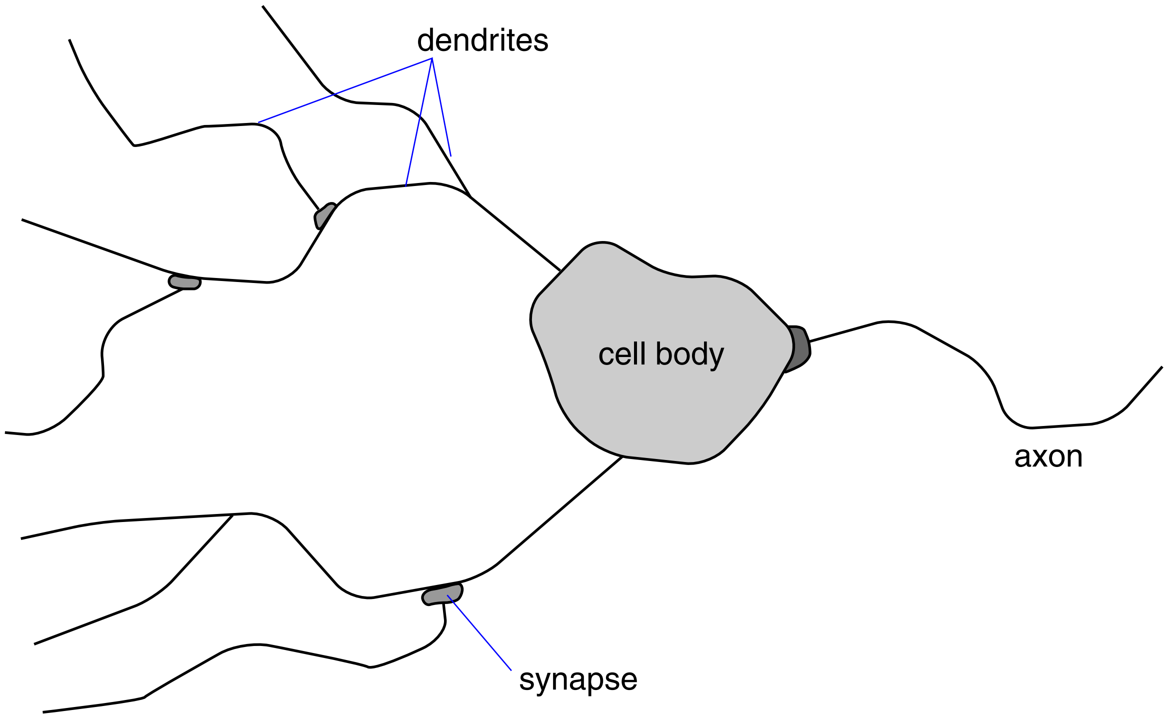
Chapter ML:VI

VI. Neural Networks

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, where 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. (those 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

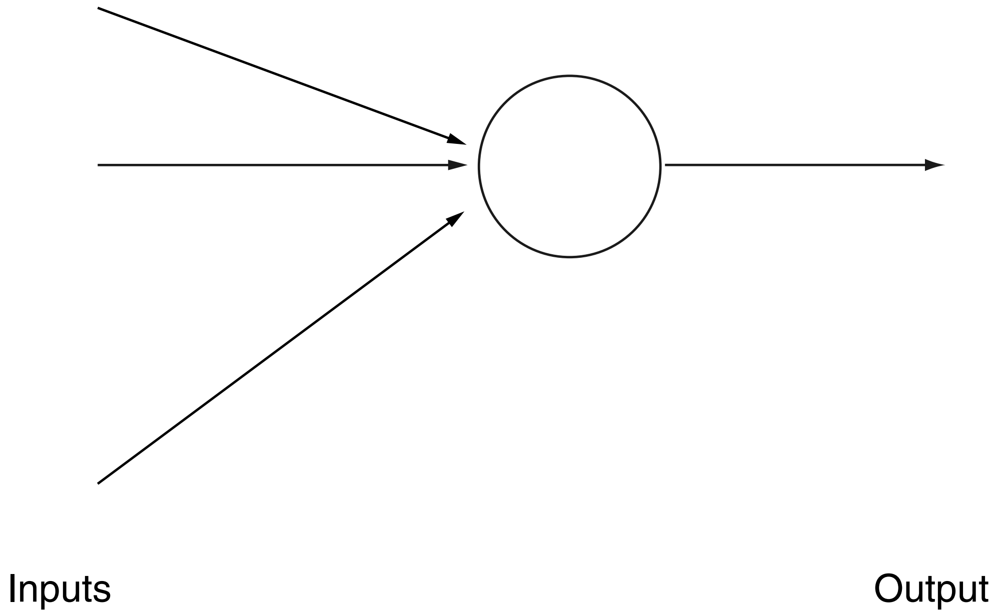
⋮ ANN research paused.

1985

1986 David Rumelhart and James McClelland present the multilayer perceptron.

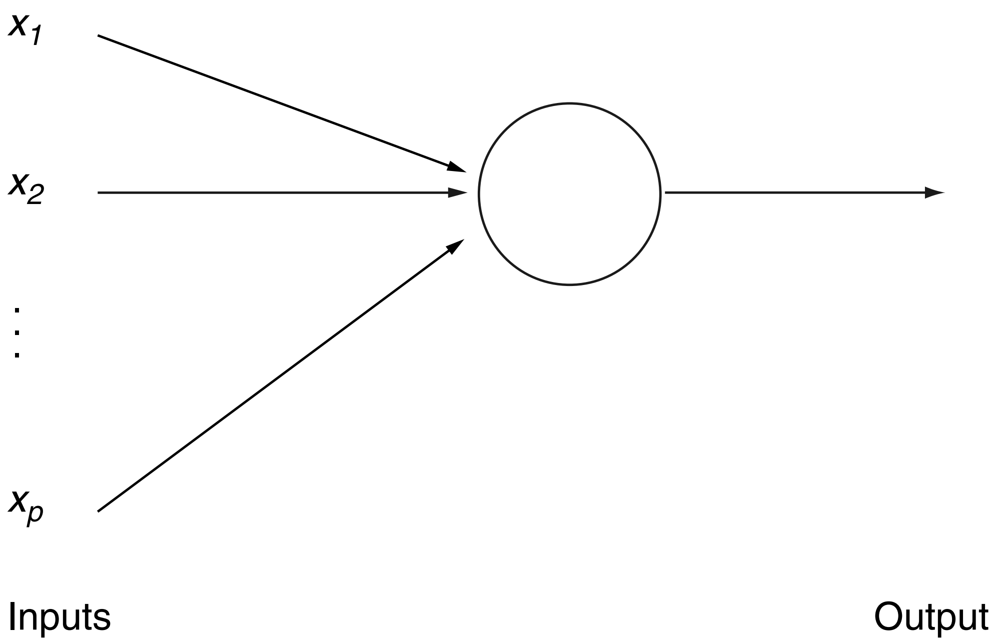
Perceptron Learning

The Perceptron of Rosenblatt [1958]



Perceptron Learning

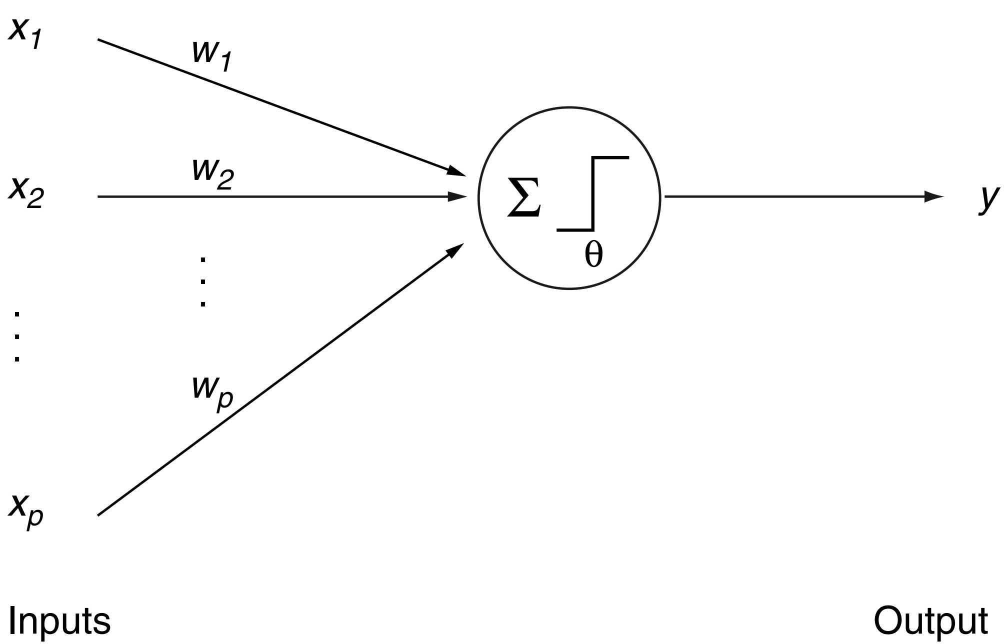
The Perceptron of Rosenblatt [1958]



$x_j, w_j \in \mathbf{R}, \quad j = 1 \ldots p$

Perceptron Learning

The Perceptron of Rosenblatt [1958]



$x_j, w_j \in \mathbf{R}, \quad j = 1 \dots p$

Remarks:

- The perceptron of Rosenblatt is based on the neuron model of [McCulloch/Pitts 1943].
- 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. $X \subseteq \mathbf{R}^p$
- $C = \{0, 1\}$ is a set of classes. $C = \{-1, 1\}$ in the regression setting.
- $c : X \to C$ is the ideal classifier for X. c is approximated by y (perceptron).
- $D = \{(\mathbf{x}_1, c(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, c(\mathbf{x}_n))\} \subseteq X \times C$ is a set of examples.

