Chapter ML:VI

VI. Neural Networks

- Perceptron Learning
- Gradient Descent
- Multilayer Perceptron
- Radial Basis Functions
Perceptron Learning
The Biological Model

Simplified model of a neuron:
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.

1985  David Rumelhart and James McClelland present the multilayer perceptron.
Perceptron Learning
The Perceptron of Rosenblatt [1958]
Perceptron Learning
The Perceptron of Rosenblatt [1958]

$x_j, w_j \in \mathbb{R}, \quad j = 1 \ldots p$
Perceptron Learning
The Perceptron of Rosenblatt [1958]

\[ x_j, w_j \in \mathbb{R}, \quad j = 1 \ldots p \]
Remarks:

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

- $X$ is a set of feature vectors, also called feature space. $X \subseteq \mathbb{R}^p$
- $C$ is a set of classes. $C = \{0, 1\}$
- $c : X \rightarrow C$ is the ideal classifier for $X$.
- $D = \{(x_1, c(x_1)), \ldots, (x_n, c(x_n))\} \subseteq X \times C$ is a set of examples.

How could the hypothesis space $H$ look like?
Perceptron Learning
Computation in the Perceptron  [Regression]

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

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

Computation in the Perceptron  [Regression]

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

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

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

\[ \rightarrow \] A hypothesis is determined by  \( \theta, w_1, \ldots, w_p \).
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 $w$ here, irrespective 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 Oliver]
Perceptron Learning
Weight Adaptation  [IGD Algorithm]

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

\(PT(D, \eta)\)
1. \textit{initialize\_random\_weights}(w),  \(t = 0\)
2. \textbf{REPEAT}
3. \(t = t + 1\)
4. \((x, c(x)) = \text{random\_select}(D)\)
5. \(\text{error} = c(x) - \text{heaviside}(w^T x)\)
6. \textbf{FOR}  \(j = 0\)  \textbf{TO}  \(p\)  \textbf{DO}
7. \(\Delta w_j = \eta \cdot \text{error} \cdot x_j\)
8. \(w_j = w_j + \Delta w_j\)
9. \textbf{ENDDO}
10. \textbf{UNTIL}  \(\text{convergence}(D, y(D))\  \textbf{OR}  t > t_{\text{max}}\)
11. \textit{return}(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.

- The weight adaptation rule compares the true class $c(x)$ (the ground truth) to the class computed by the perceptron. In case of a wrong classification of a feature vector $x$, $Err$ is either $-1$ or $+1$—independent of the exact numeric difference between $c(x)$ and $w^T x$.

- $y(D)$ is the set of $y(x)$-values given $w$ for the elements $x$ in $D$. 
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.
Definition of an (affine) hyperplane:  \( \mathbf{w}^T \mathbf{x} = 0 \iff \sum_{j=1}^{p} w_j x_j = \theta = -w_0 \)
Remarks:

- A perceptron defines a hyperplane that is perpendicular (= normal) to \((w_1, \ldots, w_p)^T\).

- \(\theta\) 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 \(w = (w_0, w_1, \ldots, w_p)^T\) form the hypothesis space \(H\).

- Weight adaptation means learning, and the shown learning paradigm is supervised.

- The computation of the weight difference \(\Delta w_j\) in Line 7 of the \textit{PT Algorithm} considers the feature vector \(x\) componentwise. In particular, if some \(x_j\) is zero, \(\Delta w_j\) will be zero as well. Keyword: Hebbian learning [Hebb 1949]
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, $c(x)$, is encoded as a number. Examples from $A$ are labeled with 1, examples from $B$ are labeled with 0.

\[
\begin{pmatrix}
  x_{11} \\
  x_{12} \\
  \vdots \\
  x_{1p}
\end{pmatrix}
\cdots
\begin{pmatrix}
  x_{k1} \\
  x_{k2} \\
  \vdots \\
  x_{kp}
\end{pmatrix}
\]

