Outline
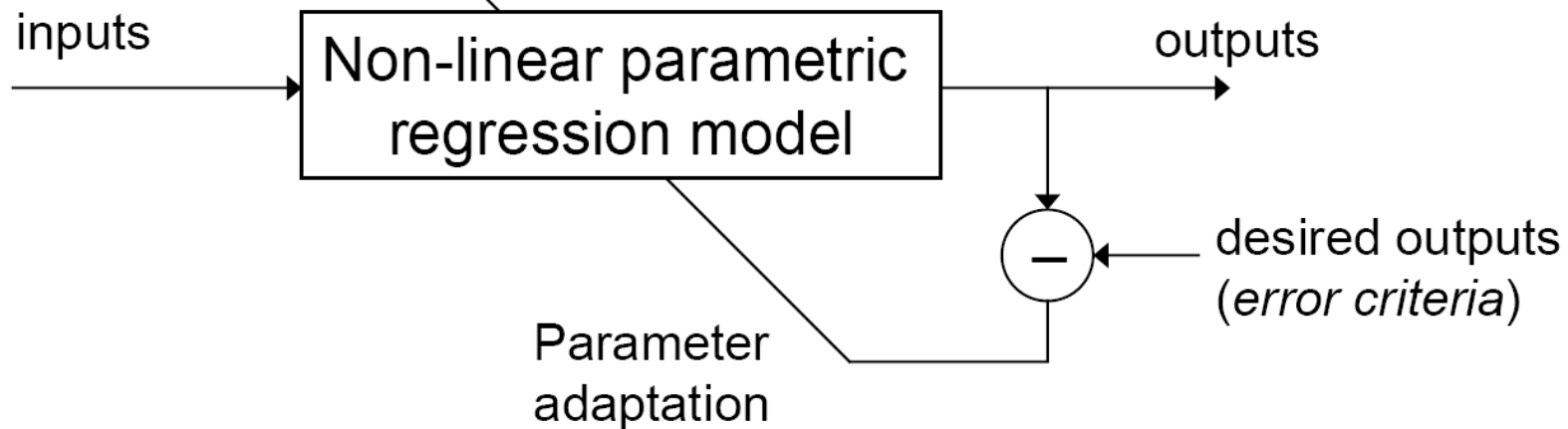
- First Example for Learning: Classification by a Perceptron

- Second Example: Regression by a Perceptron

- Perceptron Convergence Theorem

- Definition: ANN learning

- 1st Learning rule

- … and four more Learning rules

Learning in Principle

- *Fix a Model => restrictions about the possible relation*
- *Be parametric => fix the learning parameters to adjust (i.e. they "contain" the information)*
- *Choose non-linear => more powerful == expressive than linear ones*
- *NN's are a Universal approximations*

Hochschule   Fachbereich   Prof. Dr.
Bonn-Rhein-Sieg   Informatik   Paul G. Plöger

Michel Verleysen

Hands-on Example I: Perceptron learning

(1) Task : classification

(2) Network : perceptron (stepfunction)

(3) Learning: error correction

(1) Task: Classifiy

- The perceptron should classify correctly a set of examples into one of the two classes C1, C2.
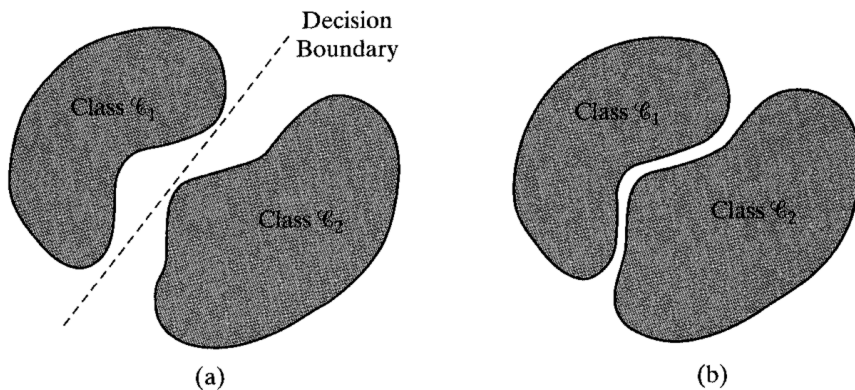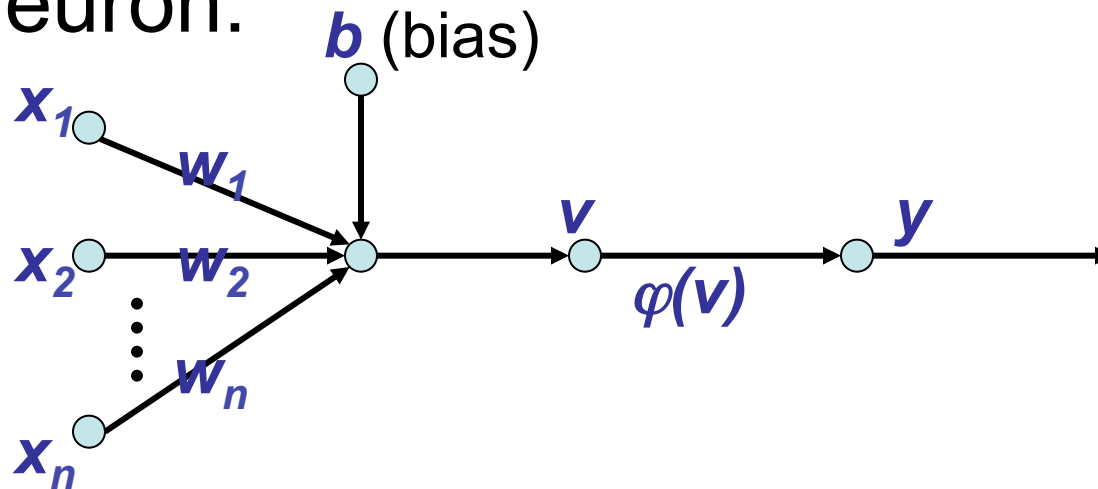

FIGURE 1.4 (a) A pair of linearly separable patterns. (b) A pair of non-linearly separable patterns.

If the output of the perceptron is +1 then the input is assigned to class C_1

If the output is 0 then the input is assigned to C_2

(2) Network: a single Perceptron
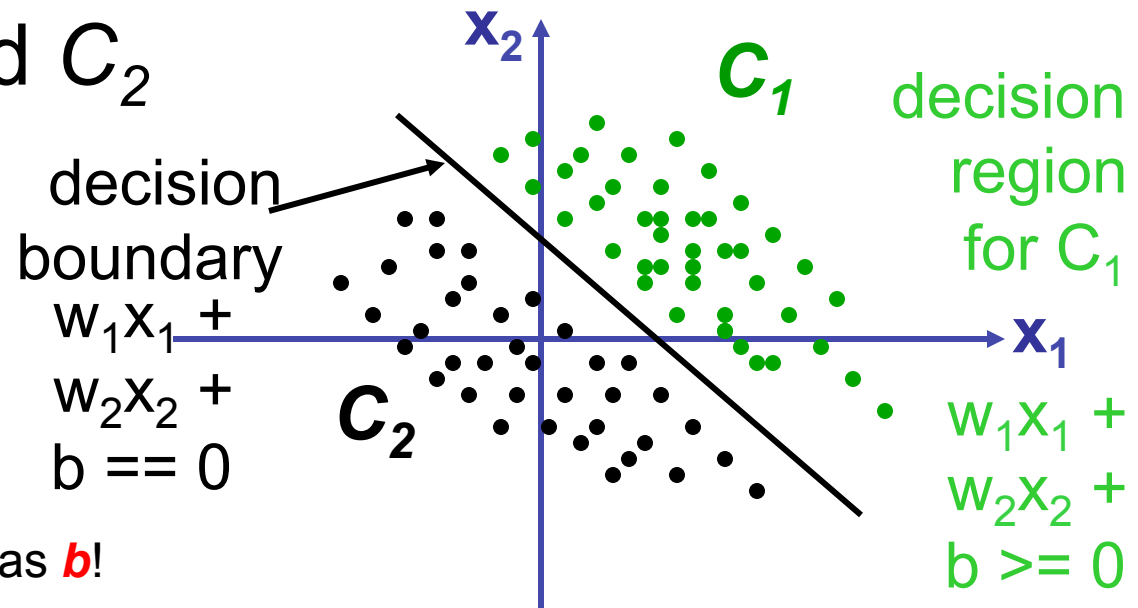
- Use a non-linear (McCulloch-Pitts) model of neuron:

b (bias)

x_1
w_1
x_2
w_2
$\vdots$
w_n
x_n

v
$\varphi(v)$
y

- φ is the step function :

$$\varphi(v) := \begin{cases} +1 & \text{if } v >= 0 \\ 0 & \text{if } v < 0 \end{cases}$$

Hochschule Bonn-Rhein-Sieg   **Fachbereich Informatik**   **Prof. Dr. Paul G. Plöger**

5

(2)Network: Output of Perceptron

- The equation below describes a hyperplane in the input space. This hyperplane is used to separate the two classes C_1 and C_2

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

$\mathbf{x_2}$

C_1

decision region for C_1

decision boundary
$w_1 x_1 +$
$w_2 x_2 +$
$b == 0$

$\mathbf{x_1}$

C_2

$w_1 x_1 +$
$w_2 x_2 +$
$b >= 0$

Wanted are: vector **w** and the bias **b**!

