Introduction to Artificial Neural Networks

Kristof Van Laerhoven
kristof @ mis.tu-darmstadt.de
http://www.mis.informatik.tu-darmstadt.de/kristof
Course Overview

• We will:
  - Learn to interpret common ANN formulas and diagrams
  - Get an extensive overview of the major ANNs
  - Understand mechanisms behind ANNs
  - Introduce it with historic background

• Left as exercises:
  - Implementations of algorithms and details
  - Evaluations and proofs
The Start of Artificial Neural Nets

• Standard ANN introduction courses just mention:

‘real neuron’… ‘simple model’… ‘some authors’…

• But let’s give a few more details…
before 1943: Neurobiology

- Trepanation: Brain = ‘cranial stuffing’
- ~400BC: Hippocrates:
  - Brain = “Seat of intelligence”
- ~350BC: Aristotle: Brain cools blood
- 160: Galen: Brain damage ~ mental functioning
- 1543: Vesalius: anatomy
- 1783: Galvani:
  - electrical excitability of muscles, neurons
before 1943: Neurobiology

- 1898: Camillo Golgi: Golgi Stains
- 1906: Golgi + Santiago Ramón y Cajal: Nobel prize
- ~1900: DuBois-Reymond, Müller, and von Helmholtz:
  - neurons electrically excitable
  - their activity affects electrical state of adjacent neurons
before 1943: Neurobiology

‘real’ neurons...

“One would assume, I think, that the presence of a theory, however strange, in a field in which no theory had previously existed, would have been a spur to the imagination of neurobiologists. But this did not occur at all! The whole field of neurology and neurobiology ignored the structure, the message, and the form of McCulloch’s and Pitts’s theory. Instead, those who were inspired by it were those who were destined to become the aficionados of a new venture, now called Artificial Intelligence, which proposed to realize in a programmatic way the ideas generated by the theory” -- Lettvin, 1989

• In 1943 there already existed a lively community of biophysicists doing mathematical work on neural nets
• Dissections, Golgi stains, and microscopes, but not: Church-Turing for the brain
Back to McCulloch-Pitts

- **Warren McCulloch:**
  - Neurophysiologist (philosophy, psychology, MD)
  - The ‘McCulloch group’

- **Walter Pitts**
  - Self-taught logic and maths, Principia @12y
  - Homeless, non-enrolled as student, eccentric
  - Manuscript on 3D networks

- **Their seminal 1943 paper:**
  - Mathematical (logic) model for neuron
  - Nervous system can be considered a kind of universal computing device as described by Leibniz

» Recommended Reading: *Dark Hero Of The Information Age: In Search of Norbert Wiener, the Father of Cybernetics*, and search the web for Jerome Lettvin
(Hebbian) Learning

• Previously:
  - Pavlov, 1927: Classical Conditioning
  - Thorndike, 1898: Operant Conditioning

• Donald Hebb (psychology, teacher)

  *The Organization of Behavior*

  What fires together, wires together: “When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased”  -- D. Hebb

• Others: Steven Grossberg, Gail Carpenter: Adaptive Resonance Theory
The Perceptron

\[ y = \sum_{i=1,2,3,...} (w_i x_i) + b \]

- \(x\): inputs
- \(w\): weights
- \(b\): bias
- \(0 < \alpha < 1\): learning rate
- \(\text{Sign}(y)\): output (usually -1 or 1)
- \(y_T\): target output (usually -1 or 1)

\[ y = \text{Sign}(\sum_{i=1,2,3,...} (w_i x_i) + b) \]

- 1957: Frank Rosenblatt “The perceptron: A perceiving and recognizing automaton (project PARA).”
The Perceptron

The Perceptron is a fundamental model in artificial neural networks. It is a linear classifier that makes a prediction based on a set of inputs.

\[ y = \sum_{i=1,2,3,...} (w_i x_i) + b \]

- \( x \): inputs
- \( w \): weights
- \( b \): bias
- \( 0 < \alpha < 1 \): learning rate
- \( \text{Sign}(y) \): output
- \( y_T \): target output

**Learning / Training Phase**: *(Widrow-Hoff, or Delta Rule)*

1. Start with random \( w_1, w_2, w_3, \ldots, b \) (usually near 0)
2. Calculate \( y \) for the current input \( x_1, x_2, x_3, \ldots \)
3. Update weights and bias: 
   \[ w_i' = w_i + \alpha (y_T - y) x_i \]
4. Go to step 2
The Perceptron: Example

