INM427 Neural Computing

Artur S. d'Avila Garcez
a.garcez@city.ac.uk
http://www.staff.city.ac.uk/~aag/

Neural Computing

- The term "neural network" refers to a family of algorithms inspired by models of the brain.
- Such models are employed in statistics, cognitive psychology and artificial intelligence.
- Some models seek to emulate the central nervous system: computational neuroscience.
- Others (deep networks) have become very popular recently in large-scale data analysis (data science), particularly in image, video, audio and language modelling.

Deep networks

Google brain simulator identifies cats on YouTube:
http://www.wired.co.uk/news/archive/2012-06/26/google-brain-recognises-cats

Facebook is working on deep learning neural networks to learn even more about your personal life
http://www.extremetech.com/computing/167179-facebook-is-working-on-deep-learning-neural-networks-to-learn-even-more-about-your-personal-life

Google and Microsoft using recurrent neural nets for speech recognition in mobile phones (LSTMs)

DARPA's Explainable AI programme:
http://www.darpa.mil/program/explainable-artificial-intelligence

Brain-mind dichotomy in AI: numerical models of the brain or logical systems that manipulate symbols?

Contents

- What is a Neural Network?
- History of Artificial Neural Networks
- The Perceptron and the Linear Separability Problem (Minsky and Papert)
- The Multilayer Perceptron and the Backpropagation Learning Algorithm
- Network Training and Evaluation in practice (the Problem of Local Minima)
- Support Vector Machines

Contents (cont.)

The Team

Artur d'Avila Garcez, FBCS
Professor of Computer Science
Director of the Research Centre for Machine Learning
http://www.city.ac.uk/machine-learning
Chair, Data Science Institute (launching event: 2nd April 2019)

Abi Sowri (TA in Data Science)

Alex Galkin (TA in Data Science)

Simon Odense (PhD student, knowledge extraction from neural nets)

Benedikt Wagner (PhD student, explainable AI)

Lab-based Tutorials

Lectures: Fri, 1-3pm BG02

Labs: using Matlab and Matlab's Neural Net toolkit

Lab 1: 3-4pm, ELG05 (surnames starting: W, S, H, P, L)
Lab 2: 3-4pm, C301 (surnames: J, B, T, G, D)
Lab 3: 4-5pm, C301 (surnames: M, R, C, A, K)
Lab 4: 4-5pm, ELG05 (surnames: E, Y, F, N, Other)

Assessment

30% coursework (in groups of two): comparing two methods on a data set of your choice using Matlab

UCI Machine Learning repository: http://archive.ics.uci.edu/ml/

(more for those interested in Python next week)

70% exam (in April/May)

Example exam and model answers will be provided so you know what to expect

Coursework

Coursework submission deadline:

Fri, 29 March 2019, 5pm

Submit (all through Moodle): Matlab code and 6-page paper with critical comparison of two methods studied in class (e.g. MLP and SVM)

(submit best trained models and test set so that test results can be reproduced)

More about the coursework including marking scheme on Moodle…

Matlab training

Every Data Science student has free access to Matlab training leading to MathWorks certificates, including:

MATLAB Fundamentals
MATLAB Programming Techniques
MATLAB for Data Processing and Visualisation

You need to be registered to use Matlab first and then request access to the training.

Please raise a service request on www.city.ac.uk/itservicedesk to create a 'Student/Home use software request' for Matlab and ask to be registered on the Matlab online training

Any problems, please contact Paul Roberts P.Roberts-1@city.ac.uk

Check also the matlab primer file on Moodle, which is a quick start guide to Matlab for use in the lab tutorials

Schedule

Weeks 1 to 5, 1pm, BG02, Labs 3pm to 5pm

Reading week (no lectures or labs)

Weeks 7 to 10, 1pm, BG02, Labs 3pm to 5pm

Coursework submission: week 10

Week 11 or 12 Revision

References

- S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd Edition, Prentice Hall, 1999.
- Tricks of the trade: https://www.amazon.co.uk/Neural-Networks-Second-Computer-Theoretical/dp/364235288X
- A. d'Avila Garcez, K. Broda and D. Gabbay, Neural-Symbolic Learning Systems, Springer-Verlag, 2002.

Resources / further reading

Behavioral and Brain Sciences (BBS), Cambridge University Press

Neural Information Processing Systems (NeurIPS), MIT Press

IEEE/INNS International Joint Conference on Neural Networks IJCNN

IEEE Transactions on Neural Networks and Learning Systems

Neural Computation, MIT Press

International Conference on Learning Representations ICLR

Matlab primer: *www.math.toronto.edu/mpugh/**primer**.pdf*

Modus Operandi (1)

Please make every effort to attend lectures and tutorials.

Ask questions in class or after the lectures and in the tutorials.

There is a discussion board on Moodle; post questions (and answers) there. The teaching team will monitor the board and respond when appropriate.

Email me only if absolutely necessary, e.g. if it can't wait till the next lecture.

Lecture notes and lab exercises will be made available on Moodle before the lectures. Read the materials before the lectures and labs. The best way to prepare for the exam is to engage during term and keep up with the material!

Modus Operandi (2)

Sample solutions of the exercises will be made available only later…

Discuss your coursework choices after the class and at the tutorials with the teaching team.

Show me your progress on the coursework and I'll seek to give you formative feedback (but do this well before the deadline).

An example coursework (6-page paper) and exam will be made available and at a revision lecture we will go through the example exam so that you know what is expected of you in terms of assessment for full marks!

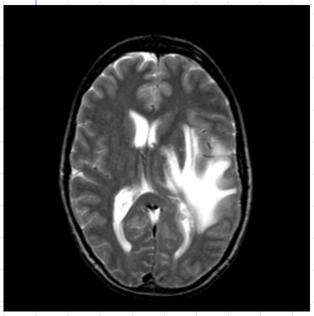
What is a Neural Network?

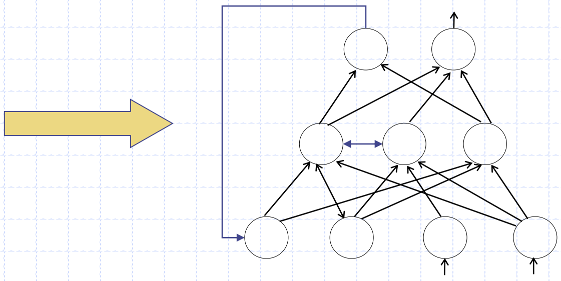
A massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

- Knowledge is acquired from the environment through a learning process
- Inter-neuron connection strength, known as synaptic weights, are used to store the acquired knowledge.

