Neuroinformatics

Marcus Kaiser

Week 9: Feed-forward mapping networks
(textbook chapter 6)
(slides edited by Cheol Han)

Peceptrons (single layer)

- linearly separable
- error function, gradient descent

Multi-layer perceptrons

- back-propagation error signal
- overfitting and underfitting
- test and training data

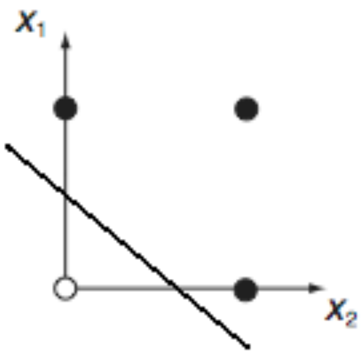
- Radial Basis Functions
- Elman network (recurrent networks)

What can a neuron do?

Can a neuron represent Boolean functions? (McCulloch and Pitts, 1943)
Features x_1 and x_2 Feature vector $x=(x_1\ x_2)$

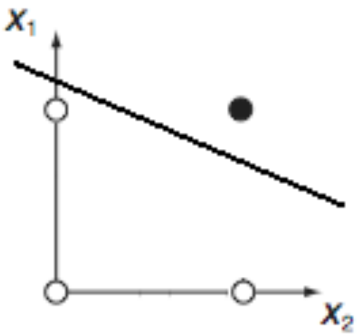
OR function

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

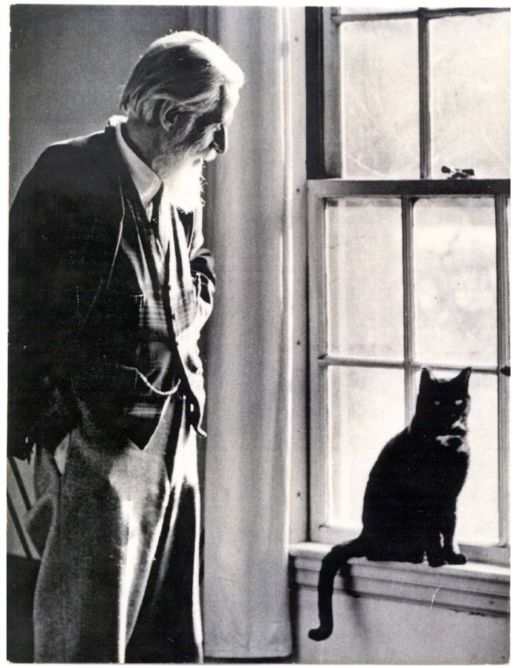


AND function

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



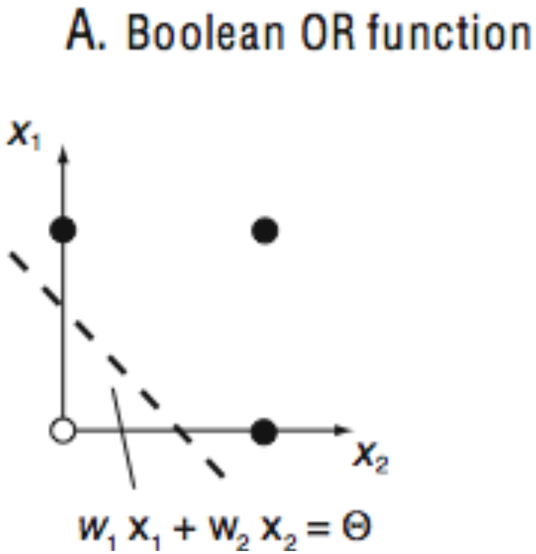
A classifier …
What is a decision boundary?



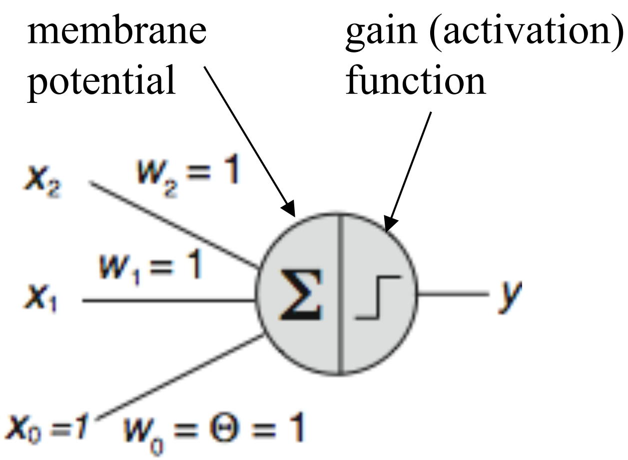
(Warren McCulloch with his cat, from Dr Arbib's class note)

Perceptron (Rosenblatt, 1962)

A single layer neural network
The weights are learnable
Various activation functions (sigmoid, linear, threshold ….)



State space

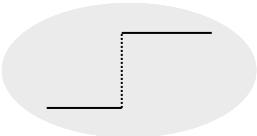
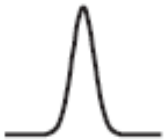


Network topology

$y(\mathbf{x})=g(w_1x_1+w_2x_2)$ $$y = f(\vec{w} \cdot \vec{x}) = f\left(\sum_j w_j x_j\right),$$

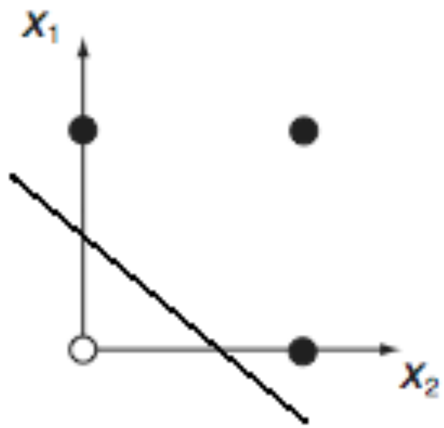
(The leftmost figure is from Bishop, 1995)

Other gain / activation functions

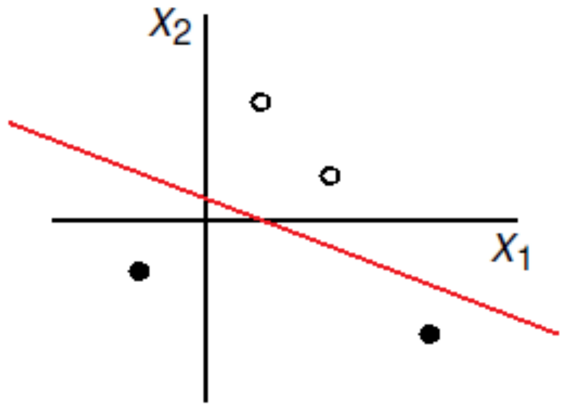
Type of function	Graphical represent.	Mathematical formula	MATLAB implementation
Linear		$g^{\text{lin}}(x) = x$	`X`
Step		$g^{\text{step}}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{elsewhere} \end{cases}$	`floor(0.5*(1+sign(x)))`
Threshold-linear		$g^{\text{theta}}(x) = x\,\Theta(x)$	`x.*floor(0.5*(1+sign(x)))`
Sigmoid		$g^{\text{sig}}(x) = \frac{1}{1+\exp(-x)}$	`1./(1+exp(-x))`
Radial-basis		$g^{\text{gauss}}(x) = \exp(-x^2)$	`exp(-x.^2)`

Linear Separability

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



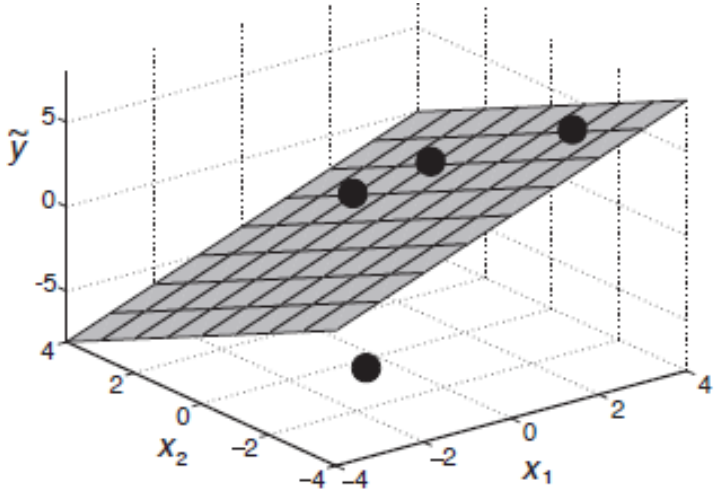
x_1	x_2	y
1	2	0
2	1	0
3	-2	1
-1	-1	1
...	...	...



