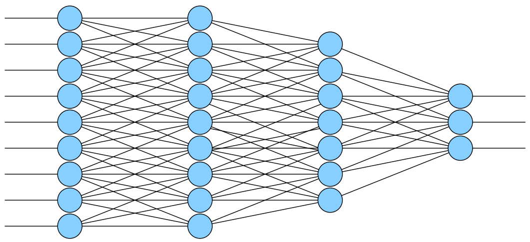
Artificial Neural Networks

Artificial Neural Networks (ANN)

- Inspired from the nervous system
- Parallel processing

We will focus on one class of ANNs:

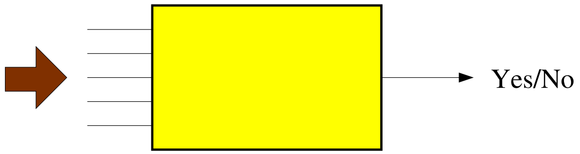
Feed-forward Layered Networks



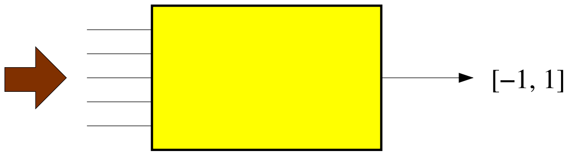
Applications

Operates like a general "Learning Box"!

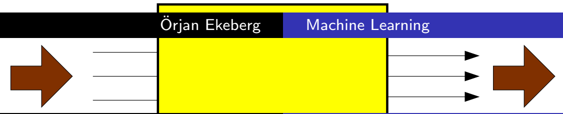
Classification



Function Approximation



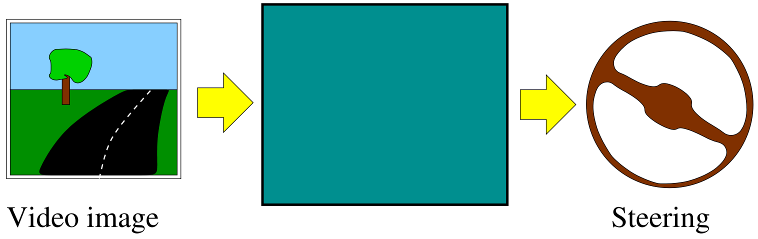
Multidimensional Mapping



Classical Examples

ALVINN

Autonomous driving

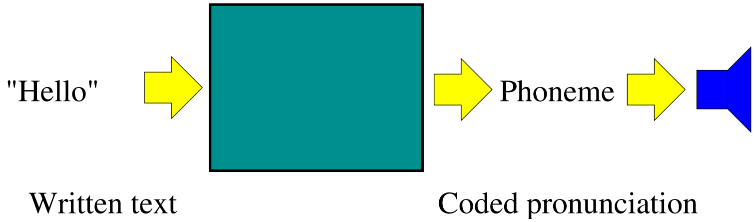


Trained to mimic the behavior of human drivers

Classical Examples

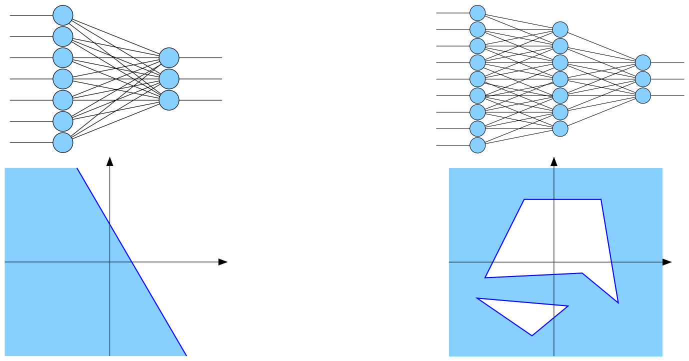
NetTalk

Speech Synthesis



Trained using a large database of spoken text

What is the point of having multiple layers?



