Artificial Neural Networks

Feed Forward Networks

- Applications
- Classical Examples

Multi Layer Networks

- Possible Mappings
- Backprop Algorithm
- Practical Problems

Deep Networks

- Vanishing Gradients
- Convolutional Networks

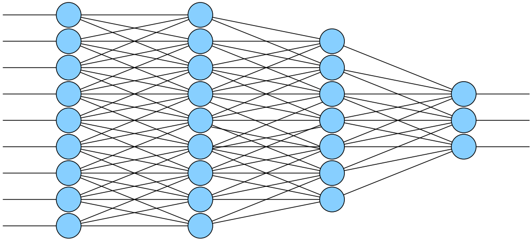
- Feed Forward Networks
  - Applications
  - Classical Examples
- Multi Layer Networks
  - Possible Mappings
  - Backprop Algorithm
  - Practical Problems
- Deep Networks
  - Vanishing Gradients
  - Convolutional Networks

Artificial Neural Networks (ANN)

- Inspired from the nervous system
- Parallel processing

We will focus on one class of ANNs:

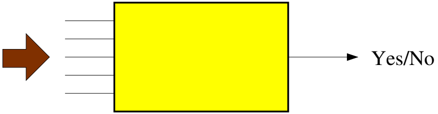
Feed-forward Layered Networks



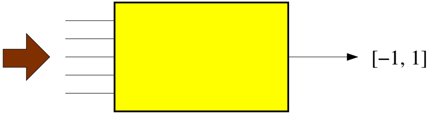
Applications

Operates like a general ”Learning Box”!

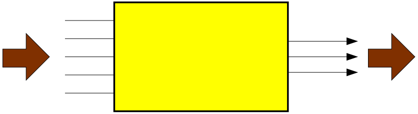
Classification



Function Approximation



Multidimensional Mapping



Classical Examples

ALVINN

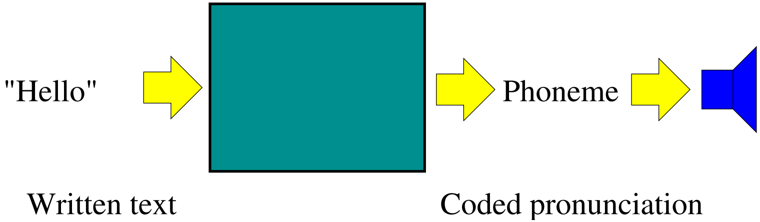
Autonomous driving

Trained to mimic the behavior of human drivers

Classical Examples

NetTalk

Speech Synthesis



Trained using a large database of spoken text

Feed Forward Networks

- Applications
- Classical Examples

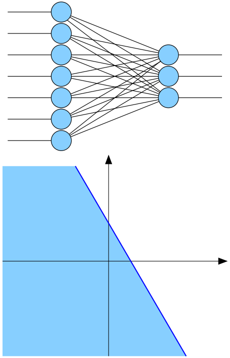
Multi Layer Networks

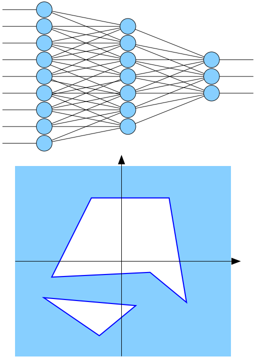
- Possible Mappings
- Backprop Algorithm
- Practical Problems

Deep Networks

- Vanishing Gradients
- Convolutional Networks

What is the point of having multiple layers?





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

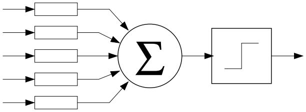
How can we train a multi layer network?

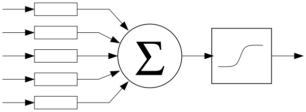
Neither perceptron learning, nor the delta rule can be used

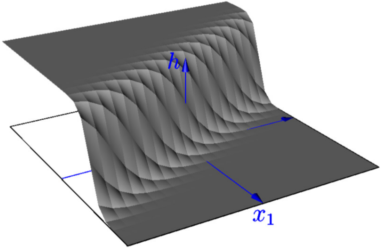
Fundamental problem:

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

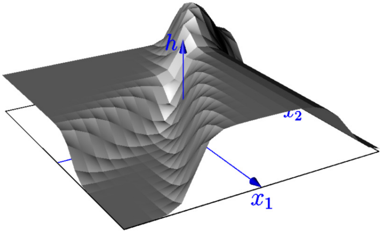
Trick:
Use threshold-like, but continuous functions



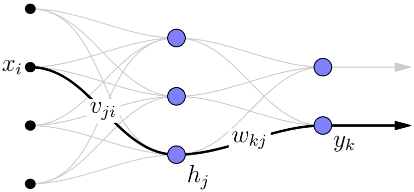




First layer response



Second layer response



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

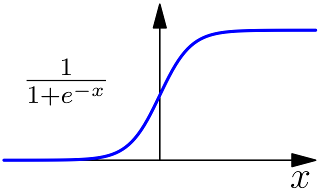
$$w_i \leftarrow w_i - \eta \frac{\partial E}{\partial w_i}$$

Normally one can use the error from each example separately (Stochastic Gradient Descent)

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

A common "threshold-like function" is

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



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:

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

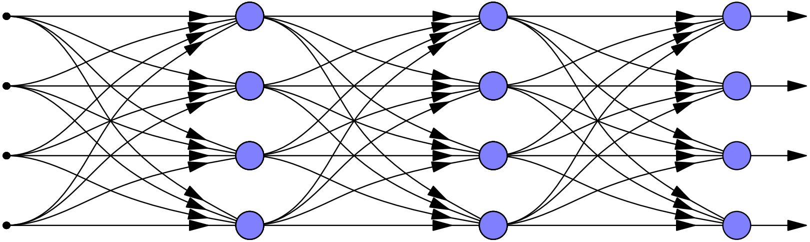
Hidden layers:

$$\delta_j = h_j \cdot (1 - h_j) \cdot \sum_k w_{kh} \delta_k$$

The errors δ propagate 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



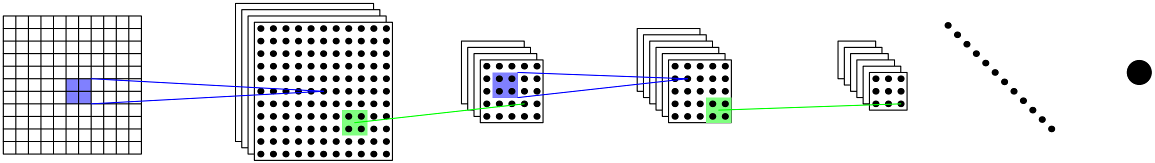
Deep networks — Networks with many layers

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

Deep Belief Networks

- Unsupervised learning of features
  Restricted Boltzmann Machine
- Greedy learning from the bottom, layer by layer
  Optimize for ability to reconstruct previous layer
- Supervised Backprop to finalize classifier

Convolutional Networks



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