9a. Neural Networks

New lectures
Didn’t change the canonical numbering yet
Pigeons as art experts (Watanabe et al. 1995)

Experiment:
- Pigeon in Skinner box
- Present paintings of two different artists (e.g. Chagall / Van Gogh)
- Reward for pecking when presented a particular artist (e.g. Van Gogh)

This section is based on slides by Torsten Reil
Pigeons were able to discriminate between Van Gogh and Chagall with 95% accuracy (when presented with pictures they had been trained on)

Discrimination still 85% successful for previously unseen paintings of the artists

Pigeons do not simply memorise the pictures
They can extract and recognise patterns (the style )
They generalise from the already seen to make predictions

This is what neural networks (biological and artificial) are good at (unlike conventional computer)
Feed-forward Network Functions

also know as multilayer perceptrons (MLP)
Warning

Some type of NNs try to model biological systems
But: biological realism imposes unnecessary constraints
We want to model data!

Airplanes don't flap their wings!
Neuron vs. Node

$\begin{align*}
x_1 & \quad \text{Input} \\
x_2 & \\
x_3 & \\
x_4 & \\
\end{align*}$

$\begin{align*}
\text{Node} & \\
\text{sum + squash} & \\
\end{align*}$

$\begin{align*}
y_1 & \quad \text{Output} \\
\end{align*}$
Formalization of a neuron

Linear combination of D inputs $x_i$

$$a_j = \sum_{i=1}^{D} w_{ji} x_i + w_{j0}$$

$w_{ji}$ : weights

$w_{j0}$ : biases

Activation function

$$z_j = h(a_j)$$
Popular activation functions

\[ y_k = \tanh(a_k) \]

Logistic sigmoid function

\[ y_k = \frac{1}{1 + \exp(a_k)} \]

Step (heaviside) function

\[ y_k = \text{Heaviside}(a_k) \]
Exercise: what does this NN do?

Which type of decision boundary do we get?

Classify as $C_1$ or $C_2$

Rule: classify as $C_2$ if $z > 0$
Two layer feed forward network
Feeding data through the net:

\[(1 \times 0.25) + (0.5 \times (-1.5)) = 0.25 + (-0.75) = -0.5\]

Squashing: \[\frac{1}{1 + e^{-0.5}} = 0.3775\]
Example for a general feed forward neural network
Example of classification problems

Green: true decision boundary
Red: neural network
2 layers
2 hidden units
tanh-activation on hidden layer
Log-sig-activation on output
Expressive Power of multi-layer Networks

**Question:**
Can every decision be implemented by a three-layer network?

**Answer:**
Yes (due to A. Kolmogorov)

Any continuous function from input to output can be implemented in a three-layer net, given sufficient number of hidden units $n_H$, proper nonlinearities, and weights.

**Unfortunately:**
Kolmogorov's theorem tells us very little about how to find the nonlinear functions based on data; this is the central problem in network-based pattern recognition.
Network Training
Let $t_n$ be the $n$-th target (or desired) output and $y(x_n, w)$ be the $n$-th computed output with $n = 1, \ldots, N$ and $w$ represents all the weights of the network.

The training error:

$$E(\hat{w}) = \frac{1}{2} \sum_{n=1}^{N} (y(\bar{x}_n, \hat{w}) - t_n)^2$$
Key Idea: Gradient Descent

Backpropagation

- Requires training set (input / output pairs)
- Starts with small random weights
- Error is used to adjust weights (supervised learning)
  - Gradient descent on error landscape
Is there a unique set of parameters?

Suppose you have the optimal solution. Is there a second equivalent one?
Gradient Descent

**learning rule is based on gradient descent**

The weights are initialized with pseudo-random values and are changed in a direction that will reduce the error:

\[
\omega^{(\tau+1)} = \omega^{(\tau)} + \Delta \omega^{(\tau)}
\]

\[
\Delta \omega^{(\tau)} = -\eta \frac{\partial E(\omega^{(\tau)})}{\partial \omega^{(\tau)}}
\]

**Problem:** how to calculate the gradient
Calculating the Gradient 1: Finite Differences

Asymmetric difference

\[
\frac{\partial E(w_{ji})}{\partial w_{ji}} = \frac{E(w_{ji} + \varepsilon) - E(w_{ji})}{\varepsilon} + O(\varepsilon)
\]

Symmetric central difference

\[
\frac{\partial E(w_{ji})}{\partial w_{ji}} = \frac{E(w_{ji} + \varepsilon) - E(w_{ji} - \varepsilon)}{2\varepsilon} + O(\varepsilon^2)
\]

Use a sufficiently small \( \varepsilon \)

But: there can be issue with numerical stability

Effort!
Back Propagation

Propagates the error back to each node
Exact calculation of the derivative
Complexity: linear in number of weights
Example application of a feed forward network: ALVINN

Wikipedia:

**Mobile robot**
Milestones: 1995

Semi-autonomous ALVINN steered a car coast-to-coast under computer control for all but about 50 of the 2850 miles. Throttle and brakes, however, were controlled by a human driver.

From: http://www.nku.edu/~foxr/CSC625/nn-alvinn.jpg
Hopfield Networks

Sub-type of recurrent neural nets

- Fully recurrent
- Weights are symmetric
- Nodes can only be on or off
- Random updating

Learning: **Hebb rule**

Can recall a memory, if presented with a corrupt or incomplete version
Elman Nets

*Elman nets* are feed forward networks with partial recurrency

Unlike feed forward nets, Elman nets have a *memory* or *sense of time*
Classic experiment on language acquisition and

Task
Elman net to predict successive letters in sentences.

Data
Suite of sentences, e.g.
The boy catches the ball.
The girl eats an apple.
Letters are input one at a time

Representation
Binary representation for each letter, e.g.
0-1-1-0 for \( m \)

Training method
Backpropagation
Summary

Simplified neurons

Feed forward networks
  Can model any decision boundary in principle
  Training: back propagation

Other networks
  Hopfield
  Elman