(3) Learning: Perceptron error

Test problem:

Let the set of training examples be

x_1 = [1 ; 2]; d_1 = 1

x_2 = [-1; 2]; d_2 = 0
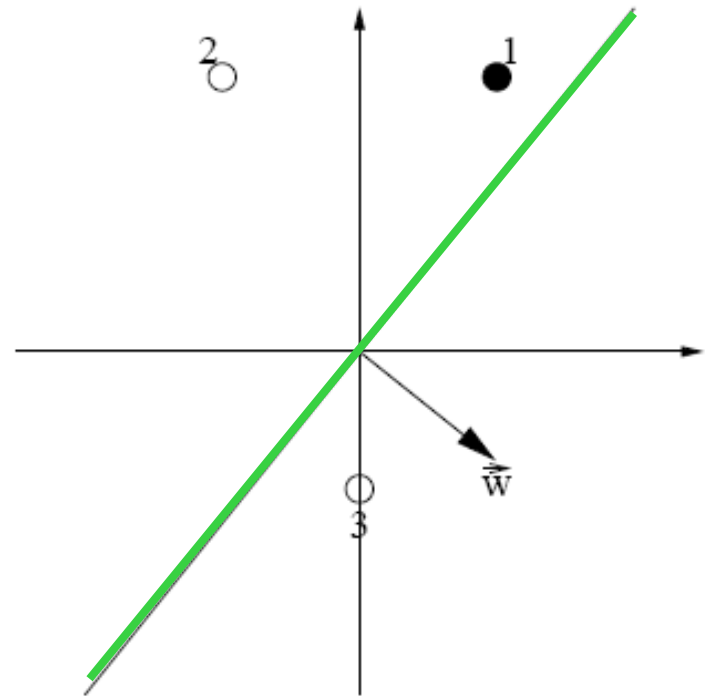
x_3 = [0 ;-1]; d_3 = 0

"d"== desired

error = d - $y_{current}$

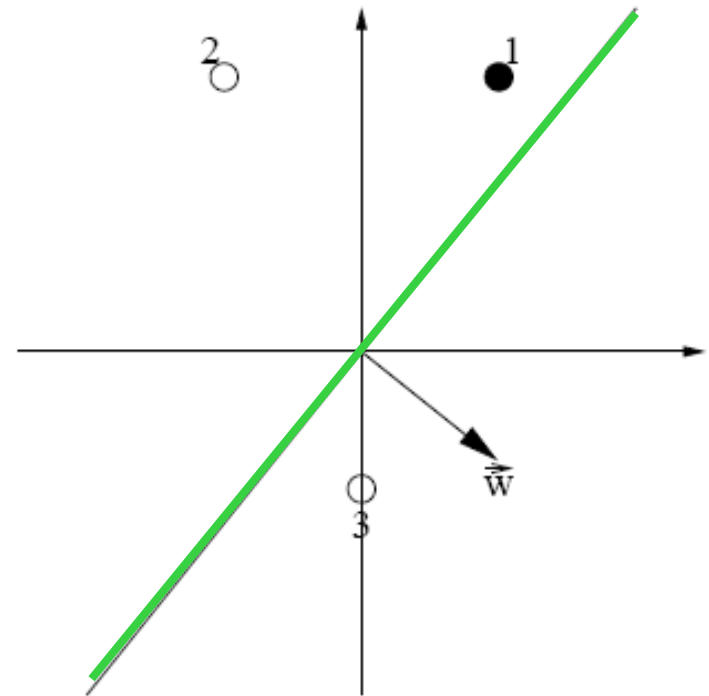Let bias be b = 0,learning rate η=1.

Let initial weight vector be
w = [1; -0.8]

We want to obtain a learning algorithm that finds a weight vector w which will correctly classify (separate) the given examples.
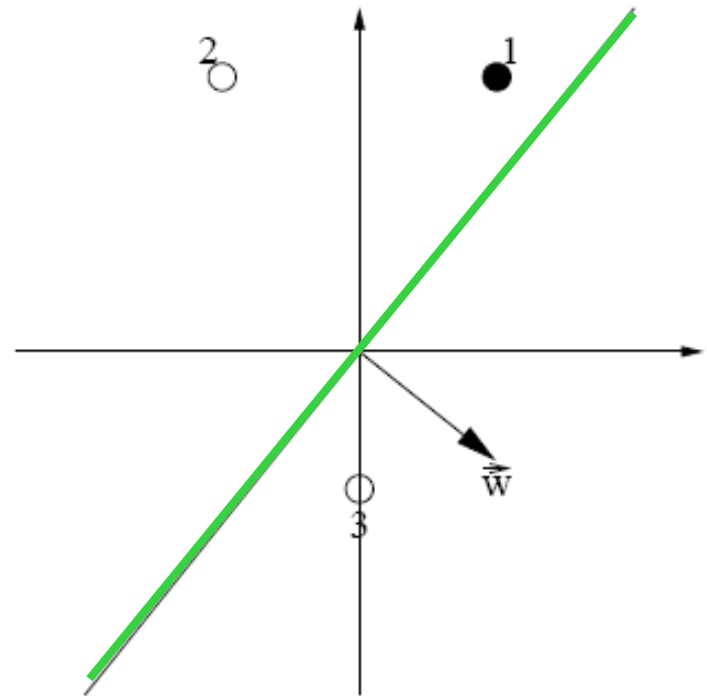
(3) Learning: Perceptron error …

- First input x_1 is misclassified with positive error. What to do?

(3) Learning: Perceptron error …

- First input x_1 is misclassified with positive error. What to do?

- Idea: move hyperplane towards separating position

Hochschule Bonn-Rhein-Sieg   **Fachbereich Informatik**   **Prof. Dr. Paul G. Plöger**

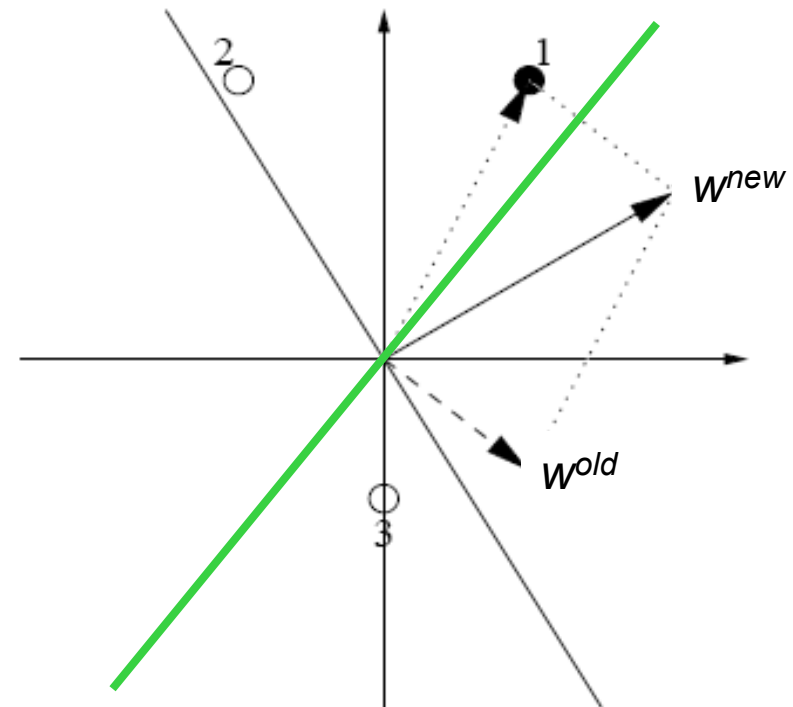**Dr. Alioune Ngom, University of Windsor, Ontario, Ca**

(3) Learning: Perceptron error …

- First input x_1 is misclassified with positive error. What to do?
- Idea: move hyperplane towards separating position

(3) Learning: Perceptron error …

- First input x_1 is misclassified with positive error. What to do?

- Idea: move hyperplane towards separating position

- So:

- Move w closer to x_1:
  add x_1 to w
  $w = w + x_1$

Hochschule Bonn-Rhein-Sieg   **Fachbereich Informatik**   **Prof. Dr. Paul G. Plöger**   11

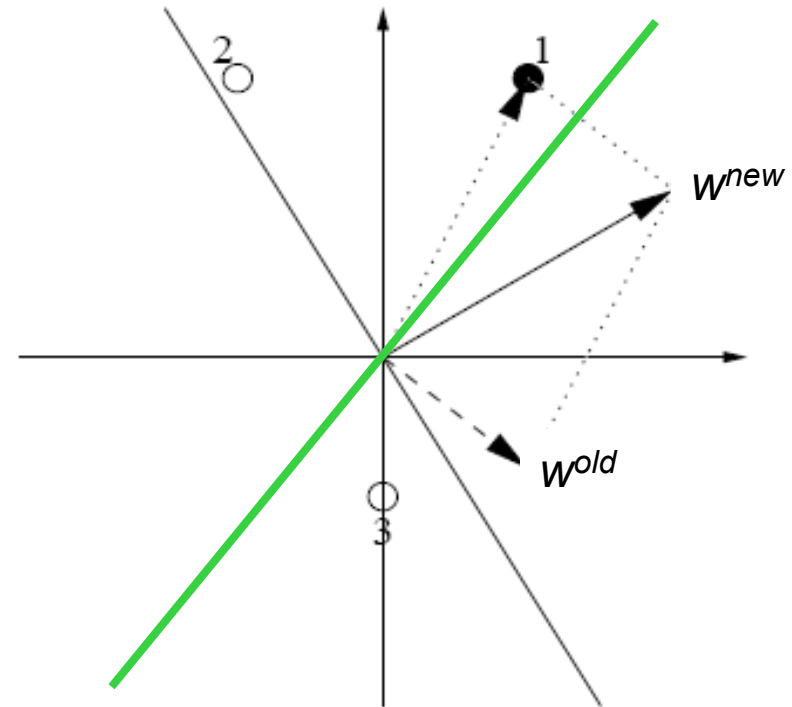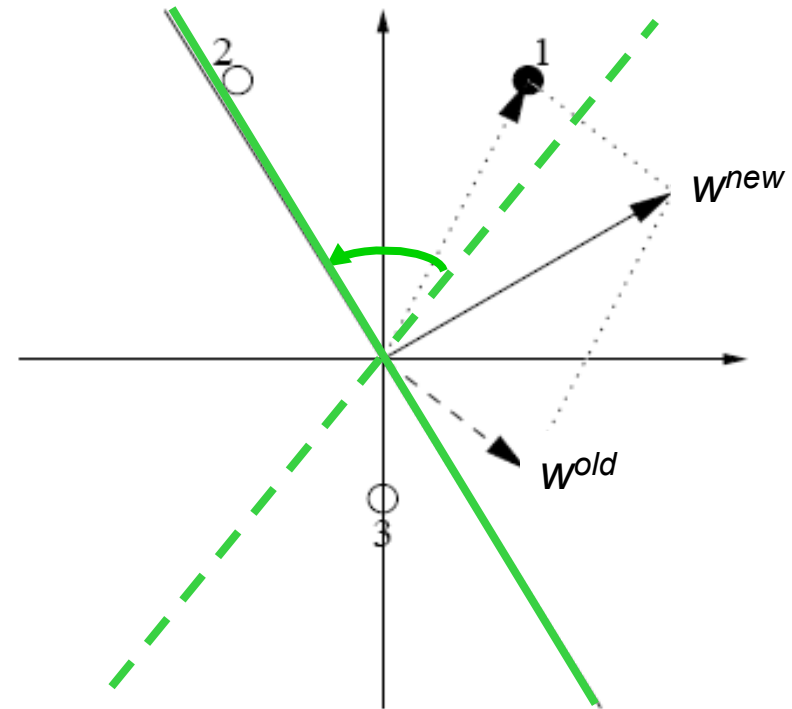**Dr. Alioune Ngom**, **University of Windsor, Ontario, Ca**

(3) Learning: Perceptron error ...

