Chapter ML:VI

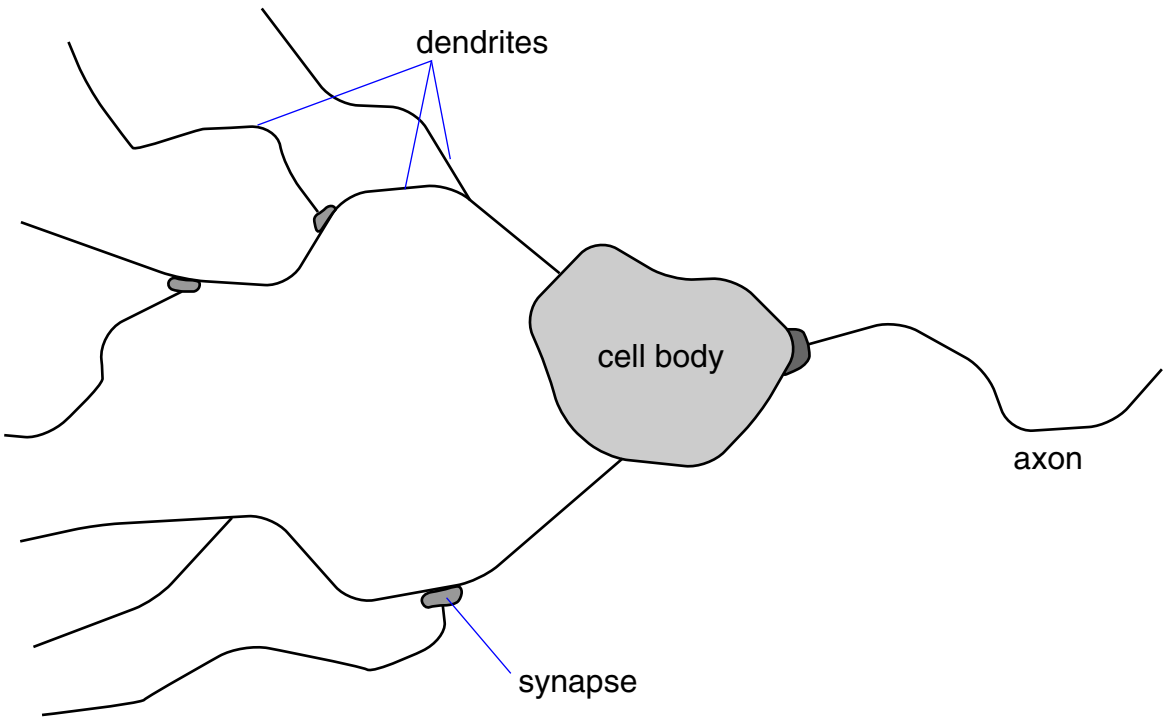
VI. Neural Networks

- Perceptron Learning
- Gradient Descend
- Multilayer Perceptron
- Radial Basis Functions

Perceptron Learning

The Biological Model

Simplified model of a neuron:



Perceptron Learning

The Biological Model (continued)

Neuron characteristics:

- The numerous dendrites of a neuron serve as its input channels for electrical signals.
- At particular contact points between the dendrites, the so-called synapses, electrical signals can be initiated.
- A synapse can initiate signals of different strengths, whereas the strength is encoded by the frequency of a pulse train.
- The cell body of a neuron accumulates the incoming signals.
- If a particular stimulus threshold is exceeded, the cell body generates a signal, which is output via the axon.
- The processing of the signals is unidirectional. (from left to right in the figure)

Perceptron Learning

History

1943 Warren McCulloch and Walter Pitts present a model of the neuron

1949 Donald Hebb postulates a new learning paradigm: reinforcement only for active neurons (neurons that are involved in a decision process)

1958 Frank Rosenblatt develops the perceptron model

1962 Rosenblatt proves the perceptron convergence theorem

1969 Marvin Minsky and Seymour Papert publish a book on the limitations of the perceptron model

1970

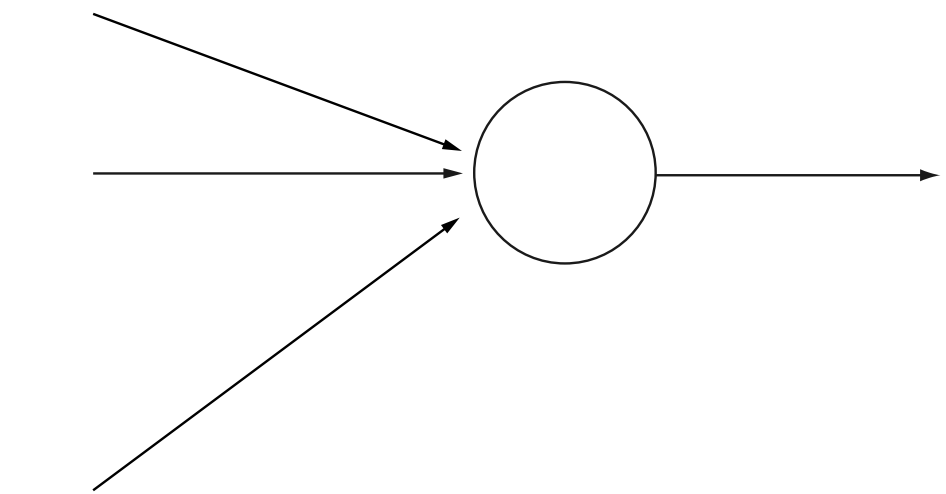
⋮

1985

1986 David Rumelhart and James McClelland present the multilayer perceptron

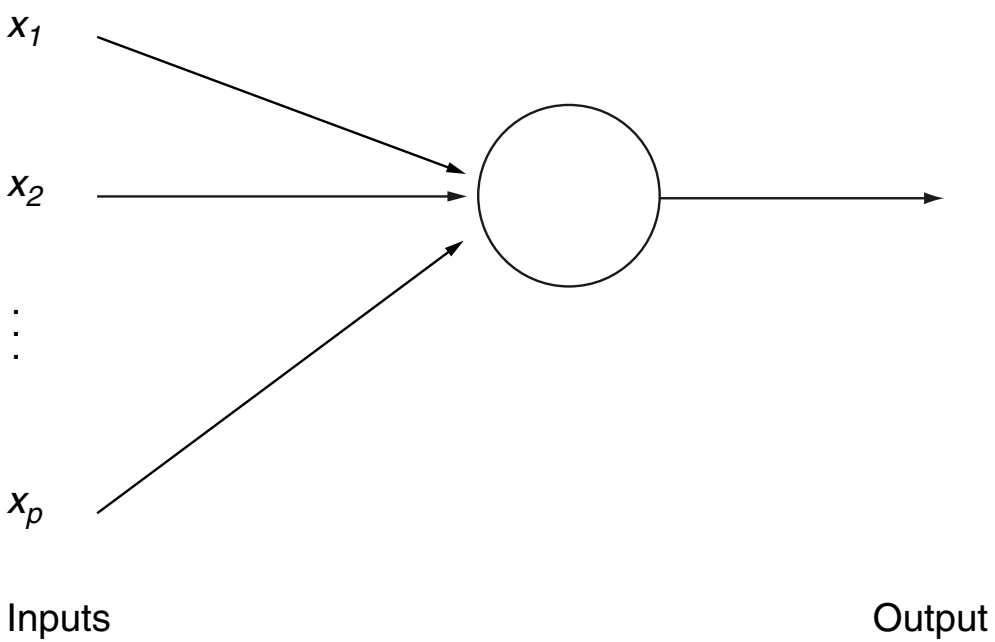
Perceptron Learning

The Perceptron of Rosenblatt [1958]



Perceptron Learning

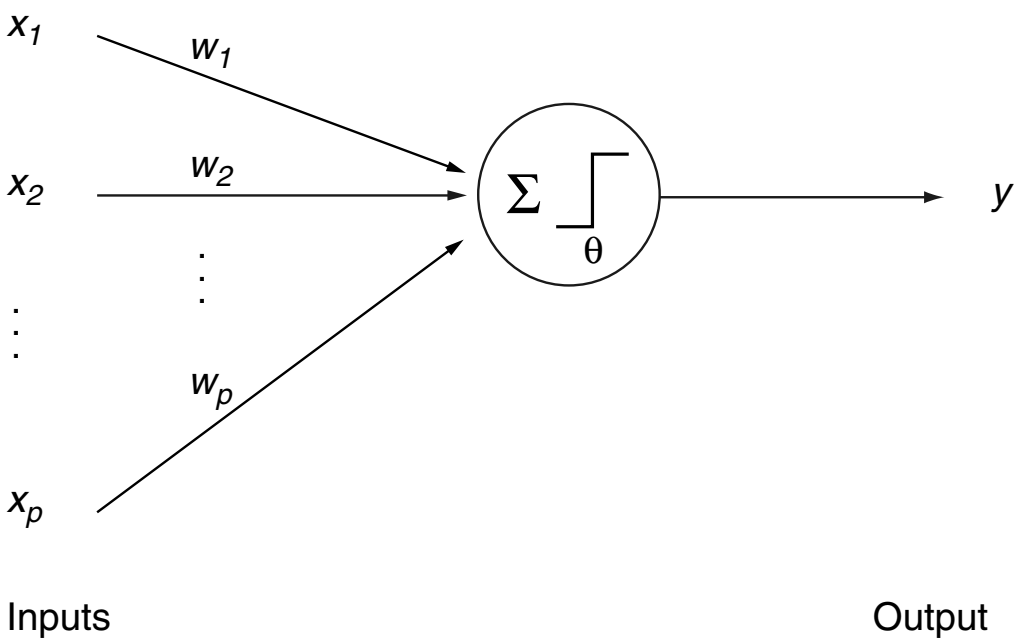
The Perceptron of Rosenblatt [1958]



$x_i, w_i \in \mathbf{R}, \quad i = 1 \ldots p$

Perceptron Learning

The Perceptron of Rosenblatt [1958]



$x_i, w_i \in \mathbf{R}, \quad i = 1 \ldots p$

Remarks:

- The perceptron of Rosenblatt is based on the neuron model of McCulloch and Pitts.
- The perceptron is a “feed forward system”.

Perceptron Learning

Specification of Classification Problems [ML Introduction]

Characterization of the model (model world):

- X is a set of feature vectors, also called feature space. $X \subseteq \mathbf{R}^p$
- C is a set of classes. $C = \{0, 1\}$
- $c : X \to C$ is the ideal classifier for X.
- $D = \{(\mathbf{x}_1, c(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, c(\mathbf{x}_n))\} \subseteq X \times C$ is a set of examples.

How could the hypothesis space H look like?

Perceptron Learning

Computation in the Perceptron [Regression]

If $\sum_{i=1}^{p} w_i x_i \geq \theta$ then $y(\mathbf{x}) = 1$, and

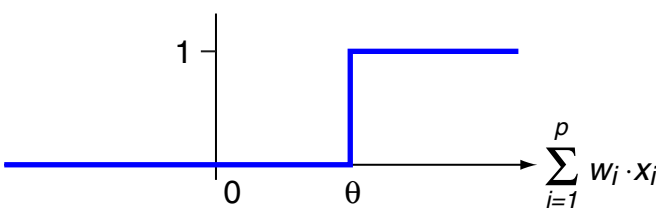
if $\sum_{i=1}^{p} w_i x_i < \theta$ then $y(\mathbf{x}) = 0$.

Perceptron Learning

Computation in the Perceptron [Regression]

If $\sum_{i=1}^{p} w_i x_i \geq \theta$ then $y(\mathbf{x}) = 1$, and

if $\sum_{i=1}^{p} w_i x_i < \theta$ then $y(\mathbf{x}) = 0$.



where $\sum_{i=1}^{p} w_i x_i = \mathbf{w}^T \mathbf{x}$. (or different notations for the scalar product)

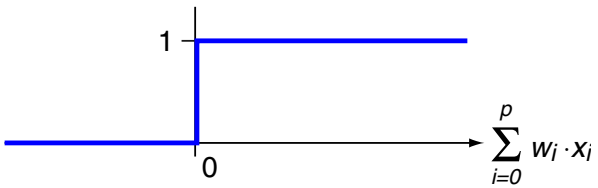
➜ The hypotheses are determined by $\theta, w_1, \ldots, w_p$.

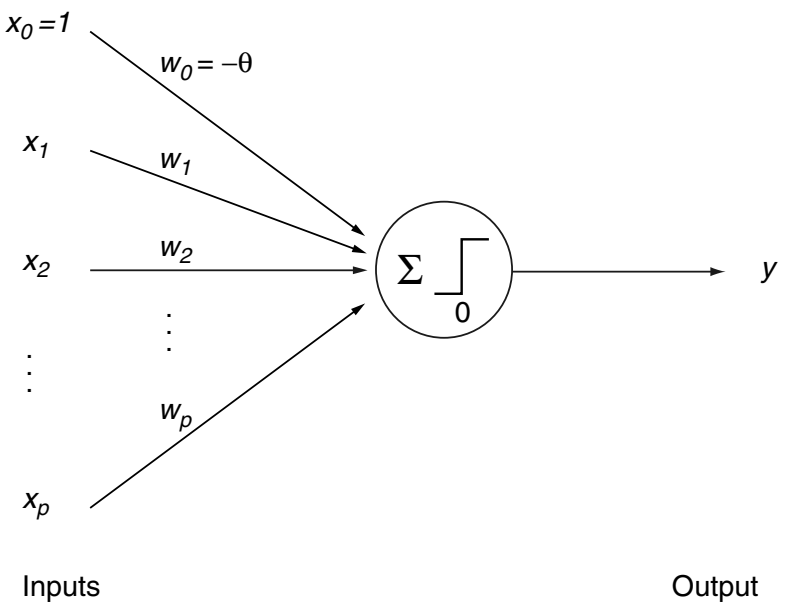
Perceptron Learning

Computation in the Perceptron (continued)

$$y(\mathbf{x}) = \textit{heaviside}(\sum_{i=1}^{p} w_i x_i - \theta)$$

$$= \textit{heaviside}(\sum_{i=0}^{p} w_i x_i) \quad \text{with } w_0 = -\theta,\ x_0 = 1$$





➜ The hypotheses are determined by $w_0, w_1, \ldots, w_p$.