Can examples be separated by a line?
Yes = **linearly separable**

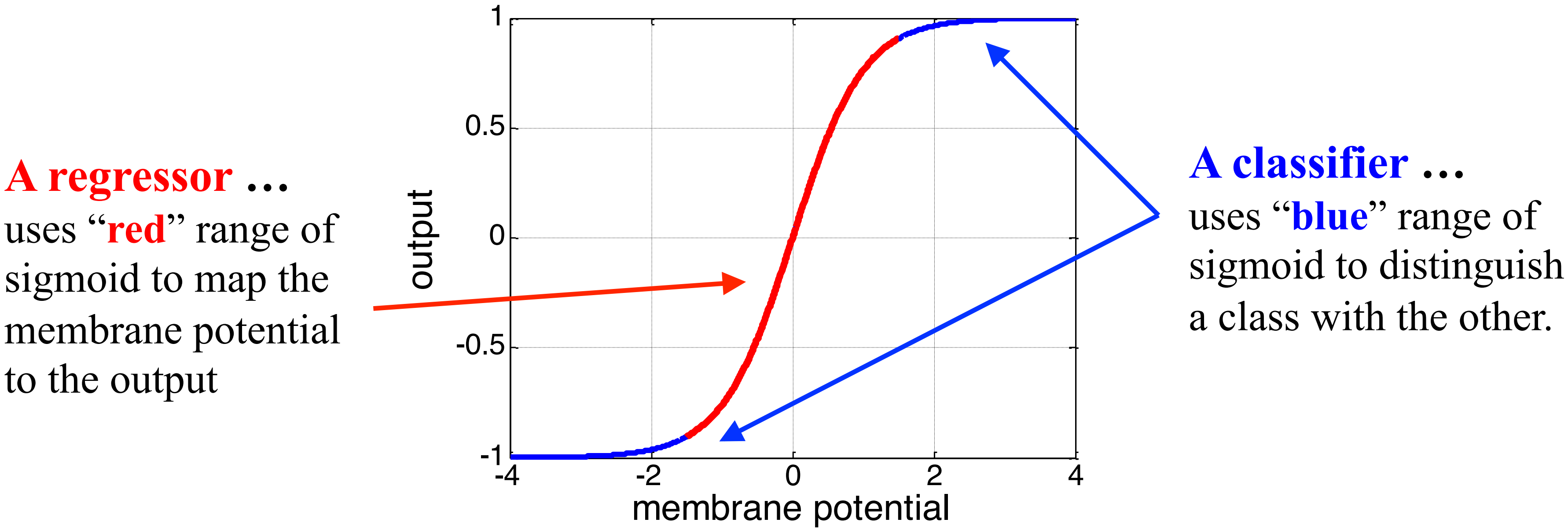
Can a neuron approximate an arbitrary function?

x_1	x_2	y
1	2	-1
2	1	1
3	-2	5
-1	-1	7
...	...	...



A neuron can "associate" inputs with a specific output.

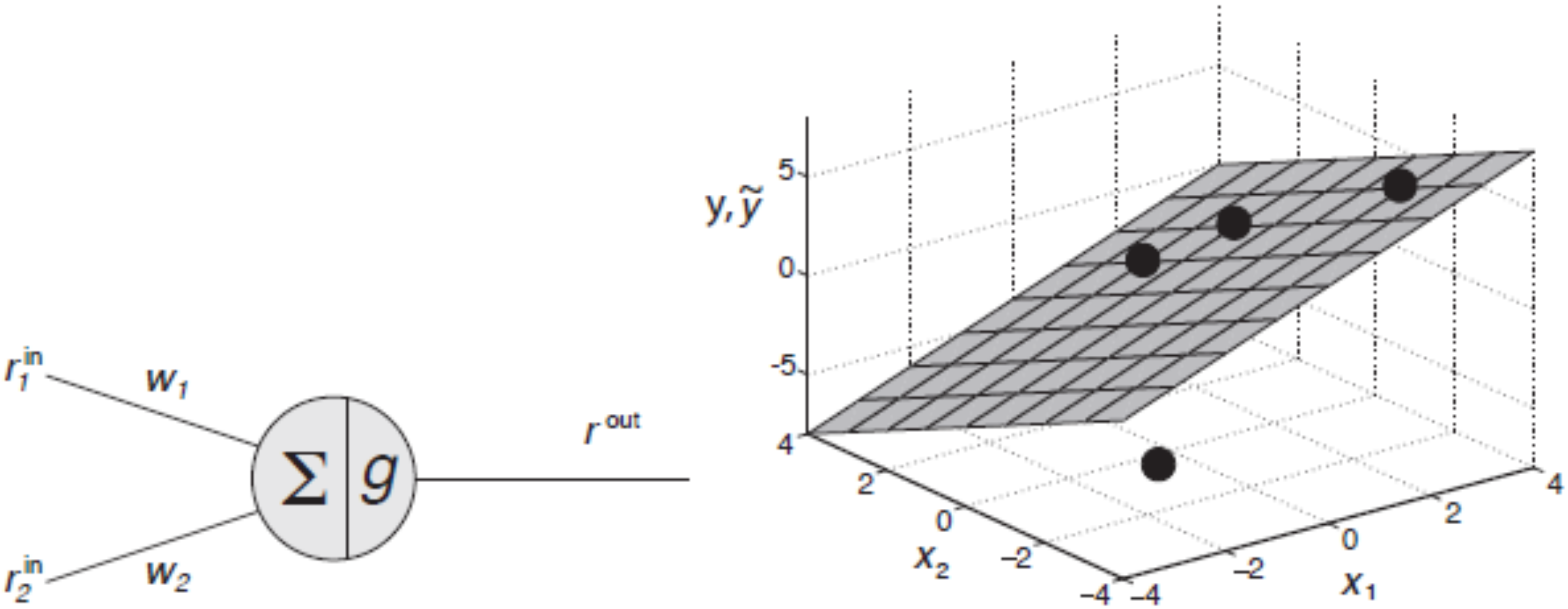
Use a linear gain function or a sigmoid.



The population node as perceptron

Update rule: $\mathbf{r}^{\text{out}} = g(\mathbf{w}\mathbf{r}^{\text{in}})$ (component-wise: $r_i^{\text{out}} = g(\sum_j w_{ij} r_j^{\text{in}})$)
For example: $r_i^{\text{in}} = x_i$, $\tilde{y} = r^{\text{out}}$, linear grain function $g(x) = x$:

$$\tilde{y} = w_1 x_1 + w_2 x_2$$



How to find the right weight values: learning

Objective (error) function, for example: mean square error (MSE)

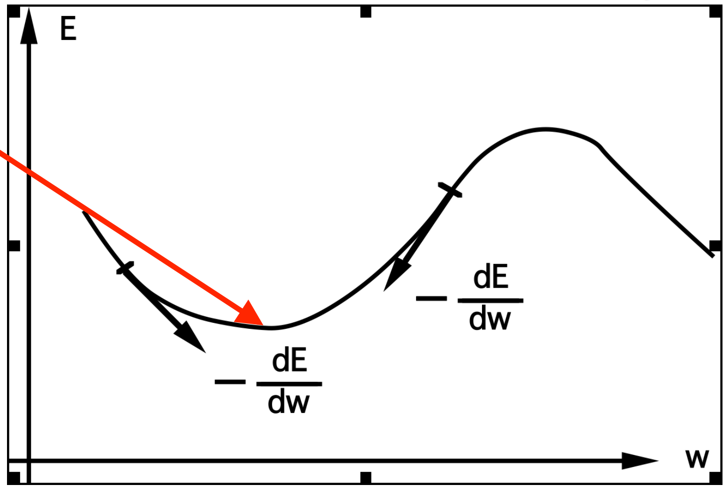
$$E = \frac{1}{2}\sum_i (r_i^{\text{out}} - y_i)^2 \qquad \textbf{<= MINIMIZE THIS}$$

Gradient descent method: $w_{ij} \leftarrow w_{ij} - \epsilon \frac{\partial E}{\partial w_{ij}}$

$= w_{ij} - \epsilon (y_i - r_i^{\text{out}}) r_j^{\text{in}}$ for MSE, linear gain

The best weight

However, is it the global minimum? (start with multiple initial weights!)