A two layer network can implement arbitrary decision surfaces
...provided we have *enough hidden units*

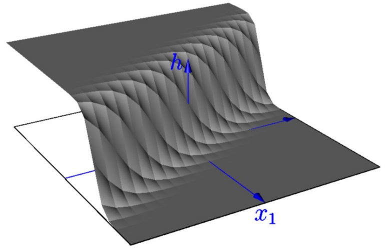
How can we train a multi layer network?

Neither perceptron learning, nor the delta rule can be used

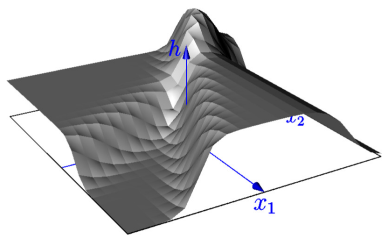
Fundamental problem:

When the network gives the wrong answer
there is no information on in which direction
the weights need to change to improve the result

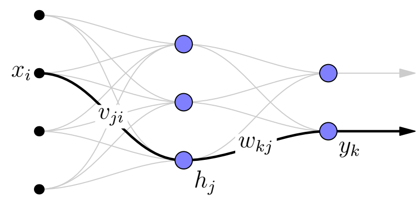
Trick:
Use threshold-like, but continuous functions



First layer response



Sum of base functions



Learning strategy:

Minimize the error (E) as a function of all weights ($\vec{w}$)

1. Compute the direction in weight space where the error increases the most $\mathrm{grad}_{\vec{w}}(E)$
2. Change the weights in the opposite direction

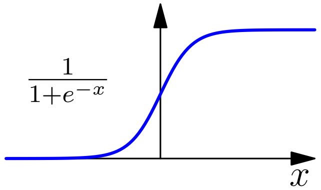
$$w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}$$

Normally one can use the error from each example separately

$$E = \frac{1}{2}\sum_{k\in\text{Out}}(t_k - o_k)^2$$

A common "threshold-like function" is

$$\rho(y) = \frac{1}{1+e^{-y}}$$



The gradient can be expressed as a function of a *local generalized error* δ

$$\frac{\partial E}{\partial w_{ji}} = -\delta_i x_j \qquad w_{ji} \leftarrow w_{ji} + \eta\delta_i x_j$$

Output layer:

$$\delta_k = o_k \cdot (1 - o_k) \cdot (t_k - o_k)$$

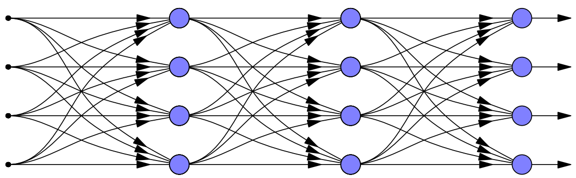
Hidden layers:

$$\delta_h = o_h \cdot (1 - o_h) \cdot \sum_{k\in\text{Out}} w_{kh}\delta_k$$

The error δ propagates backwards through the layers
Error backpropagation (*BackProp*)

Things to think about when using BackProp

- Sloooow
  Normal to require thousands of iterations through the dataset
- Gradient following
  Risk of getting stuck in local minima
- Many parameters
  - Step size η
  - Number of layers
  - Number of hidden units
  - Input and output representation
  - Initial weights



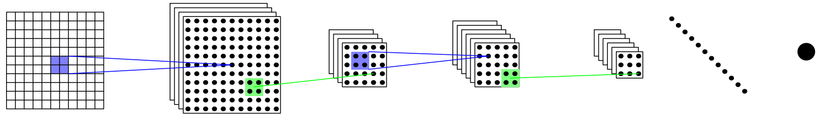
Deep networks — Networks with many layers

- Error gradients become smaller from layer to layer
- Backprop becomes unusable for deep networks

Deep Belief Networks

- Unsupervised learning of features
- Greedy learning from the bottom, layer by layer
- Supervised Backprop to finalize classifier

Convolutional Networks



- Alternating convolution and subsamping layers
- Weight sharing
- Trained using Backprop