Remarks:

- If the weight vector is extended by $w_0 = -\theta$, and if the feature vectors are extended by the constant feature $x_0 = 1$, the learning algorithm gets a canonical form. Implementations of neural networks introduce this extension often implicitly.
- The function *heaviside* is named after the mathematician Oliver Heaviside.

Perceptron Learning

Weight Adaptation [*LMS* Algorithm]

Algorithm:	*PT*	Perceptron Training
Input:	D	Training examples of the form $(\mathbf{x}, c(\mathbf{x}))$ with $\|\mathbf{x}\| = p+1,\ c(\mathbf{x}) \in \{0,1\}$.
	η	Learning rate, a small positive constant.
Output:	$\mathbf{w}$	Weight vector.

PT(D, η)

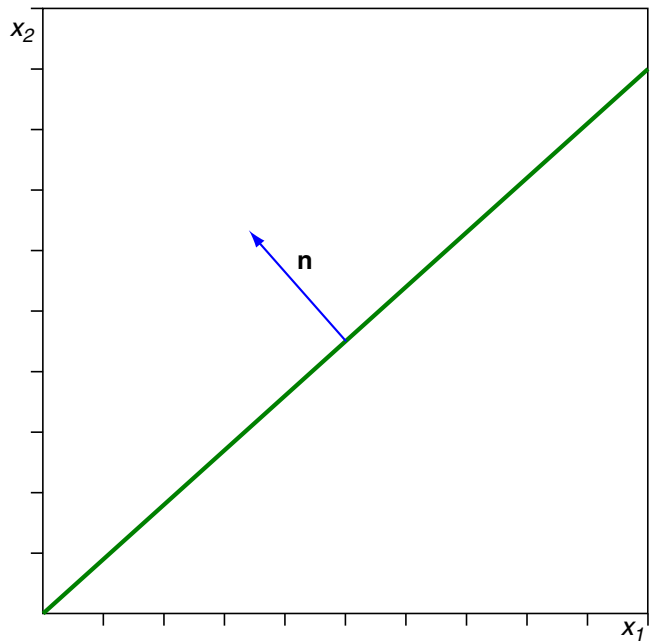
1. *initialize_random_weights*$(\mathbf{w})$, $t = 0$
2. **REPEAT**
3. $t = t + 1$
4. $(\mathbf{x}, c(\mathbf{x})) =$ *random_select*(D)
5. *error* $= c(\mathbf{x}) -$ *heaviside*$(\mathbf{w}^T\mathbf{x})$
6. **FOR** $i = 0$ **TO** p **DO**
7. $\Delta w_i = \eta \cdot$ *error* $\cdot\, x_i$
8. $w_i = w_i + \Delta w_i$
9. **ENDDO**
10. **UNTIL**(*convergence*$(D, y(D))$ **OR** $t > t_{\text{max}}$)
11. *return*$(\mathbf{w})$

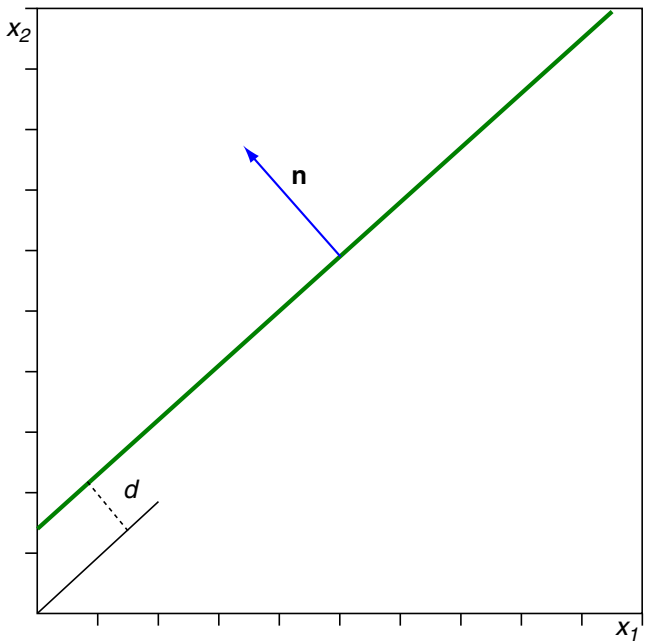
Remarks:

- The variable t denotes the time. At each point in time the learning algorithm gets an example presented and, as a consequence, may adapt the weight vector.
- $y(D)$ comprises the current classification results for D under $\mathbf{w}$.
- The weight adaptation rule compares the true class $c(\mathbf{x})$ (the ground truth) to the class computed by the perceptron. In case of a wrong classification of a feature vector $\mathbf{x}$ either -1 or $+1$ is returned—independent of the exact numeric difference between $c(\mathbf{x})$ and $\mathbf{w}^T\mathbf{x}$.

Perceptron Learning

Weight Adaptation (continued)



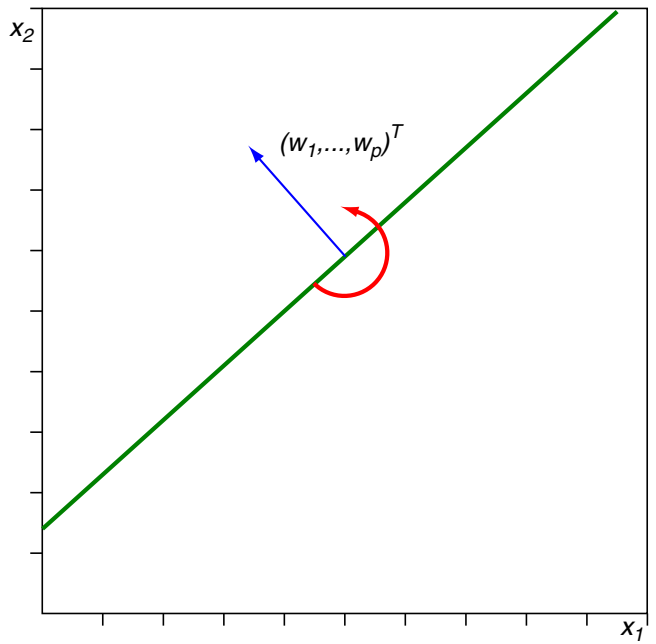


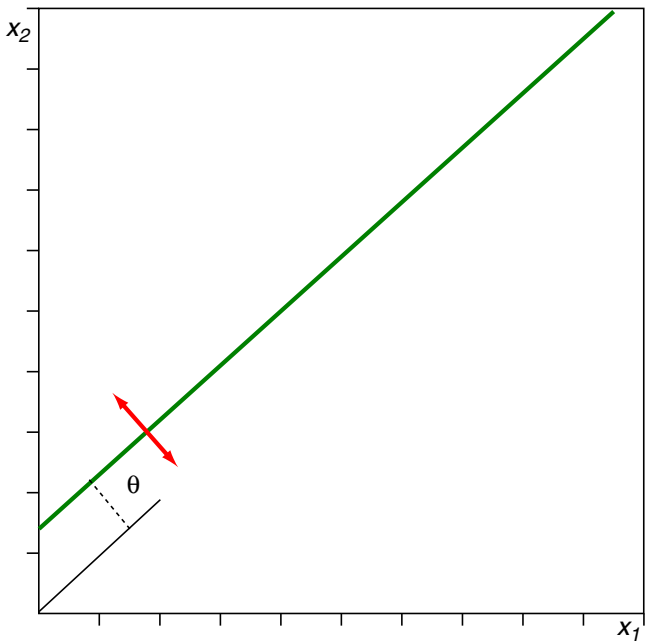
Definition of an (affine) hyperplane: $\mathbf{n}^T\mathbf{x} = d$.

- $\mathbf{n}$ denotes a normal vector that is perpendicular to the hyperplane.
- If $||\mathbf{n}|| = 1$ then $|d|$ corresponds to the distance of the origin to the hyperplane.
- If $\mathbf{n}^T\mathbf{x} < d$ and $d \geq 0$ then $\mathbf{x}$ and the origin lie on the same side of the hyperplane.

Perceptron Learning

Weight Adaptation (continued)





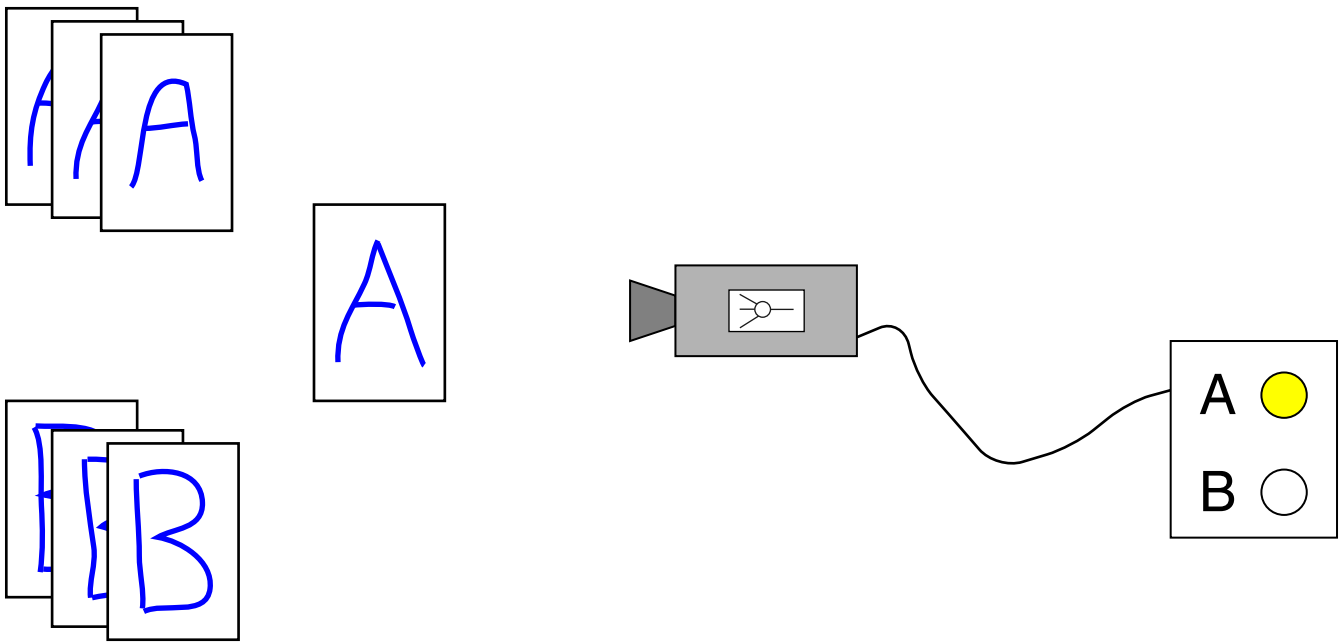
Definition of an (affine) hyperplane: $\mathbf{w}^T\mathbf{x} = 0 \Leftrightarrow \sum_{i=1}^{p} w_i x_i = \theta = -w_0$.

Remarks:

- A perceptron defines a hyperplane that is perpendicular (= normal) to $(w_1, \ldots, w_p)^T$.
- θ or $-w_0$ specify the offset of the hyperplane from the origin, along $(w_1, \ldots, w_p)^T$.
- The set of all weight vectors $\mathbf{w} = (w_0, w_1, \ldots, w_p)^T$ form the hypothesis space H.
- Weight adaptation means learning, and the shown learning paradigm is supervised.
- The computation of the weight difference Δw_i in Line 7 of the *PT* Algorithm considers a feature vector $\mathbf{x}$ componentwise. In particular, if some x_i is zero, Δw_i will be zero as well. Keyword: Hebbian learning [Hebb 1949]

Perceptron Learning

Illustration



- The examples are presented to the perceptron.
- The perceptron computes a value that is interpreted as class label.