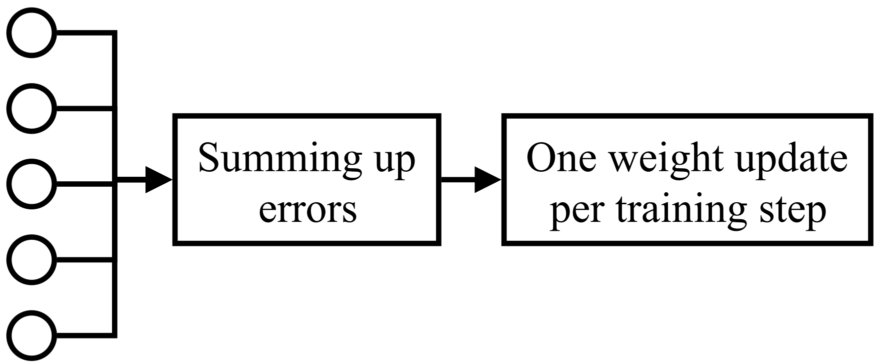


(The figure is from Bishop, 1995)

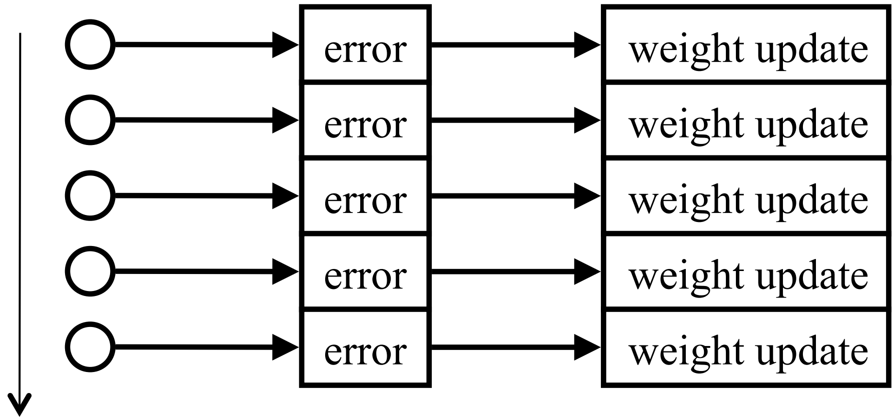
Initialize weights arbitrarily
Repeat until error is sufficiently small
- Apply a sample pattern to the input nodes: $r_i^0 = r_i^{\text{in}} = \xi_i^{\text{in}}$
- Calculate rate of the output nodes: $r_i^{\text{out}} = g(\sum_j w_{ij} r_j^{\text{in}})$
- Compute the delta term for the output layer: $\delta_i = g'(h_i^{\text{out}})(\xi_i^{\text{out}} - r_i^{\text{out}})$
- Update the weight matrix by adding the term: $\Delta w_{ij} = \epsilon \delta_i r_j^{\text{in}}$

Batch algorithm vs. online algorithm

Batch: training step



Online: training step



In order

training step
:training over all given examples once

Why multiple training steps with a small learning rate, instead of a training step with a larger learning rate?

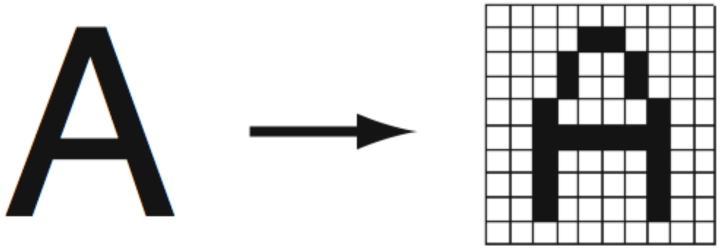
Which one does our brain use?

PerceptronTrain.m

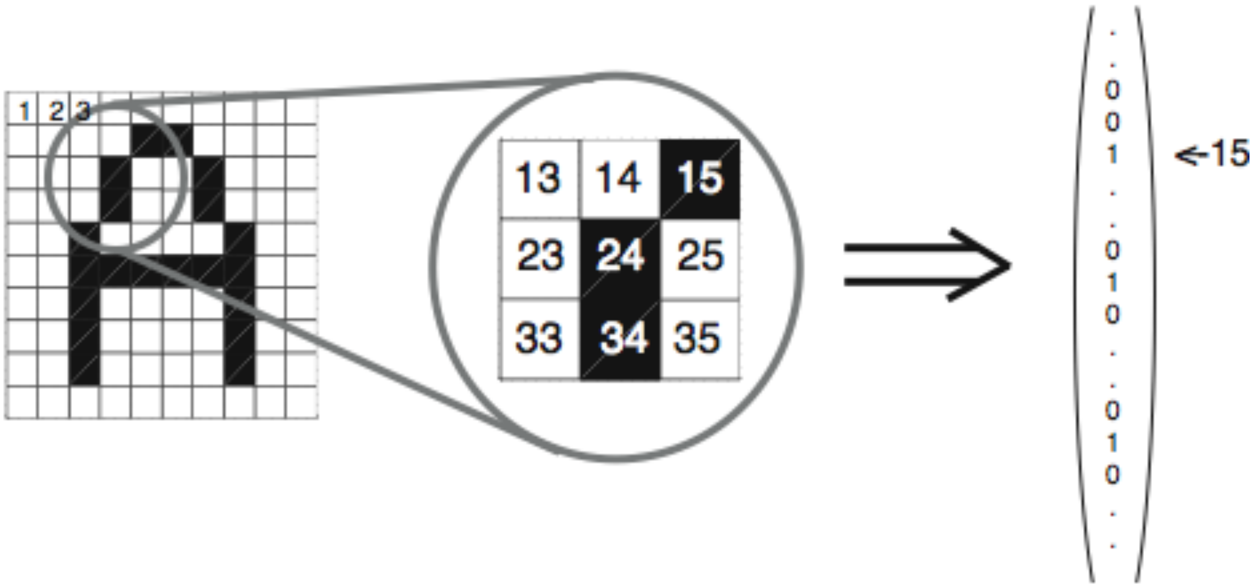
```
%% Letter recognition with threshold perceptron
 clear; clf;
 nIn=12*13; nOut=26;
 wOut=rand(nOut,nIn)-0.5;

% training vectors
 load pattern1;
 rIn=reshape(pattern1’, nIn, 26);
 rDes=diag(ones(1,26));

% Updating and training network
 for training_step=1:20;
     % test all pattern
      rOut=(wOut*rIn)>0.5;
      distH=sum(sum((rDes-rOut).^2))/26;
      error(training_step)=distH;
     % training with delta rule
      wOut=wOut+0.1*(rDes-rOut)*rIn’;
 end

 plot(0:19,error)
 xlabel(’Training step’)
 ylabel(’Average Hamming distance’)
```

It’s a batch algorithm…

Example: OCR (digital representation of a letter)



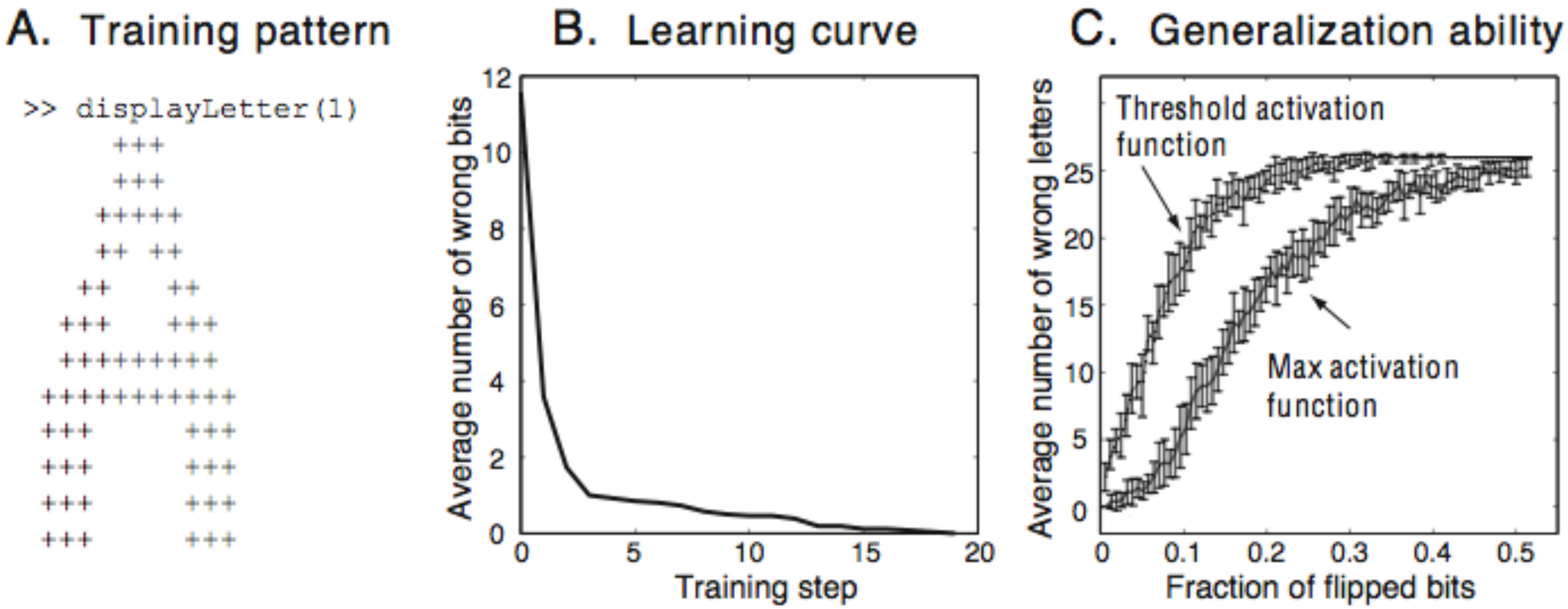
preprocessing …
(How to encode input vectors)