How could the hypothesis space H look like?

Perceptron Learning

Computation in the Perceptron [Regression]

If $\displaystyle\sum_{j=1}^{p} w_j x_j \geq \theta$ then $y(\mathbf{x}) = 1$, and

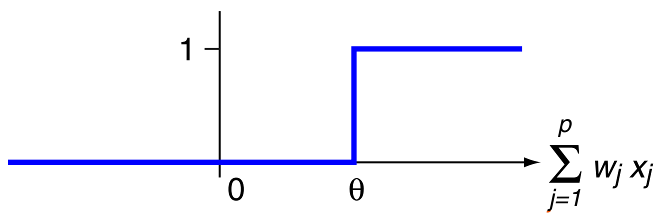
if $\displaystyle\sum_{j=1}^{p} w_j x_j < \theta$ then $y(\mathbf{x}) = 0$.

Perceptron Learning

Computation in the Perceptron [Regression]

If $\sum_{j=1}^{p} w_j x_j \geq \theta$ then $y(\mathbf{x}) = 1$, and

if $\sum_{j=1}^{p} w_j x_j < \theta$ then $y(\mathbf{x}) = 0$.



where $\sum_{j=1}^{p} w_j x_j = \mathbf{w}^T \mathbf{x}$. (or other notations for the scalar product)

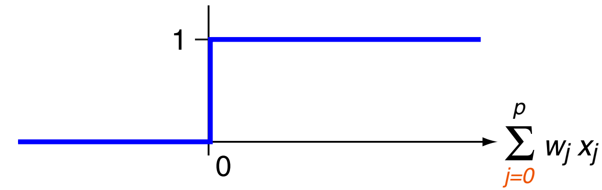
➜ A hypothesis is determined by $\theta, w_1, \ldots, w_p$.

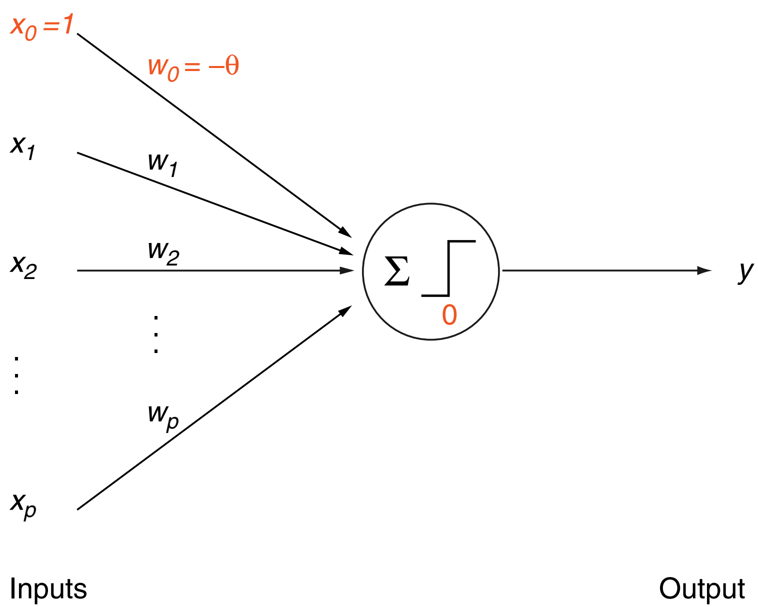
Perceptron Learning

Computation in the Perceptron (continued)

$$y(\mathbf{x}) = \textit{heaviside}\Big(\sum_{j=1}^{p} w_j x_j - \theta\Big)$$

$$= \textit{heaviside}\Big(\sum_{j=0}^{p} w_j x_j\Big) \quad \text{with } w_0 = -\theta,\ x_0 = 1$$





➜ A hypothesis is 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.
- Be careful with regard to the dimensionality of the weight vector: it is always denoted as $\mathbf{w}$ here, regardless of the fact whether the w_0-dimension, with $w_0 = -\theta$, is included.
- The function *heaviside* is named after the mathematician Oliver Heaviside.
  [Heaviside: step function O. Heaviside]

Perceptron Learning

Weight Adaptation [Algorithms: BGD IGD]

Algorithm:	*PT*	Perceptron Training
Input:	D	Training examples $(\mathbf{x}, c(\mathbf{x}))$ with $\lvert\mathbf{x}\rvert = p+1,\ c(\mathbf{x}) \in \{0,1\}$.
	η	Learning rate, a small positive constant.
Internal:	$y(D)$	Set of $y(\mathbf{x})$-values computed from the elements $\mathbf{x}$ in D given some $\mathbf{w}$.
Output:	$\mathbf{w}$	Weight vector.

PT(D, η)

initialize_random_weights$(\mathbf{w})$, $t = 0$

REPEAT

    $t = t + 1$

    $(\mathbf{x}, c(\mathbf{x})) =$ *random_select*(D)

    error $= c(\mathbf{x}) -$ *heaviside*$(\mathbf{w}^T\mathbf{x})$ // $c(\mathbf{x}) \in \{0,1\}$, *heaviside* $\in \{0,1\}$, *error* $\in \{0,1,-1\}$

    $\Delta\mathbf{w} = \eta \cdot$ *error* $\cdot\, \mathbf{x}$

    $\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$

UNTIL(*convergence*$(D, y(D))$ **OR** $t > t_{\text{max}}$)

return$(\mathbf{w})$

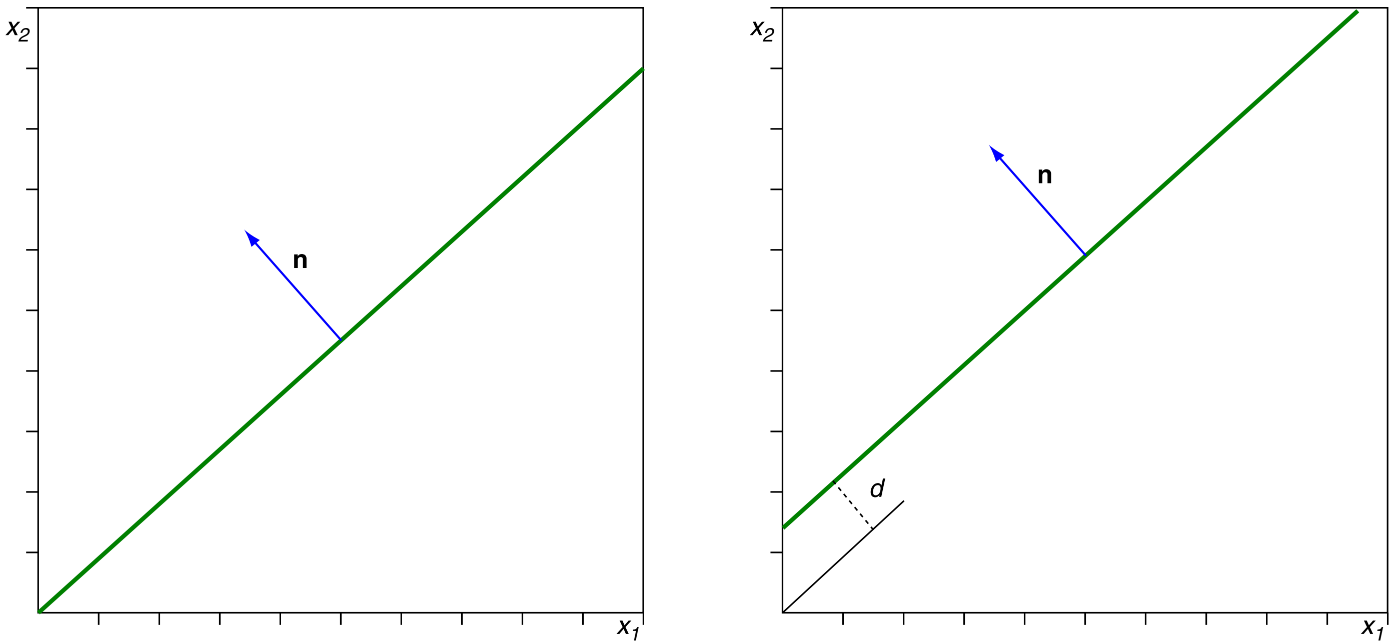
Remarks:

- The variable t denotes the time. The learning algorithm gets an example presented at each point in time and, as a consequence, may adapt the weight vector.