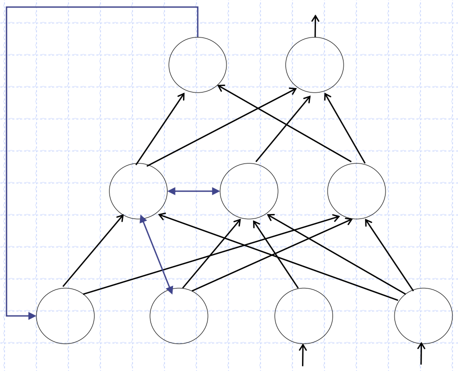
Human Brain

- Robust, Fault Tolerant, Massively Parallel, Capable of Learning from Examples…



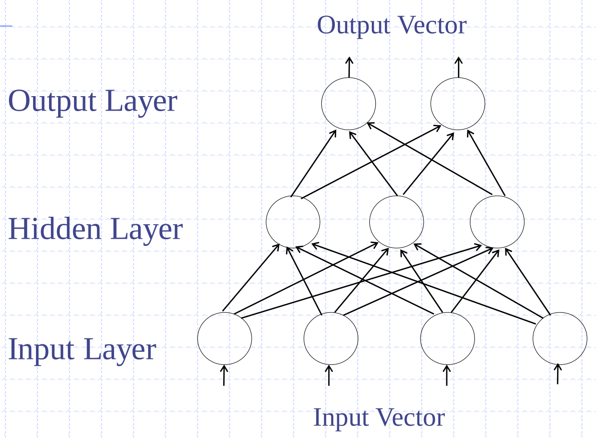


Artificial Neural Networks



The most interesting properties of neural networks do not arise from the functionality of each neuron, but from the effect of the interconnection of neurons.

Feed-forward Networks



The network computes a function $f : \mathfrak{R}^r \rightarrow \mathfrak{R}^s$

Learning from Examples in Neural Networks

The network computes a function

$$f : \mathfrak{R}^r \rightarrow \mathfrak{R}^s$$

A set of examples (input vectors and their respective *target output vectors*) defines a new function

$$g : \mathfrak{R}^r \rightarrow \mathfrak{R}^s$$

We want to change the function f computed by the network - by changing its weights (W_{ij}) - according to the set of examples, in order to approximate g.

Applications

- Pattern Recognition (e.g.: face/image recognition)
- Classification (e.g.: fault diagnosis, DNA analysis)
- Associative Memory (e.g.: image compression)
- Clustering (e.g.: credit analysis, fraud prevention)
- with applications in Aerospace, Finance, Medical, Security, Transport, etc.

Example: DARPA Challenge

http://en.wikipedia.org/wiki/DARPA_Grand_Challenge

DARPA Urban challenge

Self-driving cars: Uber AI Labs, Tesla X, Google Waimo, self-driving taxis in Singapore…

Back in 1995… No hands across America:

- Car trip from east coast to west coast
- 2849 miles of which 2797 miles (98.2%) were driven with *no hands* by the RALPH computer program (the human *driver* handled the throttle and brake)
- RALPH (Rapidly Adapting Lateral Position Handler) uses video images and a multi-layer perceptron with backpropagation as learning algorithm to keep the vehicle on the road

History of Neural Nets

- 1940s
  - McCulloch and Pitts neuron (biological motivation)
- 1950s
  - Hebbian learning (first learning algorithm by Donald Hebb)
  - Frank Rosenblatt's Perceptron (neural net for pattern recognition)
- 1960s
  - Widrow-Hoff learning rule (similar to perceptron's) First application: used in signal processing, e.g. echo cancellation in phone lines

History of Neural Nets (cont.)

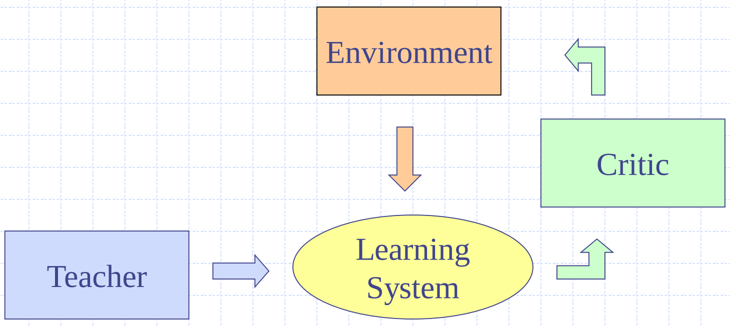
- 1969
  - Minsky and Papert's critique of perceptron (shows perceptron's linear separability problem, and practically halts all NN research for twenty years)
- 1987
  - Backpropagation algorithm (overcomes the limitations of perceptron)
- 1990s
  - Powerful computers
  - Statistical foundations of neural nets established

Neural Networks today

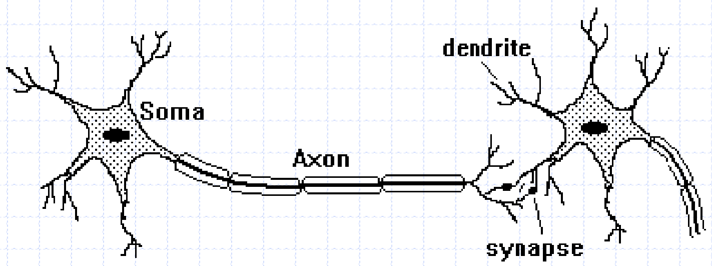
- 2000s
  - A standard tool to solve many practical problems
  - Logical foundations of neural nets established
  - Biological motivation stronger than ever (fMRI)
  - New insights into Consciousness
- 2010s
  - Big Data and Deep networks: fast algorithms to train large nets (state-of-the-art speech and image recognition and language translation)
  - Social computing, human-like computation (see HCOMP conference series)
  - Human Brain Projects (US and EU)

Classification of Neural Networks w.r.t. Learning

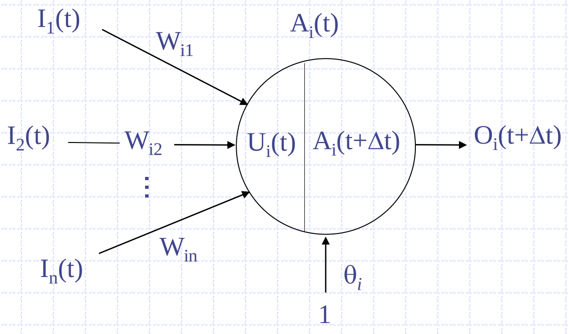
- 🟦 Supervised Learning
- 🟧 Unsupervised (Semi-supervised) Learning
- 🟩 Reinforcement Learning



