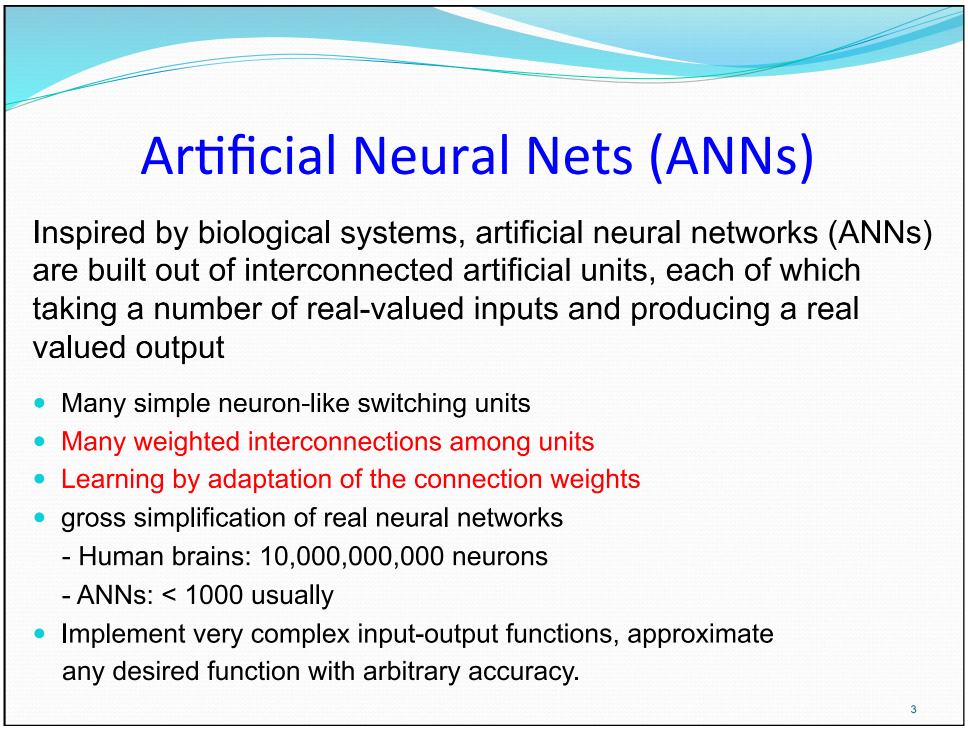
Artificial Neural Networks

Ning Xiong
Mälardalen University

Biological Inspiration

- Some numbers…
  - The human brain contains about 10 billion nerve cells (neurons)
  - Each neuron is connected to other (10^4 - 10^5) neurons through synapses
- Properties of the brain
  - It can learn from experience
  - It adapts to the environment
  - It has a high degree of parallel information processing

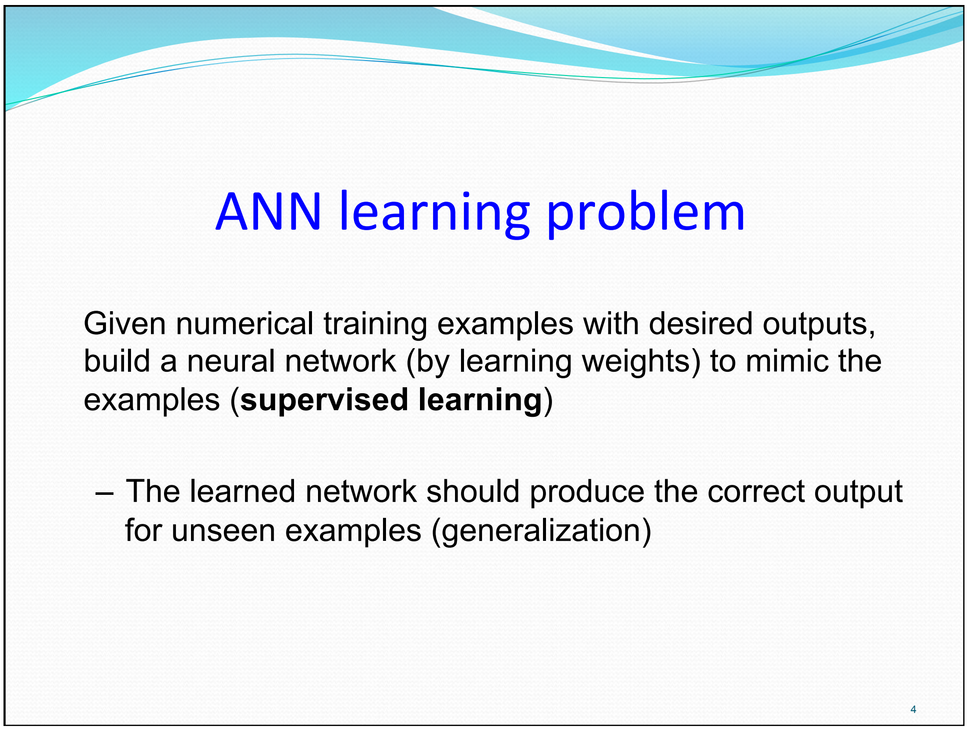


Artificial Neural Nets (ANNs)

Inspired by biological systems, artificial neural networks (ANNs) are built out of interconnected artificial units, each of which taking a number of real-valued inputs and producing a real valued output

- Many simple neuron-like switching units
- Many weighted interconnections among units
- Learning by adaptation of the connection weights
- gross simplification of real neural networks
  - Human brains: 10,000,000,000 neurons
  - ANNs: < 1000 usually
- Implement very complex input-output functions, approximate any desired function with arbitrary accuracy.

3



ANN learning problem

Given numerical training examples with desired outputs, build a neural network (by learning weights) to mimic the examples (**supervised learning**)

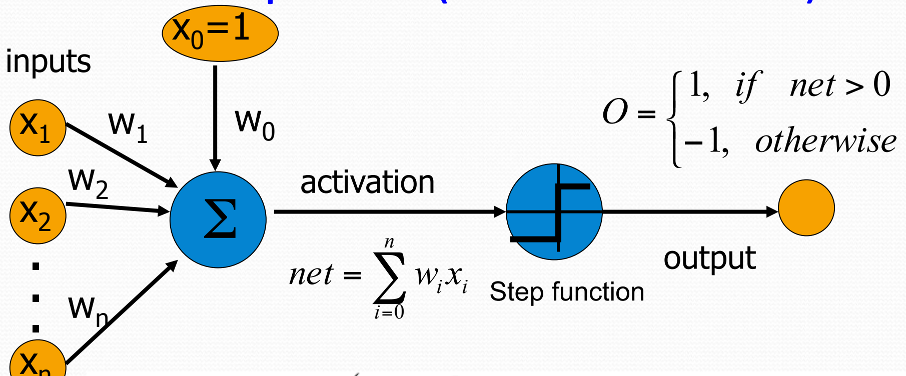
– The learned network should produce the correct output for unseen examples (generalization)

4

Agenda

- Perceptrons (Threshold units) and learning
- Multi-layer networks
- Backpropagation algorithm (gradient descent)
- Counter-measure against overfitting