Optical character recognition: Predict meaning from features.
E.g., given features **x**, what is the character **y**

$$f : \mathbf{x} \in \mathbf{S}_1^n \rightarrow \mathbf{y} \in \mathbf{S}_2^m$$

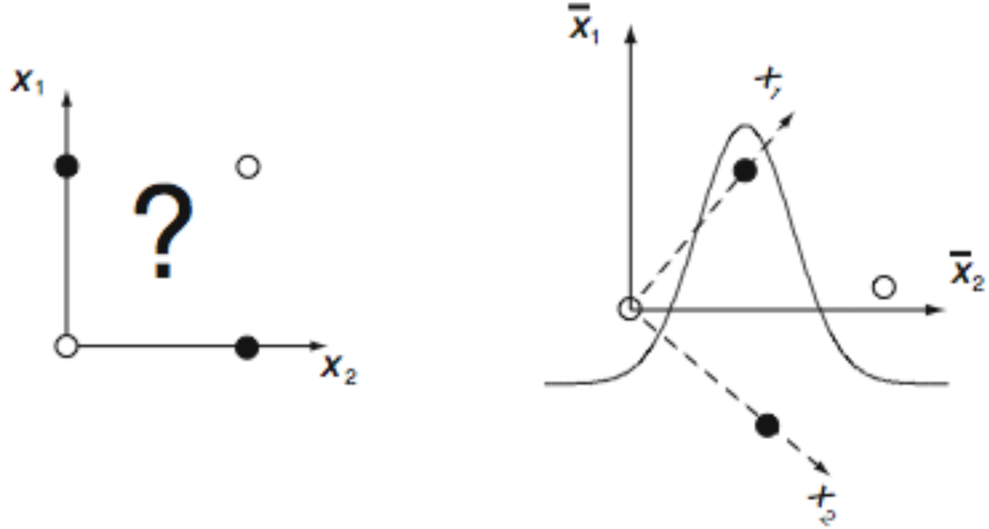
Example: OCR


A. Training pattern
>> displayLetter(1)
B. Learning curve
Average number of wrong bits
Training step
C. Generalization ability
Average number of wrong letters
Fraction of flipped bits
Threshold activation function
Max activation function

Limitation of a Perceptron

B. Boolean XOR function

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Can a Perceptron represent a XOR function?

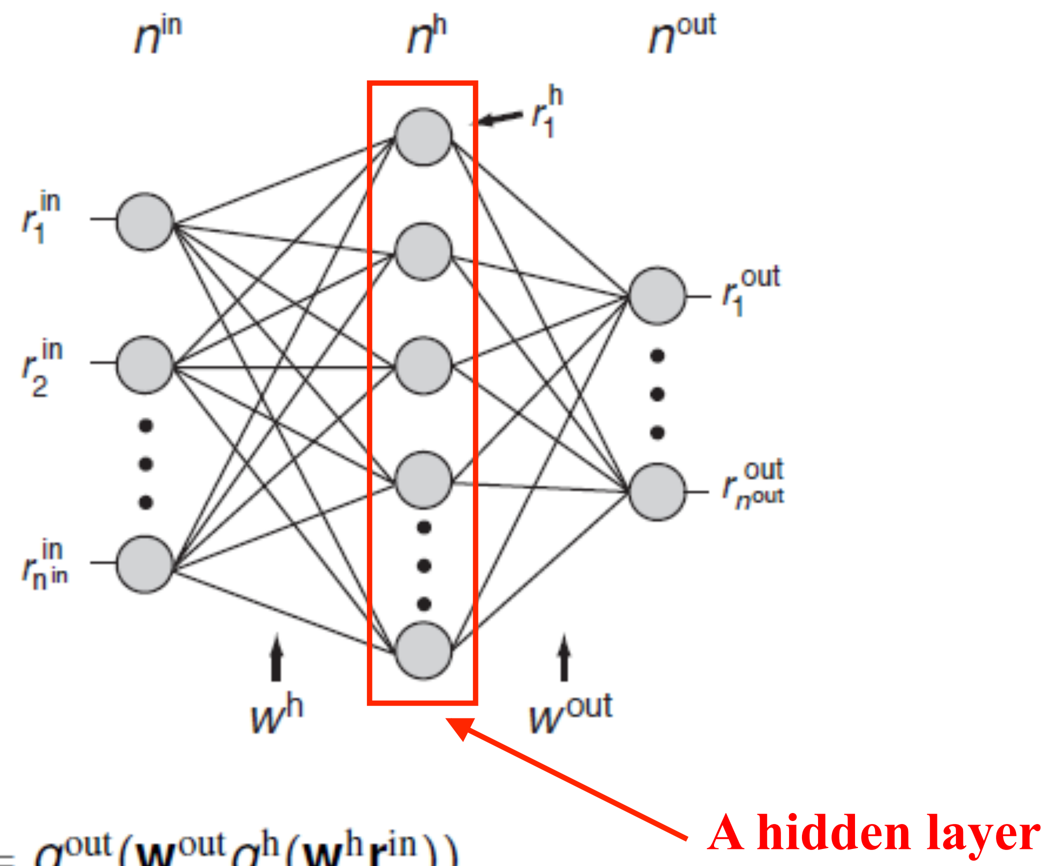
It is not linearly separable!

How about different activation function?

How about more complex problems?

Multi-layer perceptrons

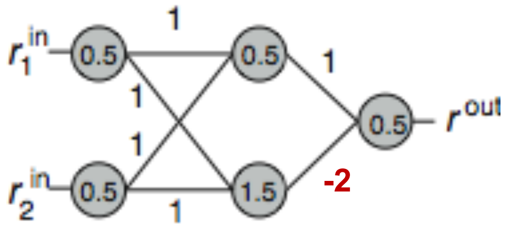
The multilayer perceptron (MLP) or Neural Networks (NN)



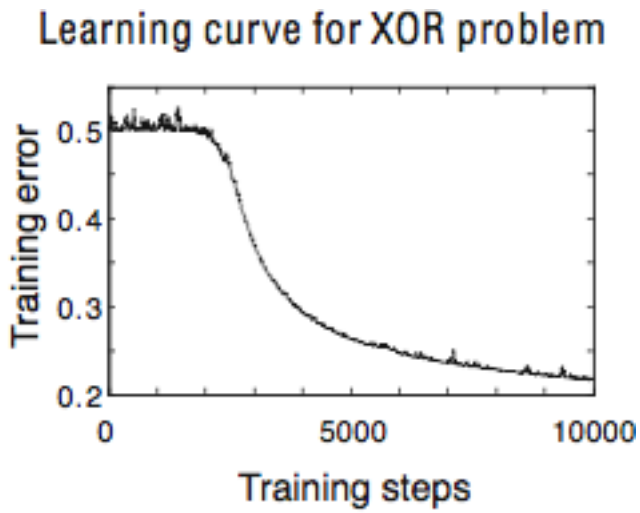
Update rule: $\mathbf{r}^{\text{out}} = g^{\text{out}}(\mathbf{w}^{\text{out}} g^{\text{h}}(\mathbf{w}^{\text{h}} \mathbf{r}^{\text{in}}))$