- First input x_1 is misclassified with positive error. What to do?

- Idea: move hyperplane towards separating position

- So:

- Move w closer to x_1:
  add x_1 to w
  $w = w + x_1$

- First rule:
  positive error rule (err=(d-y)>0)

  If $d = 1$ and $y = 0$ then
  $$w^{new} = w^{old} + x$$

(3) Learning: Perceptron error …

- Now sec. input x_2 is misclassified with negative error. What to do?
- Idea: move hyperplane away from separating position
- So:

- Move w away from x_2: subtract x_2 to w
  $$w = w - x_2$$

Hochschule Bonn-Rhein-Sieg **Fachbereich Informatik** **Prof. Dr. Paul G. Plöger**

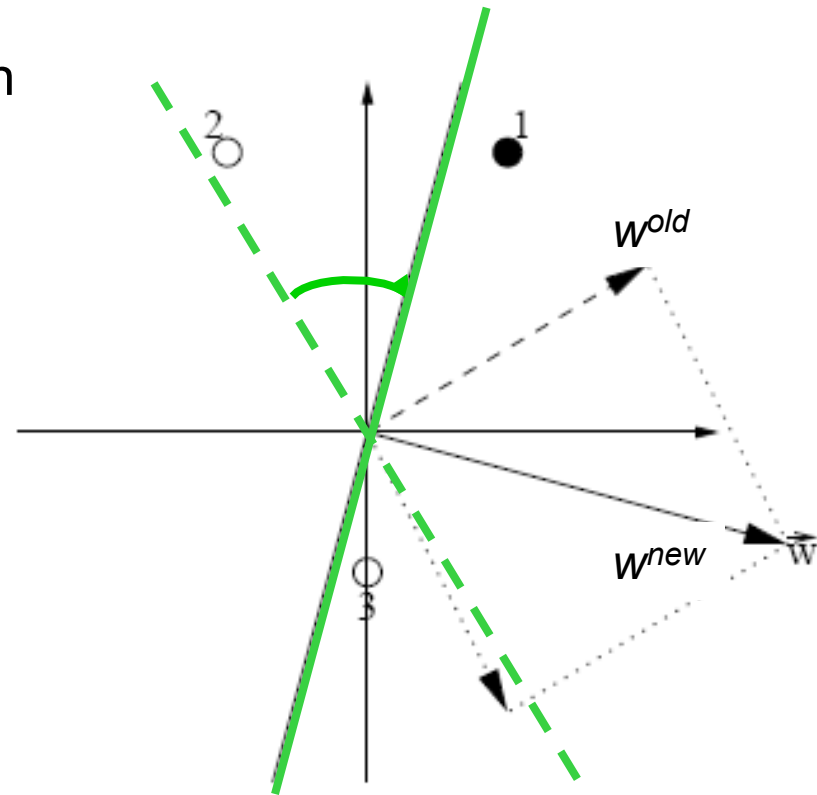**Dr. Alioune Ngom, University of Windsor, Ontario, Ca**

(3) Learning: Perceptron error …

- Now sec. input x_2 is misclassified with negative error. What to do?

- Idea: move hyperplane away from separating position

- So:
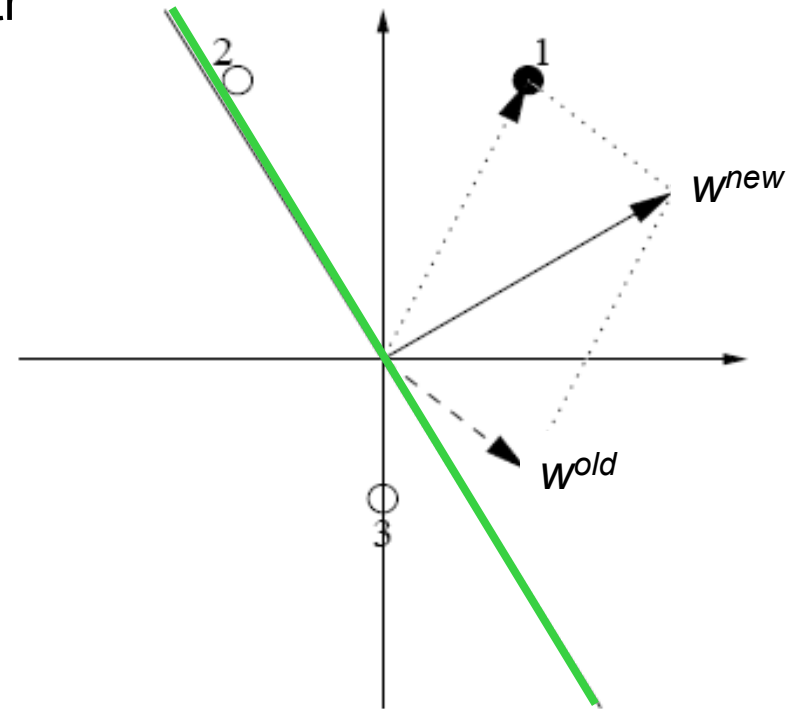
- Move w away from x_2:
  subtract x_2 to w
  $w = w - x_2$

- Second rule:
 *negative error rule (d-y)<0:*

  If $d = 0$ and $y = 1$ then
  $$w^{new} = w^{old} - x$$

Hochschule
Bonn-Rhein-Sieg

Fachbereich
Informatik

Prof. Dr.
Paul G. Plöger

14

Dr. Alioune Ngom, University of Windsor, Ontario, Ca

(3) Learning: Perceptron error …

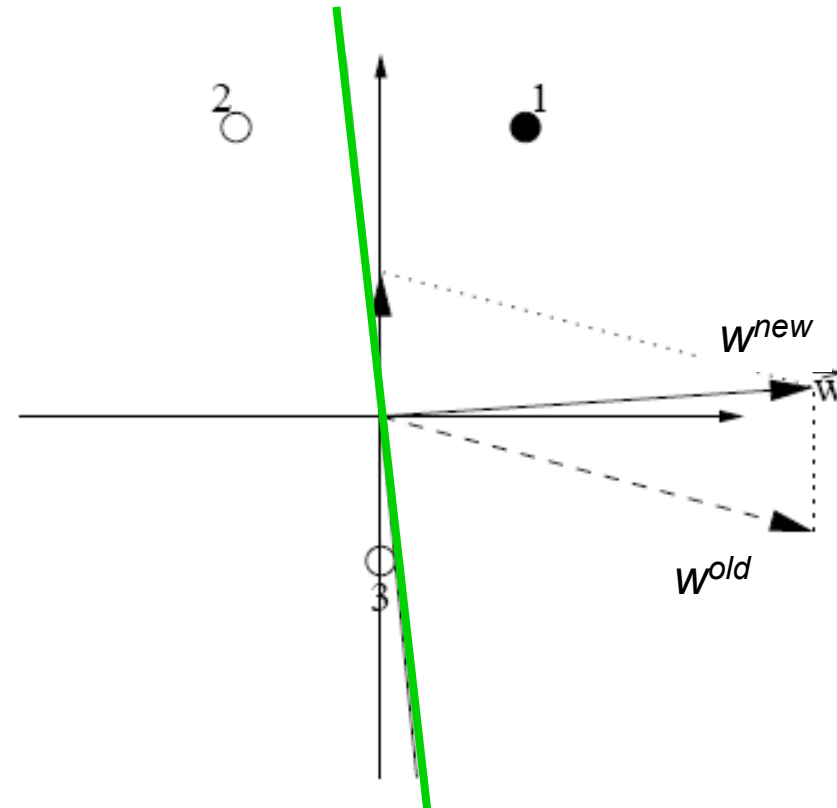- Third input x_3 is misclassified with negative error

- So

- Move w away from x_3:
  by subtracting x_3 from w

  $$w = w - x_3$$

- Second rule:
  negative error rule: *(d-y)<0*

  If d = 0 and y = 1 then
  $w^{new} = w^{old} - x$
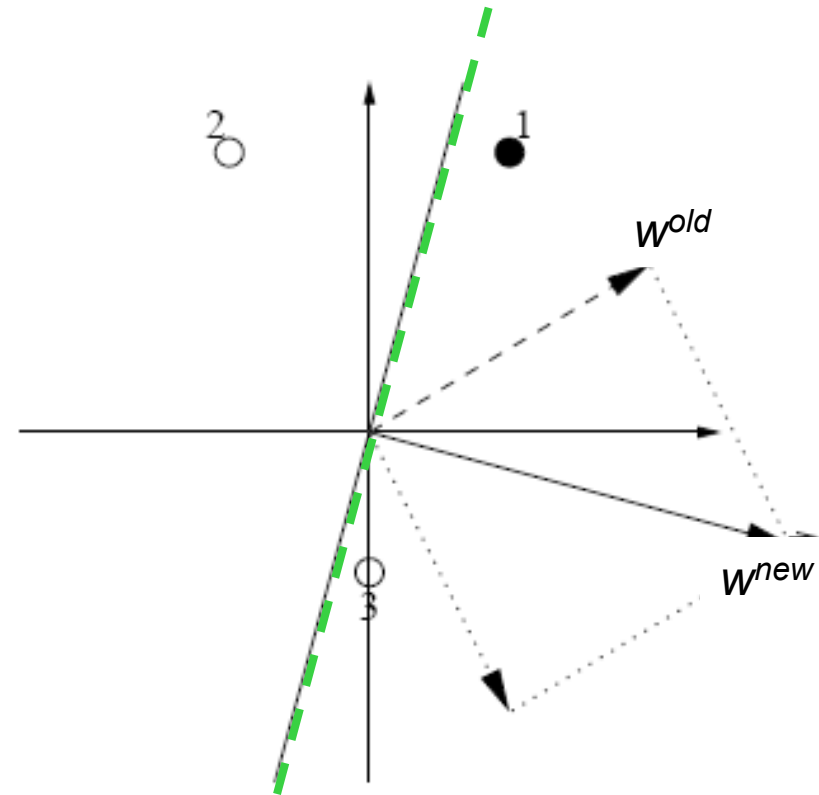
(3) Learning: Perceptron error …

- No data misclassified

w^{new}

$\vec{w}$

w^{old}

- The perceptron will now correctly classify all inputs x_1, x_2, x_3 if presented to it again. There will be no errors

- Third rule: no error rule

- If d = y then
  $$w^{new} = w^{old} .$$

Hochschule Bonn-Rhein-Sieg **Fachbereich Informatik** **Prof. Dr. Paul G. Plöger** 16