- 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 $\mathbf{x}$, *error* is either -1 or $+1$, regardless of the exact numeric difference between $c(\mathbf{x})$ and $\mathbf{w}^T\mathbf{x}$.

Perceptron Learning

Weight Adaptation: Illustration in Input Space

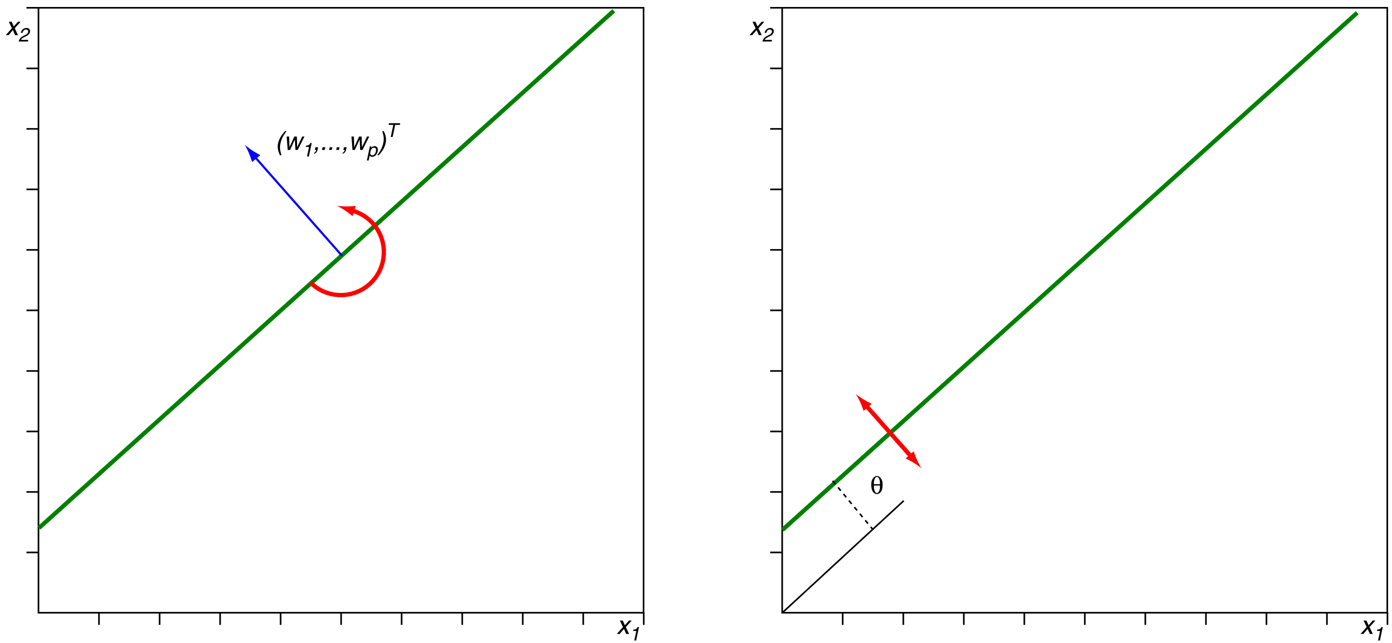


Definition of an (affine) hyperplane: $L = \left\{\mathbf{x} \mid \mathbf{n}^T\mathbf{x} = d\right\}$ [Wikipedia]

- $\mathbf{n}$ denotes a normal vector that is perpendicular to the hyperplane L.
- If $||\mathbf{n}|| = 1$ then $|\mathbf{n}^T\mathbf{x} - d|$ gives the distance of any point $\mathbf{x}$ to L.
- If $\text{sgn}(\mathbf{n}^T\mathbf{x_1} - d) = \text{sgn}(\mathbf{n}^T\mathbf{x_2} - d)$, then $\mathbf{x_1}$ and $\mathbf{x_2}$ lie on the same side of the hyperplane.

Perceptron Learning

Weight Adaptation: Illustration in Input Space (continued)



Definition of an (affine) hyperplane: $\mathbf{w}^T\mathbf{x} = 0 \Leftrightarrow \sum_{j=1}^{p} w_j x_j = \theta = -w_0$

Hyperplane definition as before, with notation taken from the classification problem setting.

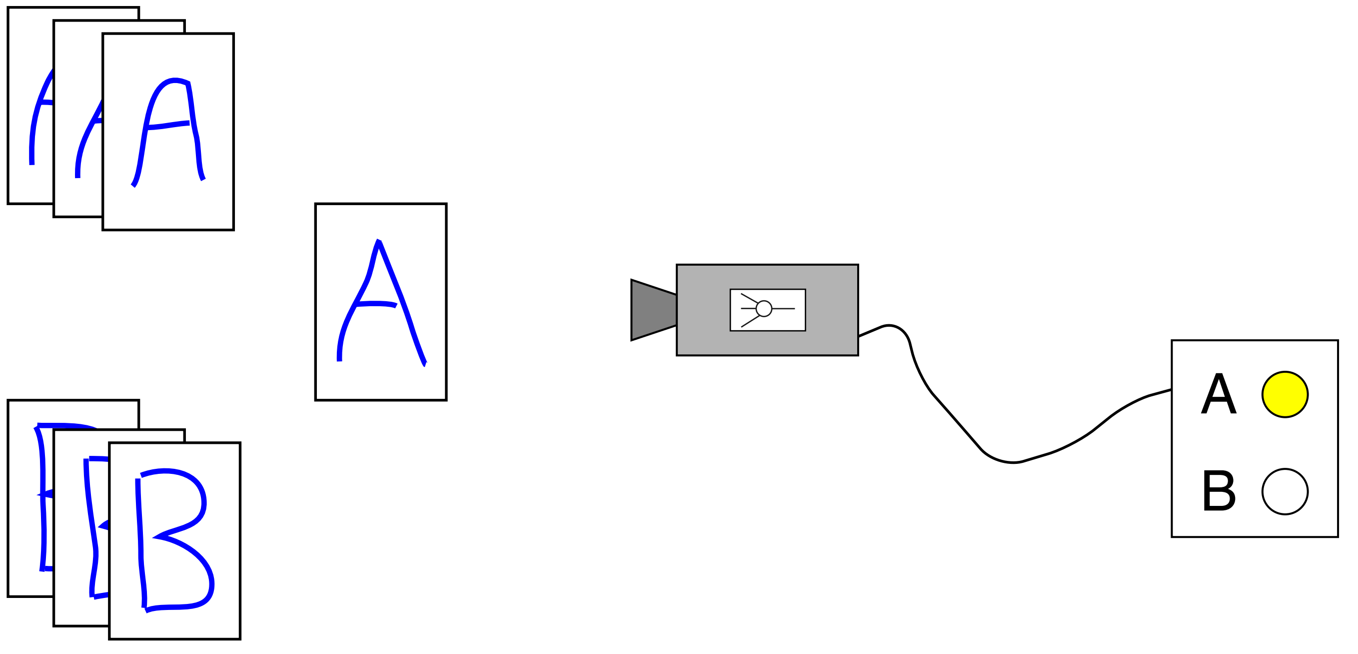
Remarks:

- 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 $(w_1, \ldots, w_p)^T$ and as multiple of $1/||(w_1, \ldots, w_p)^T||$.
- The set of possible weight vectors $\mathbf{w} = (w_0, w_1, \ldots, w_p)^T$ forms the hypothesis space H.
- Weight adaptation means learning, and the shown learning paradigm is supervised.
- For the weight adaptation in Line 6–7 of the *PT* Algorithm, note that if some x_j is zero, Δw_j will be zero as well. Keyword: Hebbian learning [Hebb 1949]
- Note that here (and in the following illustrations) the hyperplane movement is not the result of solving a regression problem in the $(p+1)$-dimensional input-output-space, where the sum of the residuals is to be minimized.

  Rather, the *PT* Algorithm takes each missclassified example $\mathbf{x}$ as an event to update the hyperplane's normal vector by a fixed amount that is proportional to $\mathbf{x}$. In particular, the update, $\Delta\mathbf{w}$, does not exploit the residual associated with $\mathbf{x}$ at time t, i.e., the absolute value of the distance of $\mathbf{x}$ from the hyperplane is disregarded.