Learning rule (error backpropagation): $w_{ij} \leftarrow w_{ij} - \epsilon \frac{\partial E}{\partial w_{ij}}$

MLP for XOR function

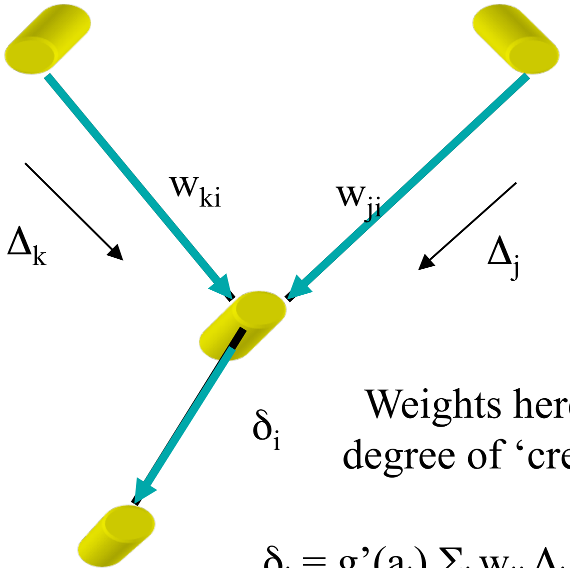


Error in textbook
figure 6.9, p. 159



The error-backpropagation algorithm (Rumelhart, Hinton and Williams, 1986)

Initialize weights arbitrarily
Repeat until error is sufficiently small
- Apply a sample pattern to the input nodes: $r_i^0 := r_i^{\mathrm{in}} = \xi_i^{\mathrm{in}}$
- Propagate input through the network by calculating the rates of nodes in successive layers l: $r_i^l = g(h_i^l) = g(\sum_j w_{ij}^l r_j^{l-1})$
- Compute the delta term for the output layer: $\delta_i^{\mathrm{out}} = g'(h_i^{\mathrm{out}})(\xi_i^{\mathrm{out}} - r_i^{\mathrm{out}})$
- Back-propagate delta terms through the network: $\delta_i^{l-1} = g'(h_i^{l-1}) \sum_j w_{ji}^l \delta_j^l$
- Update weight matrix by adding the term: $\Delta w_{ij}^l = \epsilon \delta_i^l r_j^{l-1}$



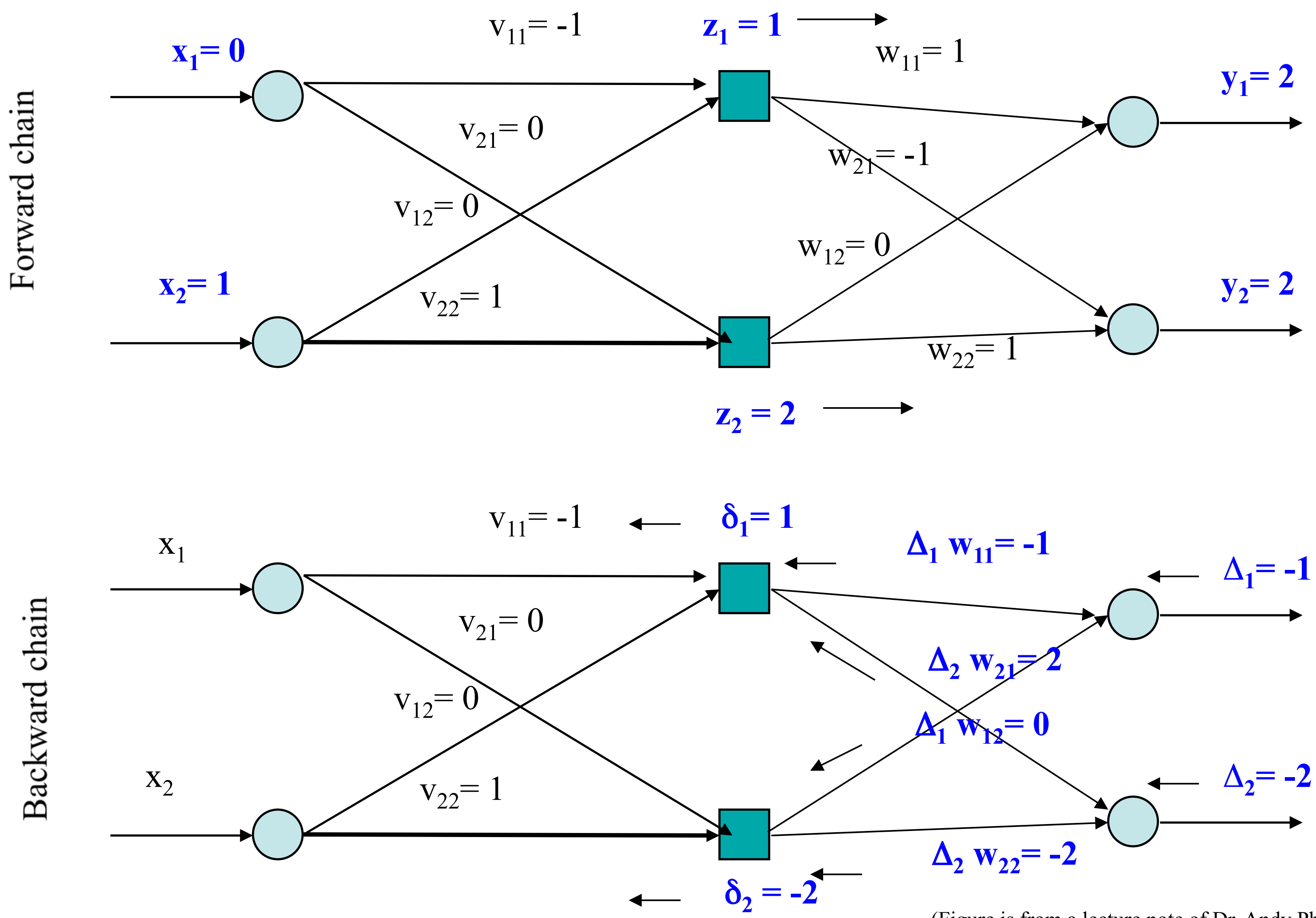
Weights here can be viewed as providing degree of ‘credit’ or ‘blame’ to hidden units

$\delta_i = g'(a_i)\, \Sigma_j\, w_{ji}\, \Delta_j$

(Figure is from a lecture note of Dr. Andy Philippides http://www.cogs.susx.ac.uk/users/andrewop/Courses/NN/NNIndex.html)

Example of delta computation

input [0 1] with target [1 0], Learning rate η = 0.1, all activation functions are linear units

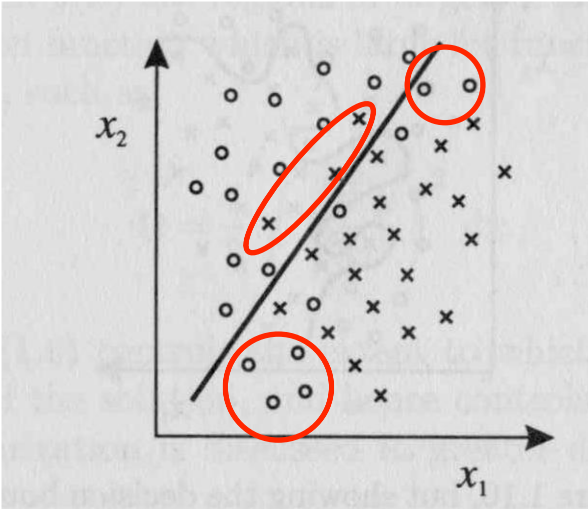


(Figure is from a lecture note of Dr. Andy Philippides
http://www.cogs.susx.ac.uk/users/andrewop/Courses/NN/NNIndex.html)