Class $A \simeq c(x) = 1$

\[
\begin{pmatrix}
  x_{l1} \\
  x_{l2} \\
  \vdots \\
  x_{lp}
\end{pmatrix}
\cdots
\begin{pmatrix}
  x_{m1} \\
  x_{m2} \\
  \vdots \\
  x_{mp}
\end{pmatrix}
\]

Class $B \simeq c(x) = 0$
A possible configuration of encoded objects in the feature space $X$:
Perceptron Learning
Illustration (continued) [PT Algorithm]

[Diagram showing a scatter plot with points labeled A and B, and a line dividing them.]
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?
Perceptron Learning

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 $x = (1, x_1, \ldots, x_p)^T$, let $X_1 \cap X_0 = \emptyset$, and let $\widehat{w}$ define a separating hyperplane with respect to $X_0$ and $X_1$. Moreover, let $D$ be a set of examples of the form $(x, 0)$, $x \in X_0$ and $(x, 1)$, $x \in X_1$. Then holds:

If the examples in $D$ are processed with the $PT$ Algorithm, the underlying weight vector $w$ 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 \( \hat{w} \). The proof requires that for all \( x \in X_1 \) the inequality \( \hat{w}^T x > 0 \) holds. This condition is always fulfilled, as the following consideration shows.

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

Preliminaries:

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

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

- For the weight vector $w$ that is to be constructed by the PT Algorithm, the two inequalities must hold as well: $w^T x < 0$ for all $x \in X_0$, and $w^T x > 0$ for all $x \in X_1$.

- Consider the set $X' = X_1 \cup \{-x \mid x \in X_0\}$: the searched $w$ fulfills $w^T x > 0$ for all $x \in X'$. 
Perceptron Learning

Perceptron Convergence Theorem: Proof

Preliminaries:

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

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

- For the weight vector \( w \) that is to be constructed by the \( PT \) Algorithm, the two inequalities must hold as well: \( w^T x < 0 \) for all \( x \in X_0 \), and \( w^T x > 0 \) for all \( x \in X_1 \).

- Consider the set \( X' = X_1 \cup \{ -x \mid x \in X_0 \} \): the searched \( w \) fulfills \( w^T x > 0 \) for all \( x \in X' \).

- The \( PT \) Algorithm performs a number of iterations, where \( w(t) \) denotes the weight vector for iteration \( t \), which form the basis for the weight vector \( w(t + 1) \). \( x(t) \in X' \) denotes the feature vector chosen in round \( t \), and \( c(x(t)) \) denotes the respective class label. The first (and randomly chosen) weight vector is denoted as \( w(0) \).

- Recall the Cauchy-Schwarz inequality: \( ||a||^2 \cdot ||b||^2 \geq (a^T b)^2 \), where \( ||x|| := \sqrt{x^T x} \) denotes the Euclidean norm.
Perceptron Learning

Perceptron Convergence Theorem: Proof (continued)

Line of argument:

(a) A lower bound for the adaptation of $w$ 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 $\hat{w}$.

(b) An upper bound for the adaptation of $w$ 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 $n$, where the lower bound grows faster than the upper bound. Hence, in order to fulfill the inequality, the number of iterations is finite.
1. The $PT$ Algorithm computes in iteration $t$ the scalar product $w(t)^T x(t)$. If classified correctly, $w(t)^T x(t) > 0$ and $w$ is unchanged. Otherwise, $w(t + 1) = w(t) + \eta \cdot x(t)$.
1. The *PT* Algorithm computes in iteration $t$ the scalar product $w(t)^T x(t)$. If classified correctly, $w(t)^T x(t) > 0$ and $w$ is unchanged. Otherwise, $w(t + 1) = w(t) + \eta \cdot x(t)$.