The Neuron



Model of a Neuron



$I_i(t)$: Input Vector

W_{ij}: Weight Vector; θ_i : Bias

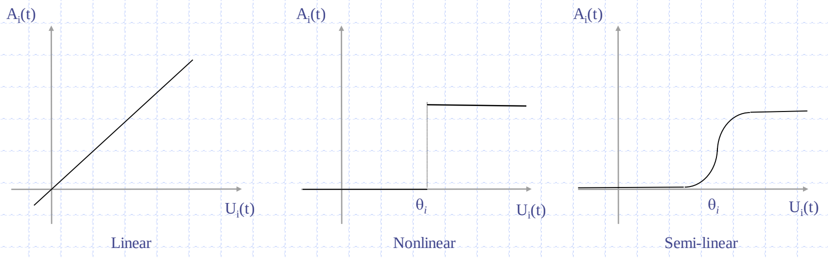
$U_i(t)$: Input Potential; $U_i(t) = g_i(I_i(t),W_{ij}, \theta_i)$

$A_i(t)$: Activation State; $A_i(t+\Delta t) = h_i(A_i(t), U_i(t))$

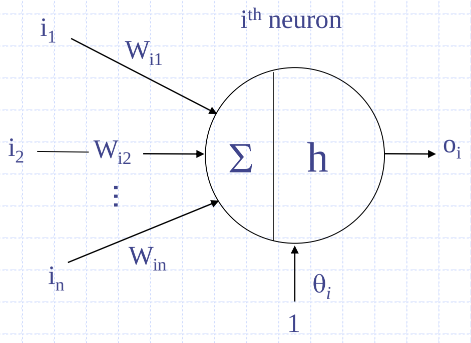
$O_i(t)$: Output; $O_i(t) = f_i(A_i(t))$

Types of Activation Function h(x)

Linear, Non-Linear (step function), Semi-Linear (sigmoid function), etc.



McCulloch-Pitts Neuron



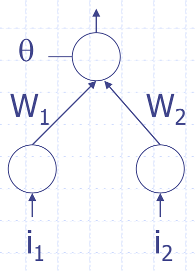
$U_i = \Sigma_j (W_{ij} . i_j) + \theta_i$

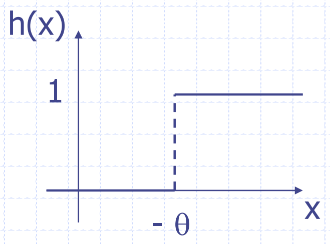
$o_i = h(U_i)$, where $h(x) = 1$ if $x > 0$ and 0 otherwise.

The Perceptron

- Uses McCulloch-Pitts Neurons
- Contains n input neurons, no hidden neuron, and 1 output neuron

$o = h(W_1 i_1 + W_2 i_2 + \theta)$





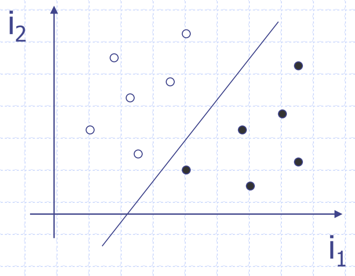
Bias = - Threshold

Note: Input neurons have *identity* as activation function!

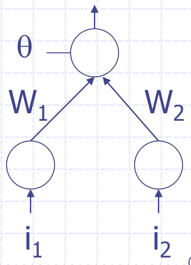
© Artur Garcez

The Perceptron (Cont.)

- Distinguishes Two Classes of Data (in n-dimensional space, by applying a n-dimensional hyper-plane)



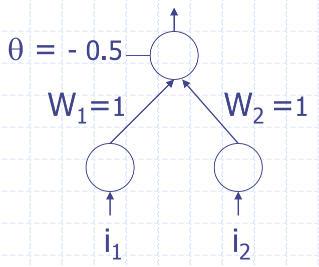
$o = h(W_1 i_1 + W_2 i_2 + \theta)$

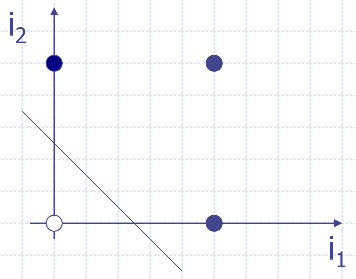


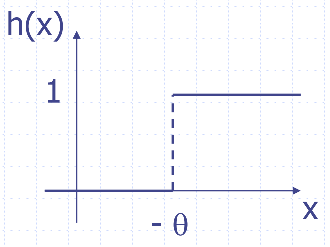
© Artur Garcez

Example: Logical OR

$o = h(W_1 i_1 + W_2 i_2 + \theta)$







i in {0,1}

Learning Algorithm (Perceptron)

1. Initialise the weights randomly;
2. For each example (**i**,t) do:

$$\Delta \mathbf{W} = \eta\, (t - o(i))\, \mathbf{i}$$

Until t = o(i) for all examples.

i = input vector
o = network's output
t = target output ($t \in \{0,1\}$)
$\eta \in \Re^+$ is called the Learning Rate

Note: The algorithm can be proven to terminate whenever the set of examples is linearly separable.

Learning (i_1 OR i_2)

Complete the table below
4 training epochs are sufficient

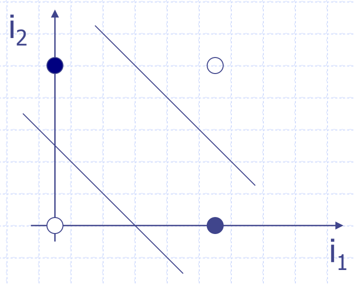
$i = (1, i_1, i_2)$

$W = (\theta, W_1, W_2)$

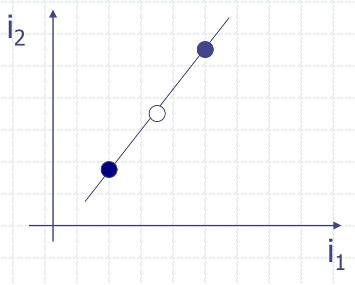
W	i	t	o	ΔW
(-2,-2,0)	(1,0,0)	**0**	0	(0,0,0)
(-2,-2,0)	(1,0,1)	**1**	0	(1,0,1)
(-1,-2,1)	(1,1,0)	**1**	0	(1,1,0)
(0,-1,1)	(1,1,1)	**1**	0	?
(1,0,2)	(1,0,0)	**0**	?	?
(0,0,2)	(1,0,1)	**1**	?	?
(0,0,2)	(1,1,0)	**1**	?	?
(1,1,2)	(1,1,1)	**1**	?	?
(1,1,2)	(1,0,0)	**0**	?	?
(0,1,2)	(1,0,1)	**1**	?	?
(0,1,2)	(1,1,0)	**1**	?	?
(0,1,2)	(1,1,1)	**1**	?	?
(0,1,2)	(1,0,0)	**0**	?	?
(0,1,2)	(1,0,1)	**1**	?	?
(0,1,2)	(1,1,0)	**1**	?	?
(0,1,2)	(1,1,1)	**1**	?	?