Perceptrons (Threshold Unit)



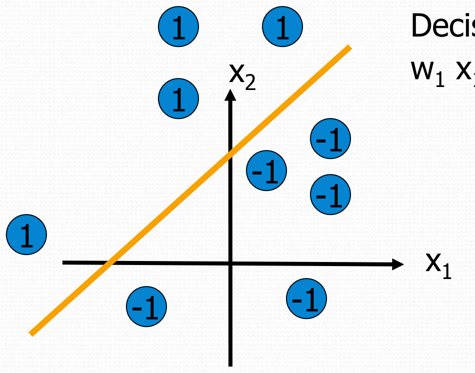
$$net = \sum_{i=0}^{n} w_i x_i$$

$$O = \begin{cases} 1, & if \quad net > 0 \\ -1, & otherwise \end{cases}$$

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Decision Surface of Perceptron



Decision line

$w_1 x_1 + w_2 x_2 + w_0 = 0$

Perceptron represents a hyperplane decision surface in the n-dimensional space of instances

Perceptron Learning Rule

Given a training example, revise weights using the following rule:

$$\Delta w_i = \eta(t-o)x_i, \quad \Delta w_0 = \eta(t-o)\bullet 1$$

$$w_i \leftarrow w_i + \Delta w_i$$

where

- t is the target value specified by the training example
- o is the perceptron output
- x_i is the attribute value in the example
- The parameter η is called the *learning rate*.

- If the output is correct (t-o=0) the weights are not changed (Δw_i =0).
- If the output is incorrect ($t-o \neq o$), the weights w_i are changed such that the activation for perceptron is increased or decreased

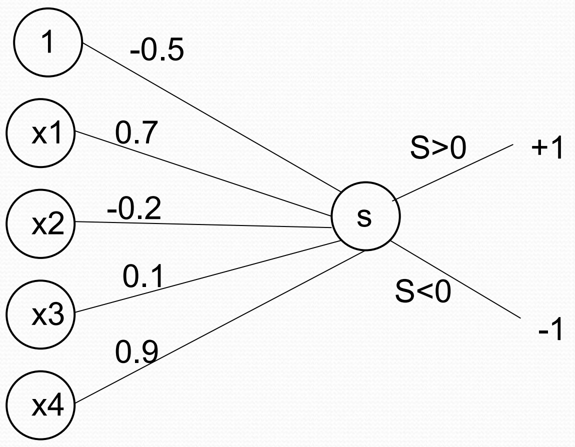
Perceptron Training Algorithm

```
Repeat
  For each training example (x,t)
    compute the output o using x as the input
        if o≠t then
                form a new weight vector w' according to
                  $w' = w + \eta (t-o) x$
        else
          do nothing
        end if
  End for (epoch)
Until o=t for all training examples
```

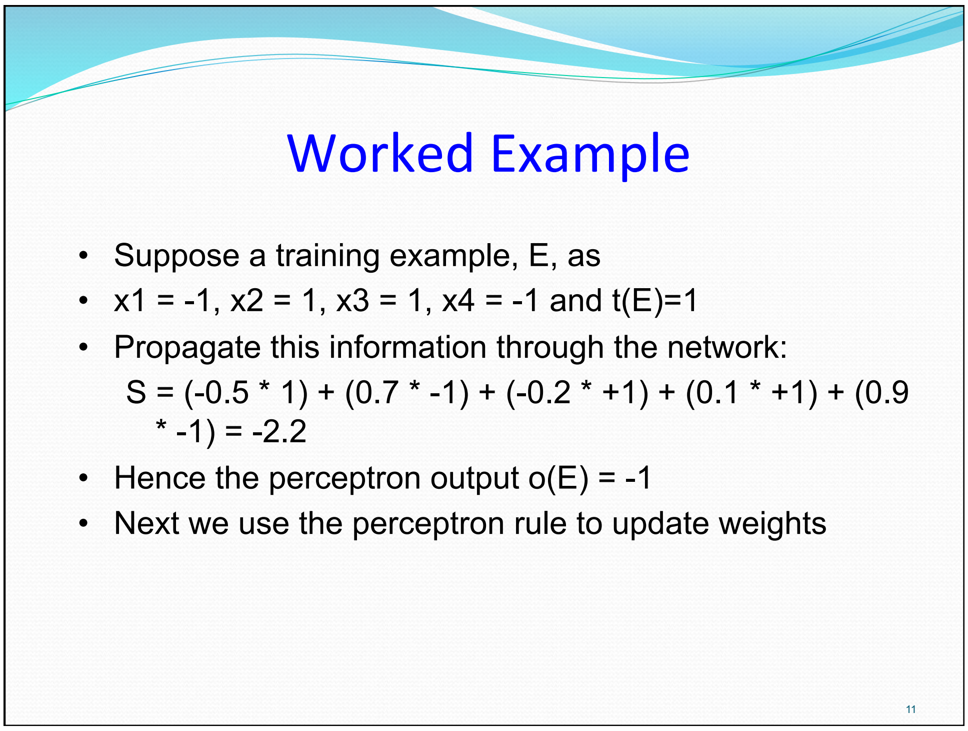
9

Worked Example

- Suppose we have a set of random weights for the perceptron
- Use a learning rate of η = 0.1



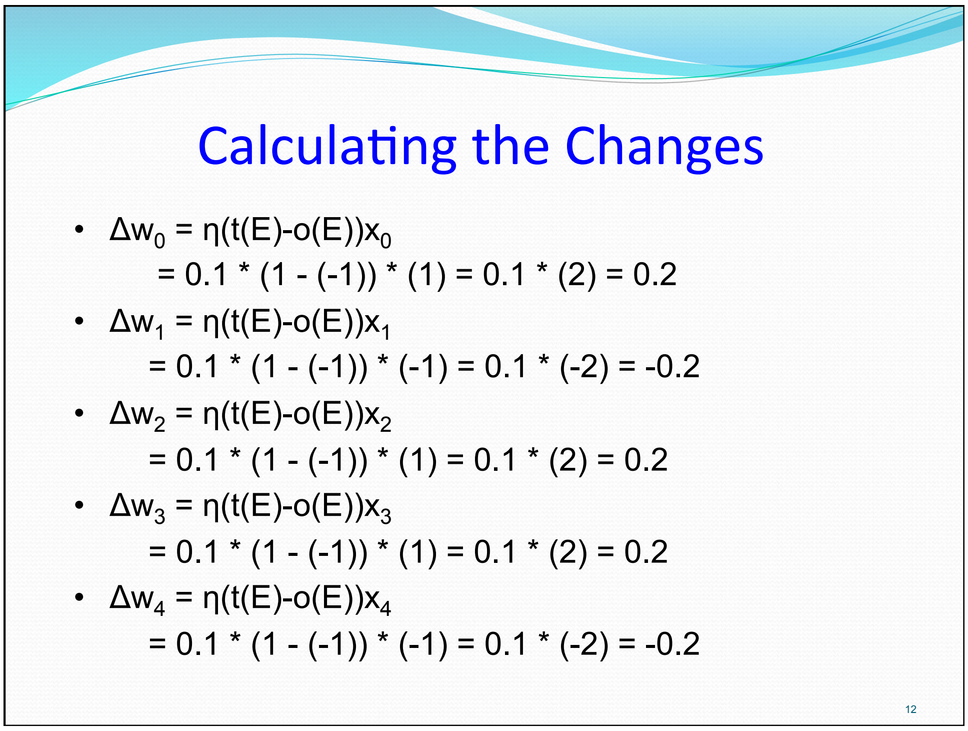
10



Worked Example

- Suppose a training example, E, as
- x1 = -1, x2 = 1, x3 = 1, x4 = -1 and t(E)=1
- Propagate this information through the network:

  S = (-0.5 * 1) + (0.7 * -1) + (-0.2 * +1) + (0.1 * +1) + (0.9 * -1) = -2.2
- Hence the perceptron output o(E) = -1
- Next we use the perceptron rule to update weights



Calculating the Changes

- $\Delta w_0 = \eta(t(E)\text{-}o(E))x_0$

  = 0.1 * (1 - (-1)) * (1) = 0.1 * (2) = 0.2
- $\Delta w_1 = \eta(t(E)\text{-}o(E))x_1$

  = 0.1 * (1 - (-1)) * (-1) = 0.1 * (-2) = -0.2
- $\Delta w_2 = \eta(t(E)\text{-}o(E))x_2$

  = 0.1 * (1 - (-1)) * (1) = 0.1 * (2) = 0.2
- $\Delta w_3 = \eta(t(E)\text{-}o(E))x_3$

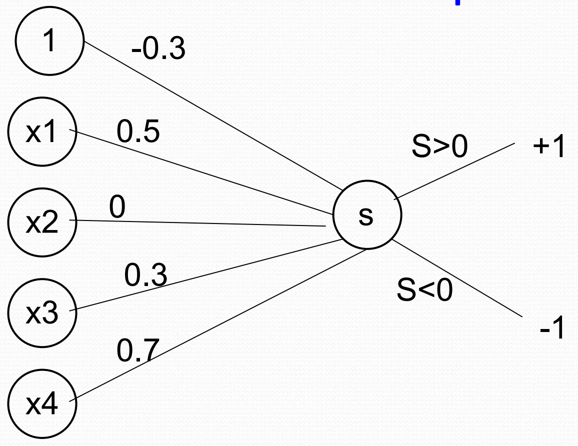
  = 0.1 * (1 - (-1)) * (1) = 0.1 * (2) = 0.2
- $\Delta w_4 = \eta(t(E)\text{-}o(E))x_4$

  = 0.1 * (1 - (-1)) * (-1) = 0.1 * (-2) = -0.2

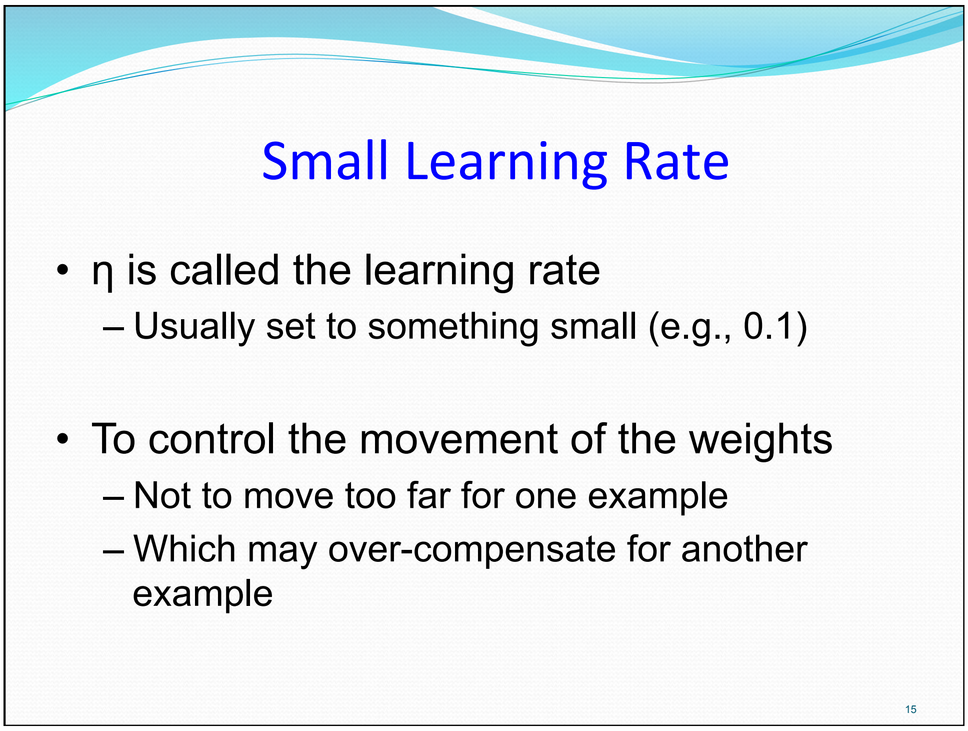
Calculating the New Weights

- $w'_0 = -0.5 + \Delta w_0 = -0.5 + 0.2 = -0.3$
- $w'_1 = 0.7 + \Delta w_1 = 0.7 -0.2 = 0.5$
- $w'_2 = -0.2 + \Delta w_2 = -0.2 + 0.2 = 0$
- $w'_3= 0.1 + \Delta w_3 = 0.1 + 0.2 = 0.3$
- $w'_4 = 0.9 + \Delta w_4 = 0.9 - 0.2 = 0.7$

New Revised Perceptron



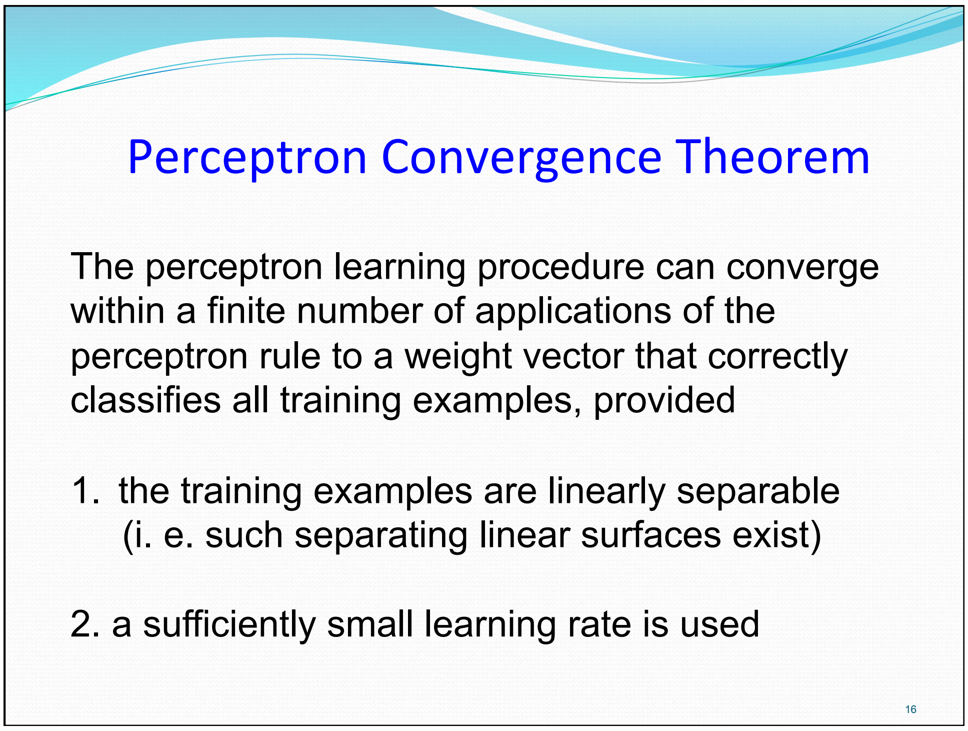
- Calculate for the example, E, again:
  S = (-0.3 * 1) + (0.5 * -1) + (0 * +1) + (0.3 * +1) + (0.7 * -1) = -1.2
- Still gets the wrong categorisation
  - But the value is closer to zero (from -2.2 to -1.2)
  - In a few epochs time, this example will be correctly classified