- Teach the neuron to spot male authors of emails on a mailing list, given two inputs:
  - \( x_1 \): counts of {'around', 'what', 'more'} minus counts of {'with', 'if', 'not'}
  - \( x_2 \): counts of {'at', 'it', 'many'} minus counts of {'myself', 'hers', 'was'}

- Learning by examples:
  - Negative samples: 
  - Positive samples: 

- (loosely based on [http://www.bookblog.net/gender/genie.php](http://www.bookblog.net/gender/genie.php) and "Gender, Genre, and Writing Style in Formal Written Texts", Shlomo Argamon, Moshe Koppel, Jonathan Fine, Anat Rachel Shimoni)
The Perceptron: Example

• Teach the neuron to spot male authors of emails on a mailing list, given two inputs
  - $x_1$: counts of ‘around’, ‘what’, ‘more’} minus counts of {'with’,’if’,’not’}
  - $x_2$: counts of {'at’,’it’,’many’} minus counts of {'myself’,’hers’, ‘was’}

• Learning by examples:
  - Negative samples: ●
  - Positive samples: ○

• (loosely based on http://www.bookblog.net/gender/genie.php and “Gender, Genre, and Writing Style in Formal Written Texts”, Shlomo Argamon, Moshe Koppel, Jonathan Fine, Anat Rachel Shimoni)
The Perceptron: Example

\[ y = \sum_{i=1,2,3,...} (w_i x_i) + b \]

\[ y = \frac{1}{2} x_1 - \frac{1}{2} x_2 + 0 = 0 \]

Initial Values:

\[ w_1 = 1/2 \]
\[ w_2 = -1/2 \]
\[ b = 0 \]

\[ y = \frac{1}{2} x_1 + (\frac{-1}{2}) x_2 + 0 \]
The Perceptron: Example

\[ y = \sum_{i=1,2,3,...} (w_i x_i) + b \]

Values after training for all \( \bullet \) and \( \circ \) (with \( \alpha = 0.3 \)):

- \( w_1 = \frac{2}{5} \)
- \( w_2 = -\frac{1}{3} \)
- \( b = -\frac{2}{7} \)

\[ y = \frac{2}{5} x_1 - \frac{1}{3} x_2 - \frac{2}{7} = 0 \]
The Perceptron: Example

\[ y = \sum_{i=1,2,3,...} (w_i x_i) + b \]

Values after training 5 times for all \( \bullet \) and \( \circ \) (with \( \alpha = 0.3 \)):

\[ w_1 = \frac{9}{12} \]
\[ w_2 = -\frac{2}{23} \]
\[ b = -2 \]

\[ y = \frac{9}{12} x_1 - \frac{2}{23} x_2 - 2 = 0 \]
The Perceptron’s Delta Rule

- **Delta Rule or Gradient Descent:**
- **Error:** \( E = (y_T - y)^2 / 2 \)
- **Calculate partial derivative of the Error w.r.t. each weight:**
  \[
  \frac{\partial E}{\partial w_1} = -(y_T - y)x_1
  \]

\[
\Delta w_1 = w'_1 - w_1 = \alpha(y_T - y)x_1
\]

Local Minima only!
The Perceptron

- Perceptron has 2 phases:
  - Training/Learning Phase: adaptive weights
  - Testing Phase: for checking performance and generalisation
- Neurons and their outputs are grouped in 1 layer:
Output Functions

- Output function per neuron: \( g(y) \) limits and scales output
  - Simple threshold
  - Sigmoid
  - Gaussian
  - Piecewise Linear
  - \[ g(x) = \begin{cases} 0 & \text{if } x < -1 \\ \frac{x+1}{2} & -1 < x < 1 \\ 1 & x > 1 \end{cases} \]

- \[ g(x) = \frac{1}{1 + e^{-x}} \]
- \( g(x) = \tanh(x) \)
Parameters

• Bias can be seen as weight to constant input 1, or input with constant weight 1
• Output parameters (e.g., steepness for sigmoid)
• Momentum term to avoid local minima
• Initialisation: Use a ‘Committee of networks’ to avoid random init effects (many networks training on same data, different initialised weights)
The Perceptron’s Issues

\[ y = \sum_{i=1,2,3,...} (w_i x_i) + b \]