2. Consider a sequence of $n$ incorrectly classified feature vectors, $(x(t))$, along with the corresponding weight vector adaptation, $w(t + 1) = w(t) + \eta \cdot x(t)$:
   - $w(1) = w(0) + \eta \cdot x(0)$
   - $w(2) = w(1) + \eta \cdot x(1) = w(0) + \eta \cdot x(0) + \eta \cdot x(1)$
   - $\vdots$
   - $w(n) = w(0) + \eta \cdot x(0) + \ldots + \eta \cdot x(n - 1)$
1. The *PT* Algorithm computes in iteration $t$ the scalar product $w(t)^T x(t)$. If classified correctly, $w(t)^T x(t) > 0$ and $w$ is unchanged. Otherwise, $w(t + 1) = w(t) + \eta \cdot x(t)$.

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

3. The hyperplane defined by $\hat{w}$ separates $X_1$ and $X_0$: $\forall x \in X' : \hat{w}^T x > 0$

   Let $\delta := \min_{x \in X'} \hat{w}^T x$. Observe that $\delta > 0$ holds.
Perceptron Learning

Perceptron Convergence Theorem: Proof (continued)

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

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

   $\vdots$

   $w(n) = w(0) + \eta \cdot x(0) + \ldots + \eta \cdot x(n - 1)$

3. The hyperplane defined by $\hat{w}$ separates $X_1$ and $X_0$: $\forall x \in X': \hat{w}^T x > 0$

   Let $\delta := \min_{x \in X'} \hat{w}^T x$. Observe that $\delta > 0$ holds.

4. Analyze the scalar product between $w(n)$ and $\hat{w}$:

   $\hat{w}^T w(n) = \hat{w}^T w(0) + \eta \cdot \hat{w}^T x(0) + \ldots + \eta \cdot \hat{w}^T x(n - 1)$

   $\Rightarrow \hat{w}^T w(n) \geq \hat{w}^T w(0) + n\eta\delta \geq 0$

   $\Rightarrow (\hat{w}^T w(n))^2 \geq (\hat{w}^T w(0) + n\eta\delta)^2$

5. Apply the Cauchy-Schwarz inequality:

   $||\hat{w}||^2 \cdot ||w(n)||^2 \geq (\hat{w}^T w(0) + n\eta\delta)^2 \Rightarrow ||w(n)||^2 \geq \frac{(\hat{w}^T w(0) + n\eta\delta)^2}{||\hat{w}||^2}$
6. Consider again the weight adaptation \( w(t + 1) = w(t) + \eta \cdot x(t) \):

\[
||w(t + 1)||^2 = ||w(t) + \eta \cdot x(t)||^2
\]

\[
= (w(t) + \eta \cdot x(t))^T (w(t) + \eta \cdot x(t))
\]

\[
= w(t)^T w(t) + \eta^2 \cdot x(t)^T x(t) + 2\eta \cdot w(t)^T x(t)
\]

\[
\leq ||w(t)||^2 + ||\eta \cdot x(t)||^2, \quad \text{since} \quad w(t)^T x(t) < 0
\]
Perceptron Learning
Perceptron Convergence Theorem: Proof (continued)

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

\[
||\mathbf{w}(t + 1)||^2 = ||\mathbf{w}(t) + \eta \cdot \mathbf{x}(t)||^2
\]

\[
= (\mathbf{w}(t) + \eta \cdot \mathbf{x}(t))^T(\mathbf{w}(t) + \eta \cdot \mathbf{x}(t))
\]

\[
= \mathbf{w}(t)^T \mathbf{w}(t) + \eta^2 \cdot \mathbf{x}(t)^T \mathbf{x}(t) + 2\eta \cdot \mathbf{w}(t)^T \mathbf{x}(t)
\]

\[
\leq ||\mathbf{w}(t)||^2 + ||\eta \cdot \mathbf{x}(t)||^2, \quad \text{since } \mathbf{w}(t)^T \mathbf{x}(t) < 0
\]