Perceptron Learning

Example



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

Perceptron Learning

Example (continued)

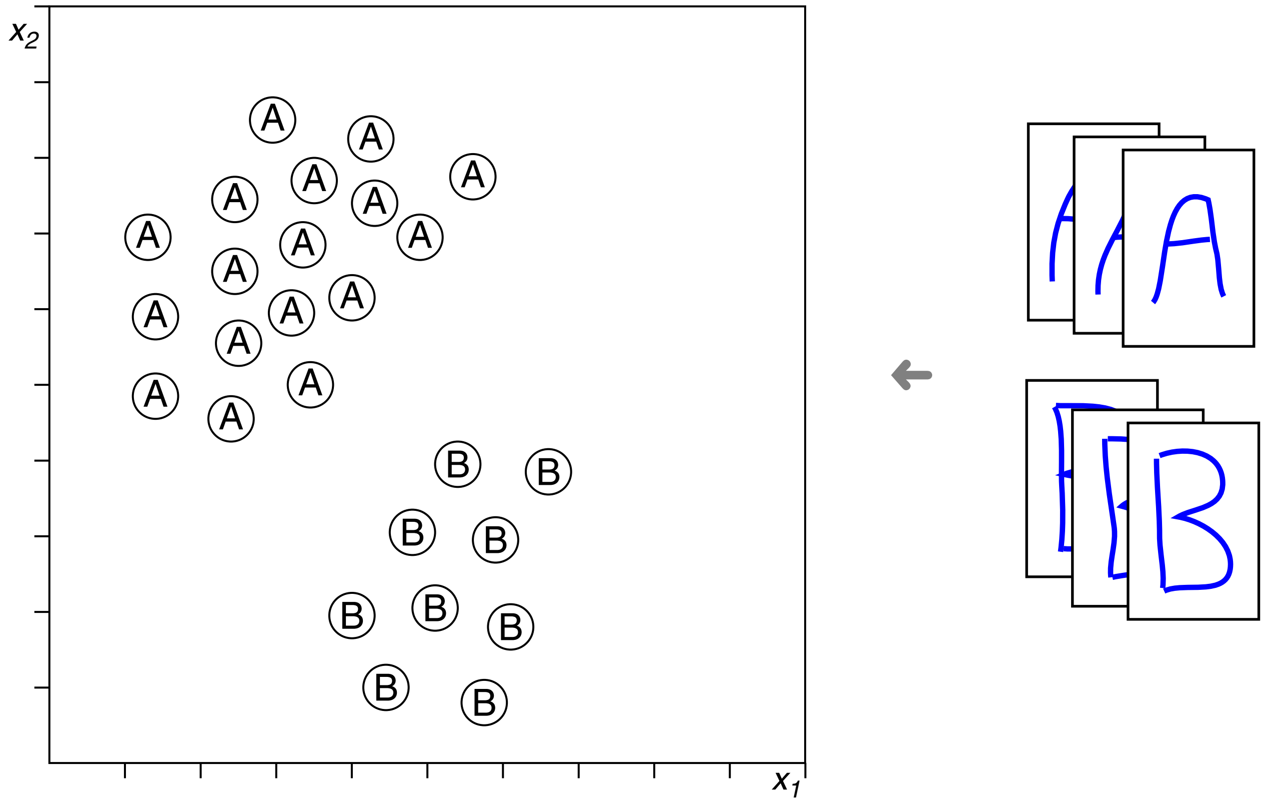
Encoding:

- The encoding of the examples is based on expressive features such as the 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.

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

Perceptron Learning

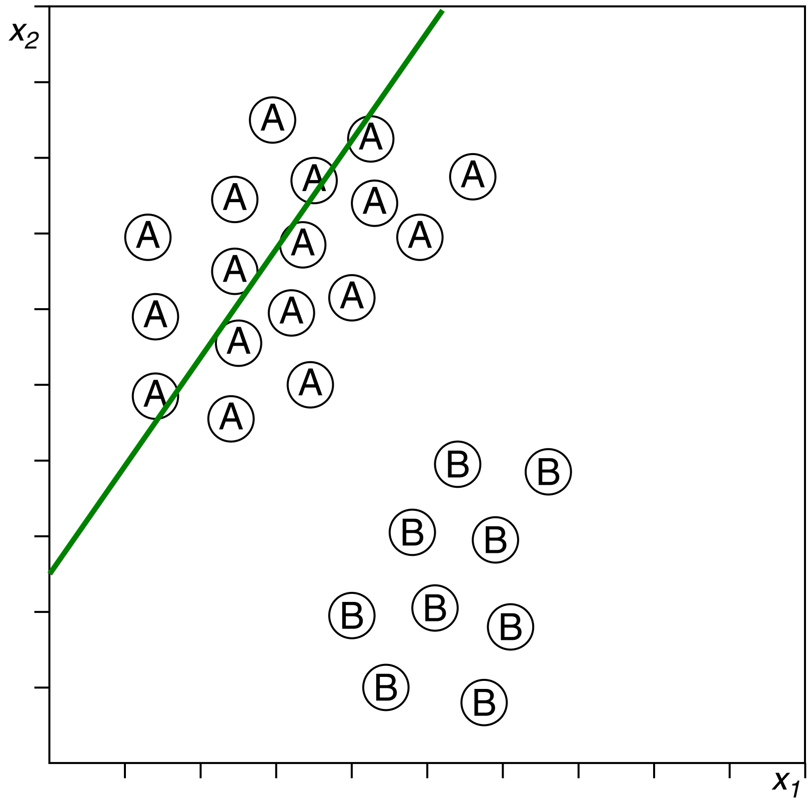
Example: Illustration in Input Space [*PT* Algorithm]



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

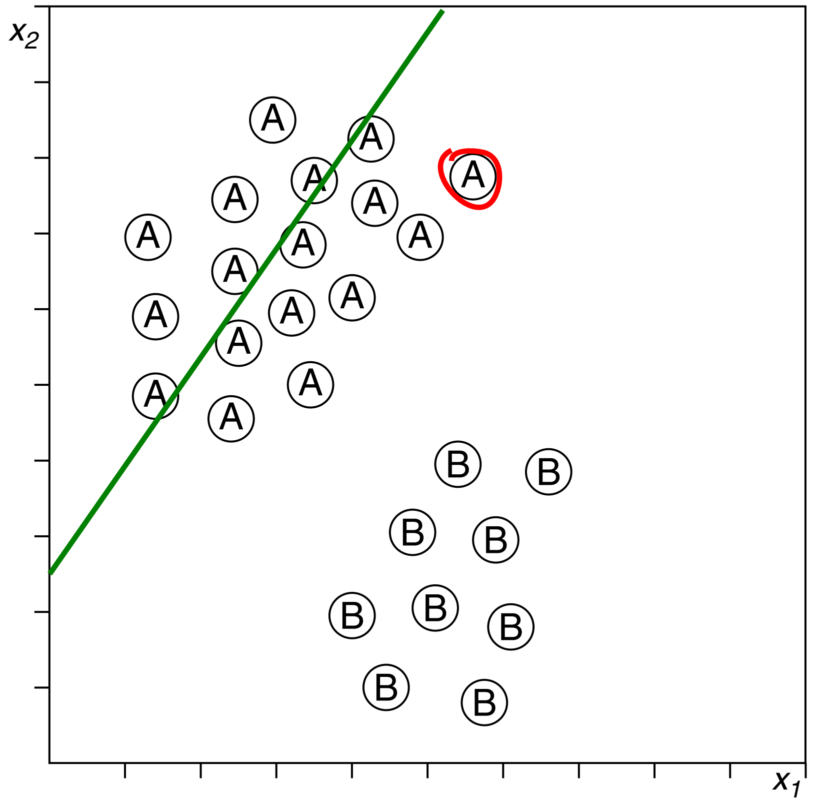
Perceptron Learning

Example: Illustration in Input Space [*PT* Algorithm]



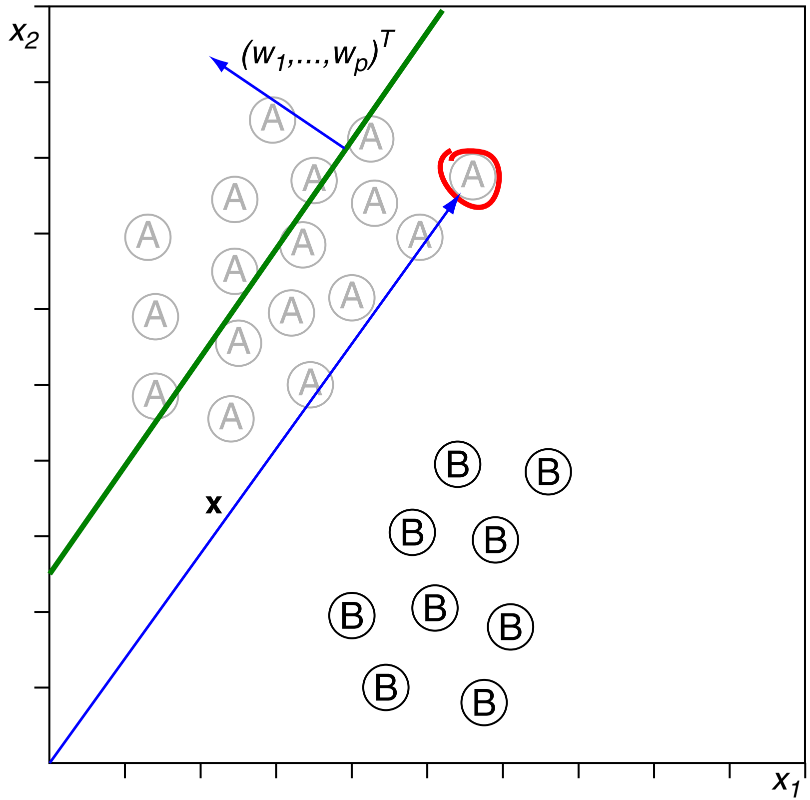
Perceptron Learning

Example: Illustration in Input Space [*PT* Algorithm]



