithm:

initialize the weights

repeat

for each training sample (\mathbf{x}, \mathbf{z}) **do**

begin

$\mathbf{o} = \mathcal{M}(\mathbf{w}, \mathbf{x})$; forward pass

 calculate error $\mathbf{z} - \mathbf{o}$ at the output units

for all $w^{(2)}$ compute $\delta_{w^{(2)}}$; backward pass

for all $w^{(1)}$ compute $\delta_{w^{(1)}}$; backward pass continued

 update the weights using $\partial E / \partial w_{ij} = \delta_j x_i$

end

 (this is called one epoch)

until stopping criterion satisfied

better? algorithm for backprop (“batch learning”)

back-propagation algorithm:

initialize the weights

repeat

for each training sample (\mathbf{x}, \mathbf{z}) **do**

begin

$\mathbf{o} = \mathcal{M}(\mathbf{w}, \mathbf{x})$; forward pass

 calculate error $\mathbf{z} - \mathbf{o}$ at the output units

for all $w^{(2)}$ compute $\delta_{w^{(2)}}$; backward pass

for all $w^{(1)}$ compute $\delta_{w^{(1)}}$; backward pass continued

 sum the delta weights using $\partial E / \partial w_{ij} = \delta_j x_i$

end

 update the weights using summed delta weights

until stopping criterion satisfied

summary of learning rules

- ▶ Hebb rule: $\Delta w_{ij} = \alpha a_i a_j$
- ▶ perceptron learning rule: $\Delta w_i = \alpha z_i x_i$
- ▶ delta rule: $\Delta \mathbf{w} = \alpha \left[z - \sum_j w_j x_j \right] \mathbf{x}$
- ▶ generalised delta rule: $\Delta \mathbf{w} = -\alpha \nabla E$

basically, nothing else than nonlinear optimisation

the “better” algorithm just computes a gradient, and we can use all well-known optimisation tools to improve on the learning algorithm. A few notes:

- ▶ tale-telling: it lasted until the mid-1980s for this algorithm to become widespread
- ▶ where is the analogy with biology? **correct answer: don't look for it**
- ▶ the on-line learning algorithm introduces stochasticity by adding many small gradients, one for each sample
- ▶ the on-line learning algorithm allows for *incremental learning*
- ▶ the on-line learning algorithm does *not* allow for 2nd-order methods, because we need decent gradients!
- ▶ watch out with vanishing gradients (t.b.c.)

some good and some bad news

universal approximation

an MLP with at least two layers of weights can approximate arbitrarily well any given mapping from one finite input space with a finite number of discontinuities to another, if we have enough hidden units [Cybenko, 1989; Hornik, 1991].

parameter finding

finding the optimum set of weights \mathbf{w} for an MLP is an NP-complete problem, i.e. it cannot be solved exactly within a time $t \propto |\mathbf{w}|^x$ [Blum et al., 1992].

the name of the game

feed-forward neural network (FFNN), neural network, multi-layer perceptron (MLP)

back-propagation (BP), backprop, generalised delta rule

FFNN's have been used in many areas, including process control, classification, stock exchange prediction, diagnosis, robot control, sensory data fusion, etc. however, since their training is cumbersome and cannot be guaranteed, after their hype (1986–1995) they lost in popularity.