$\eta = 1$

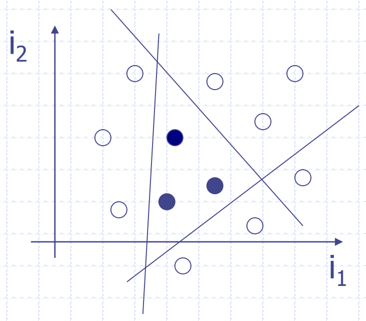
Perceptron's Linear Separability



XOR



Perceptron’s Linear Separability



Requires more Inputs, more Hidden Neurons, or the use of a Curve!

Notes

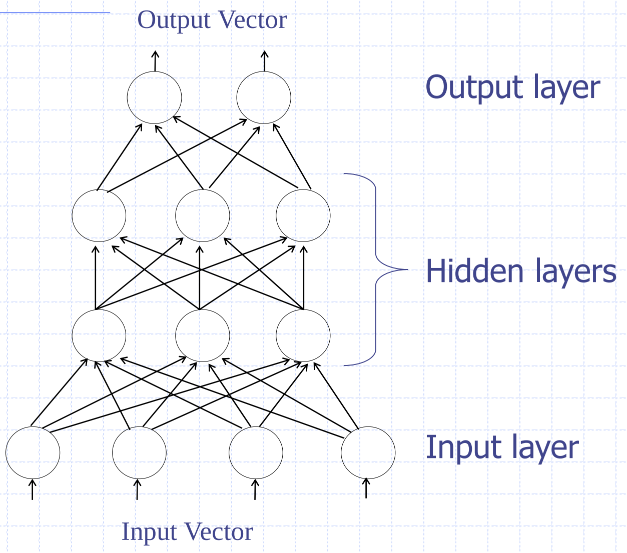
- There are two AI paradigms: symbolic and connectionist. Symbolic AI tries to model the (processes of the) *mind* by manipulating symbols. Connectionist AI tries to model (properties of) the *brain* using artificial neural networks. Hybrid Systems combine and relate the two.
- Input neurons normally use h(x) = x as activation function, i.e. they simply propagate the input.
- $\Delta\mathbf{W}$ indicates the *variation* applied to the weight vector **W** such that New_**W** = Old_**W** + $\Delta\mathbf{W}$.
- The art of defining the learning rate: normally $\eta < 1.0$; $\eta = 0.001$ (slow and safer), $\eta = 0.9$ (fast and risky).

There are many different types of neural network!

- Multilayer Feedforward Nets (classification)
- Hopfield Networks (associative memory)
- Radial Basis Function Nets (density estimation)
- Support Vector Machines (pattern recognition)
- Self Organising Maps (clustering)
- Deep Belief Networks (image/audio processing)
- Deep Boltzmann Machines (distribution estimation)
- Recurrent Neural Networks (sequence learning)

Different networks to solve different problems, but all with the same ingredients: simple neurons, distributed learning, parallel computation.

Multilayer Perceptron



Multilayer Perceptrons are Universal Approximators

Theorem [Cybenko, 1989]: Let $h:\Re \rightarrow \Re$ be a continuous and sigmoid function, i.e. $\lim_{x\rightarrow -\infty} h(x) = 0$ and $\lim_{x\rightarrow\infty} h(x) = 1$. Then, finite sums of the form:

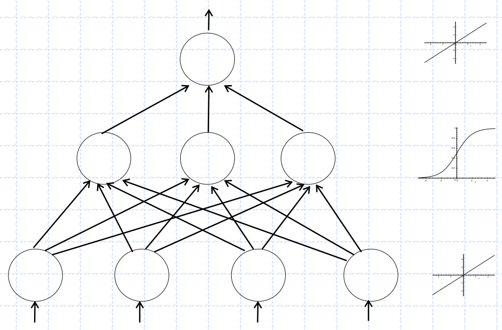
$$g(x) = \Sigma_j\ \alpha_j\, h(W_j\, x + \theta_j),\ x \in I^n$$

with parameters α_j, $\theta_j \in \Re$ and $W_j \in \Re^n$, are such that $|g(x) - f(x)| < \varepsilon$ for any continuous function f(x).

- Neural Networks with as few as a single hidden layer with sigmoid activation function can approximate virtually any function of interest.

Learning in Multilayer Perceptrons

- We need to find α_j, θ_j and W_j such that g(x) approximates f(x)

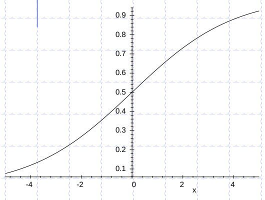


$g(x) = \Sigma_j\ \alpha_j\, h(W_j\, x + \theta_j)$

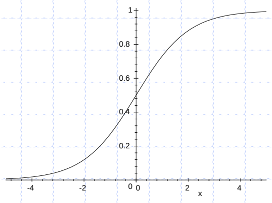
Semi-Linear Activation Function

$h(x) = 1 / (1 + e^{-\beta x})$

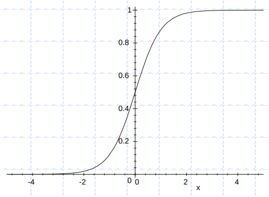
$h'(x) = \beta e^{-\beta x} / (1 + e^{-\beta x})^2$



$\beta = 1/2$



$\beta = 1$



$\beta = 2$

© Artur Garcez

Backpropagation

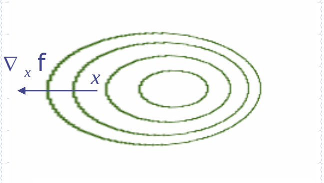
- A computationally efficient method for training multilayer perceptrons.
- The neural learning algorithm most successfully applied in industry.
- It computes an estimate of the gradient of an error function ($E_{\mathbf{w}}$) w.r.t. a set of weights **W**

$$\nabla E_{\mathbf{w}} = \frac{\partial E_{\mathbf{w}}}{\partial W_{11}}, \frac{\partial E_{\mathbf{w}}}{\partial W_{12}}, \ldots, \frac{\partial E_{\mathbf{w}}}{\partial W_{ij}}$$

© Artur Garcez

Gradient Descent

The gradient of E_W is the vector pointing in the direction of the fastest growth of E_W, perpendicular to contour lines.