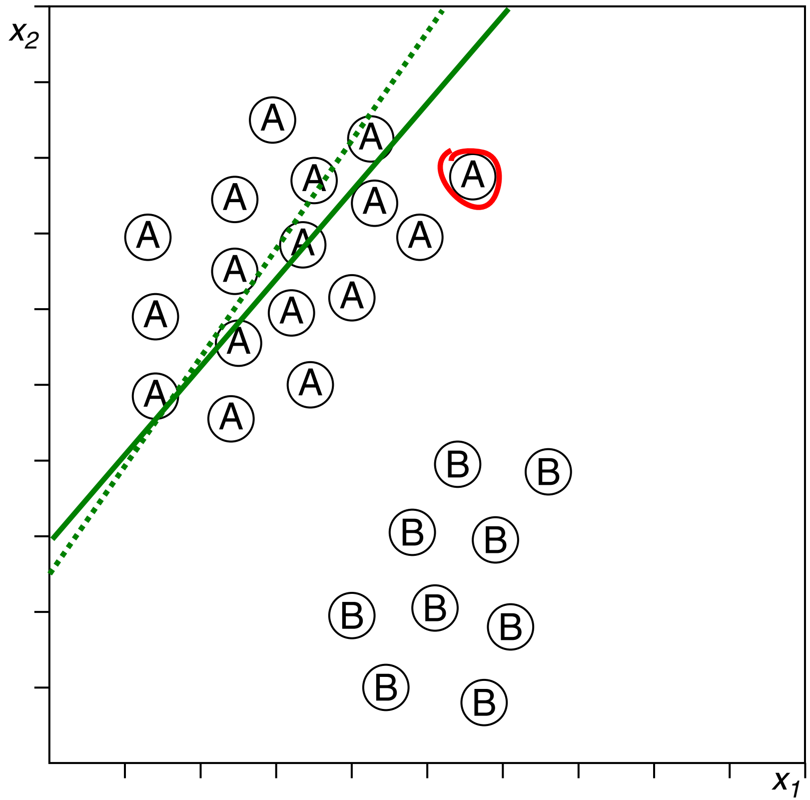
Perceptron Learning

Example: Illustration in Input Space [*PT* Algorithm]



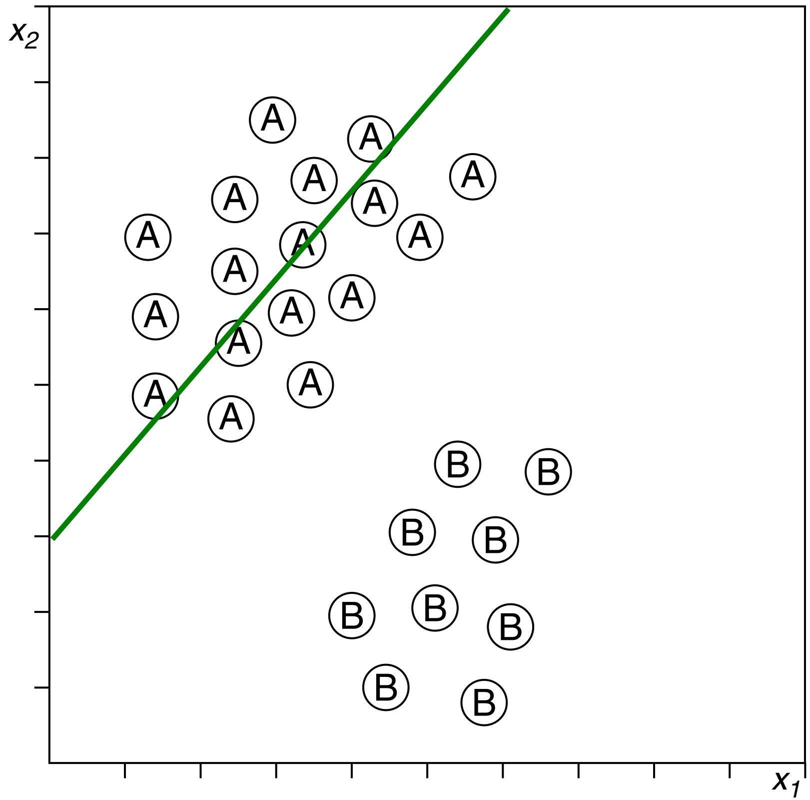
Perceptron Learning

Example: Illustration in Input Space [*PT* Algorithm]



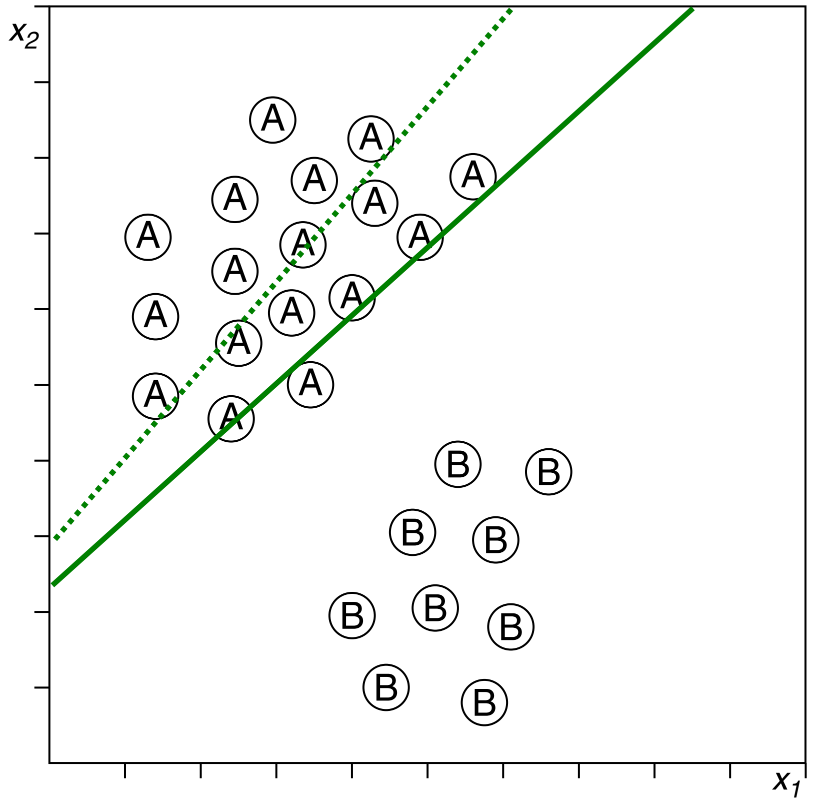
Perceptron Learning

Example: Illustration in Input Space [*PT* Algorithm]



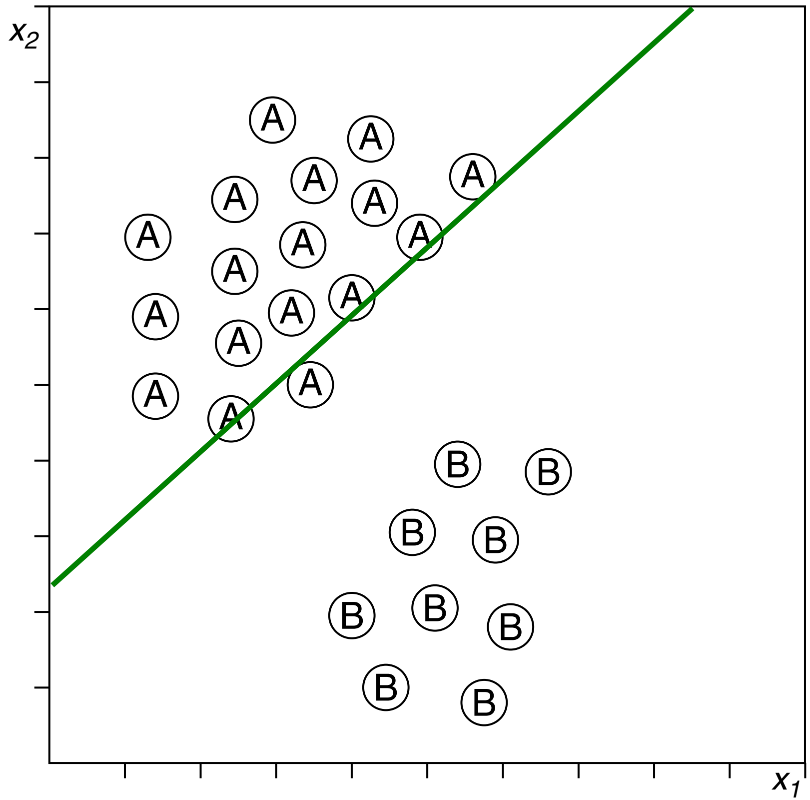
Perceptron Learning

Example: Illustration in Input Space [*PT* Algorithm]