7. Consider a sequence of \( n \) weight adaptations:

\[
||\mathbf{w}(n)||^2 \leq ||\mathbf{w}(n - 1)||^2 + ||\eta \cdot \mathbf{x}(n - 1)||^2
\]

\[
\leq ||\mathbf{w}(n - 2)||^2 + ||\eta \cdot \mathbf{x}(n - 2)||^2 + ||\eta \cdot \mathbf{x}(n - 1)||^2
\]

\[
\leq ||\mathbf{w}(0)||^2 + ||\eta \cdot \mathbf{x}(0)||^2 + \ldots + ||\eta \cdot \mathbf{x}(n - 1)||^2
\]

\[
= ||\mathbf{w}(0)||^2 + \sum_{j=0}^{n-1} ||\eta \cdot \mathbf{x}(j)||^2
\]

8. With \( \varepsilon := \max_{\mathbf{x} \in X'} ||\mathbf{x}||^2 \) follows \( ||\mathbf{w}(n)||^2 \leq ||\mathbf{w}(0)||^2 + n\eta^2 \varepsilon \)
9. Both inequalities must be fulfilled:

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

hence

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

10. Observe:

$$\frac{(\hat{\mathbf{w}}^T \mathbf{w}(0) + n\eta \delta)^2}{||\hat{\mathbf{w}}||^2} \in \Theta(n^2) \quad \text{and} \quad ||\mathbf{w}(0)||^2 + n\eta^2 \varepsilon \in \Theta(n)$$
9. Both inequalities must be fulfilled:

\[ \|w(n)\|^2 \geq \frac{(\hat{w}^T w(0) + n\eta\delta)^2}{\|\hat{w}\|^2} \quad \text{and} \quad \|w(n)\|^2 \leq \|w(0)\|^2 + n\eta^2\epsilon, \text{ hence} \]

\[ \frac{(\hat{w}^T w(0) + n\eta\delta)^2}{\|\hat{w}\|^2} \leq \|w(n)\|^2 \leq \|w(0)\|^2 + n\eta^2\epsilon \]

10. Observe:

\[ \frac{(\hat{w}^T w(0) + n\eta\delta)^2}{\|\hat{w}\|^2} \in \Theta(n^2) \quad \text{and} \quad \|w(0)\|^2 + n\eta^2\epsilon \in \Theta(n) \]

11. An upper bound for \( n \):

\[ \frac{(\hat{w}^T w(0) + n\eta\delta)^2}{\|\hat{w}\|^2} \leq \|w(0)\|^2 + n\eta^2\epsilon \]

For \( w(0) = 0 \) (set all initial weights to zero) follows:

\[ 0 < n \leq \frac{\epsilon}{\delta^2\|\hat{w}\|^2} \]

\( \rightarrow \) 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 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:

\[
y(x) = w^T x = \sum_{j=0}^{p} w_j x_j \quad \text{[Heaviside]}
\]

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

\[
Err(w) = \frac{1}{2} \sum_{(x,c(x)) \in D} (c(x) - y(x))^2 \quad \text{[Singleton error]}
\]
The gradient $\nabla Err(w)$ of $Err(w)$ defines the steepest ascent or descent:

$$\nabla Err(w) = \left( \frac{\partial Err(w)}{\partial w_0}, \frac{\partial Err(w)}{\partial w_1}, \ldots, \frac{\partial Err(w)}{\partial w_p} \right)$$
Gradient Descent
Weight Adaptation

\[ w \leftarrow w + \Delta w \quad \text{where} \quad \Delta w = -\eta \nabla \text{Err}(w) \]

Componentwise (dimension-wise) weight adaptation:

\[ w_j \leftarrow w_j + \Delta w_j \quad \text{where} \quad \Delta w_j = -\eta \frac{\partial}{\partial w_j} \text{Err}(w) \]
Gradient Descent