Dr. Alioune Ngom, University of Windsor, Ontario, Ca

(3) Learning: Perceptron error …

- Unified learning rule :
  $w^{new} = w^{old} + (d - y) x$

- Using a learning rate η :
  $w^{new} = w^{old} + \eta (d - y) x$

- Choice of learning rate η

  too large => learning oscillates
  too small => very slow learning
- $0 < \eta <= 1$, popular choices:
  $\eta = 0.5$
  $\eta = 1$
- Variable learning rate
  $\eta = |w.*x| / |x.*x|$
- Adaptive learning rate

- Perceptron convergence theorem:
  [M. Minsky and S. Papert, 1969]

- The perceptron learning algorithm terminates if and only if the task is linearly separable

- Cannot learn non-linearly separable functions

Hochschule Bonn-Rhein-Sieg **Fachbereich Informatik** **Prof. Dr. Paul G. Plöger** 17

Dr. Alioune Ngom, University of Windsor, Ontario, Ca

Hands-on Example II: Adaline learning

- Task : regression

- Network : **ADA**ptive **LIN**ear **E**lement

  (linear output activation function, "perceptron" with linear output element) (ADALINE)

- Learning: error correction
 **derived (!)** from steepest gradient (=> LMS)

Adaline: Learning Principle

- Data fitting
  (or linear regression)
  - Given a set of data
    (e.g. from measurements)
    $\{\ [x_i, d_i]\ \}$
  - Find **w** and **b** such that
    $d_i \sim\sim wx_i + b$
  - or more precisely
  - $d_i = wx_i + b + \varepsilon_i = y_i + \varepsilon_i$
    where

    ε_i = instantaneous error
    y_i = linearly fitted value
    w = line slope
    b = d-axis intercept (or bias)
- Best fit problem:
  find the best choice of
  (**w,b**) such that the fitted line
  passes closest to all points
- Solution: Least mean square

**Hochschule
Bonn-Rhein-Sieg**

**Fachbereich
Informatik**

**Prof. Dr.
Paul G. Pl**

19
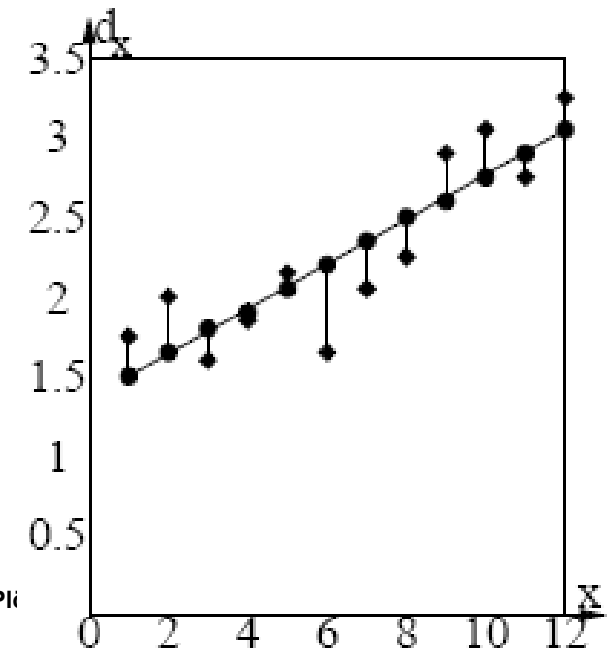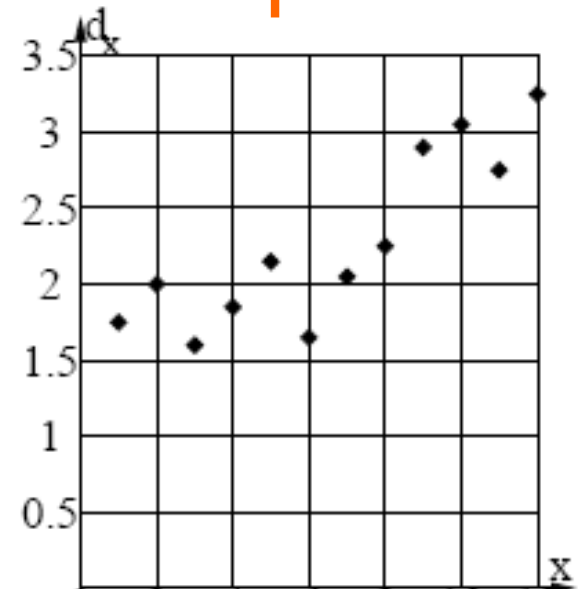
Adaline: Architecture

- Adaline: uses a linear neuron model and the Least-Mean-Square (LMS) learning algorithm

- The idea: try to minimize the mean square error, which is a function of the weights

- We may determine the minimum of the error function **E** by means of the steepest descent method

- Architecture:
  - Input:
  $$\vec{x} = (x_0 = 1, x_1, \ldots, x_n)$$
  - Weights (with bias θ):
  $$\vec{w} = (w_0 = -\theta, w_1, \ldots, w_n)$$
  - Net input:
  $$y = \vec{x}\vec{w} = \sum_{i=0}^{n} x_i w_i$$
  - Output:
  $$f(\vec{x}) = \vec{x}\vec{w}$$

- Learning Task: regression

- Learning Type: Error-correction learning (Supervised learning)
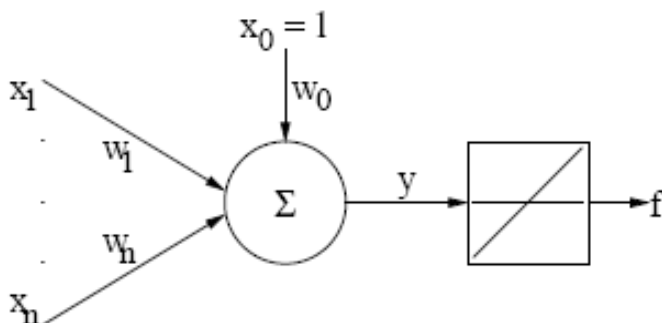
Idea ADALINE learning

- Uses a linear output neuron model

- The idea: look at the current squared error

- viewed as a function of the weights

- Minimize now by building derivative (possible!)

- Search miminimum of the error function **E(w(n))** by means of the steepest descent

- Least-Mean-Square (LMS) based learning algorithm

- $y= \varphi(x) = w*x + b$

$$E(\mathrm{w}(n)) = \tfrac{1}{2}\mathrm{e}^2(n)$$

$$e(n) = d(n) - \sum_{j=0}^{m} x_j(n)w_j(n)$$
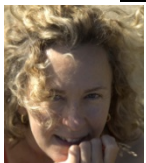
Prof. Dr.
Paul G. Plöger

21

Steepest Descent Method

- start with an arbitrary point

- find a direction in which E is decreasing most rapidly

$$- (\text{gradient of } E(\text{w})) = - \left[\frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_m} \right]$$

- make a small step in that direction

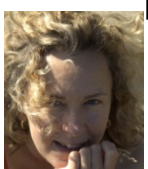$$w(n + 1) = w(n) - \eta(\text{gradient of } E(n))$$

*L*east-*M*ean-*S*quare algorithm (Widrow-Hoff algorithm)

- Approximation of gradient(E)

$$\frac{\partial E(\mathrm{w}(n))}{\partial \mathrm{w}(n)} = e(n)\frac{\partial e(n)}{\partial \mathrm{w}(n)}$$

$$= e(n)[-\mathrm{x}(n)^{\mathrm{T}}]$$

- Update rule for the weights becomes:

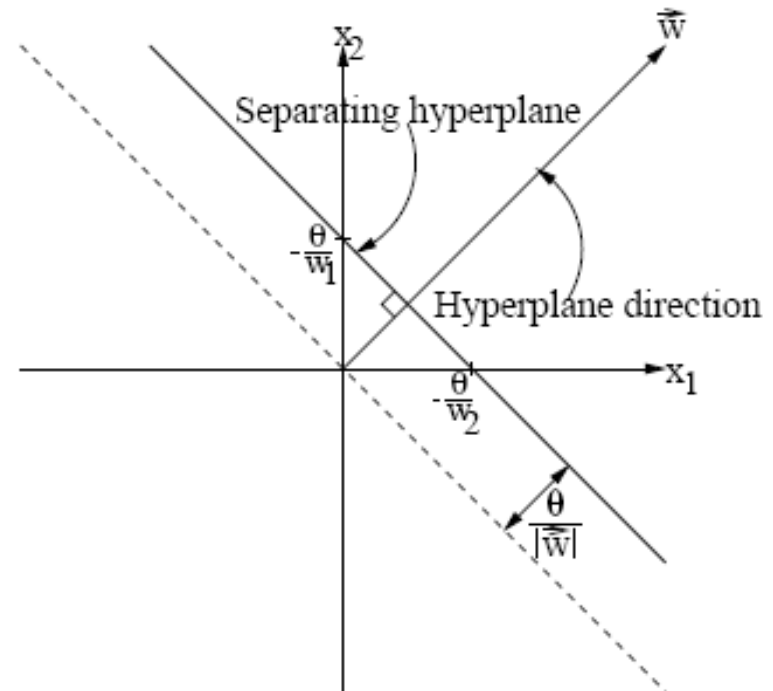$$\mathrm{w}(n+1) = \mathrm{w}(n) + \eta\,\mathrm{x}(n)e(n)$$

Decision boundary

- Adaline's decision boundary

  $$w_0 x_0 + w_1 x_1 + \ldots + w_n x_n = 0$$

- The Adaline:
  - has a decision boundary like the perceptron
  - can also be used to classify objects into two categories
  - has same limitation as the perceptron
  - **But**
    the learning can be derived from analysis (gradient descent)

ADALine: Learning Principle

- Minimize sum of squared errors (SSE) or the mean of squared errors (MSE)

$$\varepsilon_i = d_i - \widetilde{d}_i$$

$$where$$

$$\widetilde{d}_i = \vec{w}\vec{x}_i + b$$

$$MSE : E = \frac{1}{N}\sum_{i=1}^{N}\varepsilon_i^2$$

- The minimum MSE is called the least mean square (LMS), which can be obtained analytically by building:

$$\frac{\partial E}{\partial \vec{w}} = 0$$

$$\frac{\partial E}{\partial b} = 0$$

- then solve for **w** and **b**.
- LMS is difficult to obtain for larger dimensions (complex formula) and larger data sets
- ADALine:
  - learns by minimizing the MSE
  - not sensitive to noise
  - powerful and robust learning