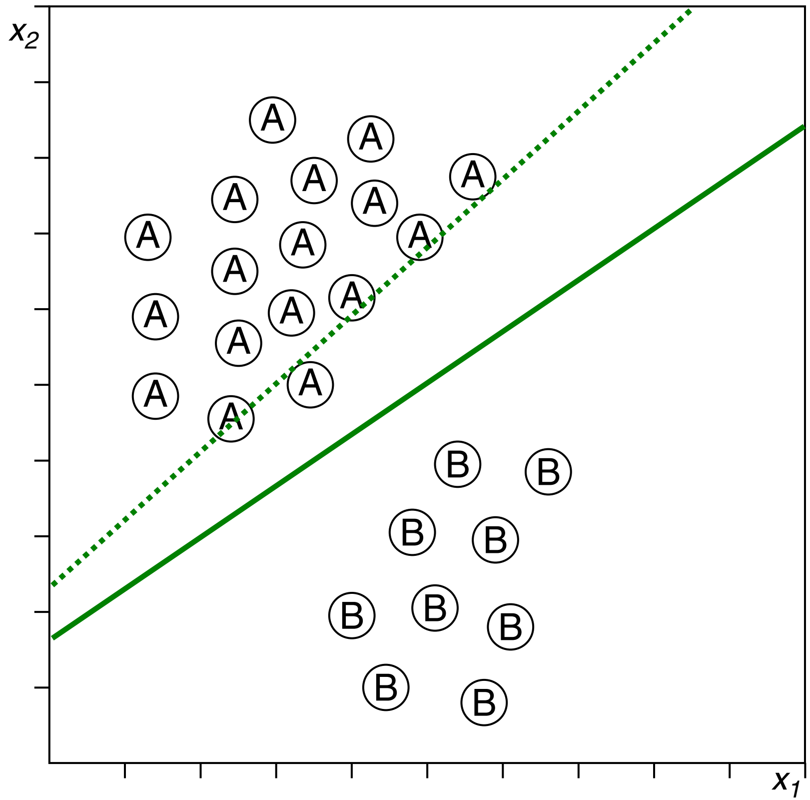
Perceptron Learning

Example: Illustration in Input Space [*PT* Algorithm]



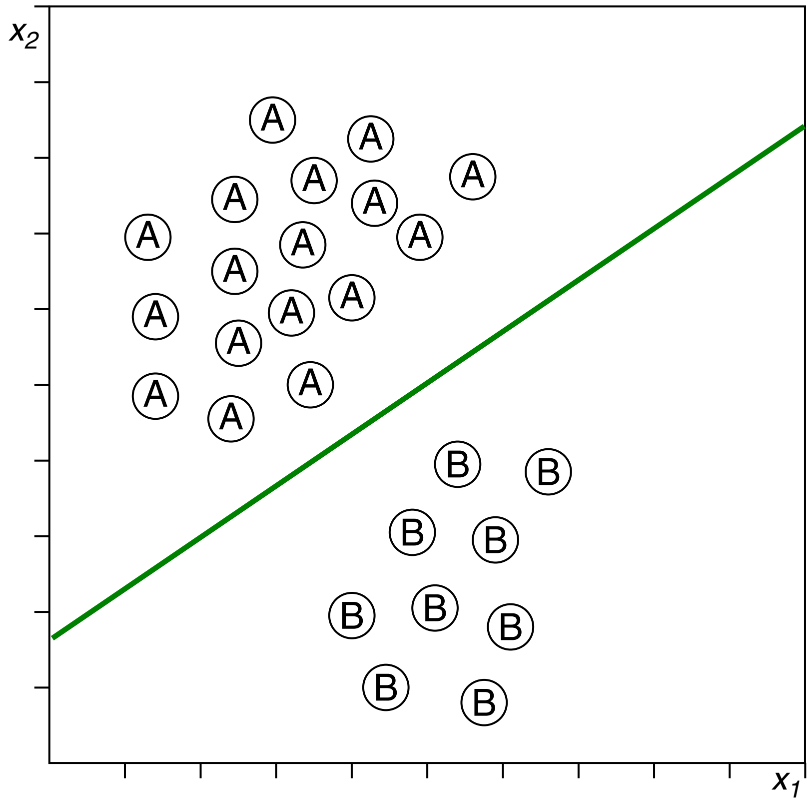
Perceptron Learning

Example: Illustration in Input Space [*PT* Algorithm]



Perceptron Learning

Example: Illustration in Input Space [*PT* Algorithm]



Perceptron Learning

Perceptron Convergence Theorem [Discussion]

Questions:

1. Which kind of learning tasks can be addressed with the functions in the hypothesis space H?

2. Can the *PT* Algorithm construct such a function for a given task?

Perceptron Convergence Theorem [Discussion]

Questions:

1. Which kind of learning tasks can be addressed with the functions in 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 constructed 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 a 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 a 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 $\mathbf{x} \in X_1$.

- By defining $X' = X_1 \cup \{-\mathbf{x} \mid \mathbf{x} \in X_0\}$, the searched $\mathbf{w}$ fulfills $\mathbf{w}^T\mathbf{x} > 0$ for all $\mathbf{x} \in X'$. Then, with $c(\mathbf{x}) = 1$ for all $\mathbf{x} \in X'$, *error* $\in \{0, 1\}$ (instead of $\{0, 1, -1\}$). [*PT* Algorithm, Line 5]

Perceptron Learning

Perceptron Convergence Theorem: Proof

Preliminaries:

- The sets X_1 and X_0 are separated by a 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 $\mathbf{x} \in X_1$.

- By defining $X' = X_1 \cup \{-\mathbf{x} \mid \mathbf{x} \in X_0\}$, the searched $\mathbf{w}$ fulfills $\mathbf{w}^T\mathbf{x} > 0$ for all $\mathbf{x} \in X'$. Then, with $c(\mathbf{x}) = 1$ for all $\mathbf{x} \in X'$, *error* $\in \{0, 1\}$ (instead of $\{0, 1, -1\}$). [*PT* Algorithm, Line 5]

- 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. The first (and randomly chosen) weight vector is denoted as $\mathbf{w}(0)$.

Perceptron Learning

Perceptron Convergence Theorem: Proof

Preliminaries:

- The sets X_1 and X_0 are separated by a 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 $\mathbf{x} \in X_1$.

- By defining $X' = X_1 \cup \{-\mathbf{x} \mid \mathbf{x} \in X_0\}$, the searched $\mathbf{w}$ fulfills $\mathbf{w}^T\mathbf{x} > 0$ for all $\mathbf{x} \in X'$. Then, with $c(\mathbf{x}) = 1$ for all $\mathbf{x} \in X'$, *error* $\in \{0, 1\}$ (instead of $\{0, 1, -1\}$). [*PT* Algorithm, Line 5]

- 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. 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) We state a lower bound for how much $||\mathbf{w}||$ must change from its initial value after n iterations (to become a separating hyperplane). The derivation of this lower bound exploits the presupposed linear separability of X_0 and X_1.

(b) We state an upper bound for how much $||\mathbf{w}||$ can change from its initial value after n iterations. The derivation of this upper bound exploits the finiteness of X_0 and X_1, which in turn guarantees the existence of an upper bound for the norm of the maximum feature vector.

(c) We observe that the lower bound grows quadratically in n, whereas the upper bound grows linearly. From the relation "lower bound $<$ upper bound" we derive a finite upper bound for n.

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, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$ [Line 5-7].

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, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$ [Line 5-7].

2. A sequence of n incorrectly classified feature vectors, $(x(t))$, along with the weight adaptation, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$, results in the series $\mathbf{w}(n)$:

$$\begin{aligned}
\mathbf{w}(1) &= \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) \\
\mathbf{w}(2) &= \mathbf{w}(1) + \eta \cdot \mathbf{x}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \eta \cdot \mathbf{x}(1) \\
&\vdots \\
\mathbf{w}(n) &= \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \ldots + \eta \cdot \mathbf{x}(n-1)
\end{aligned}$$

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, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$ [Line 5-7].

2. A sequence of n incorrectly classified feature vectors, $(x(t))$, along with the weight adaptation, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$, results in the series $\mathbf{w}(n)$:

$$
\begin{aligned}
&\mathbf{w}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) \\
&\mathbf{w}(2) = \mathbf{w}(1) + \eta \cdot \mathbf{x}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \eta \cdot \mathbf{x}(1) \\
&\quad\vdots \\
&\mathbf{w}(n) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \ldots + \eta \cdot \mathbf{x}(n-1)
\end{aligned}
$$

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, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$ [Line 5-7].

2. A sequence of n incorrectly classified feature vectors, $(x(t))$, along with the weight adaptation, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$, results in the series $\mathbf{w}(n)$:

$$\begin{aligned}
&\mathbf{w}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) \\
&\mathbf{w}(2) = \mathbf{w}(1) + \eta \cdot \mathbf{x}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \eta \cdot \mathbf{x}(1) \\
&\quad\vdots \\
&\mathbf{w}(n) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \ldots + \eta \cdot \mathbf{x}(n-1)
\end{aligned}$$

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 $\delta := \min\limits_{\mathbf{x} \in X'} \widehat{\mathbf{w}}^T\mathbf{x}$. Observe that $\delta > 0$ holds.