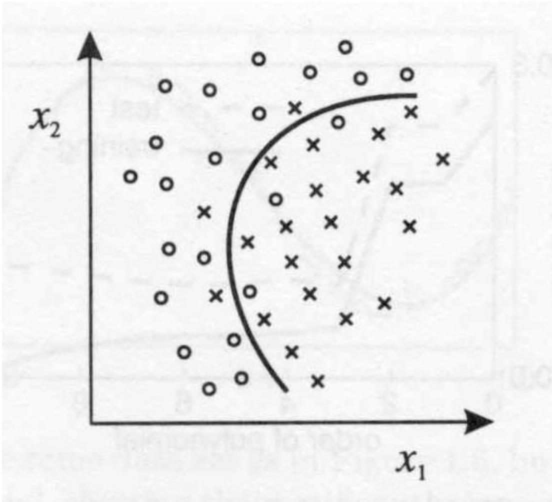
mlp.m

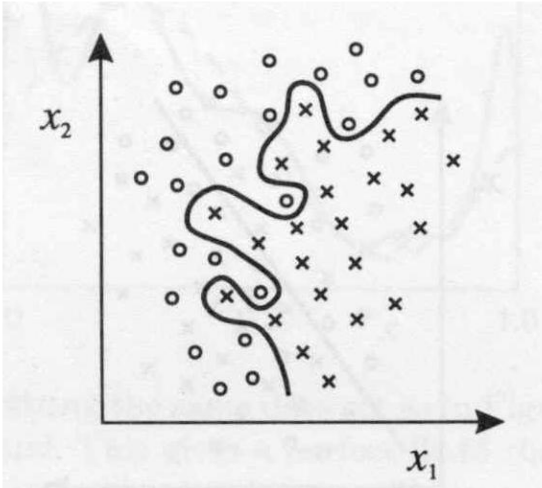
```
%% MLP with backpropagation learning on XOR problem
clear; clf;
N_i=2; N_h=2; N_o=1;
w_h=rand(N_h,N_i)-0.5; w_o=rand(N_o,N_h)-0.5;

% training vectors (XOR)
r_i=[0 1 0 1 ; 0 0 1 1];
r_d=[0 1 1 0];

% Updating and training network with sigmoid activation function
for sweep=1:10000;
  % training randomly on one pattern
    i=ceil(4*rand);
    r_h=1./(1+exp(-w_h*r_i(:,i)));
    r_o=1./(1+exp(-w_o*r_h));
    d_o=(r_o.*(1-r_o)).*(r_d(:,i)-r_o);
    d_h=(r_h.*(1-r_h)).*(w_o'*d_o);
    w_o=w_o+0.7*(r_h*d_o')';
    w_h=w_h+0.7*(r_i(:,i)*d_h')';
  % test all pattern
    r_o_test=1./(1+exp(-w_o*(1./(1+exp(-w_h*r_i)))));
    d(sweep)=0.5*sum((r_o_test-r_d).^2);
end
plot(d)
```

Overfitting and underfitting



Underfitting



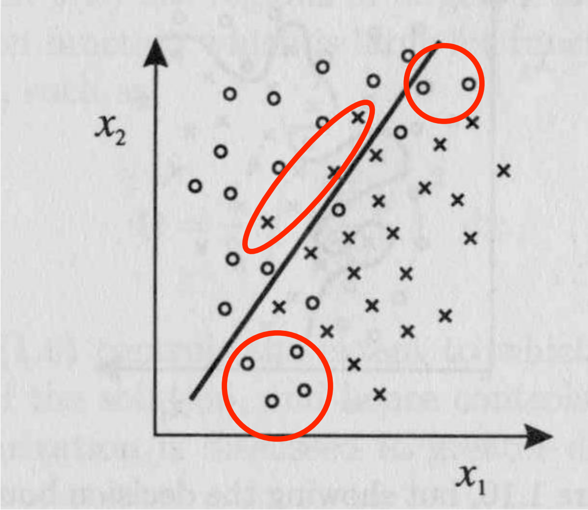


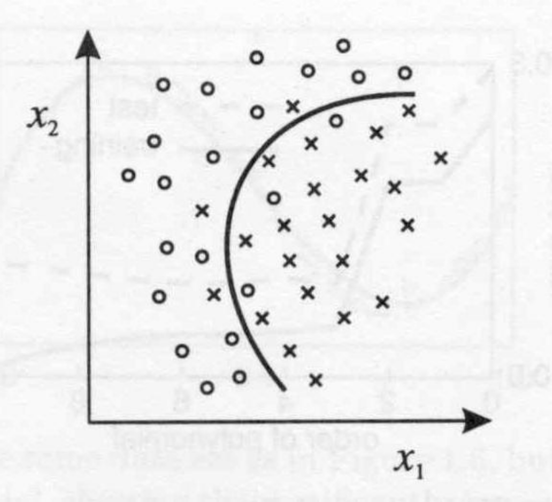
Overfitting

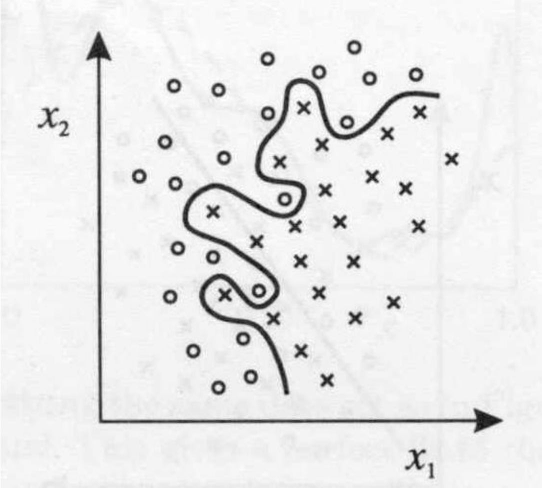
Why does it happen?

(Figures are from Bishop, 1995)

Overfitting and underfitting







Underfitting

Overfitting

Why does it happen?

Network is too simple
Training is too short

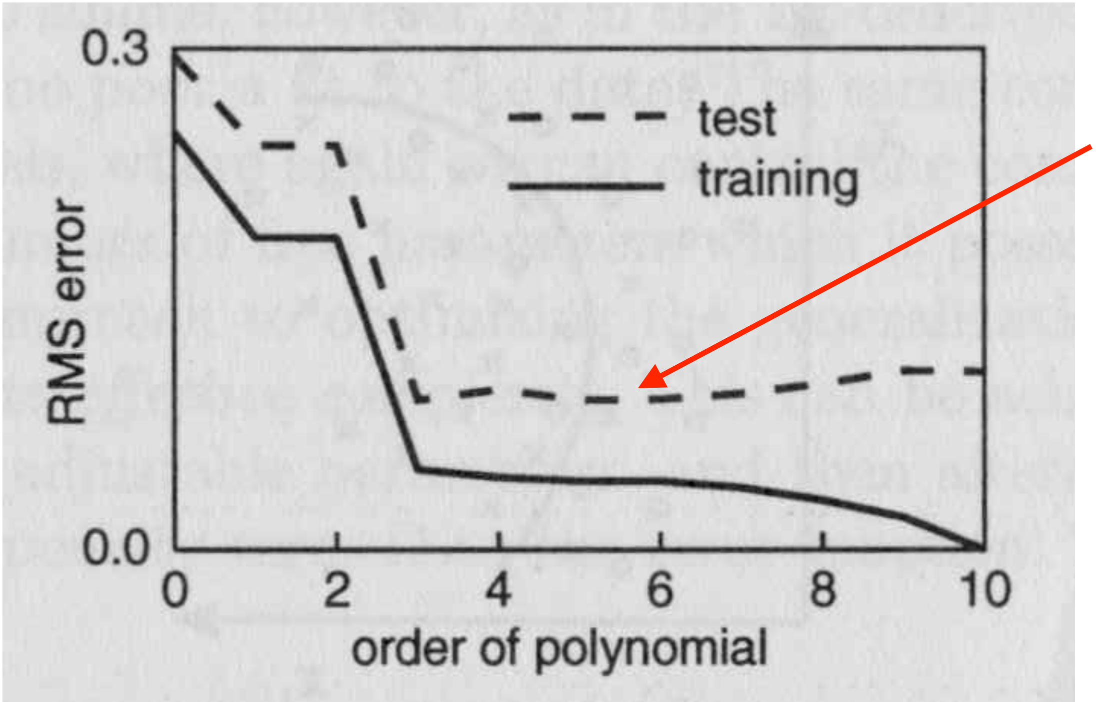
Network is too complex
Training is too long

(Figures are from Bishop, 1995)

Test error vs. Train error

Can you believe that the given training examples capture the true function?

Examples are always noisy. So though the error over training examples is low, NN can fail to capture the true function.
Test set: Another set of data, which is differently sampled from a training set.