Small Learning Rate

- η is called the learning rate
  - Usually set to something small (e.g., 0.1)
- To control the movement of the weights
  - Not to move too far for one example
  - Which may over-compensate for another example

15



Perceptron Convergence Theorem

The perceptron learning procedure can converge within a finite number of applications of the perceptron rule to a weight vector that correctly classifies all training examples, provided

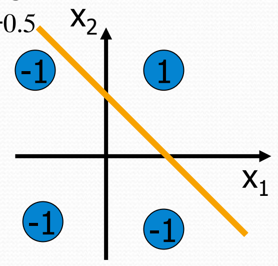
1. the training examples are linearly separable (i. e. such separating linear surfaces exist)

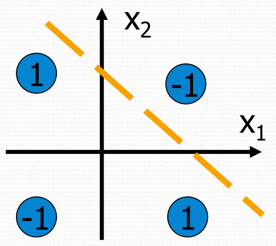
2. a sufficiently small learning rate is used

16

Linear Separability?

$w_1=0.5$
$w_2=0.5$
$w_0=-0.5$





Logical AND

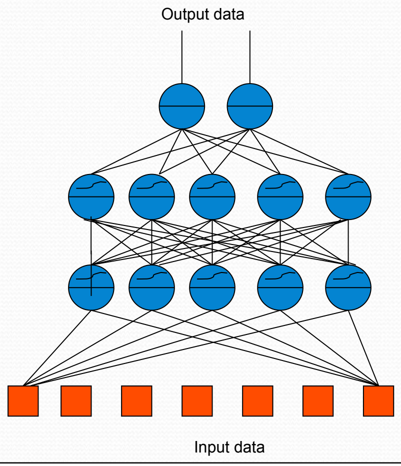
x_1	x_2	net	y
-1	-1	-1.5	-1
-1	1	-0.5	-1
1	-1	-0.5	-1
1	1	0.5	1

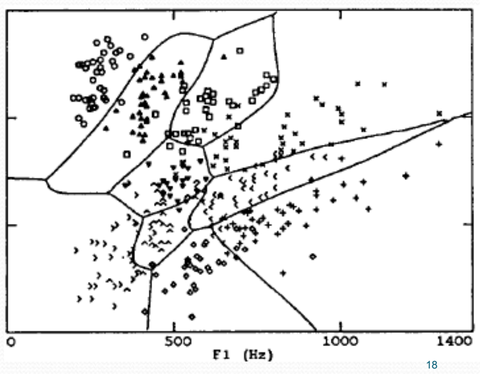
Logical XOR

x_1	x_2	y	
-1	-1	-1	
-1	1	1	
1	-1	1	
1	1	-1	

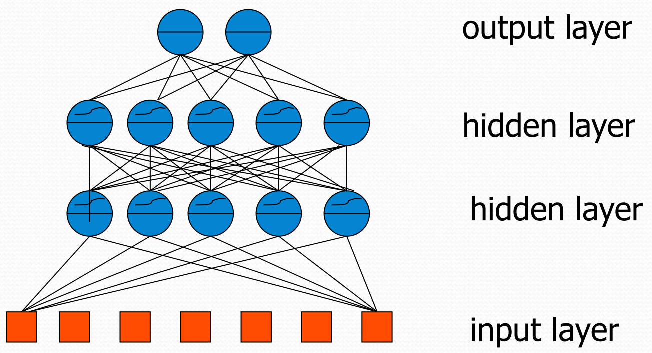
Multilayer Network of Units

To represent more complex nonlinear decision surfaces, we need a network to combine many units.





Multi-Layer Networks



Learning: Finding a weight vector to best approximate the training examples, i.e. have network outputs as close to target values as possible

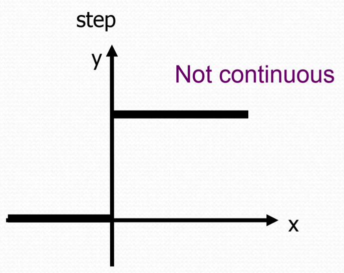
19

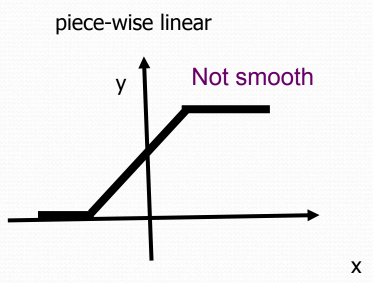
Which Units in a Network?

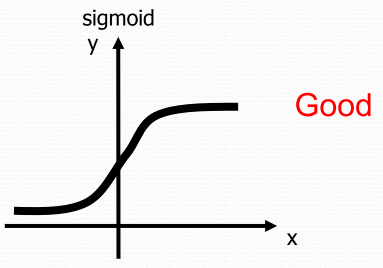
- We want to implement a complex, nonlinear but possibly smooth function by neural networks.
- Step functions aren't continuous and smooth
  - They are not continuous at the threshold
- Alternative function for unit output
  - Must be non-linear
  - Must be continuous and smooth
  - Must be similar to step function but showing behaviour as a "soft" switching