4. Analyze the scalar product of $\mathbf{w}(n)$ and $\widehat{\mathbf{w}}$:

$$\begin{aligned}
& \widehat{\mathbf{w}}^T\mathbf{w}(n) &&= \widehat{\mathbf{w}}^T\mathbf{w}(0) + \eta \cdot \widehat{\mathbf{w}}^T\mathbf{x}(0) + \ldots + \eta \cdot \widehat{\mathbf{w}}^T\mathbf{x}(n-1) \\
\Rightarrow\ & \widehat{\mathbf{w}}^T\mathbf{w}(n) &&\geq \widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta \geq 0 \quad (\text{for } n \geq n_0 \text{ with sufficiently large } n_0 \in \mathbf{N}) \\
\Rightarrow\ & (\widehat{\mathbf{w}}^T\mathbf{w}(n))^2 &&\geq (\widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta)^2
\end{aligned}$$

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, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$ [Line 5-7].

2. A sequence of n incorrectly classified feature vectors, $(x(t))$, along with the weight adaptation, $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot \mathbf{x}(t)$, results in the series $\mathbf{w}(n)$:

$$\begin{aligned}
&\mathbf{w}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0)\\
&\mathbf{w}(2) = \mathbf{w}(1) + \eta \cdot \mathbf{x}(1) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \eta \cdot \mathbf{x}(1)\\
&\quad\vdots\\
&\mathbf{w}(n) = \mathbf{w}(0) + \eta \cdot \mathbf{x}(0) + \ldots + \eta \cdot \mathbf{x}(n-1)
\end{aligned}$$

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 $\delta := \min\limits_{\mathbf{x}\in X'} \widehat{\mathbf{w}}^T\mathbf{x}$. Observe that $\delta > 0$ holds.

4. Analyze the scalar product of $\mathbf{w}(n)$ and $\widehat{\mathbf{w}}$:

$$\begin{aligned}
& \widehat{\mathbf{w}}^T\mathbf{w}(n) &&= \widehat{\mathbf{w}}^T\mathbf{w}(0) + \eta \cdot \widehat{\mathbf{w}}^T\mathbf{x}(0) + \ldots + \eta \cdot \widehat{\mathbf{w}}^T\mathbf{x}(n-1)\\
\Rightarrow\ & \widehat{\mathbf{w}}^T\mathbf{w}(n) &&\geq \widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta \geq 0 \quad \text{(for } n \geq n_0 \text{ with sufficiently large } n_0 \in \mathbf{N}\text{)}\\
\Rightarrow\ & (\widehat{\mathbf{w}}^T\mathbf{w}(n))^2 &&\geq (\widehat{\mathbf{w}}^T\mathbf{w}(0) + n\eta\delta)^2
\end{aligned}$$

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 the series $\mathbf{w}(n)$ from Step 2 :

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

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 the series $\mathbf{w}(n)$ from Step 2 :

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

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 (see Step 5 and Step 8) must be fulfilled:

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

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

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

Set $\mathbf{w}(0) = \mathbf{0}$:

$$\Rightarrow \quad \frac{n^2\eta^2\delta^2}{||\widehat{\mathbf{w}}||^2} \leq n\eta^2\varepsilon$$

$$\Leftrightarrow \quad n \leq \frac{\varepsilon}{\delta^2} \cdot ||\widehat{\mathbf{w}}||^2$$

Perceptron Learning

Perceptron Convergence Theorem: Proof (continued)

9. Both inequalities (see Step 5 and Step 8) must be fulfilled:

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

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

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

Set $\mathbf{w}(0) = \mathbf{0}$:

$$\Rightarrow \quad \frac{n^2\eta^2\delta^2}{||\widehat{\mathbf{w}}||^2} \leq n\eta^2\varepsilon$$

$$\Leftrightarrow \quad n \leq \frac{\varepsilon}{\delta^2} \cdot ||\widehat{\mathbf{w}}||^2$$

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

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

Perceptron Learning

Perceptron Convergence Theorem: Discussion [Theorem]

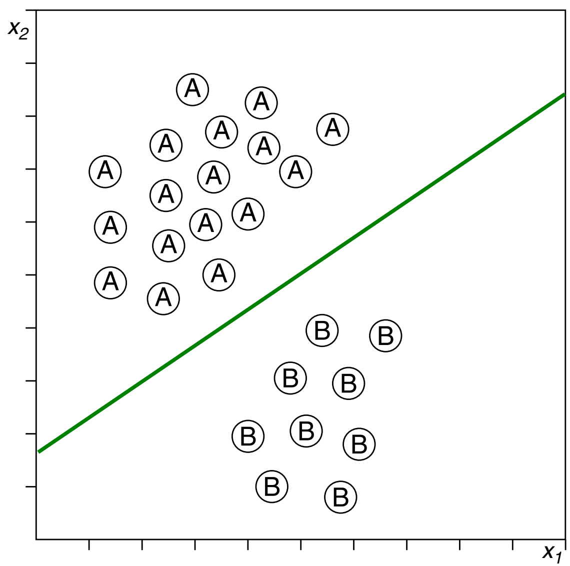
- If a separating hyperplane between X_0 and X_1 exists, the *PT* Algorithm will converge. If no such hyperplane exists, convergence cannot be guaranteed.

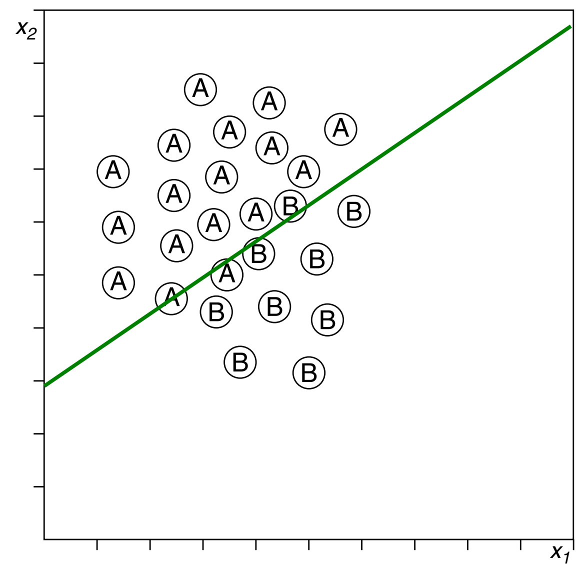
- A separating hyperplane can be found in polynomial time with linear programming. The *PT* Algorithm, however, may require an exponential number of iterations.

Perceptron Learning

Perceptron Convergence Theorem: Discussion [Theorem]

- 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 Descent
Classification Error

Gradient descent 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 an asymptotic nature and no finite iteration bound can be stated.

Gradient descent applies the so-called delta rule, which will be derived in the following. The delta rule forms the basis of the backpropagation algorithm.

Gradient Descent

Classification Error

Gradient descent 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 an asymptotic nature and no finite iteration bound can be stated.

Gradient descent applies the so-called delta rule, which will be derived in the following. The delta rule forms the basis of the backpropagation algorithm.

Consider the linear perceptron *without* a threshold function:

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \sum_{j=0}^{p} w_j x_j$$

[Heaviside]

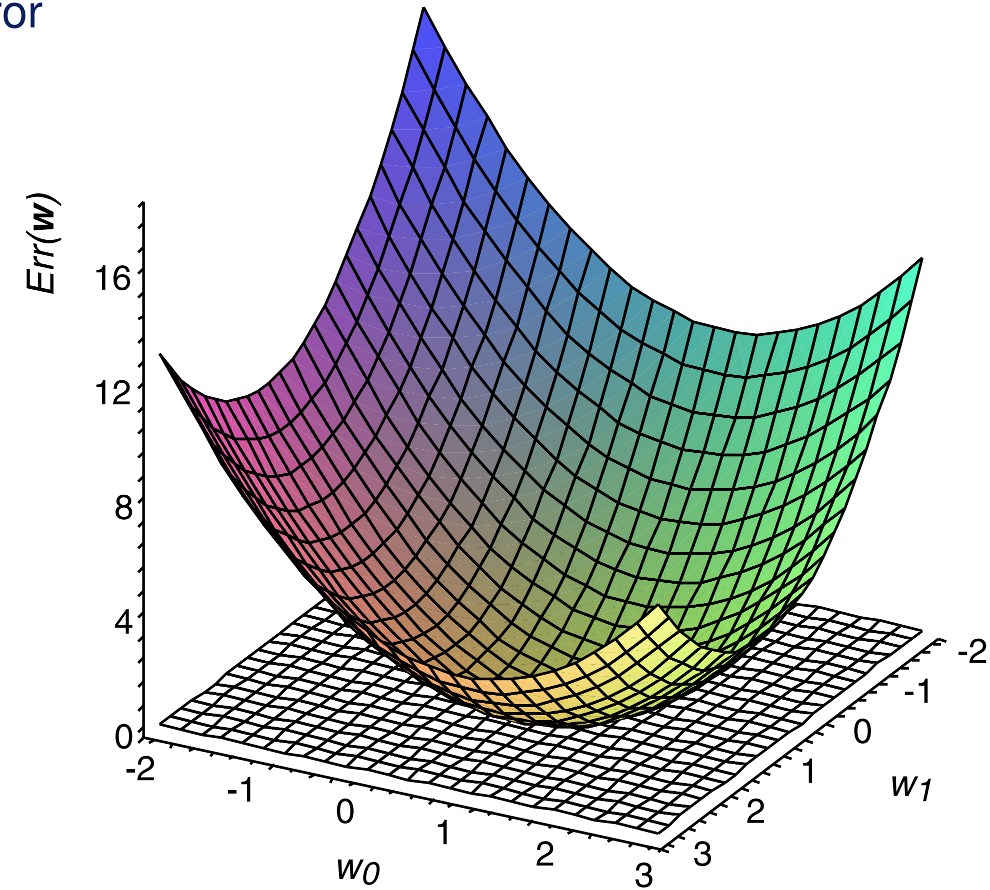
The classification error *Err*$(\mathbf{w})$ of a weight vector (= hypothesis) $\mathbf{w}$ with regard to D can be defined as follows:

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

[Singleton error]

Gradient Descent

Classification Error



The gradient of $Err(\mathbf{w})$, $\nabla Err(\mathbf{w})$, defines the steepest ascent or descent:

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

Gradient Descent

Weight Adaptation

$$\mathbf{w} \;\leftarrow\; \mathbf{w} + \Delta\mathbf{w} \quad \text{where} \quad \Delta\mathbf{w} = -\eta\nabla \mathit{Err}(\mathbf{w})$$ [*PT* Algorithm]

Componentwise ($j = 0, \ldots, p$) weight adaptation:

$$w_j \;\leftarrow\; w_j + \Delta w_j \quad \text{where} \quad \Delta w_j \;=\; -\eta \,\frac{\partial}{\partial w_j}\mathit{Err}(\mathbf{w})$$

Gradient Descent

Weight Adaptation

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w} \quad \text{where} \quad \Delta\mathbf{w} = -\eta \nabla \mathit{Err}(\mathbf{w})$$ [*PT* Algorithm]

Componentwise ($j = 0, \ldots, p$) weight adaptation:

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

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

Gradient Descent

Weight Adaptation

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w} \quad \text{where} \quad \Delta\mathbf{w} = -\eta \nabla \mathit{Err}(\mathbf{w})$$ [*PT* Algorithm]

Componentwise ($j = 0, \ldots, p$) weight adaptation:

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

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

$$= \frac{1}{2} \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} 2(c(\mathbf{x}) - y(\mathbf{x})) \cdot \frac{\partial}{\partial w_j} (c(\mathbf{x}) - y(\mathbf{x}))$$

Gradient Descent

Weight Adaptation

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w} \quad \text{where} \quad \Delta\mathbf{w} = -\eta \nabla \mathit{Err}(\mathbf{w})$$ [*PT* Algorithm]

Componentwise ($j = 0, \ldots, p$) weight adaptation:

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

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

Gradient Descent

Weight Adaptation

$$\mathbf{w} \;\leftarrow\; \mathbf{w} + \Delta\mathbf{w} \quad \text{where} \quad \Delta\mathbf{w} = -\eta \nabla \mathit{Err}(\mathbf{w})$$ [*PT* Algorithm]

Componentwise ($j = 0, \ldots, p$) weight adaptation:

$$w_j \;\leftarrow\; w_j + \Delta w_j \quad \text{where} \quad \Delta w_j \;=\; -\eta \,\frac{\partial}{\partial w_j} \mathit{Err}(\mathbf{w})$$

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

Gradient Descent
Weight Adaptation

$$\mathbf{w} \;\leftarrow\; \mathbf{w} + \Delta\mathbf{w} \quad \text{where} \quad \Delta\mathbf{w} = -\eta \nabla \mathit{Err}(\mathbf{w})$$ [*PT* Algorithm]

Componentwise ($j = 0, \ldots, p$) weight adaptation:

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

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

Gradient Descent

Weight Adaptation: Batch Gradient Descent [Algorithms: *IGD* *PT*]

Algorithm:	*BGD*	Batch Gradient Descent
Input:	D	Training examples $(\mathbf{x}, c(\mathbf{x}))$ with $\lvert\mathbf{x}\rvert = p+1,\ c(\mathbf{x}) \in \{0,1\}$. $(c(\mathbf{x}) \in \{-1,1\})$
	η	Learning rate, a small positive constant.
Internal:	$y(D)$	Set of $y(\mathbf{x})$-values computed from the elements $\mathbf{x}$ in D given some $\mathbf{w}$.
Output:	$\mathbf{w}$	Weight vector.

BGD(D, η)

initialize_random_weights$(\mathbf{w})$, $t = 0$

REPEAT

    $t = t + 1$

    $\Delta\mathbf{w} = 0$

    FOREACH $(\mathbf{x}, c(\mathbf{x})) \in D$ DO

        error $= c(\mathbf{x}) - \mathbf{w}^T\mathbf{x}$

        $\Delta\mathbf{w} = \Delta\mathbf{w} + \eta \cdot$ *error* $\cdot\, \mathbf{x}$

    ENDDO

    $\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$

UNTIL(*convergence*$(D, y(D))$ **OR** $t > t_{\text{max}}$)

return$(\mathbf{w})$

Remarks:

- $\Delta \mathbf{w} \sim -\nabla \mathit{Err}(\mathbf{w})$; i.e., proportional to "$-$" and not to "$+$" to descend to the minimum.

- Each BGD iteration "$\texttt{REPEAT} \ldots \texttt{UNTIL}$" corresponds to finding the direction of steepest error descent as $-\nabla \mathit{Err}(\mathbf{w_t}) = \sum_{(\mathbf{x},c(\mathbf{x}))\in D} \left(c(\mathbf{x}) - \mathbf{w}_t^T\mathbf{x}\right) \cdot \mathbf{x}$ and updating $\mathbf{w_t}$ by taking a step of length η in this direction.

- Using a constant step size η can severely impair the speed of convergence.
  When taking the optimal step size $\eta_t := \text{argmin}_\eta\, \mathit{Err}(\mathbf{w}_t - \eta \cdot \nabla \mathit{Err}(\mathbf{w_t}))$ at each iteration t, it can be shown that gradient descent has a linear rate of convergence, merely. [Meza 2010]

- The *convergence* function may compute the global error, either quantified as the sum of the squared residuals, $\mathit{Err}(\mathbf{w}_t)$, or as the norm of the error gradient, $||\nabla \mathit{Err}(\mathbf{w_t})||$, and compare it to some small positive bound ε.

Gradient Descent

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 $(\mathbf{x}, c(\mathbf{x})) \in D$:

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

This adaptation rule is known under different names:

- delta rule
- Widrow-Hoff rule
- adaline rule
- least mean squares (LMS) 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:

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

[Batch error]

Gradient Descent

Weight Adaptation: Incremental Gradient Descent [Algorithms: *BGD* *PT* *LMS*]

Algorithm:	*IGD*	Incremental Gradient Descent
Input:	D	Training examples $(\mathbf{x}, c(\mathbf{x}))$ with $\lvert\mathbf{x}\rvert = p+1$, $c(\mathbf{x}) \in \{0,1\}$. $(c(\mathbf{x}) \in \{-1,1\})$
	η	Learning rate, a small positive constant.
Internal:	$y(D)$	Set of $y(\mathbf{x})$-values computed from the elements $\mathbf{x}$ in D given some $\mathbf{w}$.
Output:	$\mathbf{w}$	Weight vector.

IGD(D, η)

```
initialize_random_weights(w),  t = 0
REPEAT
   t = t + 1
   FOREACH (x, c(x)) ∈ D DO
      error = c(x) − w^T x
      Δw = η · error · x
      w = w + Δw
   ENDDO
UNTIL(convergence(D, y(D)) OR t > t_max)
return(w)
```

Remarks:

- 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 descent enables larger weight adaptation increments.
- Compared to batch gradient descent, the example-based weight adaptation of incremental gradient descent can better avoid getting stuck in a local minimum of the error function.
- Incremental gradient descent is also called *stochastic* gradient descent.

Remarks (continued) :

- When, as is done here, the residual sum of squares, RSS, is chosen as error (loss) function, the incremental gradient descent algorithm [*IGD*] corresponds to the least mean squares algorithm [*LMS*].

- The incremental gradient descent algorithm [*IGD*] looks similar to the perceptron training algorithm [*PT*], since these algorithms differ only in the error computation (Line 5) where the latter applies the Heaviside function. However, this subtle syntactic difference is a significant conceptual difference, entailing a number of consequences:
  - Gradient descent is a regression approach and exploits the residua, which are provided by an error function of choice, and whose differential is evaluated to control the hyperplane movement.
  - The *PT* algorithm is not based on residuals (in the $(p+1)$-dimensional input-output-space) but refers to the input space only, where it simply evaluates the side of the hyperplane as a binary feature (correct side or not).
  - Provided linear separability, the *PT* algorithm will converge within a finite number of iterations, which, however, cannot be guaranteed for gradient descent.
  - Gradient descent will converge even if the data is not linearly separable.
  - Data sets can be constructed whose classes are linearly separable, but where gradient descent will not determine a hyperplane that classifies all examples correctly (whereas the *PT* Algorithm of course does).