Perceptron Learning

Illustration (continued)

Encoding:

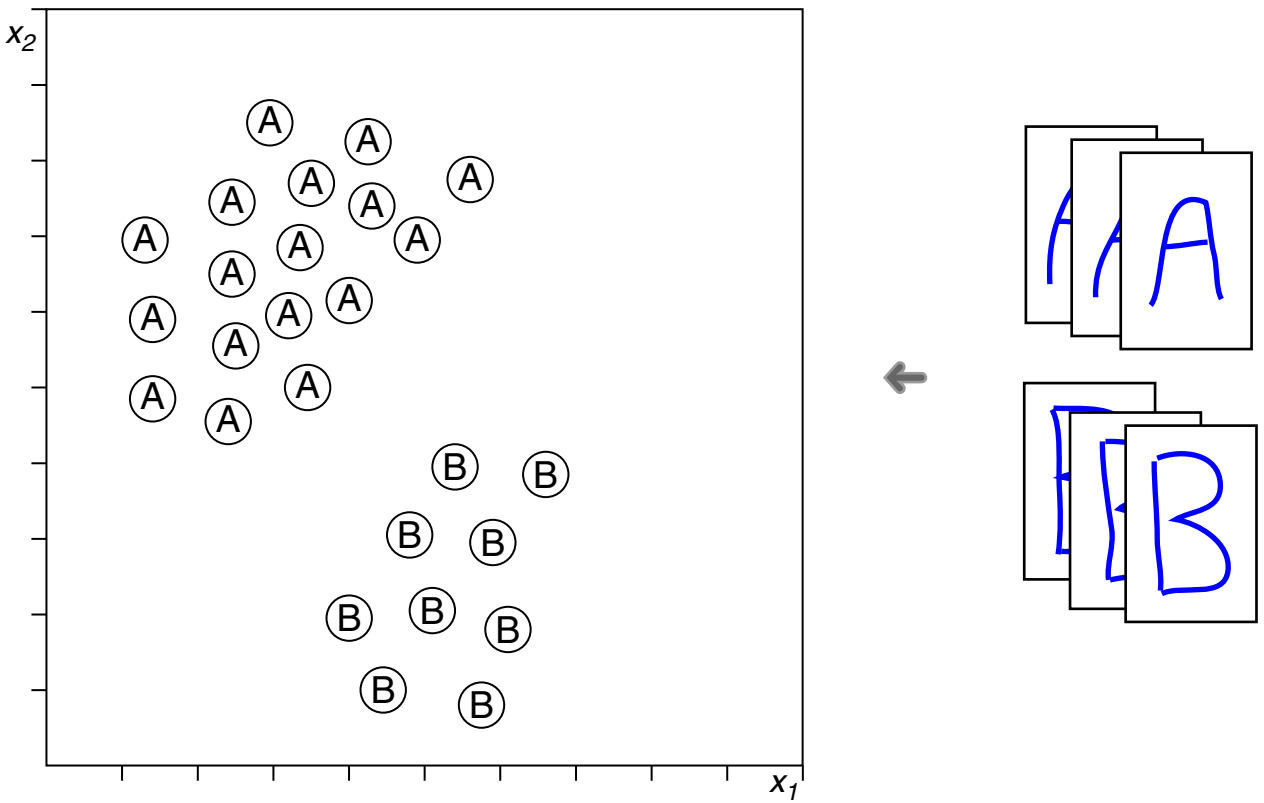
- The encoding of the examples is based on expressive features: number of line crossings, most acute angle, longest line, etc.
- The class label, $c(\mathbf{x})$, is encoded as a number. Examples from A are labeled with 1, examples from B are labeled with 0.

$$\underbrace{\begin{pmatrix} x_{1_1} \\ x_{1_2} \\ \vdots \\ x_{1_p} \end{pmatrix} \quad \cdots \quad \begin{pmatrix} x_{k_1} \\ x_{k_2} \\ \vdots \\ x_{k_p} \end{pmatrix}}_{\text{Class } A \;\simeq\; c(\mathbf{x}) = 1} \qquad \underbrace{\begin{pmatrix} x_{l_1} \\ x_{l_2} \\ \vdots \\ x_{l_p} \end{pmatrix} \quad \cdots \quad \begin{pmatrix} x_{m_1} \\ x_{m_2} \\ \vdots \\ x_{m_p} \end{pmatrix}}_{\text{Class } B \;\simeq\; c(\mathbf{x}) = 0}$$

Perceptron Learning

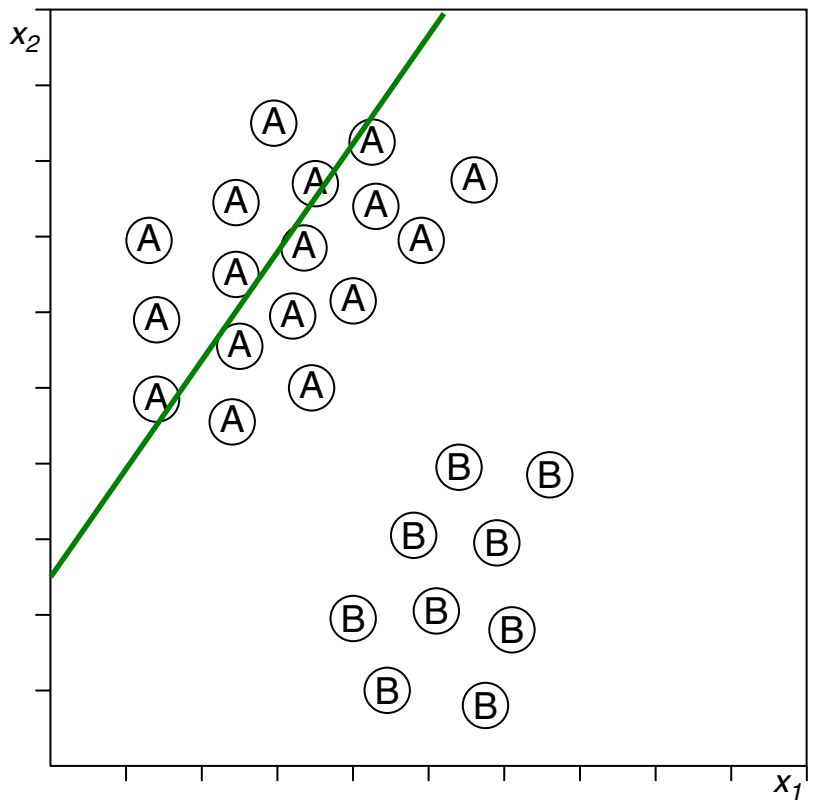
Illustration (continued)

A possible configuration of encoded objects in the feature space X :



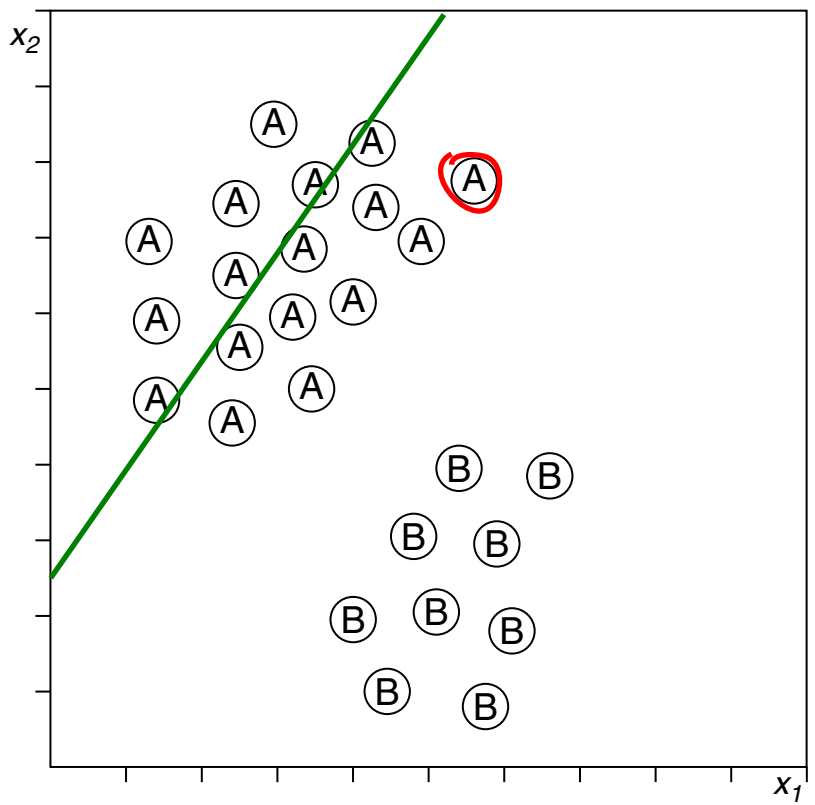
Perceptron Learning

Illustration (continued)



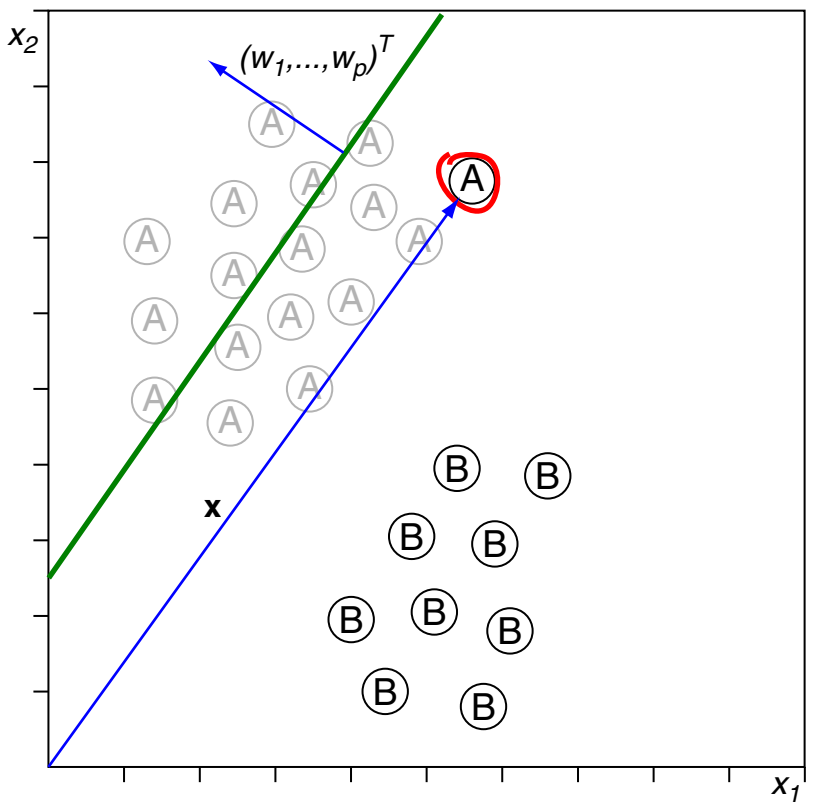
Perceptron Learning

Illustration (continued) [*PT* Algorithm]



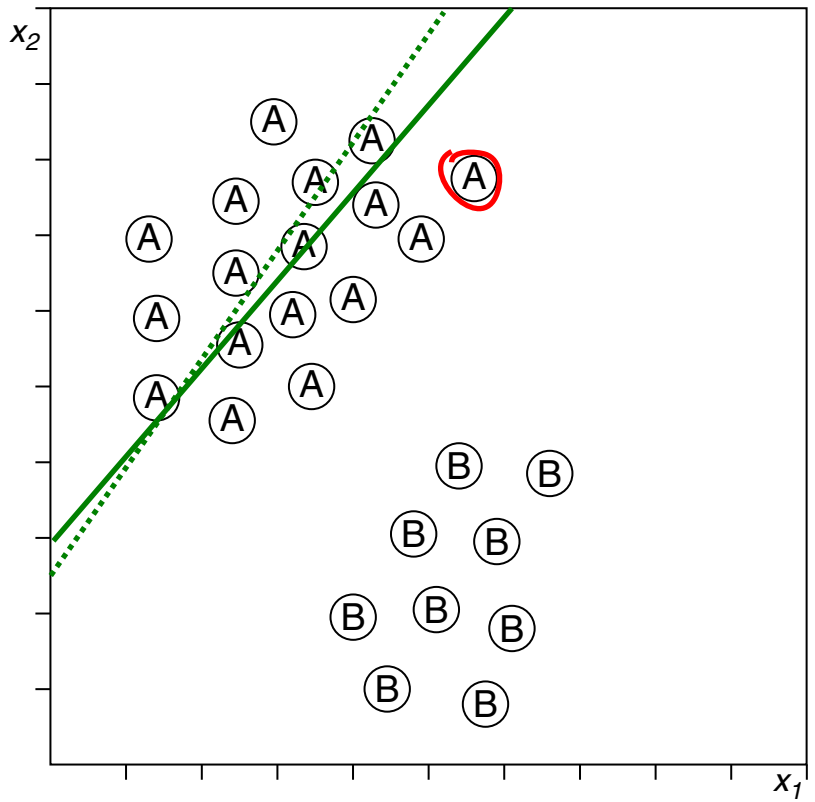
Perceptron Learning

Illustration (continued) [*PT* Algorithm]



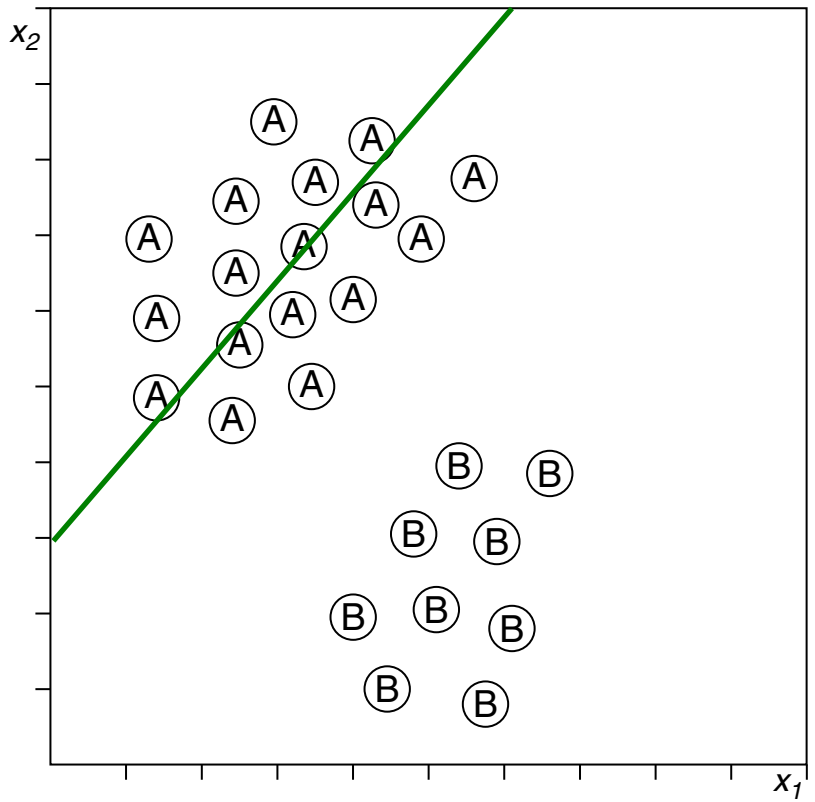
Perceptron Learning

Illustration (continued) [*PT* Algorithm]