ADALine: Learning Algorithm by Gradient descent

given a learning example $: [x_i, d_i]$
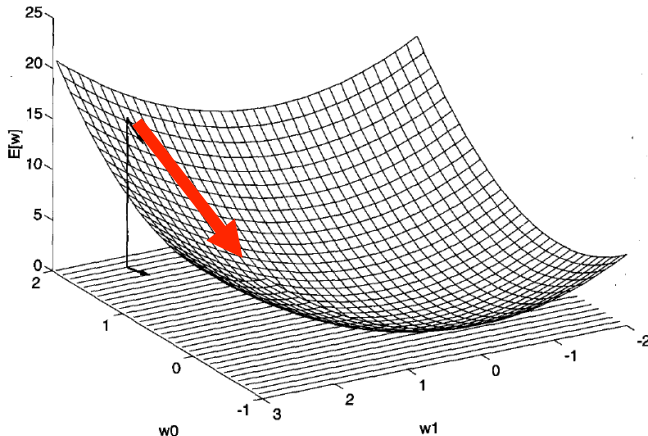
an observed output $: y_i = \ldots$

squared error $: \varepsilon_i = \left(d_i - y_i\right)^2$

gradient of $\varepsilon_i :$

$$\nabla \varepsilon_i = \frac{\partial \varepsilon_i}{\partial \vec{w}} =$$

$$\left(\frac{\partial \varepsilon_i}{\partial w_0}, \frac{\partial \varepsilon_i}{\partial w_1}, \ldots, \frac{\partial \varepsilon_i}{\partial w_n}, \right)$$

if ε_i *minimal* $\Rightarrow \nabla \varepsilon_i = 0$

- Negative gradient of ε_i gives direction of steepest descent to the minimum $-\nabla \varepsilon_i$

- Then do gradient descent

$$\Delta \vec{w} = -\eta \nabla \varepsilon_i = -\eta \frac{\partial \varepsilon_i}{\partial \vec{w}}$$

Widrow – Hoff delta rule

$(for\ component\ k):$

$$\frac{\partial \varepsilon_i}{\partial w_k} = \frac{\partial (d_i - y_i)^2}{\partial w_k} = 2(d_i - y_i) \frac{\partial (-y_i)}{\partial w_k}$$

$$= 2(d_i - y_i) \frac{\partial(-\sum_{l=0}^{n} w_l x_l)}{\partial w_k} = -2(d_i - y_i)x_k$$

Final overall learning rule $:$

$$\vec{w}^{new} = \vec{w}^{old} + \eta(d_i - y_i)\vec{x}$$

26

Summary of Adaline: Learning algorithm (LMS)

Training sample:
  input signal vector   **x(t)**
  desired response   **d(t)**

User selected parameter:
  $\eta > 0$

Computation:
  for t = 1, 2, …
  compute
  $\delta(t) = d(t) - \hat{w}^T(t)x(t)$
  $\hat{w}(t+1) = \hat{w}(t) + \eta x(t)\delta(t)$

Initialization: $\vec{w}_0 = \vec{0}$;

$t = 0$;

Repeat

$\quad t = t + 1$;

$\quad$ For each training example $[\vec{x}, d_{\vec{x}}]$ do

$\quad\quad net_{\vec{x}} = \vec{w} \cdot \vec{x}$;

$\quad\quad a_{\vec{x}} = g(net_{\vec{x}}) = net_{\vec{x}}$;

$\quad\quad \delta_{\vec{x}} = d_{\vec{x}} - a_{\vec{x}}$;

$\quad\quad \vec{w}_{t+1} = \vec{w}_t + \eta \cdot \delta_{\vec{x}} \cdot \vec{x}$;

$\quad\quad \{$

$\quad\quad\quad$ or equivalently,

$\quad\quad\quad$ For $0 \leq i \leq n$

$\quad\quad\quad\quad w_{i,t+1} = w_{i,t} + \eta \cdot \delta_{\vec{x}} \cdot x_i$;

$\quad\quad \}$

Until $MSE(\vec{w})$ is minimal;

Save last weight vector;

Hochschule Bonn-Rhein-Sieg   **Fachbereich Informatik**

ADAline END

New Classification Example (now *with* bias)

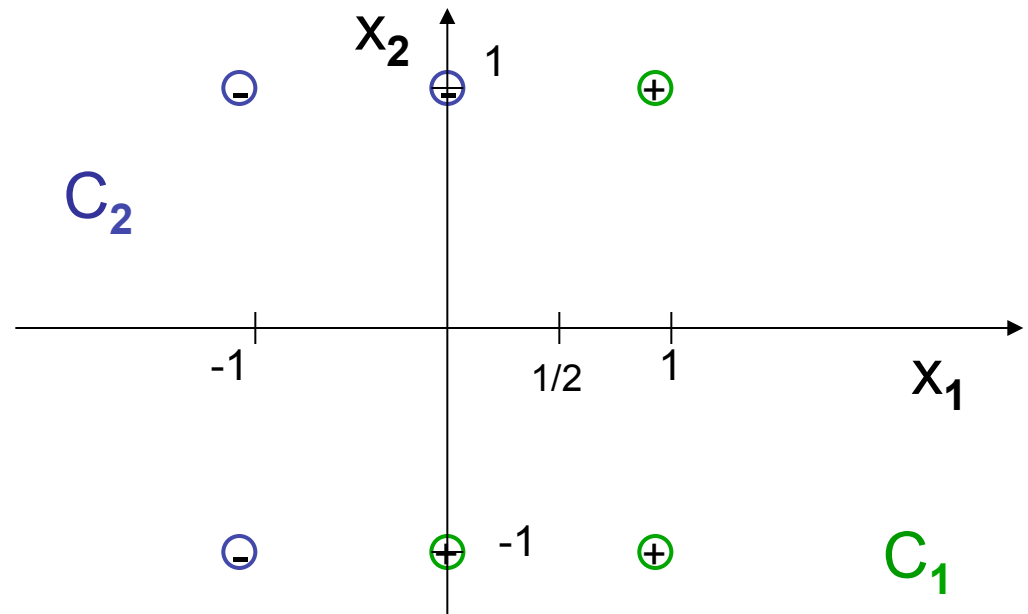Consider 2D training set
$C_1 \cup C_2$, where:

$C_1 = \{(1,1),\ (1,\ -1),\ (0,\ -1)\}$
    elements of class 1

$C_2 = \{(-1,-1),\ (-1,1),\ (0,1)\}$
    elements of class -1

- Use the perceptron learning algorithm to classify these examples.

$\mathbf{w}(1) = [1,\ 0,\ 0]^{\mathsf{T}}$
$\eta = 1$

Hochschule Bonn-Rhein-Sieg   **Fachbereich Informatik**   **Prof. Dr. Paul G. Plöger**
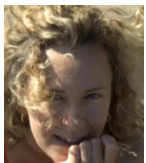
Trick for bias and learning rule

Consider the *augmented* training set C'$_1$ ∪ C'$_2$, with first entry fixed to 1 (to deal with the *bias as extra weight*):

C'$_1$ = { (1, 1, 1), (1, 1, -1), (1, 0, -1)}

C'$_2$ = { (1,-1, -1), (1,-1, 1), (1,0, 1) }

Replace tuples (a,b,c) from ∈ C$_2$' by (-a,-b,-c) and use the following simpler update rule (Note: NOT so obvious…):

$$
\mathbf{w}(n+1) =
\begin{cases}
\mathbf{w}(n) + \eta\, \mathbf{x}(n) & \text{if } \mathbf{w}^{T}(n)\, \mathbf{x}(n) \leq 0 \\[2mm]
\mathbf{w}(n) & \text{otherwise}
\end{cases}
$$

Cont…

- Training set after application of trick:
  (1, 1, 1), (1, 1, -1), (1,0, -1), (-1,1, 1), (-1,1, -1), (-1,0, -1)

- Application of perceptron learning algorithm:

Adjusted pattern	Weight applied	w(n) x(n)	Update?	New weight
(1, 1, 1)	(1, 0, 0)	1	No	(1, 0, 0)
(1, 1, -1)	(1, 0, 0)	1	No	(1, 0, 0)
(1,0, -1)	(1, 0, 0)	1	No	(1, 0, 0)
(-1,1, 1)	(1, 0, 0)	-1	Yes	(0, 1, 1)
(-1,1, -1)	(0, 1, 1)	0	Yes	(-1, 2, 0)
(-1,0, -1)	(-1, 2, 0)	1	No	(-1, 2, 0)

End epoch 1

Example

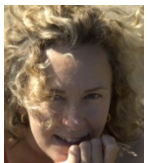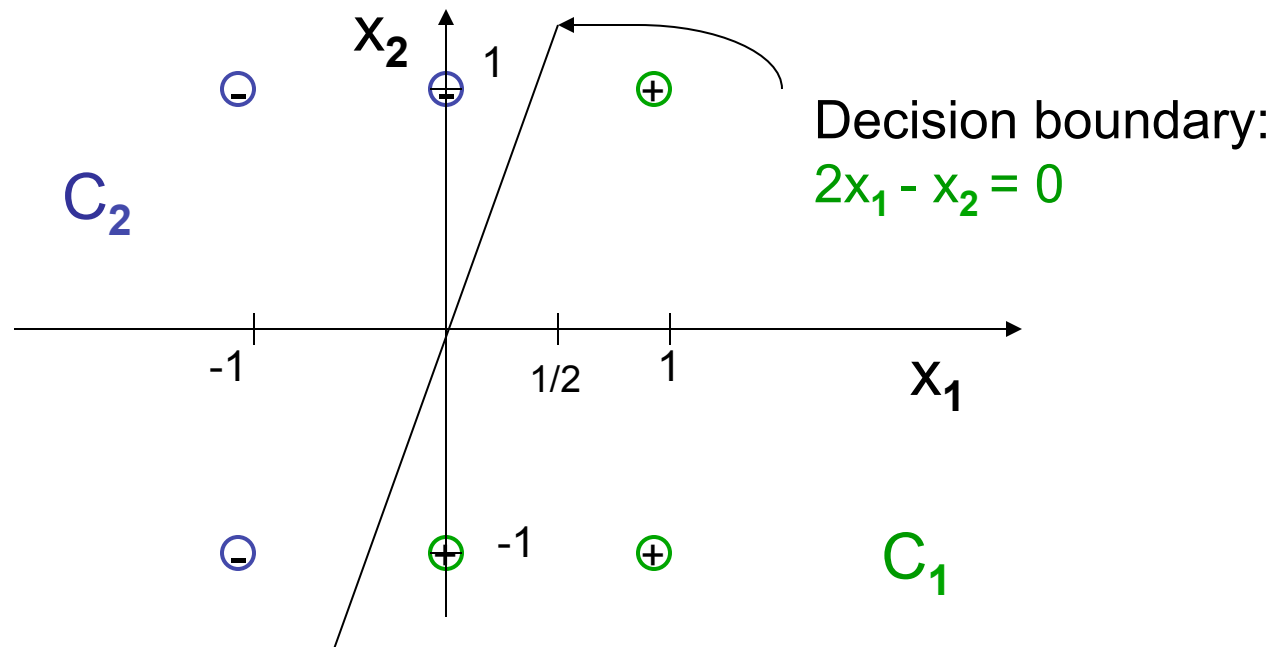| Adjusted pattern	Weight applied	$w^T(n)x(n)$	Update?	New weight
(1, 1, 1)	(-1, 2, 0)	1	No	(-1, 2, 0)
(1, 1, -1)	(-1, 2, 0)	1	No	(-1, 2, 0)
(1,0, -1)	(-1, 2, 0)	-1	Yes	(0, 2, -1)
(-1, 1, 1)	(0, 2, -1)	1	No	(0, 2, -1)
(-1, 1, -1)	(0, 2, -1)	3	No	(0, 2, -1)
(-1,0, -1)	(0,2, -1)	1	No	(0, 2, -1)

End epoch 2

At Epoch 3 no updates are performed. (check!)