  => Non-linearly separable functions not solvable!
Backpropagation

- **Network:** Multi-layer perceptron:
  - Same neurons, but:
    - ‘Hidden’ layers
  - Training propagates per layer, from output **back** to input weights

- P. Werbos in 1974
Backpropagation Test Phase

- Inputs are entered in network
Backpropagation Test Phase

- Inputs are entered in network
- Weighted sums calculate…

Diagram showing a neural network architecture with an input layer, a hidden layer, and an output layer.
Backpropagation Test Phase

- Inputs are entered in network
- Weighted sums calculate...
- Outputs of hidden layer neurons
Backpropagation Test Phase

- Inputs are entered in network
- Weighted sums calculate...
- Outputs of hidden layer neurons
- Which are weighted again to calculate…
Backpropagation Test Phase

- Inputs are entered in network
- Weighted sums calculate...
- Outputs of hidden layer neurons
- Which are weighted again to calculate...
- The outputs of the neurons in the output layer.. finally!
In the training phase, we also know the target outputs for the input...
• In the training phase, we also know the target outputs for the input…
• Which can be used to re-adjust the weights going to the output layer
In the training phase, we also know the target outputs for the input...

Which can be used to re-adjust the weights going to the output layer

Which lead to target outputs for the middle layer neurons
Backpropagation Training

- In the training phase, we also know the target outputs for the input...
- Which can be used to re-adjust the weights going to the output layer
- Which lead to target outputs for the middle layer neurons
- Which re-adjust the remaining weights (just like in the perceptron)
Backpropagation

• XOR works!

• Note: This network could be re-trained for OR, AND, NOR, or NAND
Backpropagation

• In formulas:
Multilayer Perceptron Apps

Multilayer Perceptron Apps

- ALVINN (Pomerleau, 1993)

- Image Compression using Backprop (Paul Watta, Brijesh Desaie, Norman Dannug, Mohamad Hassoun, 1996)
Underfitting and Overfitting

- Underfitting: ANN that is not sufficiently complex to correctly detect the pattern in a noisy data set
- Overfitting: ANN that is too complex so it reacts on noise in the data

- Techniques to avoid them:
  - Model selection
  - Jittering
  - Weight decay
  - Early stopping
  - Bayesian estimation
Overview so far...

• These are the most popular neural network architectures and their training algorithms:
  - **Perceptron** - *gradient descent / delta rule*
    - Only linearly separable functions
    - One layer
  - **Multilayer perceptron** - *backpropagation*
    - Hidden Layer(s)
    - More powerful
    - Training: errors are propagated back towards input weights
    - Many applications

• They are **SUPERVISED**: we train them with inputs plus known target outputs
The Kohonen Map

• Also: Self-Organising (Feature) Map
• Teuvo Kohonen (1982) “Self-organized formation of topologically correct feature maps”
• ‘Competition’ among neurons for the input:
• Topology among neurons
• Unsupervised

\[ w'_i = w_i + \alpha \eta (x_i - w_i) \]

neighbourhood learning rate
The Kohonen Map: Example

- Randomly initialised grid of (weight-) vectors:

<table>
<thead>
<tr>
<th>(72,19)</th>
<th>(22,93)</th>
<th>(24,91)</th>
<th>(31,26)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(35,102)</td>
<td>(95,67)</td>
<td>(30,11)</td>
<td>(64,19)</td>
</tr>
<tr>
<td>(12,19)</td>
<td>(83,36)</td>
<td>(16,13)</td>
<td>(24,74)</td>
</tr>
<tr>
<td>(93,10)</td>
<td>(72,56)</td>
<td>(75,55)</td>
<td>(61,42)</td>
</tr>
</tbody>
</table>

\[ w'_i = w_i + \alpha \eta (x_i - w_i) \]
neighbourhood learning rate
The Kohonen Map: Example

- Randomly initialised grid of (weight-)
  vectors:

- Input:

<table>
<thead>
<tr>
<th></th>
<th>(72,19)</th>
<th>(22,93)</th>
<th>(24,91)</th>
<th>(31,26)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(82,41)</td>
<td>(35,102)</td>
<td>(95,67)</td>
<td>(30,11)</td>
<td>(64,19)</td>
</tr>
<tr>
<td></td>
<td>(12,19)</td>
<td>(83,36)</td>
<td>(16,13)</td>
<td>(24,74)</td>
</tr>
<tr>
<td></td>
<td>(93,10)</td>
<td>(72,56)</td>
<td>(75,55)</td>
<td>(61,42)</td>
</tr>
</tbody>
</table>

\[ w'_i = w_i + \alpha \eta (x_i - w_i) \]

neighbourhood
learning rate
The Kohonen Map: Example

- Randomly initialised grid of (weight-) vectors:
  - Input:
    - $(82, 41)$
    - Minimum distance
    - $w'_i = w_i + \alpha \eta (x_i - w_i)$
    - Neighbourhood learning rate

<table>
<thead>
<tr>
<th></th>
<th>(72, 19)</th>
<th>(22, 93)</th>
<th>(24, 91)</th>
<th>(31, 26)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(35, 102)</td>
<td>(95, 67)</td>
<td>(30, 11)</td>
<td>(64, 19)</td>
<td></td>
</tr>
<tr>
<td>(12, 19)</td>
<td>(83, 36)</td>
<td>(16, 13)</td>
<td>(24, 74)</td>
<td></td>
</tr>
<tr>
<td>(93, 10)</td>
<td>(72, 56)</td>
<td>(75, 55)</td>
<td>(61, 42)</td>
<td></td>
</tr>
</tbody>
</table>
The Kohonen Map: Example

- ‘Winner Takes All’: $\eta \approx 1$ for closest neuron
- Step 1:

$$w'_i = w_i + \alpha \eta (x_i - w_i)$$

neighbourhood learning rate
The Kohonen Map: Example

- Neighbours: $\eta$ is smaller for surrounding
- Step 1:

$$w'_i = w_i + \alpha \eta (x_i - w_i)$$

neighbourhood learning rate
The Kohonen Map: Example

- Neighbours: $\eta$ gets smaller and smaller...
- Step 1:

$$w'_i = w_i + \alpha \eta (x_i - w_i)$$

neighbourhood learning rate
The Kohonen Map: Example

- Different inputs activate different regions
- Step 2: New input

\[ w'_i = w_i + \alpha \eta (x_i - w_i) \]

neighbourhood learning rate

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(74,21)</td>
<td>(26,89)</td>
<td>(27,91)</td>
<td>(38,27)</td>
<td></td>
</tr>
<tr>
<td>(41,77)</td>
<td>(91,52)</td>
<td>(53,20)</td>
<td>(68,22)</td>
<td></td>
</tr>
<tr>
<td>(80,28)</td>
<td>(82,39)</td>
<td>(34,21)</td>
<td>(32,69)</td>
<td></td>
</tr>
<tr>
<td>(88,29)</td>
<td>(78,46)</td>
<td>(77,37)</td>
<td>(63,41)</td>
<td></td>
</tr>
</tbody>
</table>
The Kohonen Map: Example

- Different inputs activate different regions

Step 2: New input:

<table>
<thead>
<tr>
<th>Input</th>
<th>Coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>(74,21)</td>
<td>(31,79)</td>
</tr>
<tr>
<td>(44,71)</td>
<td>(90,47)</td>
</tr>
<tr>
<td>(80,28)</td>
<td>(79,37)</td>
</tr>
<tr>
<td>(88,29)</td>
<td>(76,41)</td>
</tr>
</tbody>
</table>

\[ w'_i = w_i + \alpha \eta (x_i - w_i) \]

- neighbourhood
- learning rate
The Kohonen Map: Example

- The map organises along input space
- Step 3:
  
  New input:
  
  \[ w'_i = w_i + \alpha \eta (x_i - w_i) \]

  neighbourhood
  learning rate
The Kohonen Map

• The neighbourhood $\eta$ and learning rate $\alpha$ are controlled in **2 phases**:  
  ▪ Self-Organising / Ordering phase: topological ordering of the weight vectors ($\eta, \alpha$ large)  
  ▪ Convergence Phase: fine-tuning of the map ($\eta, \alpha$ small)
Kohonen Map: Applications

- Detecting structure in input data (NNRC, 1997)

Input: 39 indicators describing various quality-of-life factors, such as state of health, nutrition, educational services.
Kohonen Map: Applications