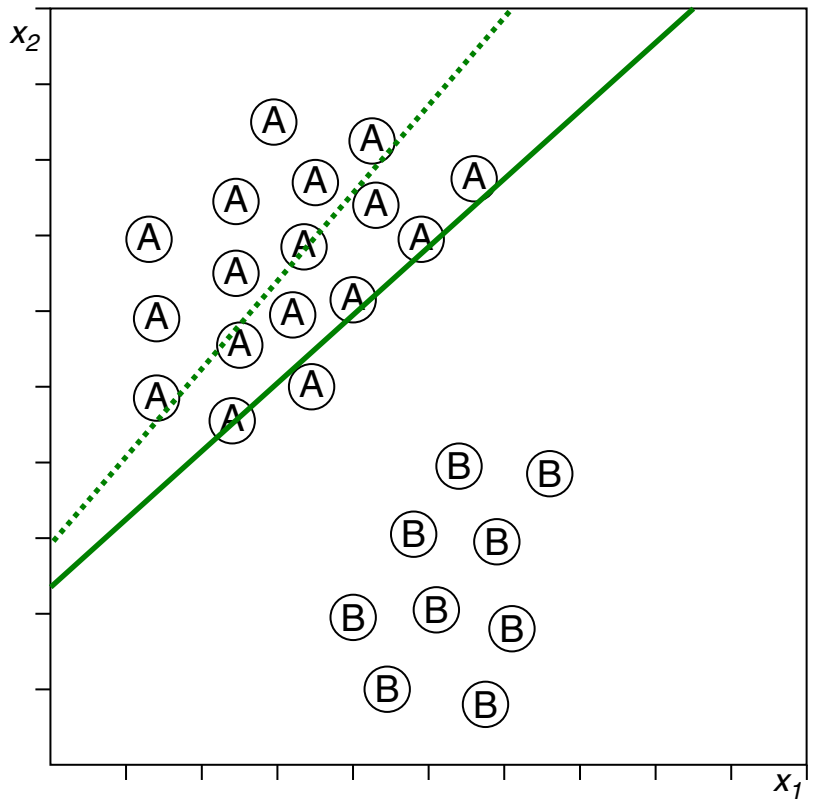
Perceptron Learning

Illustration (continued) [*PT* Algorithm]



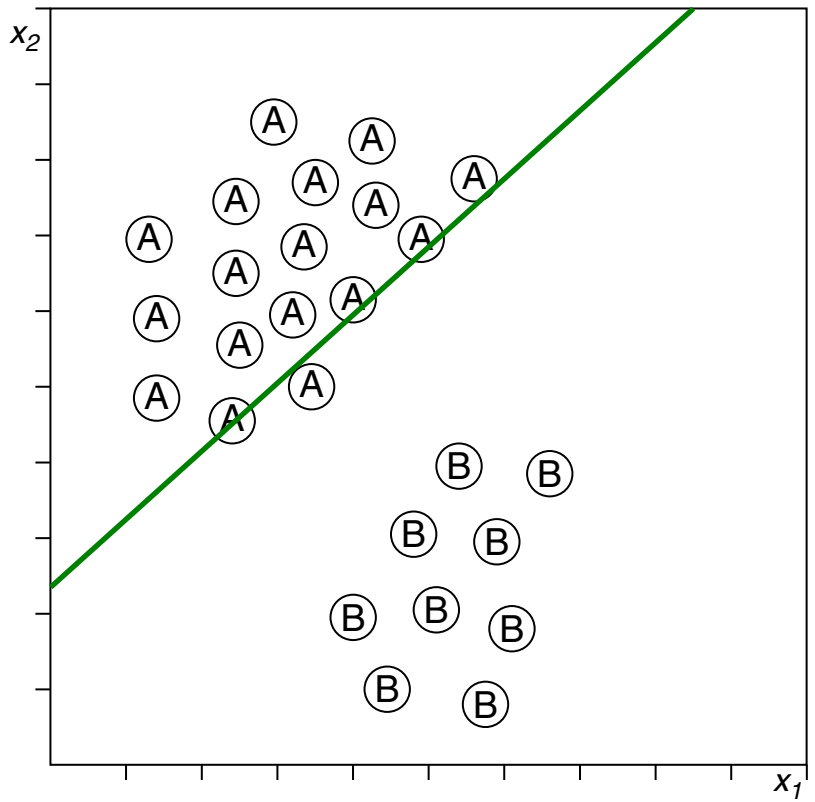
Perceptron Learning

Illustration (continued) [*PT* Algorithm]



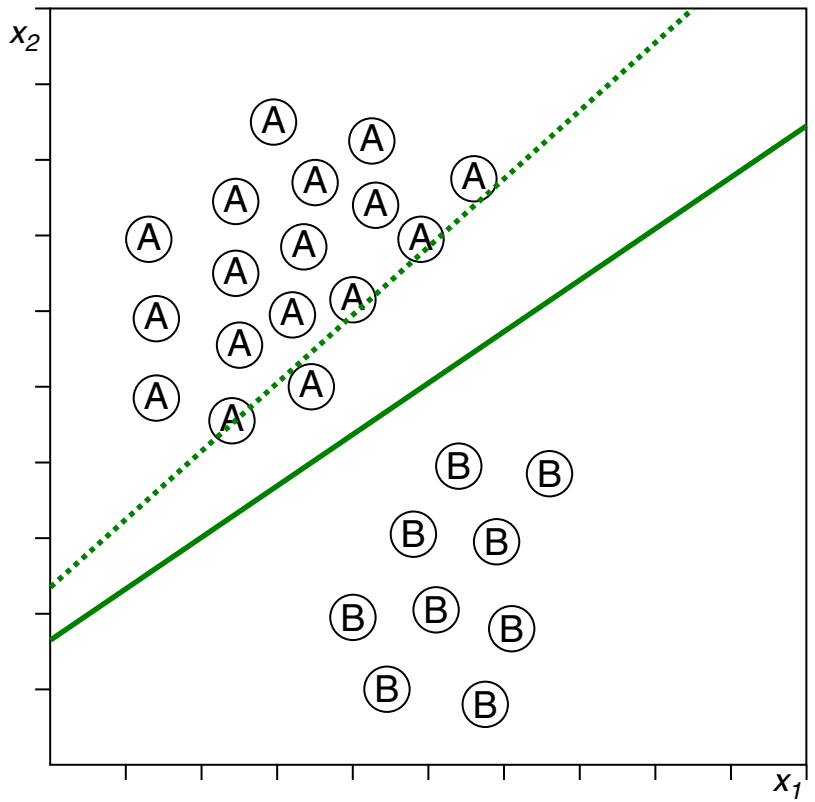
Perceptron Learning

Illustration (continued) [*PT* Algorithm]



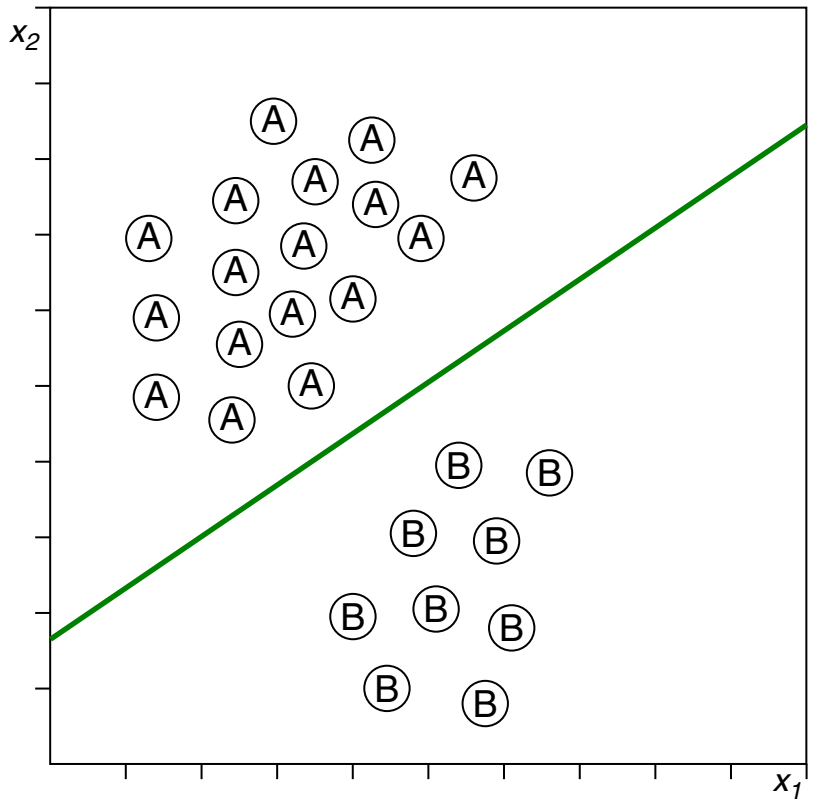
Perceptron Learning

Illustration (continued) [*PT* Algorithm]



Perceptron Learning

Illustration (continued) [*PT* Algorithm]



Perceptron Learning

Perceptron Convergence Theorem

Questions:

1. Which kind of learning tasks can be addressed with the functions of the hypothesis space H?
2. Can the *PT* Algorithm construct such a function for a given task?

Theorem 1 (Perceptron Convergence [Rosenblatt 1962]**)**
Let X_0 and X_1 be two finite sets with vectors of the form $\mathbf{x} = (-1, x_1, \ldots, x_p)^T$, let $X_1 \cap X_0 = \emptyset$, and let $\widehat{\mathbf{w}}$ define a separating hyperplane with respect to X_0 and X_1. Moreover, let D be a set of examples of the form $(\mathbf{x}, 0)$, $\mathbf{x} \in X_0$ and $(\mathbf{x}, 1)$, $\mathbf{x} \in X_1$. Then holds:

If the examples in D are processed with the *PT* Algorithm, the underlying weight vector $\mathbf{w}$ will converge within a finite number of iterations.

Perceptron Learning

Perceptron Convergence Theorem: Proof

Preliminaries:

- The sets X_1 and X_0 are separated by the hyperplane $\widehat{\mathbf{w}}$. The proof requires that for all $\mathbf{x} \in X_1$ the inequality $\widehat{\mathbf{w}}^T\mathbf{x} > 0$ holds. This condition is always fulfilled, as the following consideration shows.

  Let $\mathbf{x}' \in X_1$ with $\widehat{\mathbf{w}}^T\mathbf{x}' = 0$. Since X_0 is finite, the members $\mathbf{x} \in X_0$ have a minimum positive distance δ with regard to the hyperplane $\widehat{\mathbf{w}}$. Hence, $\widehat{\mathbf{w}}$ can be moved by $\frac{\delta}{2}$ towards X_0, resulting in a new hyperplane $\widehat{\mathbf{w}}'$ that still fulfills $(\widehat{\mathbf{w}}')^T\mathbf{x} < 0$ for all $\mathbf{x} \in X_0$, but that now also fulfills $(\widehat{\mathbf{w}}')^T\mathbf{x} > 0$ for all $\mathbf{x} \in X_1$.

Perceptron Learning

Perceptron Convergence Theorem: Proof

Preliminaries:

- The sets X_1 and X_0 are separated by the hyperplane $\widehat{\mathbf{w}}$. The proof requires that for all $\mathbf{x} \in X_1$ the inequality $\widehat{\mathbf{w}}^T\mathbf{x} > 0$ holds. This condition is always fulfilled, as the following consideration shows.
  Let $\mathbf{x}' \in X_1$ with $\widehat{\mathbf{w}}^T\mathbf{x}' = 0$. Since X_0 is finite, the members $\mathbf{x} \in X_0$ have a minimum positive distance δ with regard to the hyperplane $\widehat{\mathbf{w}}$. Hence, $\widehat{\mathbf{w}}$ can be moved by $\frac{\delta}{2}$ towards X_0, resulting in a new hyperplane $\widehat{\mathbf{w}}'$ that still fulfills $(\widehat{\mathbf{w}}')^T\mathbf{x} < 0$ for all $\mathbf{x} \in X_0$, but that now also fulfills $(\widehat{\mathbf{w}}')^T\mathbf{x} > 0$ for all $\mathbf{x} \in X_1$.
- For the weight vector $\mathbf{w}$ that is to be constructed by the *PT* Algorithm, the two inequalities must hold as well: $\mathbf{w}^T\mathbf{x} < 0$ for all $\mathbf{x} \in X_0$, and $\mathbf{w}^T\mathbf{x} > 0$ for all $\mathbf{x} \in X_1$.
- Consider the set $X' = X_1 \cup \{-\mathbf{x} \mid \mathbf{x} \in X_0\}$: the searched $\mathbf{w}$ fulfills $\mathbf{w}^T\mathbf{x} > 0$ for all $\mathbf{x} \in X'$.

