

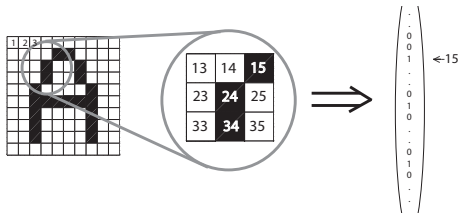
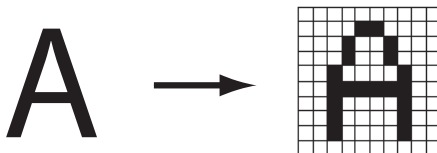
# Fundamentals of Computational Neuroscience 2e

Thomas Trappenberg

February 7, 2009

Chapter 6: Feed-forward mapping networks

# Digital representation of letter



**Optical character recognition:** Predict meaning from features.  
E.g., given features  $\mathbf{x}$ , what is the character  $\mathbf{y}$

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

# Further examples given by lookup table

A. Boolean AND function

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

B. Non-Boolean function

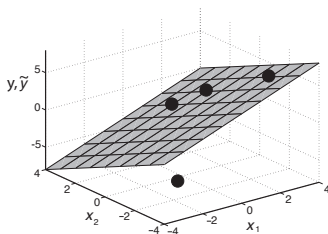
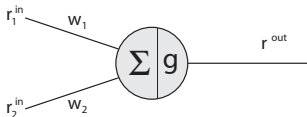
$x_1$	$x_2$	$y$
1	2	-1
2	1	1
3	-2	5
-1	-1	7
$\vdots$	$\vdots$	$\vdots$

# 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  $g(x) = x$ :

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

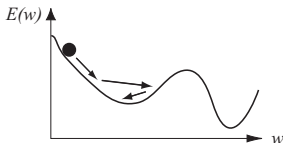


# How to find the right weight values?

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

$$E = \frac{1}{2} \sum_i (r_i^{\text{out}} - y_i)^2$$

**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



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:  $\Delta w_{ij} = \epsilon \delta_i r_j^{\text{in}}$

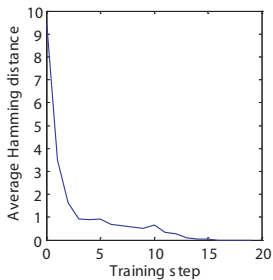
# Example: OCR

A. Training pattern

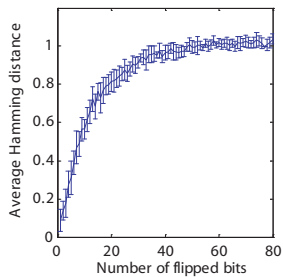
```
>> displayLetter(1)
```

```
  +++  
  +++  
 +++++  
++ ++  
++ ++  
+++ ++  
+++++++  
+++++++  
+++ ++  
+++ ++  
+++ ++  
+++ ++
```

B. Learning curve



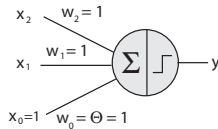
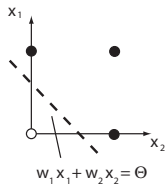
C. Generalization ability



# Example: Boolean function

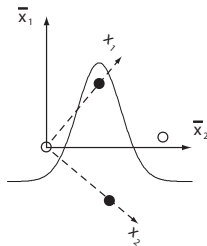
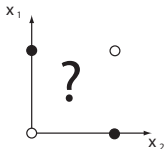
## A. Boolean OR function

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



## B. Boolean XOR function

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

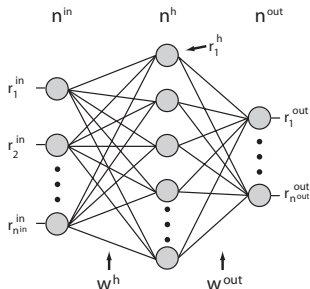


# perceptronTrain.m

```
1  %% Letter recognition with threshold perceptron
2  clear; clf;
3  nIn=12*13; nOut=26;
4  wOut=rand(nOut,nIn)-0.5;
5
6  % training vectors
7  load pattern1;
8  rIn=reshape(pattern1', nIn, 26);
9  rDes=diag(ones(1,26));
10
11 % Updating and training network
12 for training_step=1:20;
13     % test all pattern
14     rOut=(wOut*rIn)>0.5;
15     distH=sum(sum((rDes-rOut).^2))/26;
16     error(training_step)=distH;
17     % training with delta rule
18     wOut=wOut+0.1*(rDes-rOut)*rIn';
19 end
20
21 plot(0:19,error)
22 xlabel('Training step')
23 ylabel('Average Hamming distance')
```