20

Looking at Alternative Functions



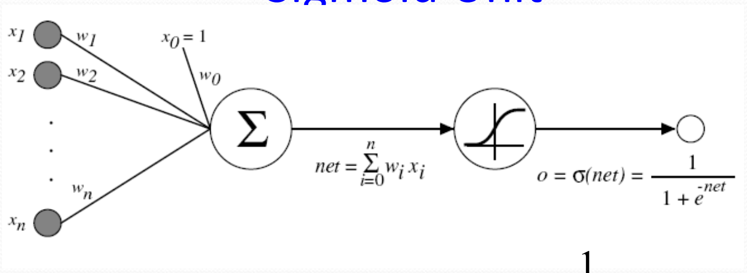




Good

21

Sigmoid Unit


x_1
x_2
x_n
w_1
w_2
w_n
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$

σ(x) is the sigmoid function: $\sigma(x) = \dfrac{1}{1+e^{-x}}$

Nice property: smooth, similar to step function, derivative expressed in terms of itself

$$\frac{d\sigma(x)}{d(x)} = \sigma(x)(1-\sigma(x))$$

22

Derivative of Sigmoid Function

$$\frac{\partial \sigma(x)}{\partial x} = -\frac{1}{(1+e^{-x})^2}\frac{\partial}{\partial x}(1+e^{-x}) = -\frac{1}{(1+e^{-x})^2}e^{-x}\frac{\partial}{\partial x}(-x)$$

$$= -\frac{1}{(1+e^{-x})^2}e^{-x}(-1) = \frac{1}{1+e^{-x}}\cdot\frac{e^{-x}}{1+e^{-x}}$$

$$= \frac{1}{1+e^{-x}}(1-\frac{1}{1+e^{-x}}) = \sigma(x)(1-\sigma(x))$$

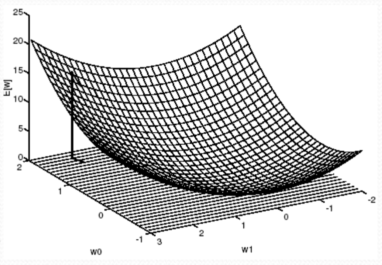
23

Gradient Descent for Learning

We **incrementally** update weights in terms of individual instances. Given a training example d, we define the error function as

$$E_d(\vec{w}) = \frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2$$

The idea is to modify the weights according to the negative of gradient of the error function to get a fast reduction of error on this example, so we revise weights according to gradient information as



$$\Delta w_{ji} = -\eta\frac{\partial E_d}{\partial w_{ji}}, w_{ji} = w_{ji} + \Delta w_{ji}$$

η is the learning rate specifying the step size in the gradient search

24

Some Notations

o_j: the output of unit j

t_j : the target output for unit j (only for output layer)

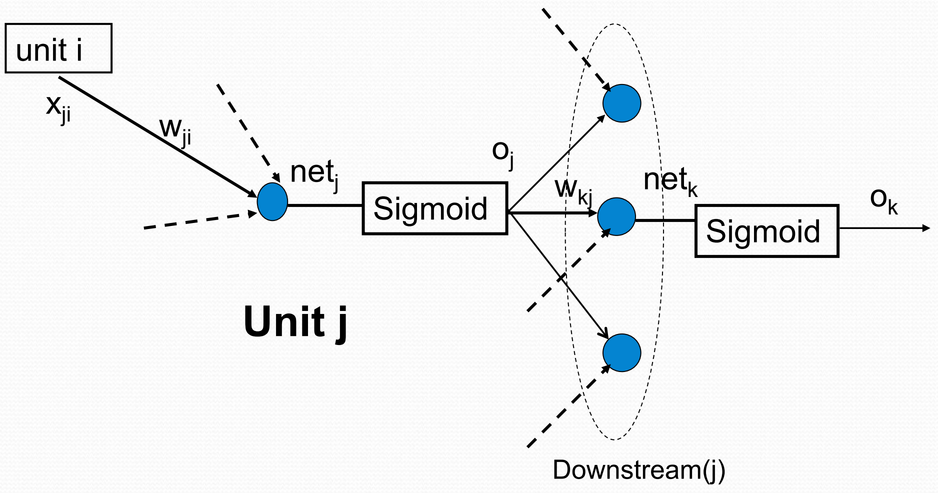
net_j: the weighted sum of inputs at unit j

x_{ji}: the input from unit i to unit j, w_{ji} denotes the corresponding weight

Downstream(j): the set of units whose immediate inputs include the output of unit j.

25

Explaining Notations



26

Updating of Weights

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j}x_{ji}$$

Now the update for weight can be rewritten as $\Delta w_{ji} = \eta(-\frac{\partial E_d}{\partial net_j})x_{ji}$

Let $\delta_j = -\frac{\partial E_d}{\partial net_j}$ referred as the error term of unit j

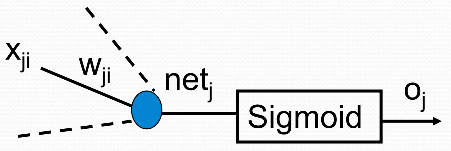
the weight update rule is formulated as

$$\Delta w_{ji} = \eta\delta_j x_{ji}$$

The key issue now turns to resolving the error term of the unit.
Next we first show how to get the error term at the output layer.
Then we illustrate that, with error terms available at layer k,
it is possible to go backwards to derive the values of error terms
for the units at the preceding layer k-1.

27

Error Term at the Output Layer



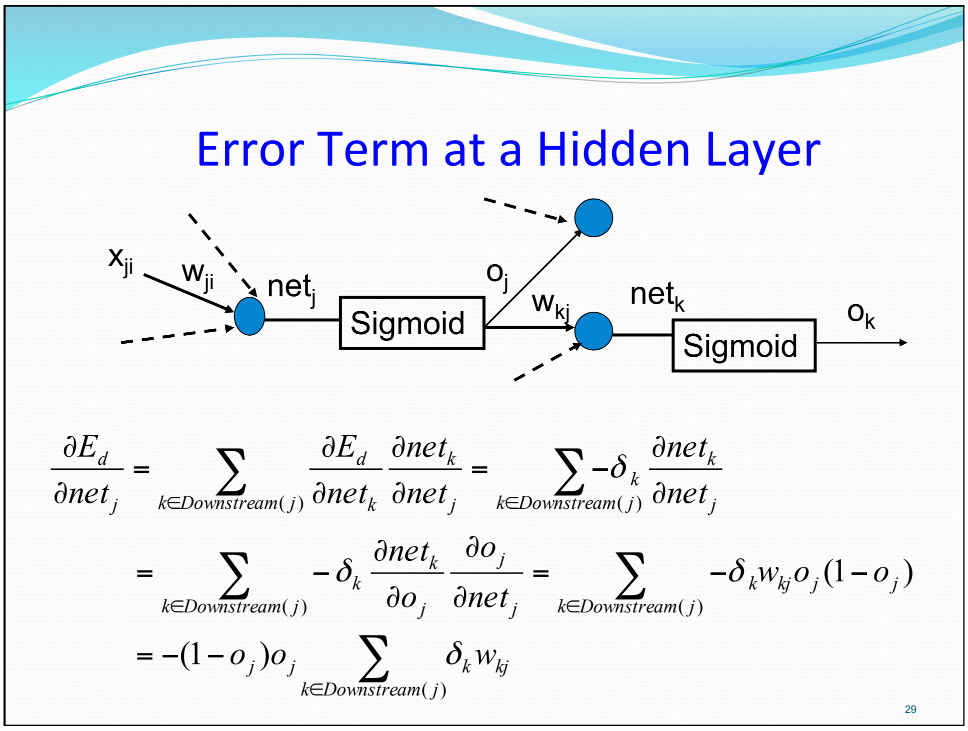
$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j} = o_j(1-o_j)\frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2$$

$$= o_j(1-o_j)\frac{1}{2}\bullet 2(t_j - o_j)\frac{\partial(t_j - o_j)}{\partial o_j} = -(t_j - o_j)o_j(1-o_j)$$