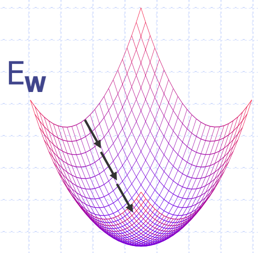


The iterative application of changes

$$\Delta W = -\eta \cdot \nabla E_{\mathbf{W}}$$

to W tries to minimise

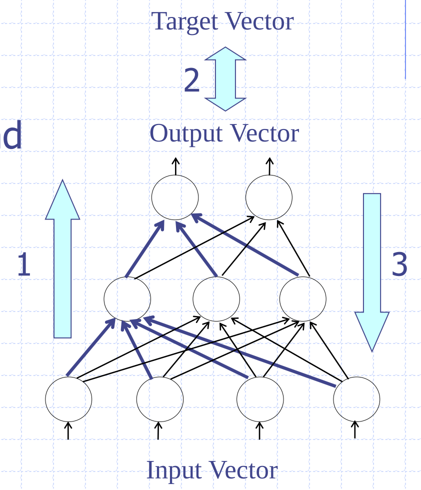
$$E_{\mathbf{W}} = \tfrac{1}{2} \Sigma_i (o_i - t_i)^2$$



© Artur Garcez

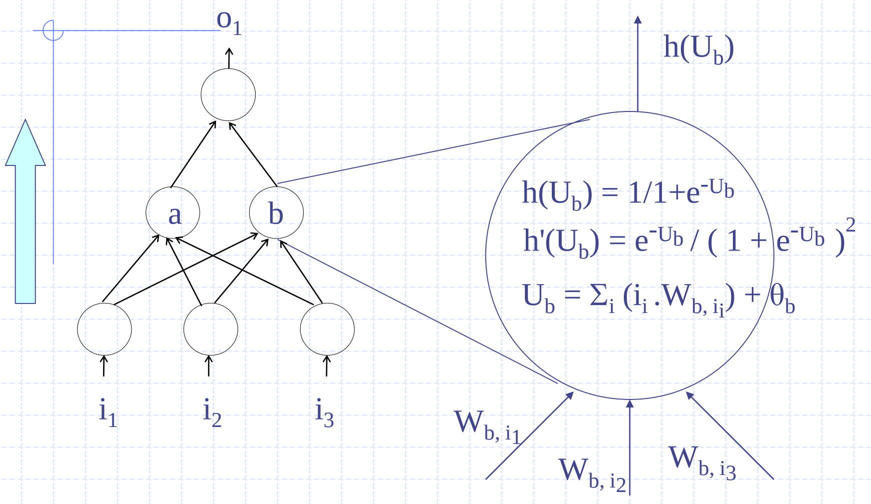
Backpropagation Learning Algorithm

1. Propagate
2. Compute Error
3. Back-propagate Error and Change Weights



© Artur Garcez

Propagate



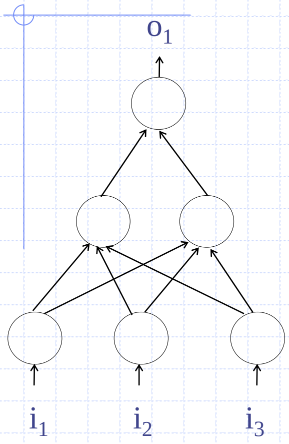
$h(U_b) = 1/1+e^{-U_b}$

$h'(U_b) = e^{-U_b} / (1 + e^{-U_b})^2$

$U_b = \Sigma_i (i_i . W_{b,i_i}) + \theta_b$

© Artur Garcez

Compute Error



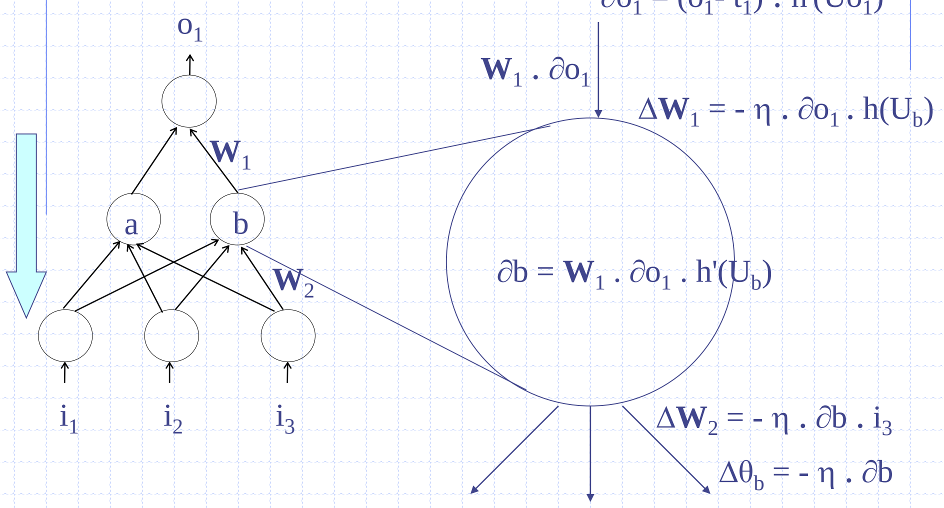
e_1 = Obtained output (o_1) - Target output (t_1)

Note:

- A Training Example is a pair (input vector, target output)
- e_1 is the (local) error of output o_1.
- $E_W = ½ \Sigma_i (o_i - t_i)^2$ is the network's global error (which we want to minimise)