# Multilayer Perceptron (MLP)



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):  $w_{ij} \leftarrow w_{ij} - \epsilon \frac{\partial E}{\partial w_{ij}}$

Initialize weights arbitrarily

Repeat until error is sufficiently small

Apply a sample pattern to the input nodes:  $r_i^0 := r_i^{\text{in}} = \xi_i^{\text{in}}$

Propagate input through the network by calculating the rates of nodes in successive layers:  $r_i^l = g(h_i^l) = g(\sum_j w_{ij}^l r_j^{l-1})$

Compute the delta term for the output layer:  $\delta_i^{\text{out}} = g'(h_i^{\text{out}})(\xi_i^{\text{out}} - r_i^{\text{out}})$

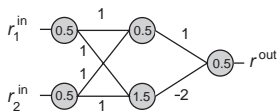
Back-propagate delta terms through the network:  $\delta_i^{l-1} = g'(h_i^{l-1}) \sum_j w_{ji}^l \delta_j^l$

Update weight matrix by adding the term:  $\Delta w_{ij}^l = \epsilon \delta_i^l r_j^{l-1}$

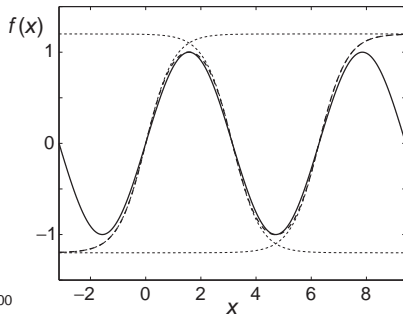
# perceptronTrain.m

```
1  %% MLP with backpropagation learning on XOR problem
2  clear; clf;
3  N_i=2; N_h=2; N_o=1;
4  w_h=rand(N_h,N_i)-0.5; w_o=rand(N_o,N_h)-0.5;
5
6  % training vectors (XOR)
7  r_i=[0 1 0 1 ; 0 0 1 1];
8  r_d=[0 1 1 0];
9
10 % Updating and training network with sigmoid activation function
11 for sweep=1:10000;
12     % training randomly on one pattern
13     i=ceil(4*rand);
14     r_h=1./(1+exp(-w_h*r_i(:,i)));
15     r_o=1./(1+exp(-w_o*r_h));
16     d_o=(r_o.*(1-r_o)).*(r_d(:,i)-r_o);
17     d_h=(r_h.*(1-r_h)).*(w_o'*d_o);
18     w_o=w_o+0.7*(r_h*d_o)';
19     w_h=w_h+0.7*(r_i(:,i)*d_h)';
20     % test all pattern
21     r_o_test=1./(1+exp(-w_o*(1./(1+exp(-w_h*r_i)))));
22     d(sweep)=0.5*sum((r_o_test-r_d).^2);
23 end
24 plot(d)
```

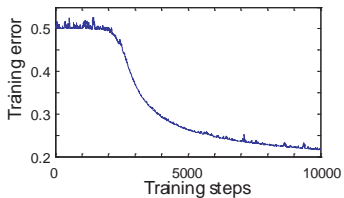
A. MLP for representing the XOR function



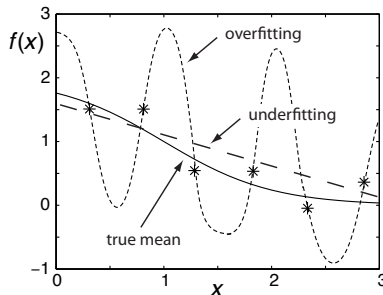
B. Approximation of sin functions by a small MLP



C. Learning curve for XOR problem



# Overfitting and underfitting

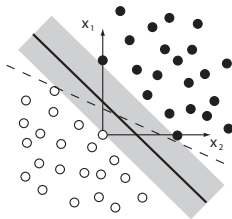


Regularization, for example

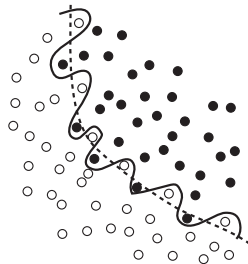
$$E = \frac{1}{2} \sum_i (r_i^{\text{out}} - y_i)^2 - \gamma_r \frac{1}{2} \sum_i w_i^2$$

# Support Vector Machines

A. Linear large margin classifier



D. Non-linear separation



B. Linear not separable case



C. Linear separable case



# 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 Schölkopf (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).