Best test error (with reasonably low training error)

Lower ← Network Complexity → Higher

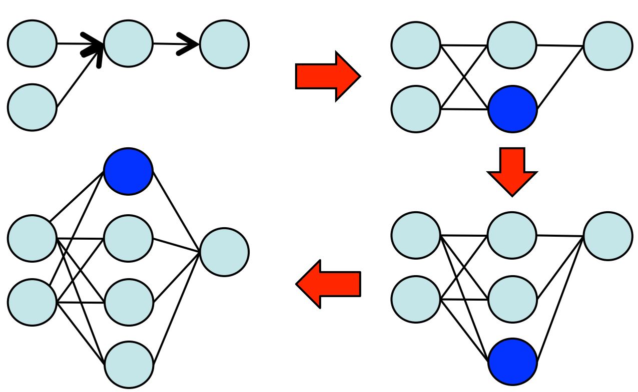
Underfitting ↔ Overfitting

cf. Cross-validation

(Figure is from Bishop, 1995)

How many hidden neurons are required? (design algorithm)

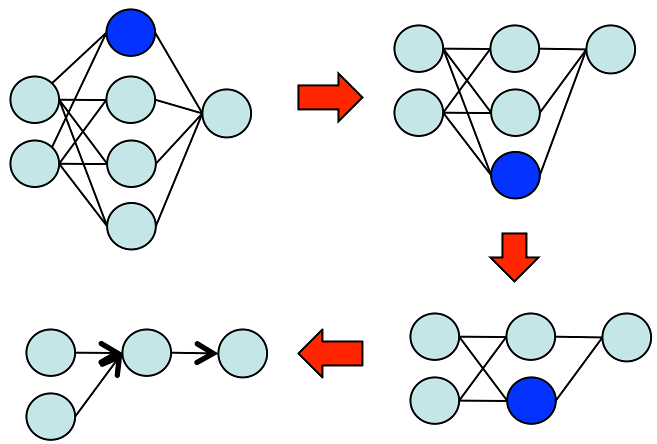
A node creation algorithm



Start with a small network and repeat adding nodes systematically until required performance is reached. (Fig 6.15)

1. Dynamic node creation (Ash) when error is not decreasing, add a node
2. Meiosis network (Hanson) when a weight varied too much, split a node into two.

A pruning algorithm



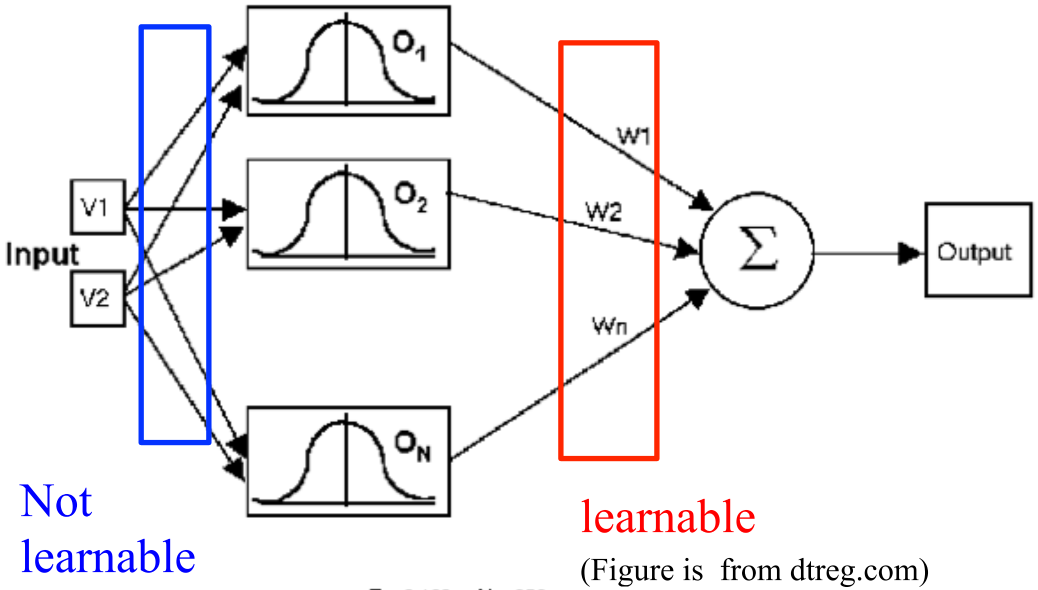
Start with a large network and remove nodes systematically.

1. Weight decay
   Weight are always decaying and too small weight (not used to generate an output) is considered 'disconnected'
2. Optimal brain damage
   If the activation of a neuron is too small compared to others, the neuron is removed.

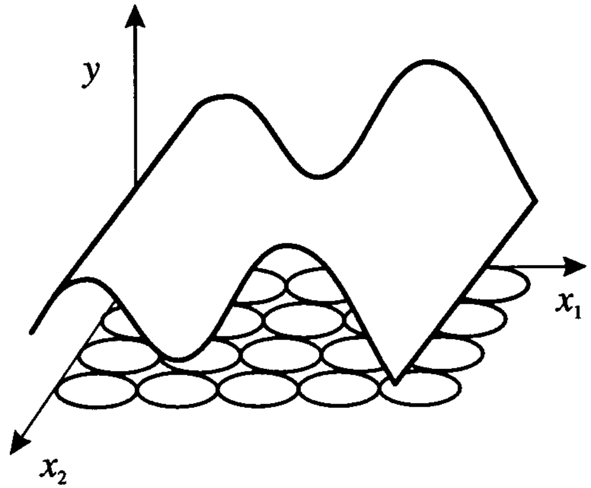
MLP: Advanced concepts (radial basis functions and recurrent networks)

Radial Basis Function (RBF) network

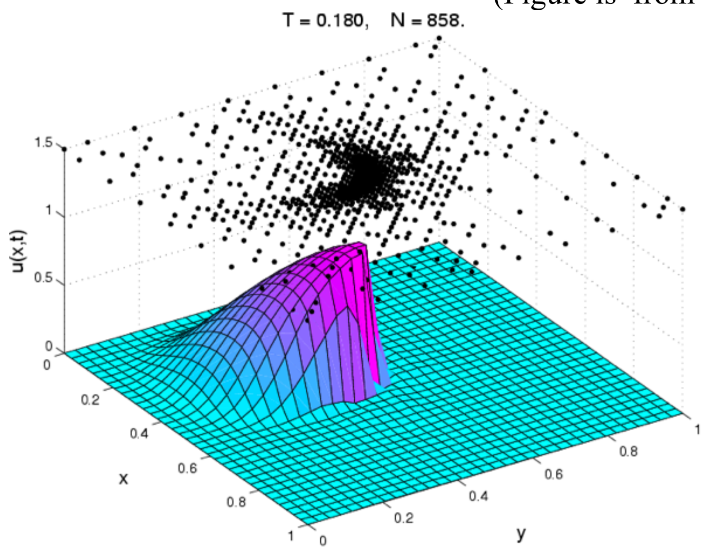
In general, "localized" radial basis function. ex. Gaussian functions.



(Figure is from dtreg.com)



(from Bishop,1995)