Perceptron Learning

Perceptron Convergence Theorem: Proof

Preliminaries:

- The sets X_1 and X_0 are separated by the hyperplane $\widehat{\mathbf{w}}$. The proof requires that for all $\mathbf{x} \in X_1$ the inequality $\widehat{\mathbf{w}}^T\mathbf{x} > 0$ holds. This condition is always fulfilled, as the following consideration shows.
  Let $\mathbf{x}' \in X_1$ with $\widehat{\mathbf{w}}^T\mathbf{x}' = 0$. Since X_0 is finite, the members $\mathbf{x} \in X_0$ have a minimum positive distance δ with regard to the hyperplane $\widehat{\mathbf{w}}$. Hence, $\widehat{\mathbf{w}}$ can be moved by $\frac{\delta}{2}$ towards X_0, resulting in a new hyperplane $\widehat{\mathbf{w}}'$ that still fulfills $(\widehat{\mathbf{w}}')^T\mathbf{x} < 0$ for all $\mathbf{x} \in X_0$, but that now also fulfills $(\widehat{\mathbf{w}}')^T\mathbf{x} > 0$ for all $\mathbf{x} \in X_1$.
- For the weight vector $\mathbf{w}$ that is to be constructed by the *PT* Algorithm, the two inequalities must hold as well: $\mathbf{w}^T\mathbf{x} < 0$ for all $\mathbf{x} \in X_0$, and $\mathbf{w}^T\mathbf{x} > 0$ for all $\mathbf{x} \in X_1$.
- Consider the set $X' = X_1 \cup \{-\mathbf{x} \mid \mathbf{x} \in X_0\}$: the searched $\mathbf{w}$ fulfills $\mathbf{w}^T\mathbf{x} > 0$ for all $\mathbf{x} \in X'$.
- The *PT* Algorithm performs a number of iterations, where $\mathbf{w}(t)$ denotes the weight vector for iteration t, which form the basis for the weight vector $\mathbf{w}(t+1)$. $\mathbf{x}(t) \in X'$ denotes the feature vector chosen in round t, and $c(\mathbf{x}(t))$ denotes the respective class label. The first (and randomly chosen) weight vector is denoted as $\mathbf{w}(0)$.
- Recall the Cauchy-Schwarz inequality: $||\mathbf{a}||^2 \cdot ||\mathbf{b}||^2 \geq (\mathbf{a}^T\mathbf{b})^2$, where $||\mathbf{x}|| := \sqrt{\mathbf{x}^T\mathbf{x}}$ denotes the Euclidean norm.

Perceptron Learning

Perceptron Convergence Theorem: Proof (continued)

Line of argument:

(a) A lower bound for the adapation of $\mathbf{w}$ can be stated. The derivation of this lower bound exploits the presupposed linear separability of X_0 and X_1, which in turn guarantees the existence of a separating hyperplane $\widehat{\mathbf{w}}$.

(b) An upper bound for the adapation of $\mathbf{w}$ can be stated. The derivation of this upper bound exploits the finiteness of X_0 and X_1, which in turn guarantees an upper bound for the norm of the maximum feature vector.

(c) Both bounds can be expressed as functions in the number of iterations n, whereas the lower bound grows faster than the upper bound. Hence, in order to fulfill the inequality, the number of iterations is finite.

Perceptron Learning

Perceptron Convergence Theorem: Proof (continued)

1. The *PT* Algorithm computes in iteration t the scalar product $\mathbf{w}(t)^T\mathbf{x}(t)$. If classified correctly, $\mathbf{w}(t)^T\mathbf{x}(t) > 0$ and $\mathbf{w}$ is unchanged. Otherwise, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$.

Perceptron Learning

Perceptron Convergence Theorem: Proof (continued)

1. The *PT* Algorithm computes in iteration t the scalar product $\mathbf{w}(t)^T\mathbf{x}(t)$. If classified correctly, $\mathbf{w}(t)^T\mathbf{x}(t) > 0$ and $\mathbf{w}$ is unchanged. Otherwise, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$.

2. Consider a sequence of n incorrectly classified feature vectors, $(x(t))$, along with the corresponding weight vector adaptation, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$:
   - $\mathbf{w}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0)$
   - $\mathbf{w}(2) = \mathbf{w}(1) + \eta \cdot \mathbf{x}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \eta \cdot \mathbf{x}(1)$
   - $\vdots$
   - $\mathbf{w}(n) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \ldots + \eta \cdot \mathbf{x}(n-1)$

Perceptron Learning

Perceptron Convergence Theorem: Proof (continued)

1. The *PT* Algorithm computes in iteration t the scalar product $\mathbf{w}(t)^T\mathbf{x}(t)$. If classified correctly, $\mathbf{w}(t)^T\mathbf{x}(t) > 0$ and $\mathbf{w}$ is unchanged. Otherwise, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$.

2. Consider a sequence of n incorrectly classified feature vectors, $(x(t))$, along with the corresponding weight vector adaptation, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$:
   - $\mathbf{w}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0)$
   - $\mathbf{w}(2) = \mathbf{w}(1) + \eta \cdot \mathbf{x}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \eta \cdot \mathbf{x}(1)$
   - $\vdots$
   - $\mathbf{w}(n) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \ldots + \eta \cdot \mathbf{x}(n-1)$

3. The hyperplane defined by $\widehat{\mathbf{w}}$ separates X_1 and X_0 : $\forall \mathbf{x} \in X' : \widehat{\mathbf{w}}^T\mathbf{x} > 0$
   Let $\delta := \min_{\mathbf{x} \in X'} \widehat{\mathbf{w}}^T\mathbf{x}$. Observe that $\delta > 0$ holds.

Perceptron Learning

Perceptron Convergence Theorem: Proof (continued)

1. The *PT* Algorithm computes in iteration t the scalar product $\mathbf{w}(t)^T\mathbf{x}(t)$. If classified correctly, $\mathbf{w}(t)^T\mathbf{x}(t) > 0$ and $\mathbf{w}$ is unchanged. Otherwise, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$.

2. Consider a sequence of n incorrectly classified feature vectors, $(x(t))$, along with the corresponding weight vector adaptation, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$:
   - $\mathbf{w}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0)$
   - $\mathbf{w}(2) = \mathbf{w}(1) + \eta \cdot \mathbf{x}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \eta \cdot \mathbf{x}(1)$
   - $\vdots$
   - $\mathbf{w}(n) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \ldots + \eta \cdot \mathbf{x}(n-1)$

3. The hyperplane defined by $\widehat{\mathbf{w}}$ separates X_1 and X_0 : $\forall \mathbf{x} \in X' : \widehat{\mathbf{w}}^T\mathbf{x} > 0$
   Let $\delta := \min_{\mathbf{x} \in X'} \widehat{\mathbf{w}}^T\mathbf{x}$. Observe that $\delta > 0$ holds.

4. Analyze the scalar product between $\mathbf{w}(n)$ and $\widehat{\mathbf{w}}$:
   $$\widehat{\mathbf{w}}^T\mathbf{w}(n) = \widehat{\mathbf{w}}^T\mathbf{w}(0) + \eta \cdot \widehat{\mathbf{w}}^T\mathbf{x}(0) + \ldots + \eta \cdot \widehat{\mathbf{w}}^T\mathbf{x}(n-1)$$
   $$\Rightarrow \quad \widehat{\mathbf{w}}^T\mathbf{w}(n) \geq \widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta \geq 0 \quad \text{for } n \geq n_0 \text{ with sufficiently large } n_0 \in \mathbf{N}$$
   $$\Rightarrow \quad (\widehat{\mathbf{w}}^T\mathbf{w}(n))^2 \geq (\widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta)^2$$

5. Apply the Cauchy-Schwarz inequality:
   $$||\widehat{\mathbf{w}}||^2 \cdot ||\mathbf{w}(n)||^2 \;\geq\; (\widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta)^2 \quad \Rightarrow \quad ||\mathbf{w}(n)||^2 \;\geq\; \frac{(\widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta)^2}{||\widehat{\mathbf{w}}||^2}$$

Perceptron Learning

Perceptron Convergence Theorem: Proof (continued)

6. Consider again the weight adaptation $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$:

$$
\begin{aligned}
||\mathbf{w}(t+1)||^2 &= ||\mathbf{w}(t) + \eta \cdot \mathbf{x}(t)||^2 \\
&= (\mathbf{w}(t) + \eta \cdot \mathbf{x}(t))^T (\mathbf{w}(t) + \eta \cdot \mathbf{x}(t)) \\
&= \mathbf{w}(t)^T \mathbf{w}(t) + \eta^2 \cdot \mathbf{x}(t)^T \mathbf{x}(t) + 2\eta \cdot \mathbf{w}(t)^T \mathbf{x}(t) \\
&\leq ||\mathbf{w}(t)||^2 + ||\eta \cdot \mathbf{x}(t)||^2, \quad \text{since } \mathbf{w}(t)^T \mathbf{x}(t) < 0
\end{aligned}
$$

Perceptron Learning

Perceptron Convergence Theorem: Proof (continued)

6. Consider again the weight adaptation $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$:

$$\begin{aligned}
||\mathbf{w}(t+1)||^2 &= ||\mathbf{w}(t) + \eta \cdot \mathbf{x}(t)||^2 \\
&= (\mathbf{w}(t) + \eta \cdot \mathbf{x}(t))^T (\mathbf{w}(t) + \eta \cdot \mathbf{x}(t)) \\
&= \mathbf{w}(t)^T \mathbf{w}(t) + \eta^2 \cdot \mathbf{x}(t)^T \mathbf{x}(t) + 2\eta \cdot \mathbf{w}(t)^T \mathbf{x}(t) \\
&\leq ||\mathbf{w}(t)||^2 + ||\eta \cdot \mathbf{x}(t)||^2, \quad \text{since } \mathbf{w}(t)^T \mathbf{x}(t) < 0
\end{aligned}$$

7. Consider a sequence of n weight adaptations:

$$\begin{aligned}
||\mathbf{w}(n)||^2 &\leq ||\mathbf{w}(n-1)||^2 + ||\eta \cdot \mathbf{x}(n-1)||^2 \\
&\leq ||\mathbf{w}(n-2)||^2 + ||\eta \cdot \mathbf{x}(n-2)||^2 + ||\eta \cdot \mathbf{x}(n-1)||^2 \\
&\leq ||\mathbf{w}(0)||^2 + ||\eta \cdot \mathbf{x}(0)||^2 + \ldots + ||\eta \cdot \mathbf{x}(n-1)||^2 \\
&= ||\mathbf{w}(0)||^2 + \sum_{i=0}^{n-1} ||\eta \cdot \mathbf{x}(i)||^2
\end{aligned}$$

8. With $\varepsilon := \max_{\mathbf{x} \in X'} ||\mathbf{x}||^2$ follows $||\mathbf{w}(n)||^2 \leq ||\mathbf{w}(0)||^2 + n\eta^2\varepsilon$

Perceptron Learning

Perceptron Convergence Theorem: Proof (continued)

9. Both inequalities must be fulfilled:

$$||\mathbf{w}(n)||^2 \geq \frac{(\widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta)^2}{||\widehat{\mathbf{w}}||^2} \quad \text{and} \quad ||\mathbf{w}(n)||^2 \leq ||\mathbf{w}(0)||^2 + n\eta^2\varepsilon, \text{ hence}$$

$$\frac{(\widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta)^2}{||\widehat{\mathbf{w}}||^2} \leq ||\mathbf{w}(n)||^2 \leq ||\mathbf{w}(0)||^2 + n\eta^2\varepsilon$$

10. Observe:

$$\frac{(\widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta)^2}{||\widehat{\mathbf{w}}||^2} \in \Theta(n^2) \quad \text{and} \quad ||\mathbf{w}(0)||^2 + n\eta^2\varepsilon \in \Theta(n)$$

Perceptron Learning

Perceptron Convergence Theorem: Proof (continued)

9. Both inequalities must be fulfilled:

$$||\mathbf{w}(n)||^2 \geq \frac{(\widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta)^2}{||\widehat{\mathbf{w}}||^2} \quad \text{and} \quad ||\mathbf{w}(n)||^2 \leq ||\mathbf{w}(0)||^2 + n\eta^2\varepsilon, \text{ hence}$$

$$\frac{(\widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta)^2}{||\widehat{\mathbf{w}}||^2} \leq ||\mathbf{w}(n)||^2 \leq ||\mathbf{w}(0)||^2 + n\eta^2\varepsilon$$

10. Observe:

$$\frac{(\widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta)^2}{||\widehat{\mathbf{w}}||^2} \in \Theta(n^2) \quad \text{and} \quad ||\mathbf{w}(0)||^2 + n\eta^2\varepsilon \in \Theta(n)$$

11. An upper bound for n:

$$\frac{(\widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta)^2}{||\widehat{\mathbf{w}}||^2} \leq ||\mathbf{w}(0)||^2 + n\eta^2\varepsilon$$

For $\mathbf{w}(0) = \mathbf{0}$ (set all initial weights to zero) follows:

$$0 < n \leq \frac{\varepsilon}{\delta^2}||\widehat{\mathbf{w}}||^2$$

➜ The *PT Algorithm* terminates within a finite number of iterations.