Weight Adaptation

\[ w \leftarrow w + \Delta w \quad \text{where} \quad \Delta w = -\eta \nabla Err(w) \]

Componentwise (dimension-wise) weight adaptation:

\[ w_j \leftarrow w_j + \Delta w_j \quad \text{where} \quad \Delta w_j = -\eta \frac{\partial}{\partial w_j} Err(w) \]

\[
\frac{\partial}{\partial w_j} Err(w) = \frac{\partial}{\partial w_j} \frac{1}{2} \sum_{(x,c(x)) \in D} (c(x) - y(x))^2 = \frac{1}{2} \sum_{(x,c(x)) \in D} \frac{\partial}{\partial w_j} (c(x) - y(x))^2
\]
Gradient Descent

Weight Adaptation

\[ w \leftarrow w + \Delta w \quad \text{where} \quad \Delta w = -\eta \nabla \text{Err}(w) \]

Componentwise (dimension-wise) weight adaptation:

\[ w_{j} \leftarrow w_{j} + \Delta w_{j} \quad \text{where} \quad \Delta w_{j} = -\eta \frac{\partial}{\partial w_{j}} \text{Err}(w) \]

\[
\frac{\partial}{\partial w_{j}} \text{Err}(w) = \frac{\partial}{\partial w_{j}} \frac{1}{2} \sum_{(x,c(x)) \in D} (c(x) - y(x))^2 = \frac{1}{2} \sum_{(x,c(x)) \in D} \frac{\partial}{\partial w_{j}} (c(x) - y(x))^2
\]

\[
= \frac{1}{2} \sum_{(x,c(x)) \in D} 2(c(x) - y(x)) \cdot \frac{\partial}{\partial w_{j}} (c(x) - y(x))
\]
Gradient Descent

Weight Adaptation

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

Componentwise (dimension-wise) weight adaptation:

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

\[
\frac{\partial}{\partial w_j} \mathbf{Err} (\mathbf{w}) = \frac{\partial}{\partial w_j} \left( \frac{1}{2} \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} (c(\mathbf{x}) - y(\mathbf{x}))^2 \right) = \frac{1}{2} \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} \frac{\partial}{\partial w_j} (c(\mathbf{x}) - y(\mathbf{x}))^2 \\
= \frac{1}{2} \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} 2(c(\mathbf{x}) - y(\mathbf{x})) \cdot \frac{\partial}{\partial w_j} (c(\mathbf{x}) - y(\mathbf{x})) \\
= \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) \cdot \frac{\partial}{\partial w_j} (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x})
\]
Gradient Descent

Weight Adaptation

\[ w \leftarrow w + \Delta w \quad \text{where} \quad \Delta w = -\eta \nabla \text{Err}(w) \]

Componentwise (dimension-wise) weight adaptation:

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

\[
\frac{\partial}{\partial w_j} \text{Err}(w) = \frac{\partial}{\partial w_j} \frac{1}{2} \sum_{(x, c(x)) \in D} (c(x) - y(x))^2 = \frac{1}{2} \sum_{(x, c(x)) \in D} \frac{\partial}{\partial w_j} (c(x) - y(x))^2
\]

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

\[
= \sum_{(x, c(x)) \in D} (c(x) - w^T x) \cdot \frac{\partial}{\partial w_j} (c(x) - w^T x)
\]

\[
= \sum_{(x, c(x)) \in D} (c(x) - w^T x)(-x_j)
\]
Gradient Descent

Weight Adaptation: Batch Gradient Descent  [IGD Algorithm]

Algorithm:  \( BGD \)  Batch Gradient Descent

Input:  \( D \)  Training examples of the form \((x, c(x))\) with \(|x| = p + 1\), \(c(x) \in \{0, 1\}\).
\( \eta \)  Learning rate, a small positive constant.

Internal:  \( y(D) \)  Set of \( y(x) \)-values computed from the elements \( x \) in \( D \) given some \( w \).