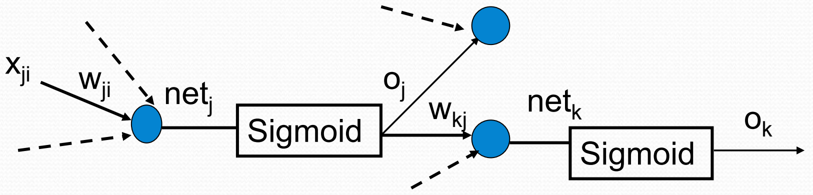
$$\delta_j = (t_j - o_j)o_j(1-o_j)$$
$$= e_j o_j(1-o_j)$$

for output units

28



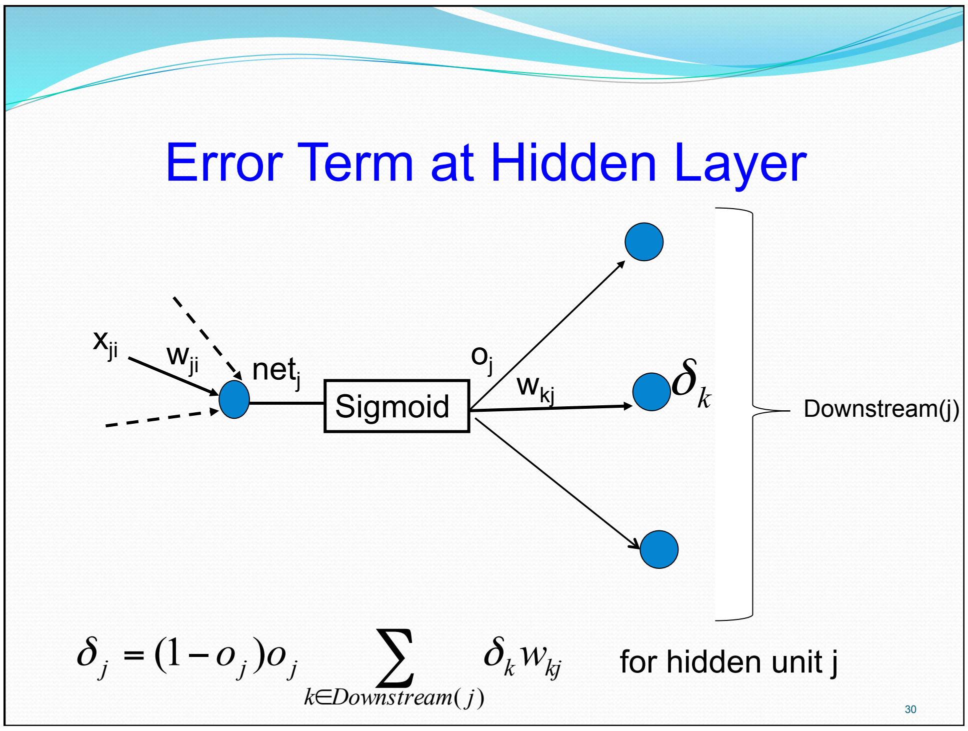
Error Term at a Hidden Layer



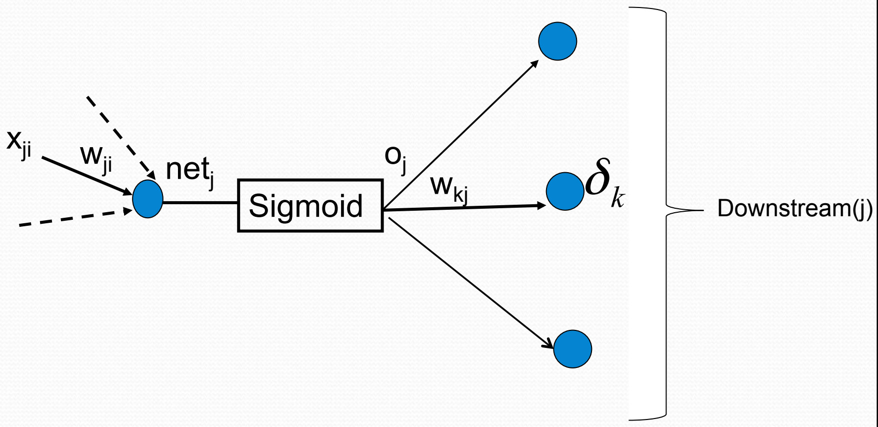
$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k}\frac{\partial net_k}{\partial net_j} = \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j}\frac{\partial o_j}{\partial net_j} = \sum_{k \in Downstream(j)} -\delta_k w_{kj} o_j (1-o_j)$$

$$= -(1-o_j)o_j \sum_{k \in Downstream(j)} \delta_k w_{kj}$$

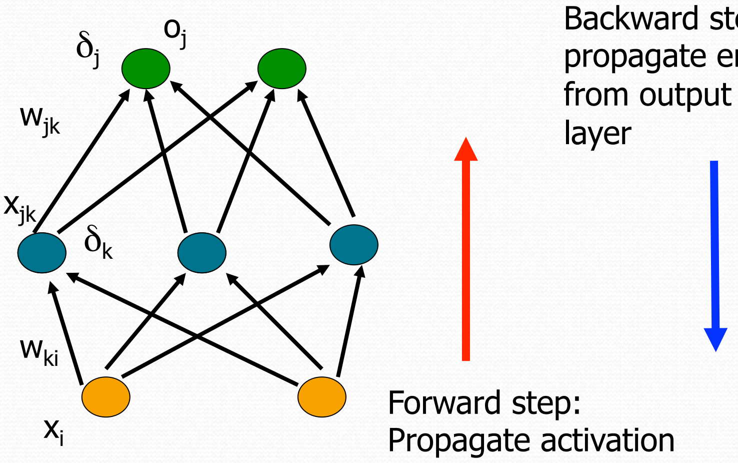


Error Term at Hidden Layer



$$\delta_j = (1-o_j)o_j \sum_{k \in Downstream(j)} \delta_k w_{kj}$$ for hidden unit j