Perceptron Learning

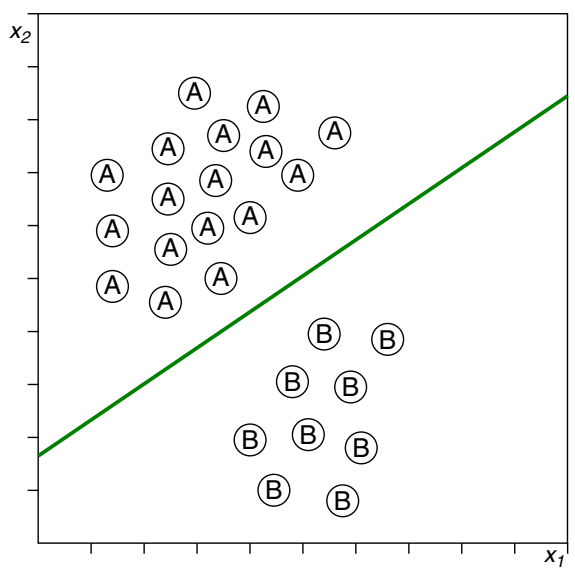
Perceptron Convergence Theorem: Discussion

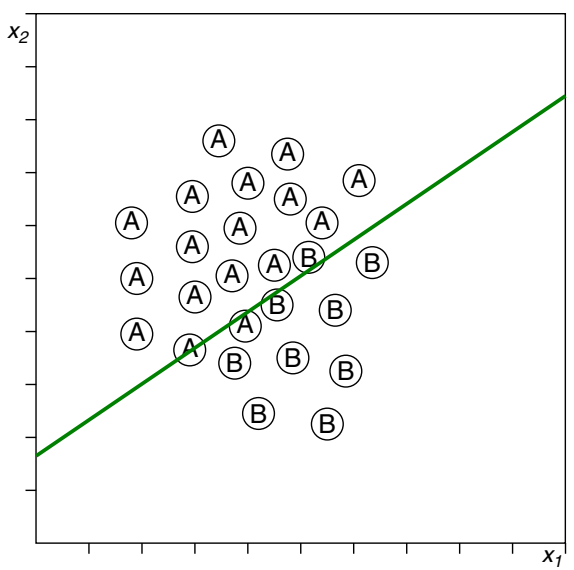
- If a separating hyperplane between X_0 and X_1 exists, the *PT* Algorithm will converge. If no such hyperplane exists, convergence cannot be guaranteed.
- A separating hyperplane can be found in polynomial time with linear programming. The *PT* Algorithm, however, may require an exponential number of iterations.

Perceptron Learning

Perceptron Convergence Theorem: Discussion

- If a separating hyperplane between X_0 and X_1 exists, the *PT* Algorithm will converge. If no such hyperplane exists, convergence cannot be guaranteed.
- A separating hyperplane can be found in polynomial time with linear programming. The *PT* Algorithm, however, may require an exponential number of iterations.
- Classification problems with noise (right-hand side) are problematic:





Gradient Descend

Classification Error

Gradient descend considers the true error (better: the hyperplane distance) and will converge even if X_1 and X_0 cannot be separated by a hyperplane. However, this convergence process is of asymptotic nature for which no finite iteration number can be estimated.

Gradient descend is based on the so-called delta rule, which in turn forms the basis of the backpropagation algorithm.

Gradient Descend

Classification Error

Gradient descend considers the true error (better: the hyperplane distance) and will converge even if X_1 and X_0 cannot be separated by a hyperplane. However, this convergence process is of asymptotic nature for which no finite iteration number can be estimated.

Gradient descend is based on the so-called delta rule, which in turn forms the basis of the backpropagation algorithm.

Consider the linear perceptron *without* a threshold function:

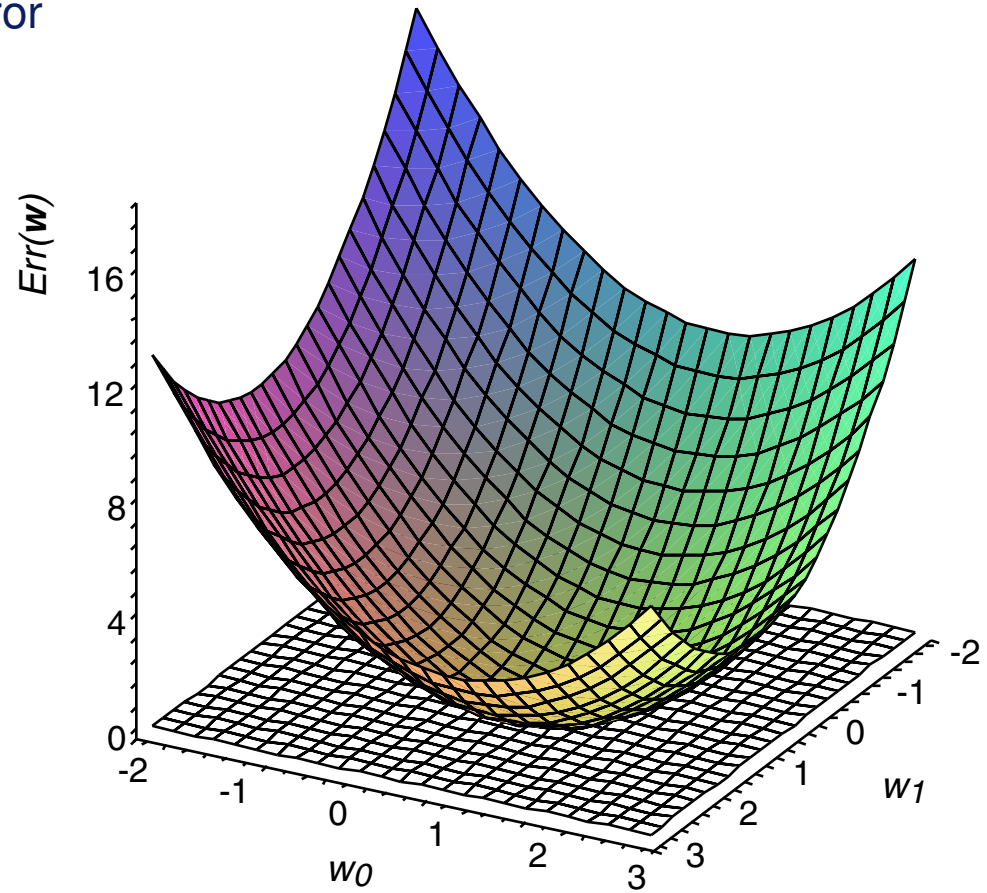
$$y(\mathbf{x}) \;=\; \mathbf{w}^T\mathbf{x} \;=\; \sum_{i=0}^{p} w_i x_i \qquad \text{[Heaviside]}$$

The classification error $\mathit{Err}(\mathbf{w})$ of a weight vector (= hypothesis) $\mathbf{w}$ with regard to D can be defined as follows:

$$\mathit{Err}(\mathbf{w}) \;=\; \frac{1}{2} \sum_{(\mathbf{x},c(\mathbf{x}))\in D} \left(c(\mathbf{x}) - y(\mathbf{x})\right)^2$$

Gradient Descend

Classification Error



The gradient $\nabla Err(\mathbf{w})$ of $Err(\mathbf{w})$ defines the steepest ascend or descend:

$$\nabla Err(\mathbf{w}) = \left(\frac{\partial Err(\mathbf{w})}{\partial w_0}, \frac{\partial Err(\mathbf{w})}{\partial w_1}, \cdots, \frac{\partial Err(\mathbf{w})}{\partial w_p} \right)$$

Gradient Descend

Weight Adaptation [*PT* Algorithm]

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w} \quad \text{mit} \quad \Delta\mathbf{w} = -\eta \nabla \mathit{Err}(\mathbf{w})$$

Componentwise (dimension-wise) weight adaptation:

$$w_i \leftarrow w_i + \Delta w_i \quad \text{mit} \quad \Delta w_i = -\eta \frac{\partial \mathit{Err}(\mathbf{w})}{\partial w_i}$$

Gradient Descend

Weight Adaptation [*PT* Algorithm]

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w} \quad \text{mit} \quad \Delta\mathbf{w} = -\eta\nabla \mathit{Err}(\mathbf{w})$$

Componentwise (dimension-wise) weight adaptation:

$$w_i \leftarrow w_i + \Delta w_i \quad \text{mit} \quad \Delta w_i = -\eta\frac{\partial \mathit{Err}(\mathbf{w})}{\partial w_i}$$

$$\begin{aligned}
\frac{\partial \mathit{Err}(\mathbf{w})}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{(\mathbf{x},c(\mathbf{x}))\in D}(c(\mathbf{x})-y(\mathbf{x}))^2 = \frac{1}{2}\sum_{(\mathbf{x},c(\mathbf{x}))\in D}\frac{\partial}{\partial w_i}(c(\mathbf{x})-y(\mathbf{x}))^2 \\
&= \frac{1}{2}\sum_{(\mathbf{x},c(\mathbf{x}))\in D} 2(c(\mathbf{x})-y(\mathbf{x}))\frac{\partial}{\partial w_i}(c(\mathbf{x})-y(\mathbf{x})) \\
&= \sum_{(\mathbf{x},c(\mathbf{x}))\in D}(c(\mathbf{x})-\mathbf{w}^T\mathbf{x})\frac{\partial}{\partial w_i}(c(\mathbf{x})-\mathbf{w}^T\mathbf{x}) \\
&= \sum_{(\mathbf{x},c(\mathbf{x}))\in D}(c(\mathbf{x})-\mathbf{w}^T\mathbf{x})(-x_i)
\end{aligned}$$

Gradient Descend

Weight Adaptation [*PT* Algorithm]

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w} \quad \text{mit} \quad \Delta\mathbf{w} = -\eta\nabla \mathit{Err}(\mathbf{w})$$

Componentwise (dimension-wise) weight adaptation:

$$w_i \leftarrow w_i + \Delta w_i \quad \text{mit} \quad \Delta w_i = -\eta\frac{\partial \mathit{Err}(\mathbf{w})}{\partial w_i} = \eta \sum_{(\mathbf{x},c(\mathbf{x}))\in D} (c(\mathbf{x}) - \mathbf{w}^T\mathbf{x}) \cdot x_i$$

$$\begin{aligned}
\frac{\partial \mathit{Err}(\mathbf{w})}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{(\mathbf{x},c(\mathbf{x}))\in D}(c(\mathbf{x}) - y(\mathbf{x}))^2 = \frac{1}{2}\sum_{(\mathbf{x},c(\mathbf{x}))\in D}\frac{\partial}{\partial w_i}(c(\mathbf{x}) - y(\mathbf{x}))^2 \\
&= \frac{1}{2}\sum_{(\mathbf{x},c(\mathbf{x}))\in D} 2(c(\mathbf{x}) - y(\mathbf{x}))\frac{\partial}{\partial w_i}(c(\mathbf{x}) - y(\mathbf{x})) \\
&= \sum_{(\mathbf{x},c(\mathbf{x}))\in D} (c(\mathbf{x}) - \mathbf{w}^T\mathbf{x})\frac{\partial}{\partial w_i}(c(\mathbf{x}) - \mathbf{w}^T\mathbf{x}) \\
&= \sum_{(\mathbf{x},c(\mathbf{x}))\in D} (c(\mathbf{x}) - \mathbf{w}^T\mathbf{x})(-x_i)
\end{aligned}$$

Gradient Descend

Weight Adaptation: Batch Gradient Descent [*IGD* Algorithm]

Algorithm:	*BGD*	Batch Gradient Descent
Input:	D	Training examples of the form $(\mathbf{x}, c(\mathbf{x}))$ with $\|\mathbf{x}\| = p+1,\ c(\mathbf{x}) \in \{0,1\}$.
	η	Learning rate, a small positive constant.
Output:	$\mathbf{w}$	Weight vector.

BGD(D, η)

1. *initialize_random_weights*$(\mathbf{w})$, $t = 0$
2. **REPEAT**
3. $t = t + 1$
4. **FOR** $i = 0$ **TO** p **DO** $\Delta w_i = 0$
5. FOREACH $(\mathbf{x}, c(\mathbf{x})) \in D$ DO
6. *error* $= c(\mathbf{x}) - \mathbf{w}^T\mathbf{x}$
7. **FOR** $i = 0$ **TO** p **DO** $\Delta w_i = \Delta w_i + \eta \cdot$ *error* $\cdot\, x_i$
8. ENDDO
9. **FOR** $i = 0$ **TO** p **DO** $w_i = w_i + \Delta w_i$
10. **UNTIL**(*convergence*$(D, y(D))$ **OR** $t > t_{\text{max}}$)
11. *return*$(\mathbf{w})$

Gradient Descend

Weight Adaptation: Delta Rule

The weight adaptation in the *BGD* Algorithm is set-based: before modifying a weight component in $\mathbf{w}$ the total error of *all* examples (the "batch") is computed.

Weight adaptation with regard to a *single* example $(\mathbf{x}, c(\mathbf{x})) \in D$:

$$\Delta w_i \;=\; \eta \cdot (c(\mathbf{x}) - \mathbf{w}^T\mathbf{x}) \cdot x_i$$

This adaptation rule is known under different names:

- delta rule
- Widrow-Hoff rule
- adaline rule

The classification error $Err_d(\mathbf{w})$ of a weight vector (= hypothesis) $\mathbf{w}$ with regard to a single example $d \in D$, $d = (\mathbf{x}, c(\mathbf{x}))$, is given as:

$$Err_d(\mathbf{w}) \;=\; \frac{1}{2}(c(\mathbf{x}) - \mathbf{w}^T\mathbf{x})^2$$

Gradient Descend

Weight Adaptation: Incremental Gradient Descent [*BGD* Algorithm]

Algorithm:	*IGD*	Incremental Gradient Descent
Input:	D	Training examples of the form $(\mathbf{x}, c(\mathbf{x}))$ with $\|\mathbf{x}\| = p+1,\ c(\mathbf{x}) \in \{0,1\}$.
	η	Learning rate, a small positive constant.
Output:	$\mathbf{w}$	Weight vector.