⇒ stop execution of algorithm.

Final weight vector: (0, 2, -1).

⇒ decision hyperplane is 2x1 - x2 = 0.

Example visualized:

Decision boundary:
$2x_1 - x_2 = 0$

C_2

C_1

Perceptron: fixed-increment learning algorithm

- Variables and parameters at iteration n of the learning algorithm:

$\mathbf{x}\,(n) =$ input vector

$\qquad = [+1, x_1(n), x_2(n), \ldots, x_m(n)]^{\top}$

$\mathbf{w}(n) =$ weight vector

$\qquad = [b(n), w_1(n), w_2(n), \ldots, w_m(n)]^{\top}$

$b(n) =$ bias

$y(n) =$ actual response

$d(n) =$ desired response

$\eta =$ learning rate parameter

Hochschule Bonn-Rhein-Sieg   **Fachbereich Informatik**   **Prof. Dr. Paul G. Plöger**   33

Perceptron: fixed-increment learning algorithm (cont)

- Initialization: n=1, w(n) = 0 (or random small val.)

- Activation: activate perceptron by applying input example (input = x (n), desired output = d(n))

- Compute actual output of perceptron:

$$y(n) = step[w^T(n) \, x(n)]$$

- Adapt weight vector: if $d(n) \neq y(n)$ then

$$w(n + 1) = w(n) + \eta \, e(n) \, x(n)$$

$$\text{where } e(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \in C_1 \\ -1 & \text{if } \mathbf{x}(n) \in C_2 \end{cases}$$

- Continuation: increment time step *n* and go to Activation:

Perceptron Learning Convergence

Suppose datasets C_1, C_2 are linearly separable. The perceptron convergence algorithm converges after n_0 iterations, with $n_0 \leq n_{max}$ on training set $C_1 \cup C_2$.
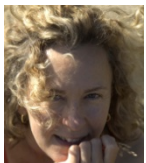
Proof:

- suppose $\mathbf{x} \in C_1 \Rightarrow$ output = 1 and $\mathbf{x} \in C_2 \Rightarrow$ output = -1.
- For simplicity assume $\mathbf{w}(1) = 0,\ \eta = 1$.
- Suppose perceptron incorrectly classifies $\mathbf{x}(1) \dots \mathbf{x}(k) \dots \in C_1$. Then $\mathbf{w^T}(k)\,\mathbf{x}(k) \leq 0$.

$\Rightarrow$ Error correction rule:

$\mathbf{w}(2) = \mathbf{w}(1) + \mathbf{x}(1)$
$\mathbf{w}(3) = \mathbf{w}(2) + \mathbf{x}(2)$
$\vdots$
$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{x}(k)$

$\Rightarrow \mathbf{w}(k+1) = \mathbf{x}(1) + \dots + \mathbf{x}(k)$

Proof : Convergence theorem (cont.)

- Let $\mathbf{w_0}$ be such that $\mathbf{w_0^T x}(n) > 0 \quad \forall \mathbf{x}(n) \in C_1$.
w_0 exists because C_1 and C_2 are linearly separable.

- Let $\alpha = \min w_0^T x(n) \quad x(n) \in C_1$.

- Then $\mathbf{w_0^T} w(k+1) = \mathbf{w_0^T x}(1) + \ldots + \mathbf{w_0^T x}(k) \geq k\alpha$

- Cauchy-Schwarz inequality:
$$\|\mathbf{w_0}\|^2 \ \|\mathbf{w}(k+1)\|^2 \ \geq \ [\mathbf{w_0^T w}(k+1)]^2$$

$$\|\mathbf{w}(k+1)\|^2 \ \geq \ \frac{k^2 \alpha^2}{\|w_0\|^2} \qquad \textbf{(A)}$$

Proof: Convergence theorem (cont.)

- Now we consider another route:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{x}(k)$$

$$\| \mathbf{w}(k+1)\|^2 = \| \mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2 + 2\,\mathbf{w}^{\mathsf{T}}(k)\mathbf{x}(k)$$
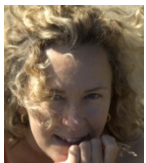
euclidean norm

≤ 0 because $\mathbf{x}(k)$ is misclassified

$$\Rightarrow \|\mathbf{w}(k+1)\|^2 \leq \|\mathbf{w}(k)\|^2 + \|\mathbf{x}(k)\|^2$$

$=0$

$$\|\mathbf{w}(2)\|^2 \leq \|\mathbf{w}(1)\|^2 + \|\mathbf{x}(1)\|^2$$

$$\|\mathbf{w}(3)\|^2 \leq \|\mathbf{w}(2)\|^2 + \|\mathbf{x}(2)\|^2$$

$$\vdots$$

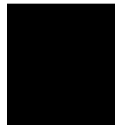$$\Rightarrow \|\mathbf{w}(k+1)\|^2 \leq \sum_{i=1}^{k} \| \mathrm{x}(i) \|^2$$

Proof: Convergence theorem (end)

- Let $\beta = \max \|\mathbf{x}(n)\|^2$ $\mathbf{x}(n) \in C_1$

- $\|\mathbf{w}(k+1)\|^2 \leq k \, \beta$ **(B)**

- For   sufficiently large values of k:
  (B) becomes in conflict with **(A).**
  So k cannot be greater than some k_{max}, where **(A)** and **(B)** are both satisfied with the equality sign.

$$\frac{k_{max}^2 \alpha^2}{\| w_0 \|^2} = k_{max}\beta \Rightarrow \boxed{k_{max} = \frac{\| w_0 \|^2}{\alpha^2}\beta}$$

- Perceptron convergence algorithm terminates in at most
  $n_{max} = \dfrac{\beta \, \|\mathbf{w_0}\|^2}{\alpha^2}$ iterations.
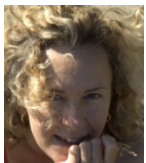
Perceptron: Limitations

- The perceptron can only model linearly separable functions.

- The perceptron can be used to model the following Boolean functions:

- AND / OR / COMPLEMENT

- But it cannot model the XOR. Why?
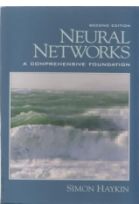
Comparison Adaline and Perceptron

- Both represent different implementations of a single-layer perceptron based on error-correction learning.

Adaline: Linear.

- Model of a neuron

  Perceptron: Non linear.
  Hard-Limiter activation function.
  McCulloch-Pitts model.

Adaline: Continuous.

- Learning Process

  Perceptron: A finite number of iterations.

Hochschule
Bonn-Rhein-Sieg

Fachbereich
Informatik

Prof. Dr.
Paul G. Plöger

40

ANN Learning

- *Learning is a process by which the free parameters of a neural network are adapted in an desired way through a process of stimulation by the environment in which the network is embedded.*
The type of learning is determined by the manner in which the parameter change takes place.
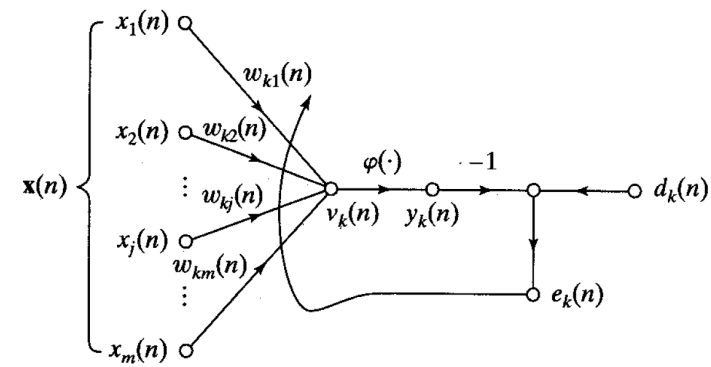
$$e.g. \quad w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}$$

1st Learning Rule: Error Correction

n	time step
k	neuron number
$d_k(n)$	desired signal
$y_k(n)$	observed signal

- Error e_k actuates a controls mechanism

$$e_k(n) = d_k(n) - y_k(n)$$

Hochschule Bonn-Rhein-Sieg   **Fachbereich Informatik**   **Prof. Dr. Paul G. Plöger**

1st Learning Rule: Error Correction

n   time step
k   neuron number
$d_k(n)$ desired signal
$y_k(n)$ observed signal

- Error e_k actuates a controls mechanism
- Cost function **E** on top (here: instantaneous at time n, local at output node k)

$$e_k(n) = d_k(n) - y_k(n)$$

$$E(n) = \frac{1}{2} e_k^{\,2}(n)$$

Hochschule Bonn-Rhein-Sieg　　**Fachbereich Informatik**　　**Prof. Dr. Paul G. Plöger**