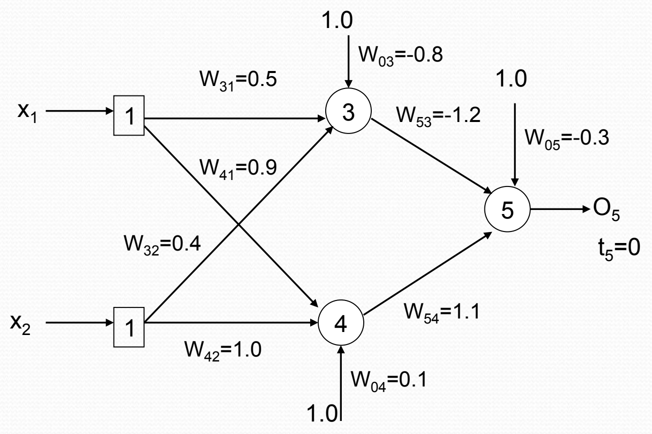
Backpropagation



Backward step: propagate error terms from output to hidden layer

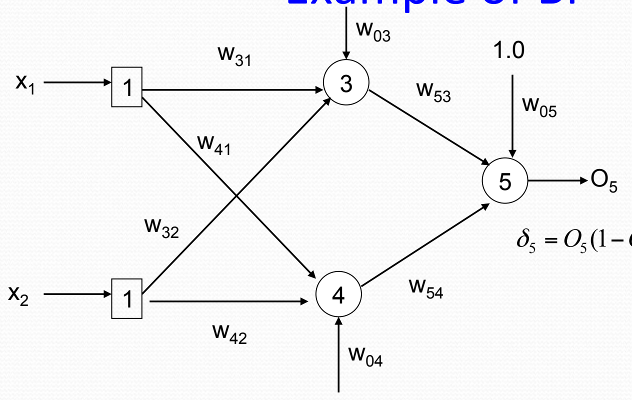
Forward step: Propagate activation from input to output layer

Example of BP



O_3=sigmoid($x_1w_{31}+x_2w_{32}+w_{03}$)=0.5250
O_4=sigmoid($x_1w_{41}+x_2w_{42}+w_{04}$)=0.8808
O_5=sigmoid($O_3w_{53}+O_4w_{54}+w_{05}$)=0.5097

Example of BP



$$\delta_5 = O_5(1-O_5)(t_5-O_5) = -0.1274$$

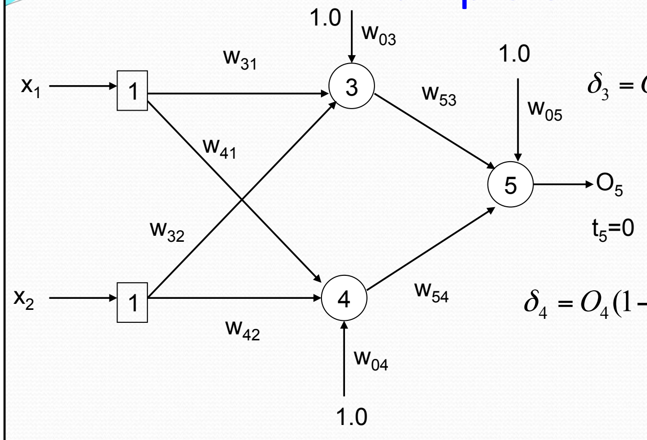
$$\Delta w_{53} = \eta \cdot \delta_5 \cdot O_3 = 0.1 \cdot (-0.1274) \cdot 0.5250 = -0.0067$$

$$\Delta w_{54} = \eta \cdot \delta_5 \cdot O_4 = 0.1 \cdot (-0.1274) \cdot 0.8808 = -0.0112$$

$$\Delta w_{05} = \eta \cdot \delta_5 \cdot 1 = 0.1 \cdot (-0.1274) \cdot 1 = 0.0127$$

33

Example of BP



t_5=0

$$\delta_3 = O_3(1-O_3)w_{53}\delta_5 = 0.0381$$

$$\delta_4 = O_4(1-O_4)w_{54}\delta_5 = -0.0147$$

$$\Delta w_{31} = \eta \cdot \delta_3 \cdot x_1 = 0.0038$$

$$\Delta w_{32} = \eta \cdot \delta_3 \cdot x_2 = 0.0038$$

$$\Delta w_{03} = \eta \cdot \delta_3 \cdot 1 = 0.0038$$

$$\Delta w_{41} = \eta \cdot \delta_4 \cdot x_1 = -0.0015$$

$$\Delta w_{42} = \eta \cdot \delta_4 \cdot x_2 = -0.0015$$

$$\Delta w_{04} = \eta \cdot \delta_4 \cdot 1 = -0.0015$$

34

Backpropagation Algorithm

- Initialize weights w_{ij} with a small random values
- Repeat
  For each training example $\{(x_1,\ldots x_n)^p,(t_1,\ldots,t_m)^p\}$ do

    1. Present $(x_1,\ldots,x_n)^p$ to the network and compute the outputs (forward step)
    2. Compute the error terms in the output layer
    3. Compute the error terms in the hidden layer (backwards)
    4. Update the weights for all units using error terms

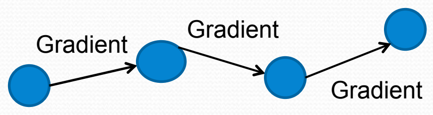
  until overall error becomes acceptably low

Problem with Local Minima

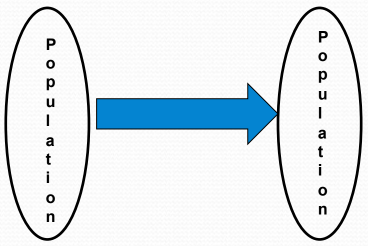
- Backpropagation is gradient descent search
  - local search mechanism
  - very high dimensional space
- Therefore backpropagation can get stuck into local minima
- One partial solution:
  Be invoked with multiple times with different initial weights