IGD(D, η)

1. *initialize_random_weights*$(\mathbf{w})$, $t = 0$
2. **REPEAT**
3. $t = t + 1$
4. FOREACH $(\mathbf{x}, c(\mathbf{x})) \in D$ DO
5. *error* $= c(\mathbf{x}) - \mathbf{w}^T\mathbf{x}$
6. **FOR** $i = 0$ **TO** p **DO**
7. $\Delta w_i = \eta \cdot$ *error* $\cdot\, x_i$
   $w_i = w_i + \Delta w_i$
8. **ENDDO**
9. ENDDO
10. **UNTIL**(*convergence*$(D, y(D))$ **OR** $t > t_{\text{max}}$)
11. *return*$(\mathbf{w})$

Remarks:

- Incremental gradient descent is also called *stochastic* gradient descent.
- The classification error *Err* of incremental gradient descent is specific for each training example $d \in D$, $d = (\mathbf{x}, c(\mathbf{x}))$: $Err_d(\mathbf{w}) = \frac{1}{2}(c(\mathbf{x}) - \mathbf{w}^T\mathbf{x})^2$
- The sequence of incremental weight adaptations approximates the gradient descent of the batch approach. If η is chosen sufficiently small, this approximation can happen at arbitrary accuracy.
- The computation of the total error of batch gradient descend enables larger weight adaptation increments.
- Compared to batch gradient descend, the example-based weight adaptation of incremental gradient descend can better avoid getting stuck in a local minimum of the error function.

Chapter ML:VI (continued)

VI. Neural Networks

- Perceptron Learning
- Gradient Descend
- Multilayer Perceptron
- Radial Basis Functions

Multilayer Perceptron

Definition 1 (Linear Separability)

Two sets of feature vectors, X_0, X_1, of a p-dimensional feature space are called linearly separable, if $p+1$ real numbers, $\theta, w_1, \ldots, w_p$, exist such that holds:

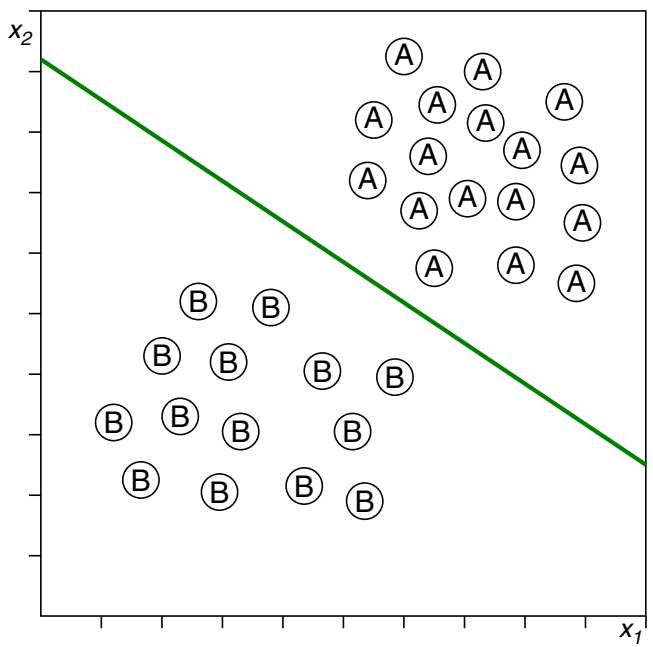
1. $\forall \mathbf{x} \in X_0$: $\sum_{i=1}^{p} w_i x_i < \theta$
2. $\forall \mathbf{x} \in X_1$: $\sum_{i=1}^{p} w_i x_i \geq \theta$

Multilayer Perceptron

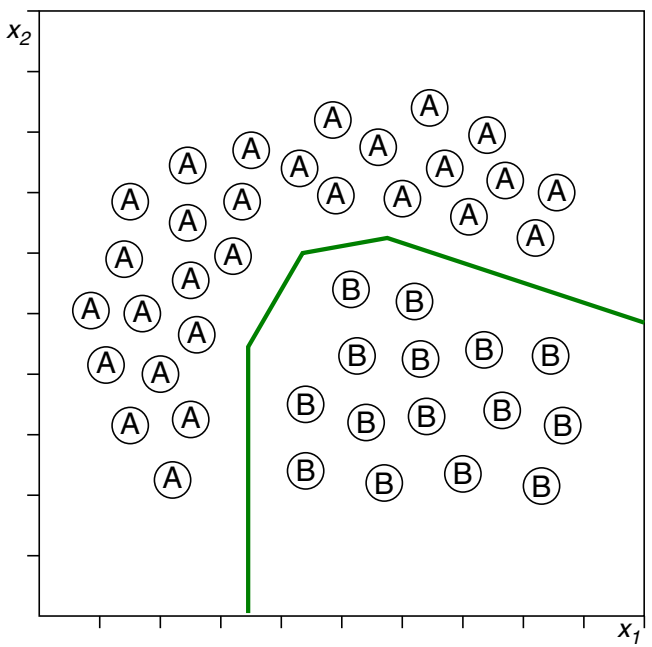
Definition 1 (Linear Separability)

Two sets of feature vectors, X_0, X_1, of a p-dimensional feature space are called linearly separable, if $p+1$ real numbers, $\theta, w_1, \ldots, w_p$, exist such that holds:

1. $\forall \mathbf{x} \in X_0$: $\sum_{i=1}^{p} w_i x_i < \theta$
2. $\forall \mathbf{x} \in X_1$: $\sum_{i=1}^{p} w_i x_i \geq \theta$



linearly separable



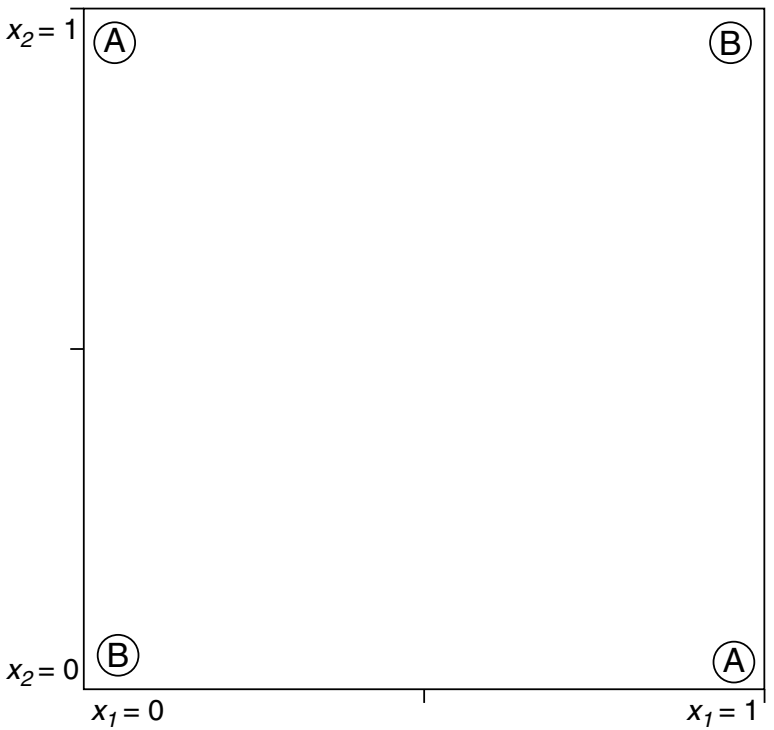
not linearly separable

Multilayer Perceptron

Separability

The *XOR* function defines the smallest example for two not linearly separable sets:

x_1	x_2	XOR	Class
0	0	0	B
1	0	1	A
0	1	1	A
1	1	0	B

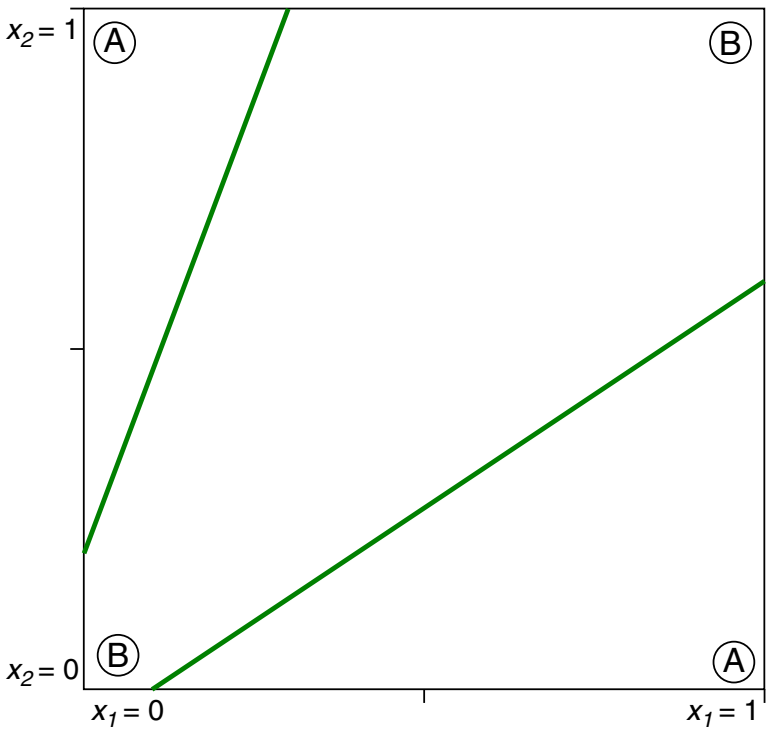


Multilayer Perceptron

Separability (continued)

The *XOR* function defines the smallest example for two not linearly separable sets:

x_1	x_2	XOR	Class
0	0	0	B
1	0	1	A
0	1	1	A
1	1	0	B



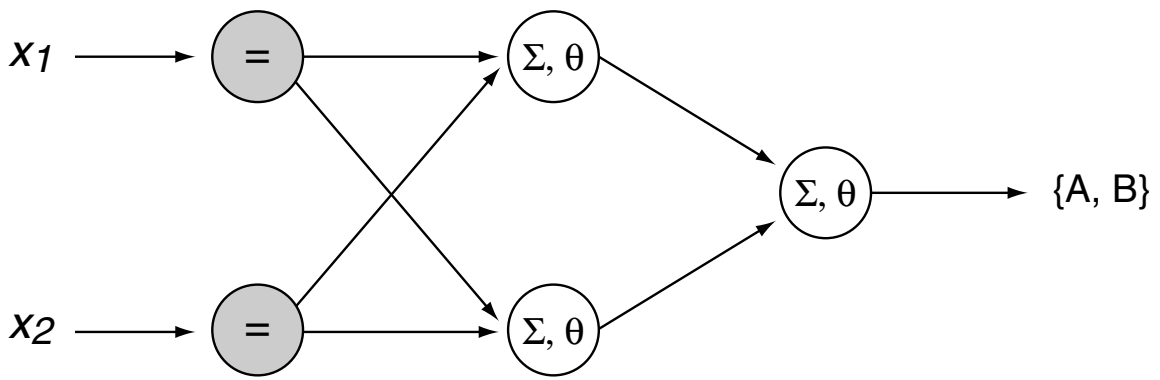
- ➜ specification of several hyperplanes
- ➜ combination of several perceptrons

Multilayer Perceptron

Separability (continued)

Layered combination of several perceptrons: the multilayer perceptron.