1st Learning Rule: Error Correction

n   time step
k   neuron number
$d_k(n)$ desired signal
$y_k(n)$ observed signal

- Error e_k actuates a controls mechanism
- Cost function E on top (here: instantaneous at time n, local at output node k)
- Minimization of cost function: **Widrow-Hoff rule (delta rule)**

$$e_k(n) = d_k(n) - y_k(n)$$

$$E(n) = \frac{1}{2} e_k^{\,2}(n)$$

$$\Delta w_{kj}(n) = e_k(n)x_j(n)$$

**Hochschule
Bonn-Rhein-Sieg**   **Fachbereich
Informatik**   **Prof. Dr.
Paul G. Plöger**   44

1st Learning Rule: Error Correction

n   time step
k   neuron number
$d_k(n)$ desired signal
$y_k(n)$ observed signal

- Error e_k actuates a controls mechanism
- Cost function E on top (here: instantaneous at time n, local at output node k)
- Minimization of cost function: **Widrow-Hoff rule (delta rule)**
- *The adjustment made to a synaptic weight of a neuron is proportional to the product of the error signal and the input signal of the synapse in question.*
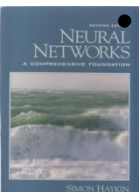
$$e_k(n) = d_k(n) - y_k(n)$$

$$E(n) = \tfrac{1}{2} e_k^{\,2}(n)$$

$$\Delta w_{kj}(n) = e_k(n) x_j(n)$$

1st Learning Rule: Error Correction

n   time step
k   neuron number
$d_k(n)$ desired signal
$y_k(n)$ observed signal

- Error e_k actuates a control mechanism
- Cost function E on top (here: instantaneous at time n, local at output node k)
- Minimization of cost function: **Widrow-Hoff rule (delta rule)**
- *The adjustment made to a synaptic weight of a neuron is proportional to the product of the error signal and the input signal of the synapse in question.*
- Learning rate η and final eq.

$$e_k(n) = d_k(n) - y_k(n)$$

$$E(n) = \frac{1}{2} e_k^{\,2}(n)$$

$$\Delta w_{kj}(n) = e_k(n) x_j(n)$$

$$w_{kj}(n+1) = w_{kj}(n) + \eta\, \Delta w_{kj}$$

2nd: Memory based learning

$$0 \qquad 0$$
$$0$$
$$0 \quad 0 \qquad 0$$
$$0$$
$$1 \qquad 0$$
$$1 \qquad 0$$
outlier $\longrightarrow 0^d \qquad 0$
$$1 \qquad 1$$
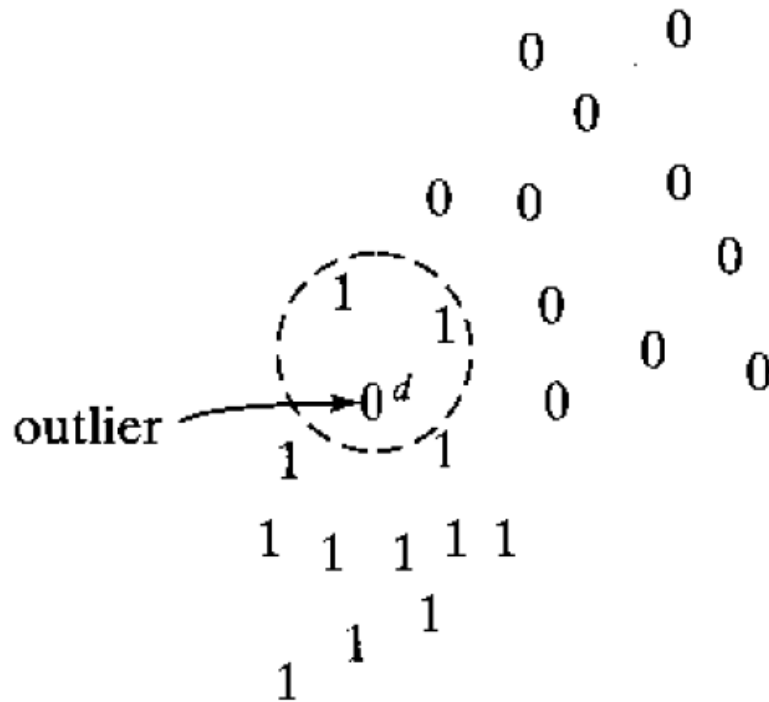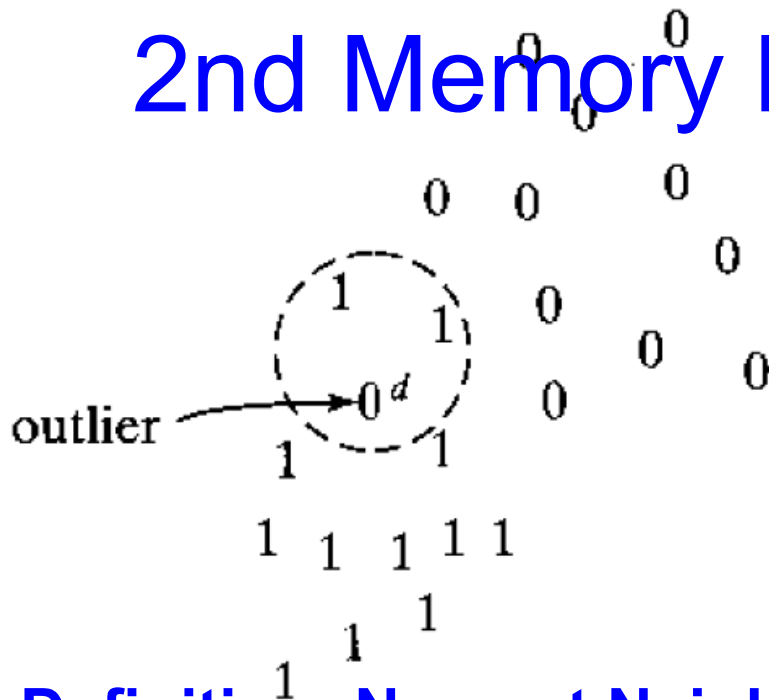$$1 \quad 1 \quad 1 \quad 1 \quad 1$$
$$1$$
$$1$$
$$1$$

FIGURE 2.2 The area lying inside the dashed circle includes two points pertaining to class 1 and an outlier from class 0. The point d corresponds to the test vector $\mathbf{x}_{test}$. With $k = 3$, the *k-nearest* neighbor *classifier* assigns class 1 to point d even though it lies closest to the *outlier*.

(majority vote)

2nd Memory based learning cont.

outlier

Algorithm: k-nearest neighbor learning

$Given\ L,\ x_{test} \notin L,\ k \in IN\ fixed,$

$class\ function : class\_of()\ on\ L$

$Set\ x' = \{\},\ L_0 = L,\ Classf = empty\_list$

$for\ j = 1...k\ do\ \{$

$\quad L_j \leftarrow L_{j-1} \setminus x';$

$\quad x' \leftarrow find\ NN\ to\ x_{test}\ in\ L_j;$

$\quad c \leftarrow class\_of(x');$

$\quad Classf \leftarrow push(c);$

$\}$

$set\ c(x_{test}) := most\ frequent\ value\ in\ Classf$

Definition: Nearest Neighbor

$Given\ L = \{x_1, x_2, \ldots, x_N\}\ and\ x_{test} \notin L.$

$Then\ x' \in L\ is\ called$

$nearest\ neighbor\ to\ x_{test}\ in\ L :\Leftrightarrow$
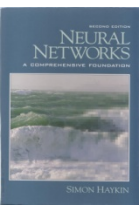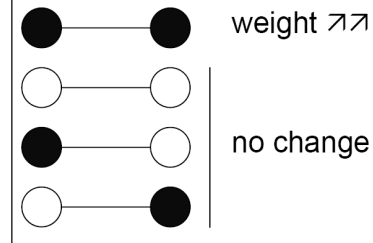
$$\min_i d(x_i, x_{test}) = d(x', x_{test})$$

3rd Hebbian learning

Donald Hebb, Canadian psychologist, wrote a revolutionary paper in 1949:

"Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability…. When an axon of cell A is near enough to excite a 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."

1. If two neurons on either side of a synapse (connection) are activated simultaneously (i.e., synchronously), then the strength of that synapse is selectively increased.
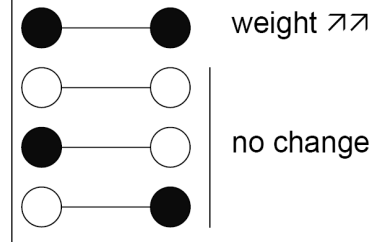
3rd Hebbian learning

Donald Hebb, Canadian psychologist, wrote a revolutionary paper in 1949:

"Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability.… When an axon of cell A is near enough to excite a 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."

1. If two neurons on either side of a synapse (connection) are activated simultaneously (i.e., synchronously), then the strength of that synapse is selectively increased.

2. If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated.
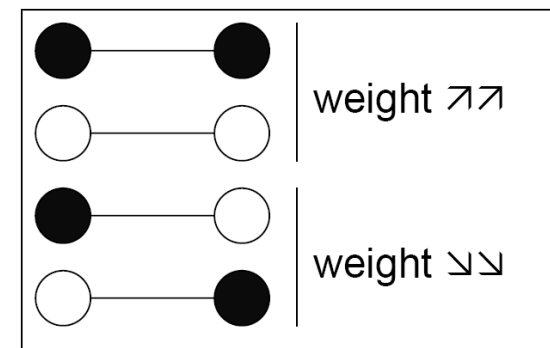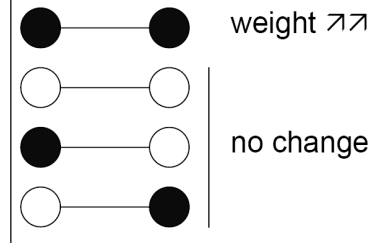
3rd Hebbian learning

Donald Hebb, Canadian psychologist, wrote a revolutionary paper in 1949:

"Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability.… When an axon of cell A is near enough to excite a 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."

1. If two neurons on either side of a synapse (connection) are activated simultaneously (i.e., synchronously), then the strength of that synapse is selectively increased.

2. If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated.
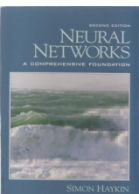
weight ↗↗

weight ↘↘

3rd Hebbian learning

- Time-dependent mechanism
- Local mechanism
- Interactive mechanism
- Conjuctional or correlational mechanism

- In general

$$\Delta w_{kj}(n) = F(y_k(n), x_j(n))$$

- Most simple

$$\Delta w_{kj}(n) = \eta y_k(n) x_j(n)$$

- Covariance hypothesis

$$\Delta w_{kj}(n) = \eta(y_k - \bar{y})(x_j - \bar{x})$$

**summarized: cells that fire together, wire together.**

Hochschule
Bonn-Rhein-Sieg

Fachbereich
Informatik

Prof. Dr.
Paul G. Plöger

52

4th Competitive Learning

- In competitive learning the output neurons compete among themselves to become active (fired).

- Hebbian learning ⇔
  several output neurons may
  be active simultaneously,

- competitive learning ⇔
  only a **single output neuron**
  is active **at any one time**.