© Artur Garcez

Backpropagate Error and Change Weights



$\partial o_1 = (o_1 - t_1) \cdot h'(Uo_1)$

$\mathbf{W}_1 \cdot \partial o_1$

$\Delta \mathbf{W}_1 = -\eta \cdot \partial o_1 \cdot h(U_b)$

$\partial b = \mathbf{W}_1 \cdot \partial o_1 \cdot h'(U_b)$

$\Delta \mathbf{W}_2 = -\eta \cdot \partial b \cdot i_3$

$\Delta \theta_b = -\eta \cdot \partial b$

$U_j = \Sigma_i (W_{j,i_i} \cdot i_i) + \theta_j$

$o_j = h(U_j)$, where $h(x) = 1/(1+e^{-x})$

$h'(U_j) = e^{-U_j} / (1 + e^{-U_j})^2$

$U_k = \Sigma_j (W_{kj} \cdot o_j) + \theta_k$

$o_k = h(U_k)$

$h'(U_k) = e^{-U_k} / (1 + e^{-U_k})^2$

$e_k = o_k - t_k$

$\partial o_k = e_k \cdot h'(U_k)$

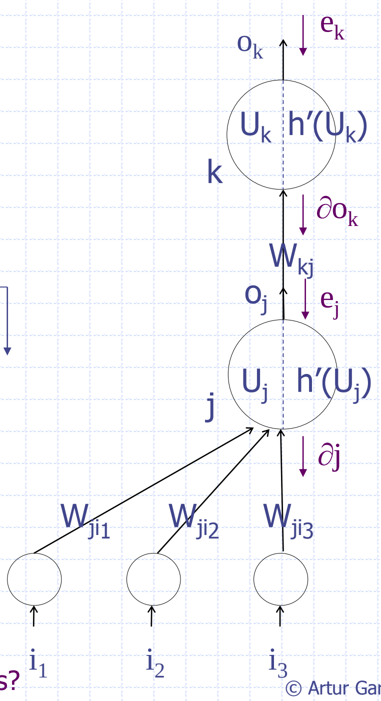
$e_j = W_{kj} \cdot \partial o_k$

$\partial j = e_j \cdot h'(U_j)$

$\Delta \mathbf{W}_{kj} = -\eta \cdot \partial o_k \cdot o_j$

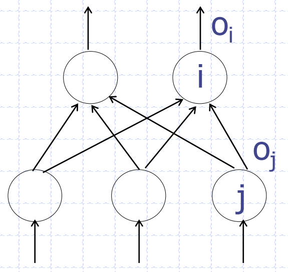
$\Delta \mathbf{W}_{ji_i} = -\eta \cdot \partial j \cdot I_i$

Don't forget the bias!
What if there are two output neurons?



Backprop. computes $\nabla E_W = \partial E_W / \partial W$ efficiently

$$\frac{\partial E_W}{\partial W_{ij}} = \underbrace{\frac{\partial E_W}{\partial e_i}}_{e_i}\ \underbrace{\frac{\partial e_i}{\partial o_i}}_{1}\ \underbrace{\frac{\partial o_i}{\partial U_i}}_{h'(U_i)}\ \underbrace{\frac{\partial U_i}{\partial W_{ij}}}_{o_j = h(U_j)}$$



<u>Note</u>:

$E_W = ½\ \Sigma_i\ (o_i - t_i)^2$ $\quad e_i = o_i - t_i$

$o_i = h(U_i)$ $\quad U_i = \Sigma_j\ (W_{ij}\ .\ o_j)$

$\partial o_i = e_i\ .\ h'(U_i)$ $\quad \Delta \mathbf{W} = -\ \eta\ .\ \partial o_i\ .\ h(U_j)$

(∂o_i is the local gradient)

© Artur Garcez

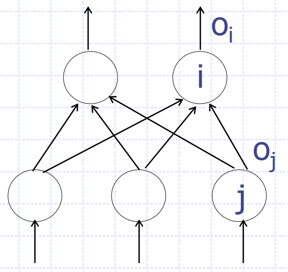
Backprop. computes $\nabla E_W = \partial E_W / \partial W$ efficiently

When i is an output neuron $\Delta W = -\ \eta\ .\ e_i\ .\ h'(U_i)\ .\ o_j$

What if i is a hidden neuron (h_i)?

What if there are many hidden layers? Vanishing gradients

$$\frac{\partial E_W}{\partial W_{ij}} = \underbrace{\frac{\partial E_W}{\partial o_i}}_{e_k\ .\ h'_k(U_k)\ .\ W_{ki}}\ \underbrace{\frac{\partial o_i}{\partial U_i}}_{h'(U_i)}\ \underbrace{\frac{\partial U_i}{\partial W_{ij}}}_{o_j}$$



$\Delta W = -\ \eta\ .\ \partial h_i\ .\ o_j$

$\partial h_i = h'(U_i)\ \Sigma_k\ (W_{ki} . \partial o_k)$ (∂h_i, ∂o_j are local gradients)

© Artur Garcez

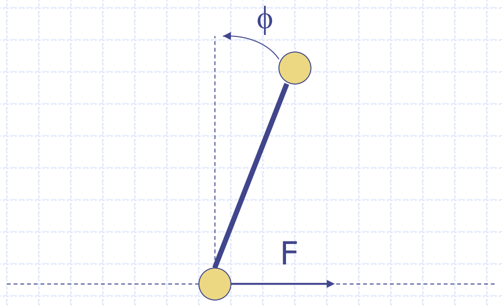
Backpropagation Algorithm

```
For each example ($\mathbf{i} = i_1,\ldots,i_p$; $\mathbf{t} = t_1,\ldots,t_q$) in the training set, do:
    For each neuron n in the network in *ascending topological order*, do:
        Compute $o_n = h(U(\mathbf{i}))$ and $d_n = h'(U(\mathbf{i}))$
    For each neuron n in the network in *descending topological order*, do:
        If n is an output neuron $o_k$ then:
            $\partial o_k = e_k \,.\, d_k$, where $e_k = o_k - t_k$
            $\Delta W_{ki} = - \eta \,.\, \partial o_k \,.\, o_i$
        If n is a hidden neuron $h_i$ then:
            $\partial h_i = d_i \,.\, \Sigma_k (W_{ki}\, \partial o_k)$
            $\Delta W_{ij} = - \eta \,.\, \partial h_i \,.\, o_j$
```

Notes: Each pass through the training set is called an epoch.
Typically, Backprop. takes several epochs to converge.
The algorithm can be extended easily to networks having more than a single hidden layer. Try it!

Lab-based Tutorial

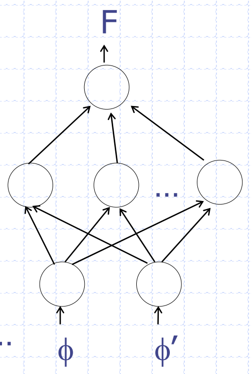
- Pole Balancing



Backpropagation Learning Algorithm
Generalisation x Overfitting

Lab-based Tutorial (cont.)

We want to estimate F given:
angle ϕ and angular velocity ϕ'



We need a training set and a test set…

Challenge: simulate backprop by hand for network with two output nodes!

Limitations of Backpropagation

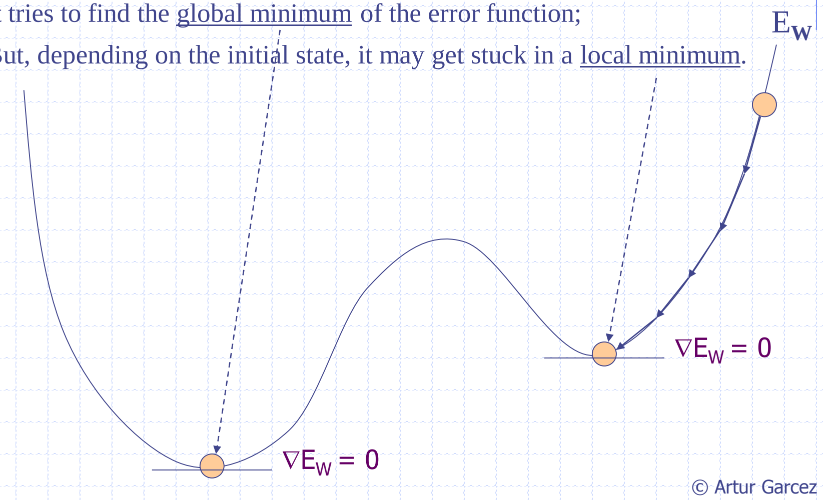
Backprop. is about trying to minimise a training set error…