Minimum multilayer perceptron that is able to handle the *XOR* problem:



Remarks:

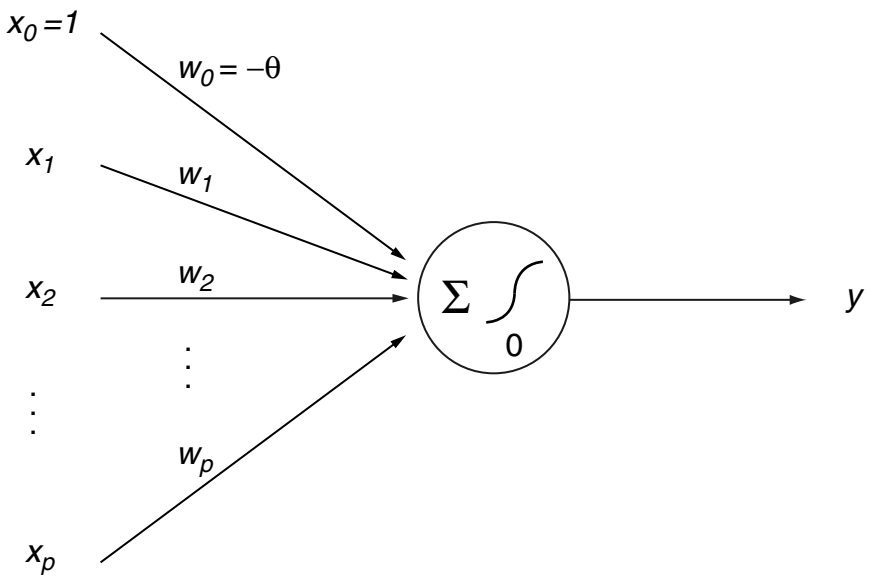
- The multilayer perceptron was presented by Rumelhart and McClelland in 1986. Earlier, but unnoticed, was a similar research work of Werbos and Parker [1974, 1982].
- Compared to a single perceptron the multilayer perceptron poses a significantly more challenging training (= learning) problem, which requires continuous threshold functions and sophisticated learning strategies.
- Marvin Minsky and Seymour Papert showed 1969 with the *XOR* problem the limitations of single perceptrons. Moreover, they assumed that extensions of the perceptron architecture (such as the multilayer perceptron) would be similarly limited as a single perceptron. A fatal mistake. In fact, they brought the research in this field to a halt that lasted 17 years. [Berkeley]

[Marvin Minsky]

Multilayer Perceptron

Computation in the Network [Heaviside]

A perceptron with a continuous, non-linear threshold function:



The sigmoid function $\sigma(z)$ as threshold function:

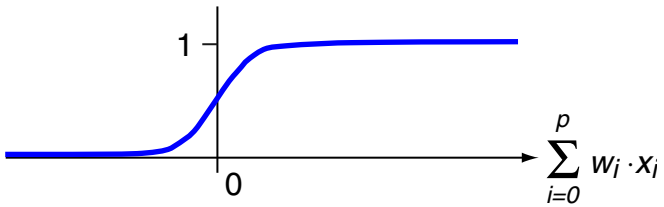
$$\sigma(z) = \frac{1}{1+e^{-z}} \quad \text{where} \quad \frac{d\sigma(z)}{dz} = \sigma(z) \cdot (1 - \sigma(z))$$

Multilayer Perceptron

Computation in the Network (continued)

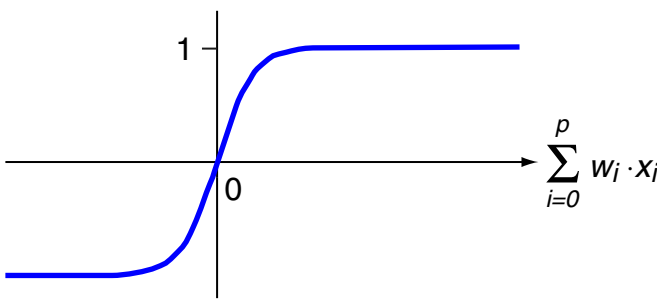
Computation of the perceptron output $y(\mathbf{x})$ via the sigmoid function σ:

$$y(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$



An alternative to the sigmoid function is the $\tanh$ function:

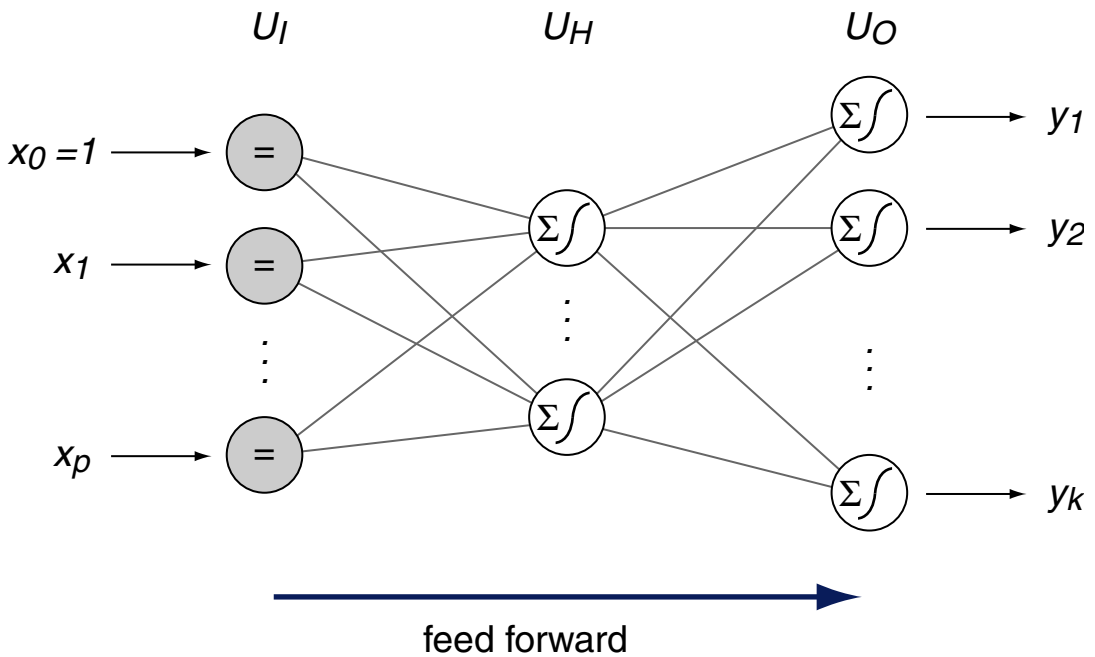
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$



Multilayer Perceptron

Computation in the Network (continued)

Distinguish units (nodes, perceptrons) of type input, hidden, and output:



U_I, U_H, U_O	Sets with units of type input, hidden, and output
$x_{u \to v}$	Input value for unit v, provided via output of unit u
w_{uv}, Δw_{uv}	Weight and weight adaptation for edge connecting units u and v
δ_u	Classification error of unit u
y_u	Output value of unit u

Remarks:

- The units of the input layer, U_I, perform no computations at all. They distribute the input values to the next layer.
- The non-linear characteristic of the sigmoid function make networks possible that approximate every function. To achieve this flexibility, only three active layers are required, i.e., two layers with hidden units and one layer with output units. Keywords: universal approximator, [Kolmogorov Theorem, 1957]
- The generic delta rule allows for a backpropagagtion of the classification error and hence the training of multi-layered networks.
- Gradient descend is based on the classification error that considers the entire network (network weight vector).
- Multilayer perceptrons are also called multilayer networks or (artificial) neural networks, abbreviated as ANN.

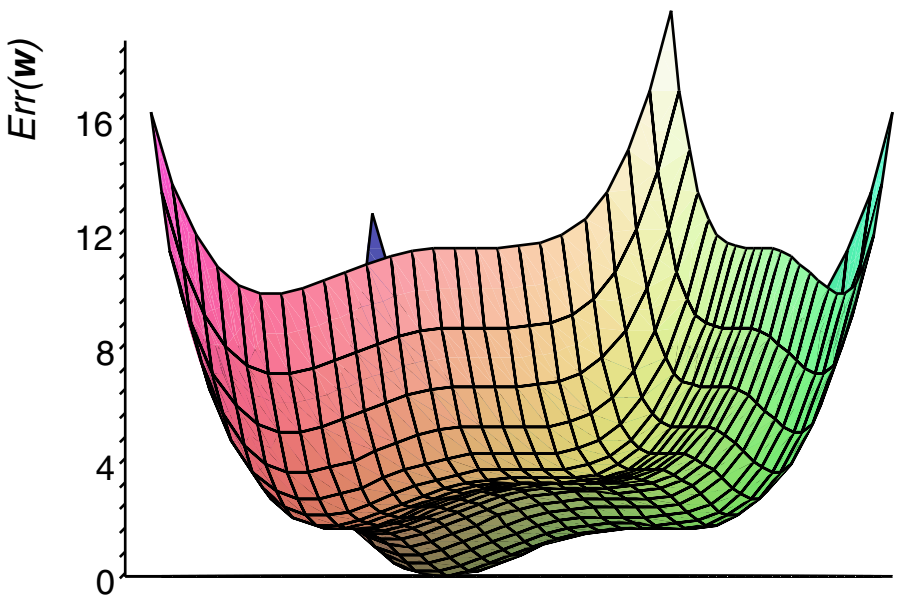
Multilayer Perceptron

Classification Error

The classification error $Err(\mathbf{w})$ is computed as sum over the $|U_O| = k$ network outputs:

$$Err(\mathbf{w}) = \frac{1}{2} \sum_{(\mathbf{x},\mathbf{c}(\mathbf{x}))\in D} \sum_{v\in U_O} (c_v(\mathbf{x}) - y_v(\mathbf{x}))^2$$

Due its complex form, $Err(\mathbf{w})$ may contain various local minima:



Multilayer Perceptron

Weight Adaptation: Incremental Gradient Descent [network]

Algorithm:	*MPT*	Multilayer Perceptron Training
Input:	D	Training examples of the form $(\mathbf{x}, c(\mathbf{x}))$ with $\|\mathbf{x}\| = p+1,\ c(\mathbf{x}) \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$\mathbf{w}$	Weights for the units in U_I, U_H, U_O.

```
1.  initialize_random_weights(U_I, U_H, U_O), t = 0
2.  REPEAT
3.    t = t + 1
4.    FOREACH (x, c(x)) ∈ D DO
5.      FOREACH u ∈ U_H DO y_u = propagate(x, u)  // layer 1
6.      FOREACH v ∈ U_O DO y_v = propagate(y_H, v)  // layer 2
7.
8.
9.
10.
11.
12.
13.   ENDDO
14. UNTIL(convergence(D, y_O(D)) OR t > t_max)
15. return(w)
```

Multilayer Perceptron

Weight Adaptation: Incremental Gradient Descent [network]

Algorithm:	*MPT*	Multilayer Perceptron Training
Input:	D	Training examples of the form $(\mathbf{x}, c(\mathbf{x}))$ with $\|\mathbf{x}\| = p+1,\ c(\mathbf{x}) \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$\mathbf{w}$	Weights for the units in U_I, U_H, U_O.

1. *initialize_random_weights*(U_I, U_H, U_O), $t = 0$
2. **REPEAT**
3. $t = t + 1$
4. FOREACH $(\mathbf{x}, \mathbf{c}(\mathbf{x})) \in D$ DO
5. **FOREACH** $u \in U_H$ **DO** $y_u =$ *propagate*$(\mathbf{x}, u)$ // layer 1
6. **FOREACH** $v \in U_O$ **DO** $y_v =$ *propagate*$(\mathbf{y}_H, v)$ // layer 2
7. FOREACH $v \in U_O$ DO $\delta_v = y_v \cdot (1 - y_v) \cdot (\mathbf{c}_v(\mathbf{x}) - y_v)$ // backpropagate layer 2
8. FOREACH $u \in U_H$ DO $\delta_u = y_u \cdot (1 - y_u) \sum_{v \in U_o} w_{uv} \cdot \delta_v$ // backpropagate layer 1
9.
10.
11.
12.
13. ENDDO
14. **UNTIL**(*convergence*$(D, y_O(D))$ **OR** $t > t_{\max}$)
15. *return*$(\mathbf{w})$

Multilayer Perceptron

Weight Adaptation: Incremental Gradient Descent [network]

Algorithm:	*MPT*	Multilayer Perceptron Training
Input:	D	Training examples of the form $(\mathbf{x}, c(\mathbf{x}))$ with $\|\mathbf{x}\| = p+1,\ c(\mathbf{x}) \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$\mathbf{w}$	Weights for the units in U_I, U_H, U_O.

1. *initialize_random_weights*(U_I, U_H, U_O), $t = 0$
2. **REPEAT**
3. $t = t + 1$
4. FOREACH $(\mathbf{x}, \mathbf{c}(\mathbf{x})) \in D$ DO
5. **FOREACH** $u \in U_H$ **DO** $y_u =$ *propagate*$(\mathbf{x}, u)$ // layer 1
6. **FOREACH** $v \in U_O$ **DO** $y_v =$ *propagate*$(\mathbf{y}_H, v)$ // layer 2
7. FOREACH $v \in U_O$ DO $\delta_v = y_v \cdot (1 - y_v) \cdot (\mathbf{c}_v(\mathbf{x}) - y_v)$ // backpropagate layer 2
8. FOREACH $u \in U_H$ DO $\delta_u = y_u \cdot (1 - y_u) \sum_{v \in U_o} w_{uv} \cdot \delta_v$ // backpropagate layer 1
9. **FOREACH** w_{ij}, $(i,j) \in (U_I \times U_H) \cup (U_H \times U_O)$ **DO**
10. $\Delta w_{ij} = \eta \cdot \delta_j \cdot x_{i \to j}$
11. $w_{ij} = w_{ij} + \Delta w_{ij}$
12. **ENDDO**
13. ENDDO
14. **UNTIL**(*convergence*$(D, y_O(D))$ **OR** $t > t_{\max}$)
15. *return*$(\mathbf{w})$

Multilayer Perceptron

Weight Adaptation: Momentum Term

Momentum idea: a weight adaptation in iteration t considers the adaptation in iteration $t-1$:

$$\Delta w_{uv}(t) = \eta \cdot \delta_v \cdot x_{u \to v} + \alpha \cdot \Delta w_{uv}(t-1)$$

The term α, $0 \leq \alpha < 1$, is called “momentum”.

Multilayer Perceptron

Weight Adaptation: Momentum Term

Momentum idea: a weight adaptation in iteration t considers the adaptation in iteration $t-1$:

$$\Delta w_{uv}(t) = \eta \cdot \delta_v \cdot x_{u \to v} + \alpha \cdot \Delta w_{uv}(t-1)$$

The term α, $0 \leq \alpha < 1$, is called “momentum”.

Effects:

- due the “adaptation inertia” local minima can be overcome
- if the direction of the descend does not change, the adaptation increment and, as a consequence, the speed of convergence is increased.

Neural Networks

Additional Sources on the Web

Application and implementation:

- JNNS. Java Neural Network Simulator.
  http://www-ra.informatik.uni-tuebingen.de/software/JavaNNS
- SNNS. Stuttgart Neural Network Simulator.
  http://www-ra.informatik.uni-tuebingen.de/software/snns