Output:  \( w \)  Weight vector.

\( BGD(D, \eta) \)

1.  \texttt{initialize\_random\_weights}(w),  \( t = 0 \)
2.  \textbf{REPEAT}
3. \( t = t + 1 \)
4.  \textbf{FOR}  \( j = 0 \)  \textbf{TO}  \( p \)  \textbf{DO}  \( \Delta w_j = 0 \)
5. \textbf{FOREACH}  \((x, c(x)) \in D\)  \textbf{DO}
6.  \texttt{error}  =  \( c(x) - w^T x \)
7. \textbf{FOR}  \( j = 0 \)  \textbf{TO}  \( p \)  \textbf{DO}  \( \Delta w_j = \Delta w_j + \eta \cdot \texttt{error} \cdot x_j \)
8.  \textbf{ENDDO}
9. \textbf{FOR}  \( j = 0 \)  \textbf{TO}  \( p \)  \textbf{DO}  \( w_j = w_j + \Delta w_j \)
10.  \textbf{UNTIL}  \((\texttt{convergence}(D, y(D)) \textbf{ OR } t > t_{max})\)
11.  \texttt{return}(w)
Gradient Descent
Weight Adaptation: Delta Rule

The weight adaptation in the **BGD Algorithm** is set-based: before modifying a weight component in \( w \), the total error of *all* examples (the “batch”) is computed.

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

\[
\Delta w_j = \eta \cdot (c(x) - w^T x) \cdot x_j
\]

This adaptation rule is known under different names:

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

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

\[
\text{Err}_d(w) = \frac{1}{2} (c(x) - w^T x)^2
\]

[Batch error]
Gradient Descent

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

Algorithm: \( IGD \)  Incremental Gradient Descent

Input:  \( D \)  Training examples of the form \((x, c(x))\) with \(|x| = p + 1\), \(c(x) \in \{0, 1\}\).

\( \eta \)  Learning rate, a small positive constant.

Internal:  \( y(D) \)  Set of \( y(x) \)-values computed from the elements \( x \) in \( D \) given some \( w \).

Output:  \( w \)  Weight vector.

\( IGD(D, \eta) \)

1.  \( \text{initialize\_random\_weights}(w), \ t = 0 \)
2.  \( \text{REPEAT} \)
3.  \( t = t + 1 \)
4.  \( \text{FOREACH} \ (x, c(x)) \in D \ \text{DO} \)
5.  \( \text{error} = c(x) - w^T x \)
6.  \( \text{FOR} \ j = 0 \ \text{TO} \ p \ \text{DO} \)
7.  \( \Delta w_j = \eta \cdot \text{error} \cdot x_j \)
8.  \( w_j = w_j + \Delta w_j \)
9.  \( \text{ENDDO} \)
10.  \( \text{ENDDO} \)
11.  \( \text{UNTIL}(\text{convergence}(D, y(D)) \ \text{OR} \ t > t_{max}) \)
12.  \( \text{return}(w) \)
The classification error $Err$ of incremental gradient descent is specific for each training example $d \in D, d = (x, c(x))$: 
$Err_d(w) = \frac{1}{2}(c(x) - w^T x)^2$

The sequence of incremental weight adaptations approximates the gradient descent of the batch approach. If $\eta$ 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.
Remarks (continued):

- The incremental gradient descend algorithm corresponds to the least mean squares (LMS) algorithm.

- The incremental gradient descend algorithm is similar to the perceptron training (PT) algorithm except for the fact that the latter applies the Heaviside function within the error computation. Consequences:
  - Gradient descend will converge even if the data is not linear separable.
  - Provided linear separability, the PT algorithm converges within a finite number of iterations, which, however, cannot be guaranteed for gradient descend.
  - The error function of the PT algorithm is not differentiable, which prohibits an effective exploitation of the residual.

- Incremental gradient descent is also called stochastic gradient descent.