Optimisation: The problem of local minima

Lack of biological plausibility…

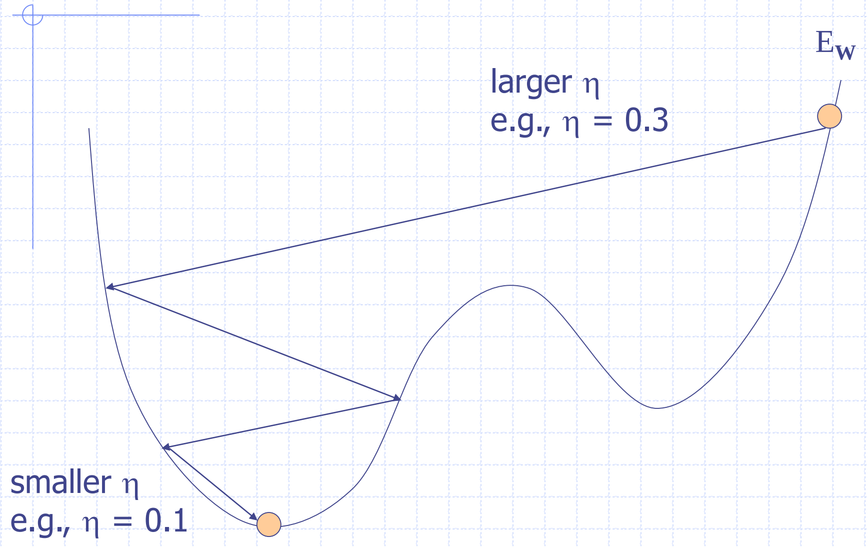
The Problem of Local Minima

- Backprop. performs gradient descent on an error surface;
- It tries to find the global minimum of the error function;
- But, depending on the initial state, it may get stuck in a local minimum.



© Artur Garcez

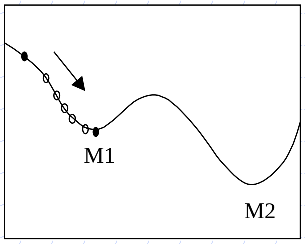
Changing the Learning Rate



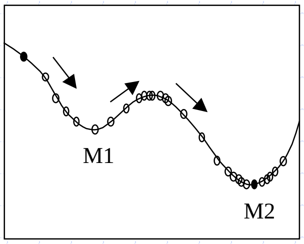
© Artur Garcez

Adding Momentum

$$\Delta W_t = -\eta \,.\, \nabla E_W + \underbrace{\mu\, \Delta \mathbf{W}_{t-1}}_{\text{Term of Momentum}}, \text{ where } 0 \le \mu \le 1$$



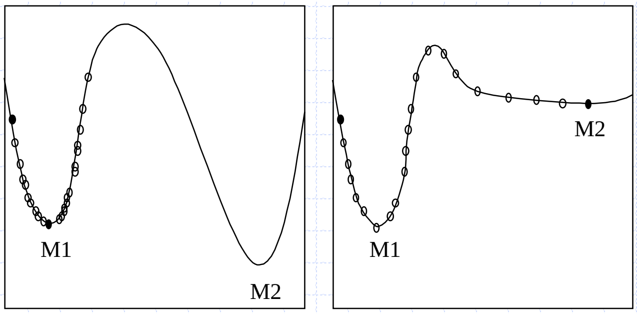
$\mu = 0.0$, η small



$\mu = 0.95$, η small

Global Minimisation

Momentum does not guarantee global minimisation of E_W



Note: E_W is a function in n-dimensional space, where n is the number of input neurons

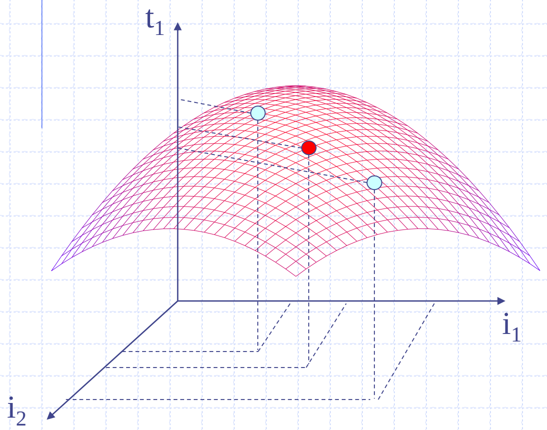
Generalisation

- We expect g (our approximation of f) to produce reasonable results for examples not seen during training, i.e. we expect the learning algorithm to *generalise* to unseen cases.
- The most likely hypothesis (g) is the *simplest* one that is consistent with all observations (*Ockham's razor*).

Generalisation

We try to minimise a training set error

We would like to minimise a generalisation error



○ training examples

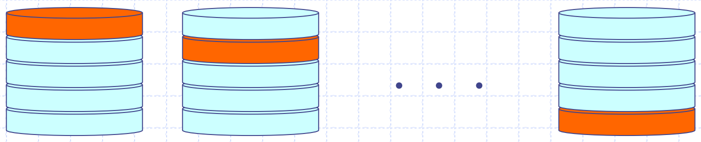
● new (unseen) case

Estimating Generalisation Error

- Take out a subset of the set of examples (the test set) randomly for comparative evaluation
- The test set is never seen by the networks during training.
- Partition the remaining set of examples into a training set and a validation set.
- The validation set error is an estimate of the generalisation error.
- The validation set can be used for model selection or to control for overfitting.

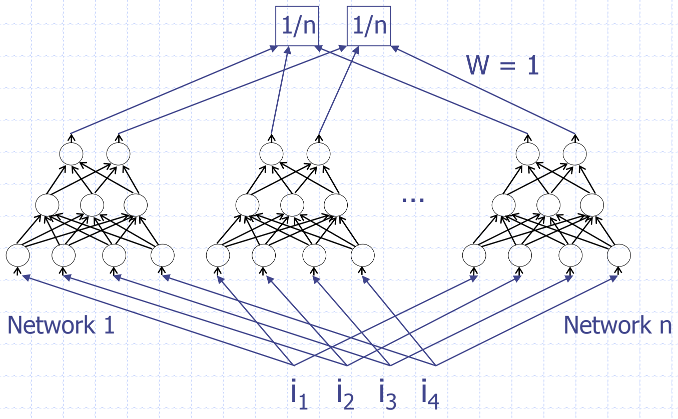
n-fold Cross-Validation

- Divide the set of examples E into n subsets $E_1, E_2, \dots, E_n$
- For each E_i ($1 \le i \le n$), train a network with $E - E_i$ and test it with E_i
- Calculate the averaged training set and validation error



Note: Leaving m out = (|E| / m)-fold cross validation.

