Neural Networks

Machine Learning I, Week 8

Sibylle Mueller

Based on a web lecture by Nicol Schraudolph and Fred Cummins

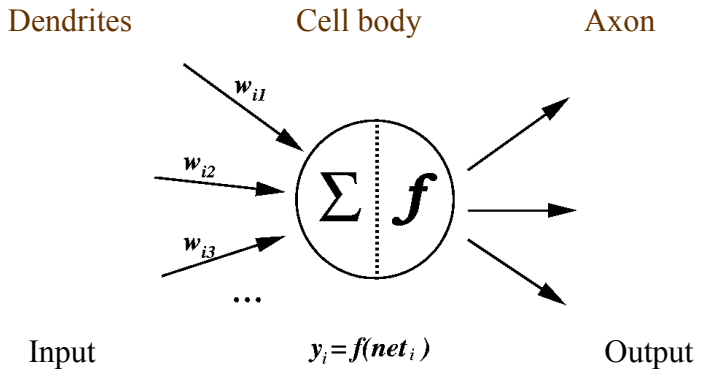
http://www.icos.ethz.ch/teaching/NNcourse

Why do we Simulate Neural Networks?

The brain is a network of neurons, forming a massively parallel information processing system.

	Brain	Computer
Speed	100 Hz	1 GHz
Computation	Parallel, distributed	Serial, centralized
Fault tolerance	yes	no
Learning	yes	?

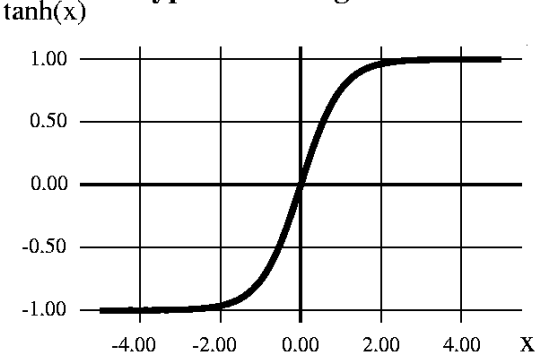
Neurons are Elements of Neural Networks



Activation Functions

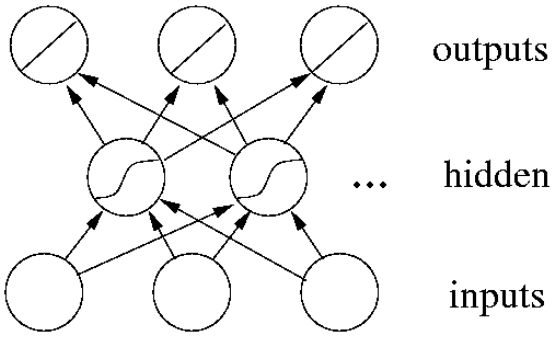
f is a monotonic increasing, limited function:

1. **Sigmoid:** **hyperbolic tangent function**



2. **Logistic Sigmoid: $1/(1+e^{-x})$**
3. **Linear (identity function: input = output)**
4. **Binary function (y = 0 for x < 0, and y = 1 for x > 0)**

Layer structure

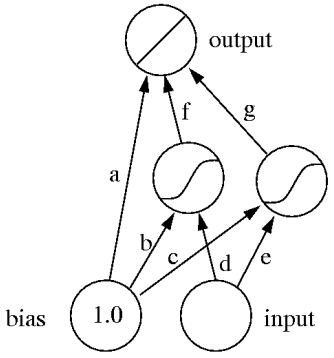


Usually output neurons have linear activation functions

This neural network is called layered feedforward net

Bias in Neural Nets

Neurons have also biases as an additional input component



This representation for the bias is useful because bias terms can be interpreted as additional weights

Learning through presenting data sets

Task: given a set of N input/output vectors, (x_i, t_i)
find a model $t_i = g(x_i)$ for the data
A least squares solution is sought for:

$$\min_W E(W) = \frac{1}{2}\sum_o [y_o(W) - t_o]^2$$

How can we minimize the error wrt the weights

- in linear networks? → relatively simple
- in multi-layer networks? → backpropagation algorithm

Online and Offline Learning

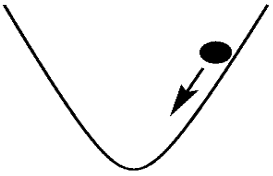
Online learning means a weight update after each pattern.
→ Order in which learning samples are presented plays a role.

A **training epoch** is said to be passed when the backprop has been applied one time to all the p examples

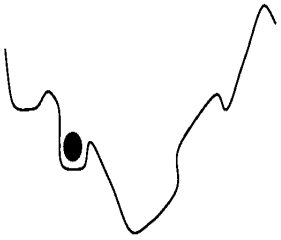
Offline / Batch learning is a variant of the backprop in which the weights variations are accumulated during a training epoch, and the network weights are actually updated at the end of a training epoch.
→ Order in which learning samples are presented does not play a role.

Local Minima

Gradient descent algorithms (such as backprop) suffer from the problem of local minima



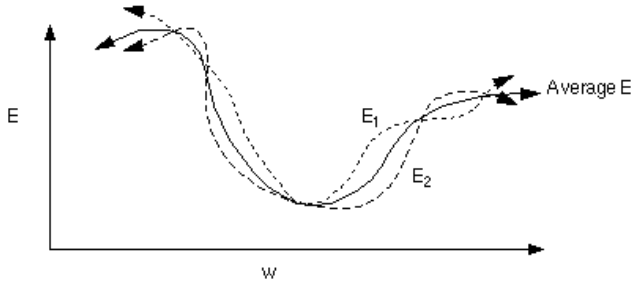
"Good" function: the gradient vanishes only in the desired goal



"Bad" function: the gradient vanishes in a local minimum, no further progress is possible

Solution 1: use online learning

The noise introduced in the error surface by online learning can by itself be sufficient to avoid local minima



Drawback: because of the random oscillation introduced, online learning is usually slower that offline learning

Solution 2: use a momentum term

With a momentum m, the weight update at a given time t becomes

$$\Delta w_{ij}(t) = \mu_i \delta_i y_j + m \Delta w_{ij}(t-1)$$

With $0 < m < 1$ (m has to be found by trial and error)

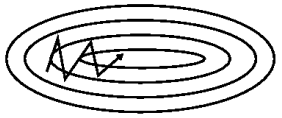
Working principle: when the gradient keeps pointing in the same direction, the adaptation steps can be increased. When in a local minimum, the first term will vanish but there will be still some contribution from the cumulated (momentum) changes

Warning! m too large with learning rate too large can make Backprop rush away from a good minimum with large steps!

Momentum term regularizing effect

Neural net is ill-conditioned ⇔ elongated error landscapes

Error landscape for a badly conditioned neural net, and path followed by backprop without momentum



The same landscape, with the path followed by backprop with momentum included