Population-based search

- Gradient descent: trajectory-based search relying on local information

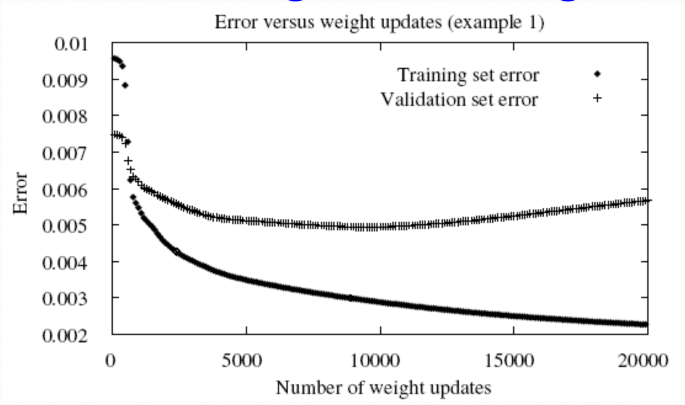


- Population based search: exploiting more global information



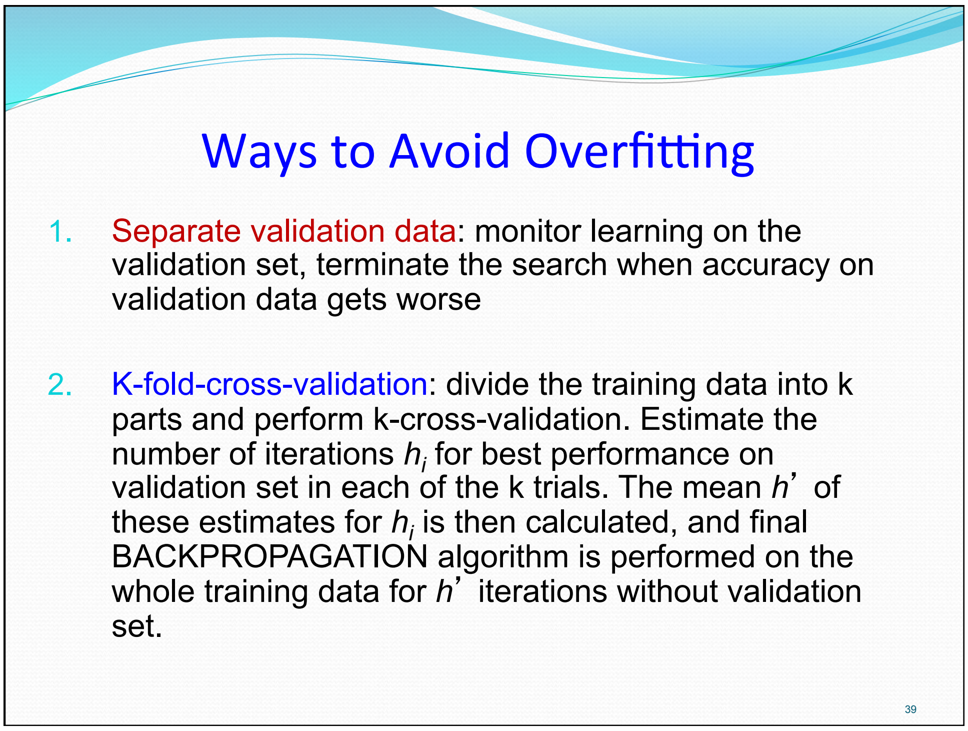
37

Overfitting the Training Data



- Learning **coincidence** in training data, not representative

38

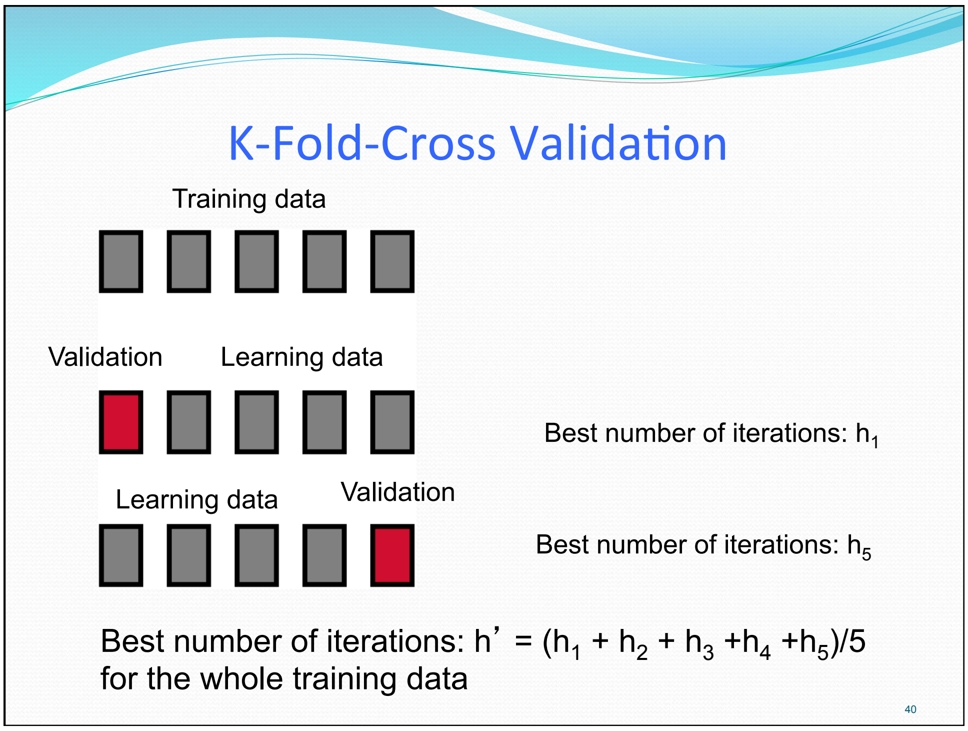


Ways to Avoid Overfitting

1. Separate validation data: monitor learning on the validation set, terminate the search when accuracy on validation data gets worse

2. K-fold-cross-validation: divide the training data into k parts and perform k-cross-validation. Estimate the number of iterations h_i for best performance on validation set in each of the k trials. The mean h' of these estimates for h_i is then calculated, and final BACKPROPAGATION algorithm is performed on the whole training data for h' iterations without validation set.

39



K-Fold-Cross Validation

Training data

Validation Learning data

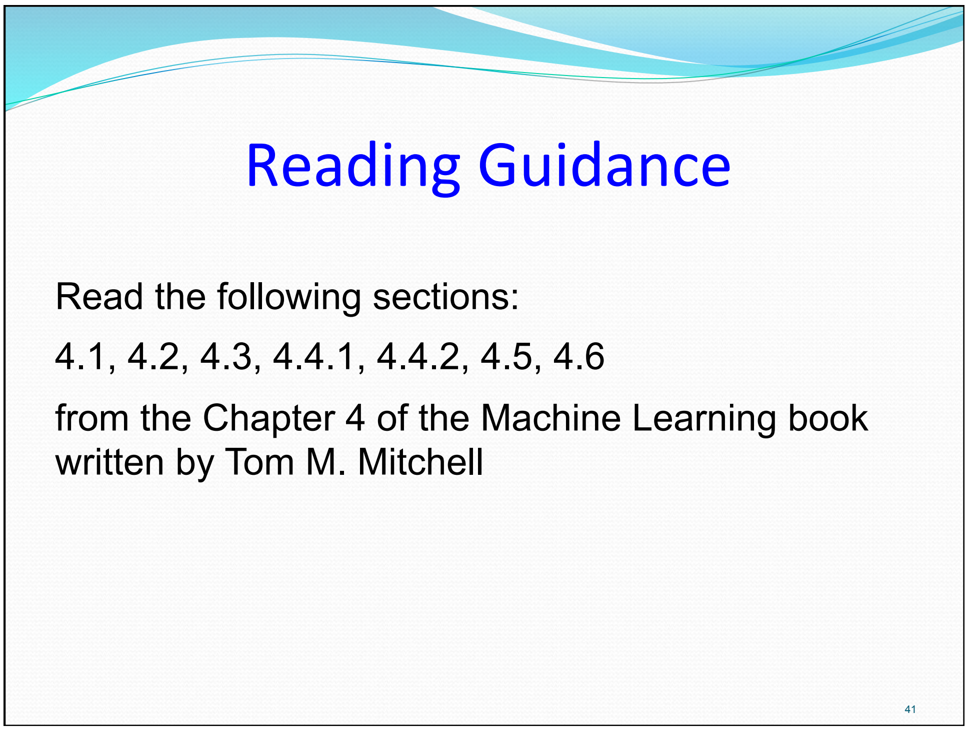
Best number of iterations: h_1

Learning data Validation

Best number of iterations: h_5

Best number of iterations: $h' = (h_1 + h_2 + h_3 + h_4 + h_5)/5$ for the whole training data

40



Reading Guidance

Read the following sections:

4.1, 4.2, 4.3, 4.4.1, 4.4.2, 4.5, 4.6

from the Chapter 4 of the Machine Learning book written by Tom M. Mitchell

41