Model selection



Note: n-fold cross-validation on m different neural models (with different numbers of hidden neurons) can also be used to select the best model.

Grid Search

E.g. learning rate vs number of hidden neurons

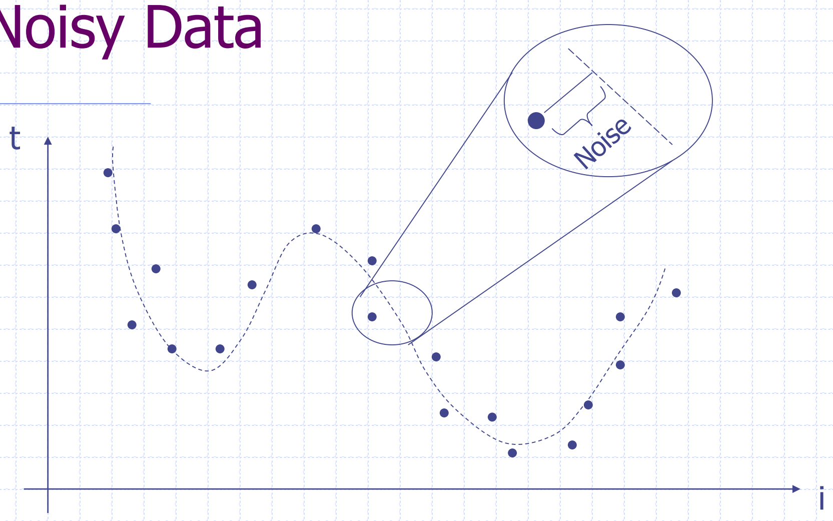
	0.1	0.05	0.01
10	Net 1	Net 2	...
20			
30			

ML Paradox: the use of ensemble methods seems to always improve performance…
Ockham’s razor?

Bootstrapping

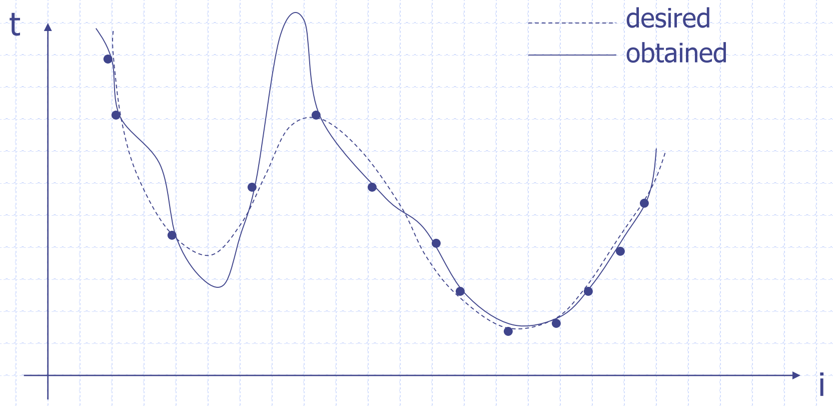
- Create k (pseudo) sets of examples $E_1, E_2, \ldots, E_k$ by randomly selecting $|E|$ elements (with replacement) from E
- For each E_i ($1 \leq i \leq k$), train a network with $E - E_i$ and test it with E_i
- Calculate the averaged training and test set error

Noisy Data



Note: If the set of training examples contains noise, the training set error must not be zero!

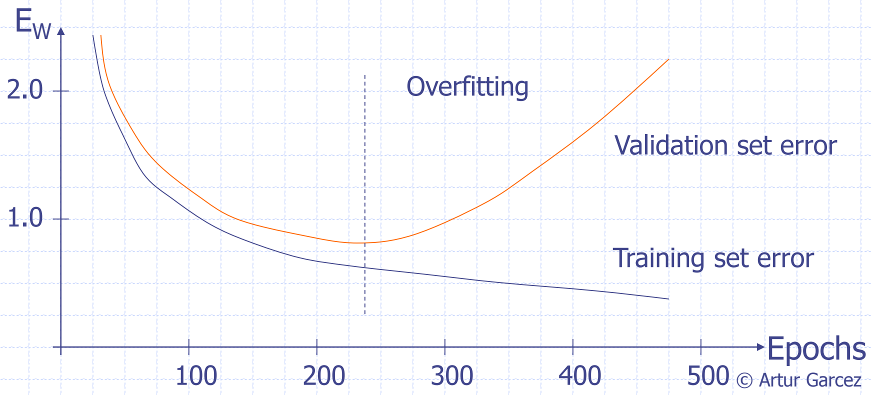
Overfitting



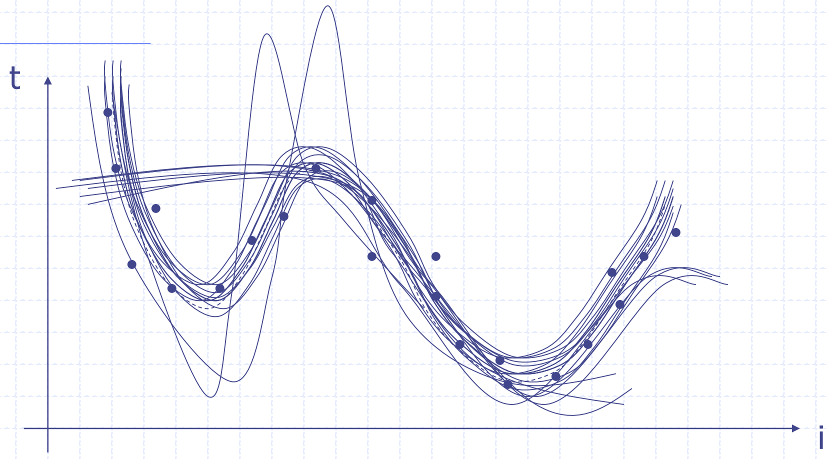
Note: Typically caused by too many hidden neurons or not enough training examples, overfitting results in good training set performance and poor test set performance.

Generalisation

- Assume Ockham's razor
- Assume set of examples is representative
- Check training and validation set performances



Testing the Network is Key!



© Artur Garcez

Learning and Generalisation

Backprop. is about trying to minimise a training set error…

- Problem of local minima: add momentum, vary learning rate

…but learning is about trying to minimise a generalisation error

- Estimate using test set error, cross-validation, bootstrapping (Note: generalisation error approaches training set error when the number of training examples grows)

In addition, in the presence of noisy training examples or to check for overfitting, testing the network is key!

© Artur Garcez