- A set of neurons that are all the same except for some randomly distributed synaptic weights, and which therefore respond differently to a given set of input patterns.

- A limit imposed on the "strength" of each neuron.

- A mechanism that permits the neurons to compete for the right to respond to a given subset of inputs, such that only one output neuron, or only one neuron per group, is active (i.e., "on") at a time.

The neuron that wins the competition is called a **winner-takes-all neuron.**
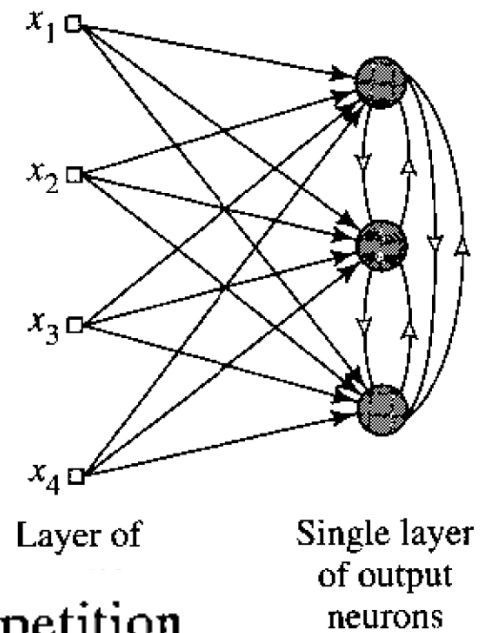
(Rumelhart and Zipser,1985)
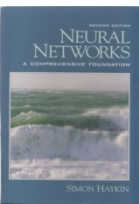
4th Competitive Learning (cont)

$$y_k = \begin{cases} 1 & \text{if } v_k > v_j \text{ for all } j, j \neq k \\ 0 & \text{otherwise} \end{cases}$$

$$\sum_j w_{kj} = 1 \qquad \text{for all } k$$

$$\Delta w_{kj} = \begin{cases} \eta(x_j - w_{kj}) & \text{if neuron } k \text{ wins the competition} \\ 0 & \text{if neuron } k \text{ loses the competition} \end{cases}$$

Layer of

Single layer of output neurons
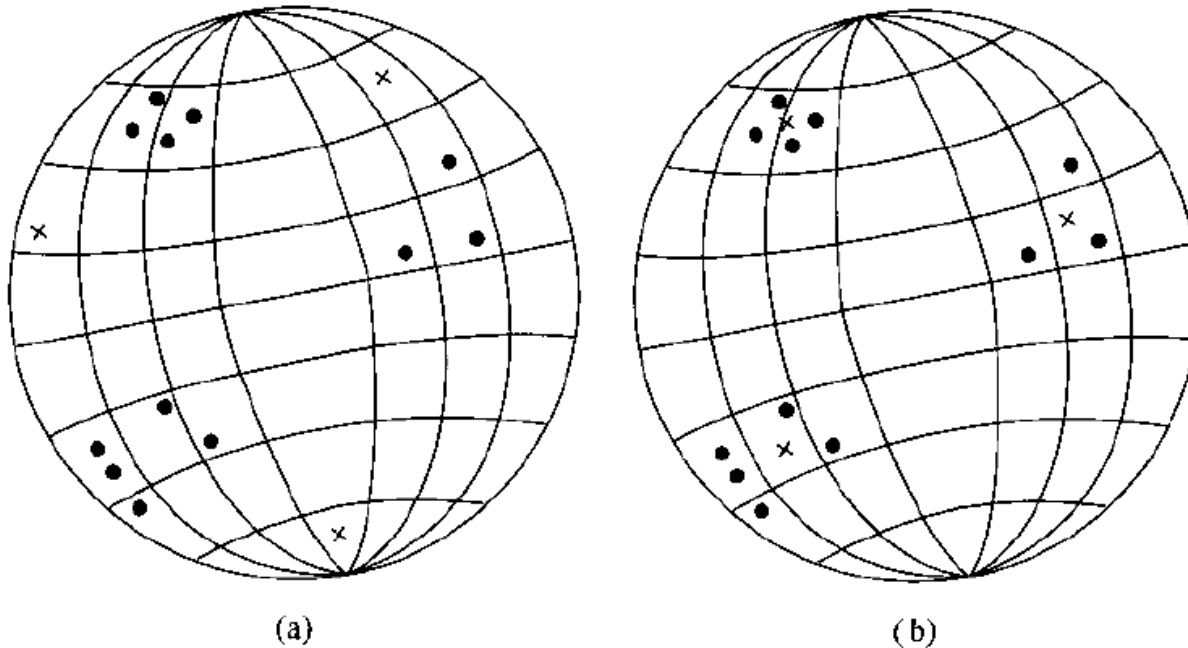
4th Competitive learning (cont)

(a)   (b)

FIGURE 2.5 Geometric interpretation of the competitive learning process. The dots represent the input vectors, and the crosses represent the synaptic weight vectors of three output neurons. (a) Initial state of the network. (b) Final state of the network.
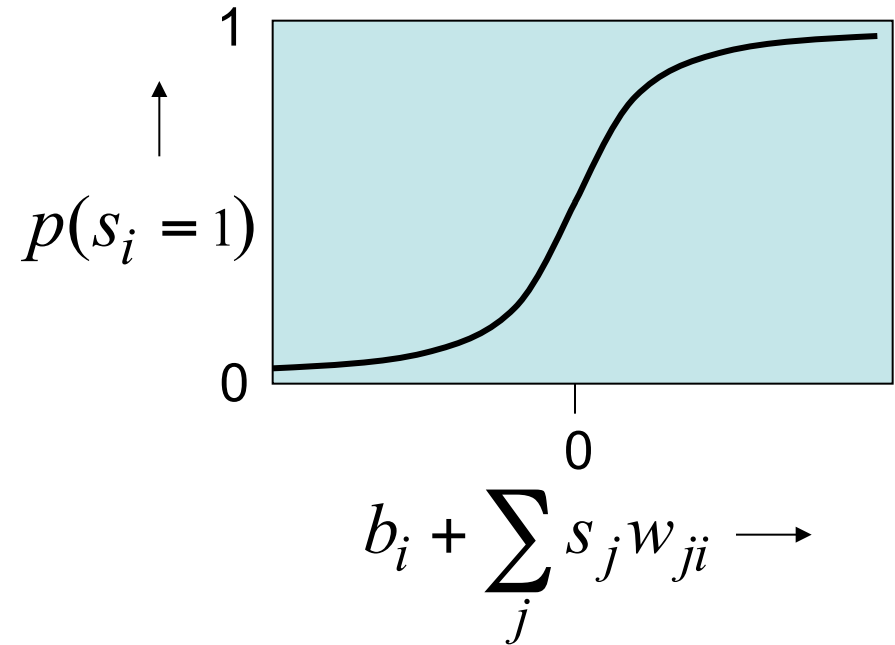
Sketch: 5th Boltzmann learning

hidden

- Recurrent ANN
- Two groups: visible and hidden neurons
- Binary neurons (+/- 1 as state)
- operate by flipping

visible
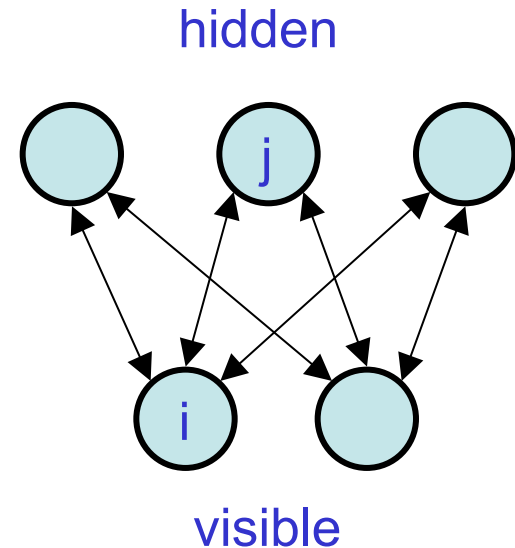
Stochastic binary units
(Bernoulli variables)

- These have a state of 1 or 0.

- The probability of turning on is determined by the weighted input from other units (plus a bias)

$$p(s_i = 1) = \frac{1}{1 + \exp(-b_i - \sum_j s_j w_{ji})}$$

$p(s_i = 1)$

$$b_i + \sum_j s_j w_{ji} \longrightarrow$$

Cited from:
2007 NIPS Tutorial on:
Deep Belief Nets
by Geoffrey Hinton

Hochschule Bonn-Rhein-Sieg   **Fachbereich Informatik**   **Prof. Dr. Paul G. Plöger**

Sketch: 5th Boltzmann learning

hidden

- Recurrent ANN
- Two groups: visible and hidden neurons
- Binary neurons (+/- 1 as state)
- operate by flipping
- Modes of operation: clamped / free running conditions

visible
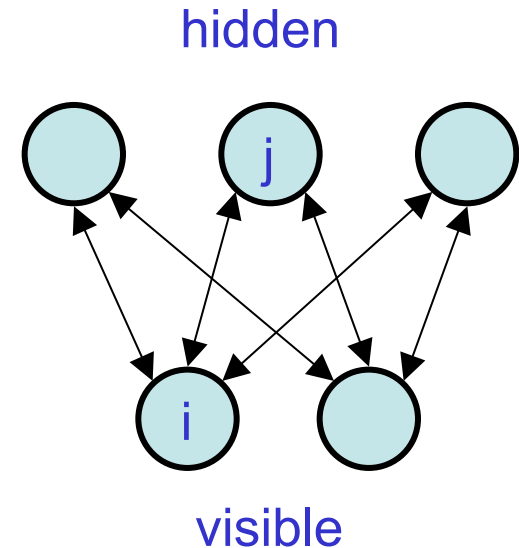
ρ_{kj}^+ *correlation in clamped state*

ρ_{kj}^- *correlation in free running state*

$$\Delta w_{kj} = \eta(\rho_{kj}^+ - \rho_{kj}^-) \quad j \neq k$$

Example

- **How to train a perceptron to recognize this 3?**

-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	+1	+1	+1	+1	-1	-1
-1	-1	-1	-1	-1	+1	-1	-1
-1	-1	-1	+1	+1	+1	-1	-1
-1	-1	-1	-1	-1	+1	-1	-1
-1	-1	-1	-1	- 1	+1	-1	-1
-1	-1	+1	+1	+1	+1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

- Assign 1 to weights of input values that are equal to +1, and -1 to weights of input values that are equal to -1

Hochschule
Bonn-Rhein-Sieg

Fachbereich
Informatik

Prof. Dr.
Paul G. Plöger

59