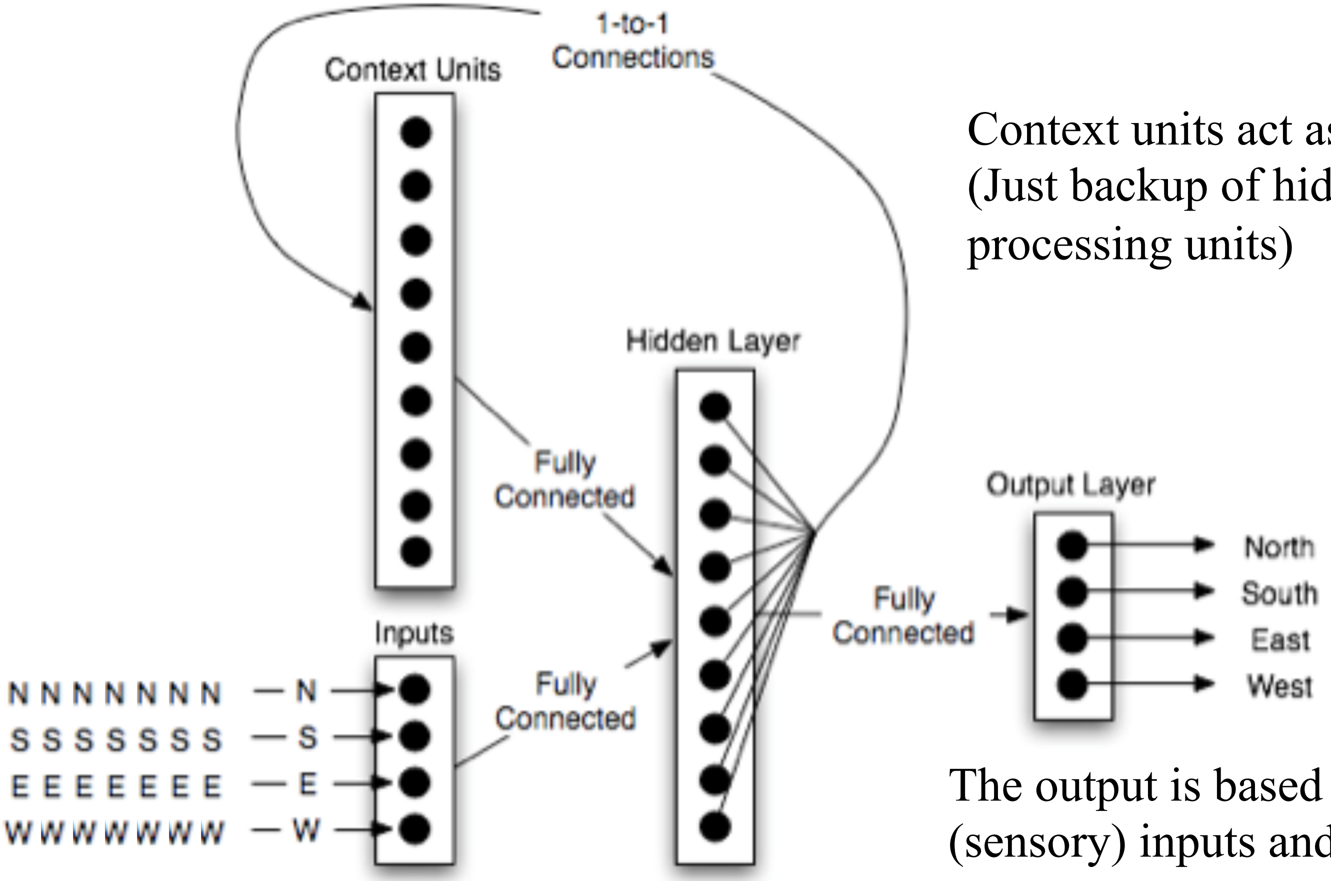


(http://www.mathworks.com/matlabcentral/fx_files/13205/1/burgers2d.png)

If RBFs are well located, we can capture a function with a small number of RBFs and higher accuracy.
What does their extents mean?
(their receptive fields)
How to locate centers?
(Self-organization or unsupervised learning.)

Elman network

Remember a previous state. (A simplified version of recurrent network.)



Context units act as a memory
(Just backup of hidden neurons, not a processing units)

The output is based on the current (sensory) inputs and context units.

Sequence of inputs coming.

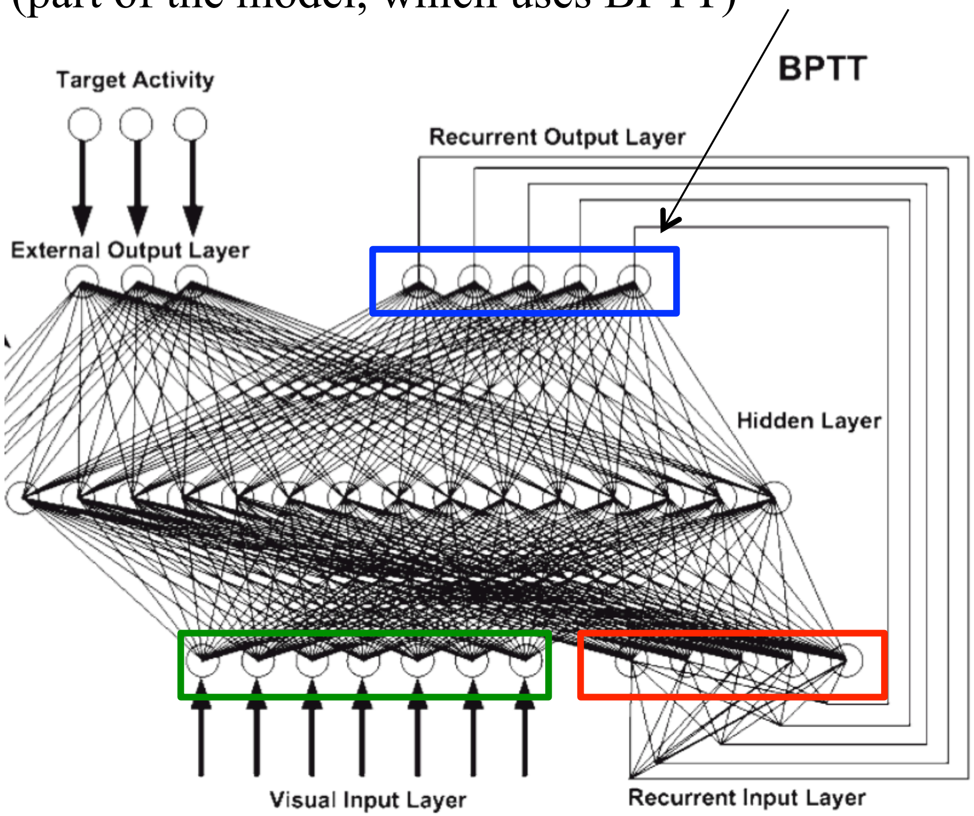
(figure from http://www.fabioruini.eu/blog/2008/02/)

Full recurrent neural network

Can be learned through Back Propagation Through Time (Werbos, 1990)

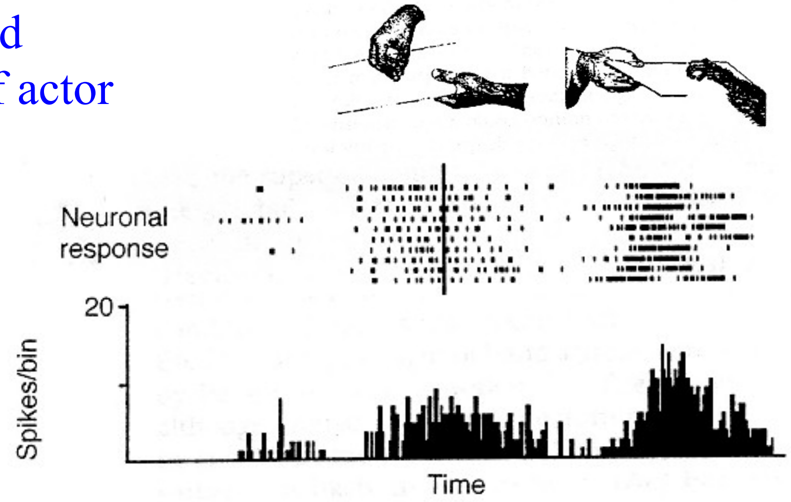
Remember a sequence of states
(part of the model, which uses BPTT)

2. Estimated
intention of actor



1. Actor's current
behavior

3. What was the last
estimated intention
of actor?



(Figure is from Rizzolatti et al.)

Continue estimation using the current belief of the actor's intention (the previous network output)

Mirror Neuron
Estimate other's intention
(Bonauito, Rosta, & Arbib,2007)

Peceptrons (single layer)

- linearly separable
- error function, gradient descent

Multi-layer perceptrons

- back-propagation error signal
- overfitting and underfitting
- test and training data

- Radial Basis Functions
- Elman network (recurrent networks)

Further readings

Simon Haykin (1999), **Neural networks: a comprehensive foundation**, MacMillan (2nd edition).

John Hertz, Anders Krogh, and Richard G. Palmer (1991), **Introduction to the theory of neural computation**, Addison-Wesley.

Berndt Müller, Joachim Reinhardt, and Michael Thomas Strickland (1995), **Neural Networks: An Introduction**, Springer

Christopher M. Bishop (2006), **Pattern Recognition and Machine Learning**, Springer

Laurence F. Abbott and Sacha B. Nelson (2000), **Synaptic plasticity: taming the beast**, in **Nature Neurosci. (suppl.)**, 3: 1178–83.

Christopher J. C. Burges (1998), **A Tutorial on Support Vector Machines for Pattern Recognition** in **Data Mining and Knowledge Discovery** 2:121–167.

Alex J. Smola and Bernhard Schölhopf (2004), **A tutorial on support vector regression** in **Statistics and computing** 14: 199-222.

David E. Rumelhart, James L. McClelland, and the PDP research group (1986), **Parallel Distributed Processing: Explorations in the Microstructure of Cognition**, MIT Press.

Peter McLeod, Kim Plunkett, and Edmund T. Rolls (1998), **Introduction to connectionist modelling of cognitive processes**, Oxford University Press.

E. Bruce Goldstein (1999), **Sensation & perception**, Brooks/Cole Publishing Company (5th edition).