- Traveling Salesman Problem (Fritzke & Wilke, 1990)
- 1D Ring structure
Kohonen Map: Applications

• Visualising sensor data from different contexts (Van Laerhoven, 1999)
The Hopfield Network

- 1982: John Hopfield, "Neural networks and physical systems with emergent collective computational abilities"
- Associative Memory: ‘pony’
- Simple threshold units
- Outputs are -1 or +1 (sometimes 0 or 1):

\[ y = \begin{cases} 
  +1 & \text{if } \sum_{i=1,2,3,...} (w_i x_i) > \theta \\
  -1 & \text{otherwise}
\end{cases} \]

- Connections are typically symmetric!
The Hopfield Network

- States:

<table>
<thead>
<tr>
<th>State</th>
<th>Neuron</th>
<th>New state if...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2 fires</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Diagram:

- Nodes 1, 2, 3 connected in a network structure.
- States: 0, 1, 2, 3.
Energy Functions

- **Energy function** of the network is a scalar value used for finding **local minima**, regarded as solutions

\[ E = -\frac{1}{2} \sum_{i<j} w_{ij}s_is_j + \sum_i \theta_is_i \]

for Hopfield net

- ‘solutions’: memorised states

- Popular method for neural network design
The Hopfield Network

- States for:

<table>
<thead>
<tr>
<th>state</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>1 fires</th>
<th>2 fires</th>
<th>3 fires</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>
Hopfield Network Apps

- Pattern recognition:
Elman Network

- Jeff Elman 1990: “Finding structure in time”
- Sequences in the input: implicit representation of time
- Hidden neuron(s) *copied* as an input
- Backpropagation as usual
Elman Network

- Input is given
Elman Network

- Input is given
- Hidden neurons’ output functions are calculated
Elman Network

- Input is given
- Hidden neurons’ output functions are calculated
- Output is calculated and hidden neuron’s output is copied
Elman Network

- Input is given
- Hidden neurons’ output functions are calculated
- Output is calculated and hidden neuron’s output is copied
- The next input...
Elman Network Apps

• Learning formal grammars
  ▪ Which strings belong to a language?

• Speech recognition:
  ▪ Recognition of Auditory Sound Sources and Events
    (Stephen McAdams, 1990)

• Music composition:
  ▪ Neural network music composition by prediction: Exploring the benefits of psychophysical constraints and multiscale processing (Michael Mozer, 1994)

• Activity recognition:
Architectures:
- McCulloch-Pitts
- Adaptive Resonance Theory
- Perceptron
- Multilayer Perceptron
- Kohonen Map
- Hopfield Network
- Elman Network

Learning:
- Hebbian
- Delta rule
- Backpropagation
- (Un)Supervised
- Competitive Learning
- Associative Memory
- Energy
Architectures

- Perceptron
- Multilayer perceptron
- Hopfield net
- Kohonen map
- Elman network
Training

- Perceptron
  - Delta rule / Gradient descent
  - Supervised
- Multilayer perceptron
  - Backpropagation
  - Supervised
- Hopfield net
  - Associative memory
  - Unsupervised
- Kohonen map
  - Competitive Learning
  - Unsupervised
- Elman network
  - Backpropagation
  - Supervised
Number of Inputs

- Perceptron: 4 inputs
- Multilayer Perceptron: 4 inputs
- Hopfield Net: 3 inputs
- Kohonen Map: 5 inputs
- Elman Network: 2 inputs
Number of Weights

perceptron

multilayer perceptron

Hopfield net

Kohonen map

Elman network

12

24

3

60

9
Number of Outputs

- **Perceptron**: 3 outputs
- **Multilayer Perceptron**: 4 outputs
- **Hopfield Net**: 3 outputs
- **Kohonen Map**: 12 outputs
- **Elman Network**: 1 output
Further Research

- Spiking / Pulsed Neural Networks
  - ‘Spike trains’: timings of pulses is crucial
  - Closer to biological neuron
  - Action potential

\[ p(t) \]

\[ t_1,1 \]

\[ t_1,2 \]

\[ t_2,1 \]
Further Research

- Principal Component Analysis
  - Dimension reduction
- Independent Component Analysis
  - Cocktail party problem
  - ‘blind source detection’
Further Research

- Support Vector Machines

![Graph showing Support Vector Machines](image)