Chapter ML:IV

IV. Neural Networks

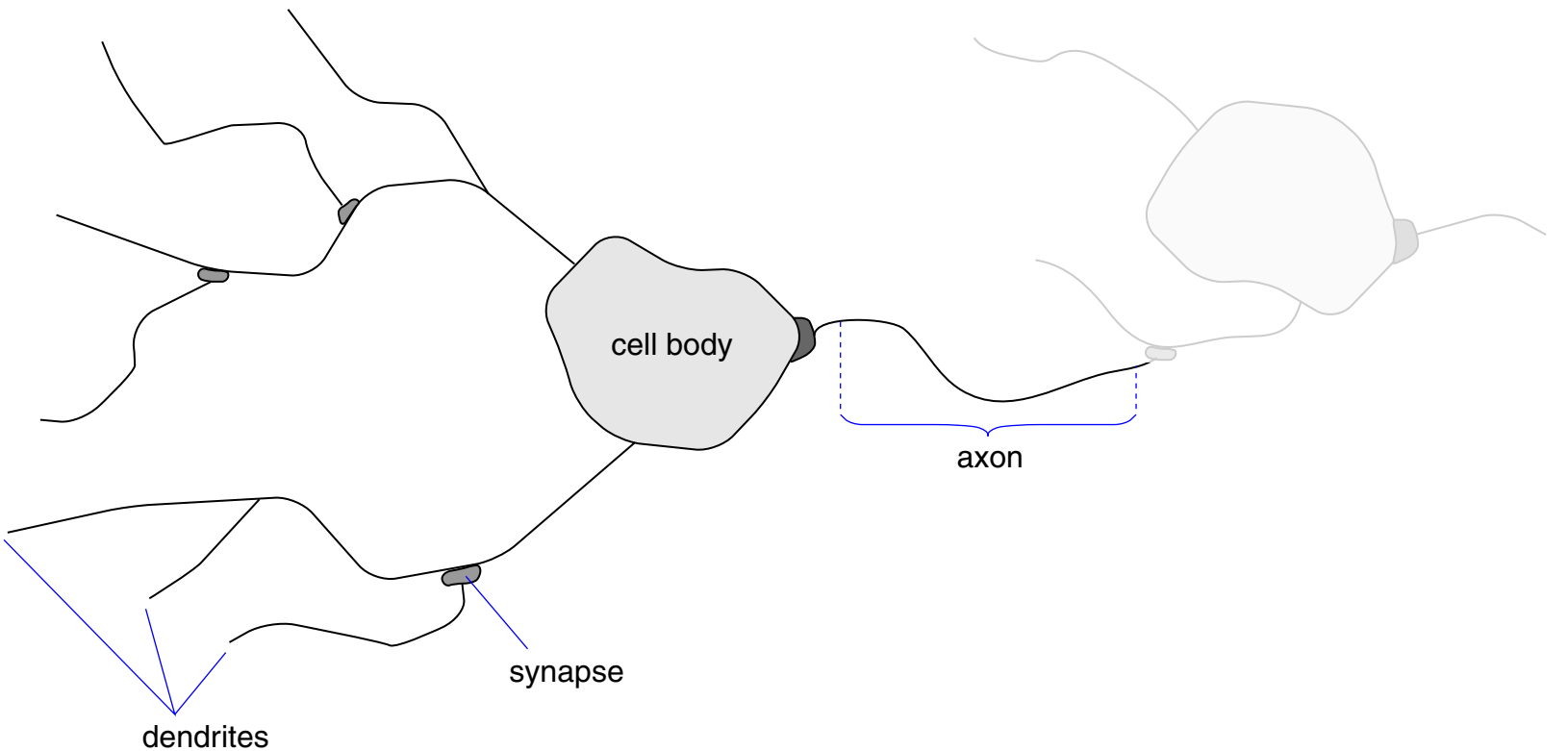
- Perceptron Learning
- Gradient Descent
- Multilayer Perceptron

© STEIN/VÖLSKE 2021

Perceptron Learning

Biological Inspiration

A *very* simplified model of a neuron:



© STEIN/VÖLSKE 2021

Perceptron Learning

Biological Inspiration (continued)

Neuron characteristics:

- The numerous dendrites of a neuron serve as its input channels for electrical signals.
- At particular contact points, 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”. Keyword: frequency modulation
- 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)

© STEIN/VÖLSKE 2021

Remarks (facts about the human brain):

- The human brain is comprised of about 10^{11} neurons.
- The length of a dendrite or an axon is about 100 micron. A micron = 10^{-6} m = 10^{-3} mm.
- The dendrites of a neuron are connected to 100 000 – 200 000 other neurons.
- An axon is connected to up to 10 000 synapses of other neurons.
- The human contains about 10^{12} synapses.
- The switching of a neuron is not faster than 10^{-3} s, which is rather slow compared to the 10^{-12} s for electrical circuits. I.e., within the typical human reaction time of about 10^{-1} s only a few 100 neuronal activities can take place.

© STEIN/VÖLSKE 2021

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 involved in a decision process)

1958 Frank Rosenblatt develops the perceptron model. (see next slide)

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.

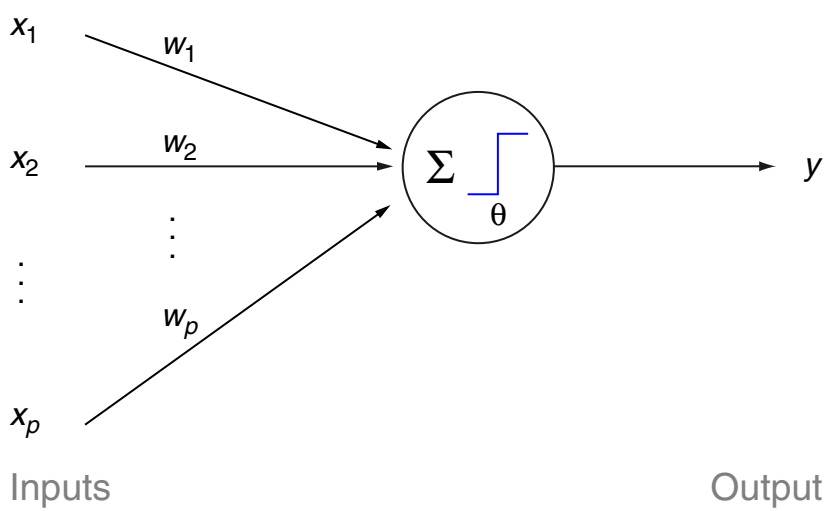
1985

1986 David Rumelhart and James McClelland present the multilayer perceptron.

© STEIN/VÖLSKE 2021

Perceptron Learning

The Perceptron of Rosenblatt [1958]

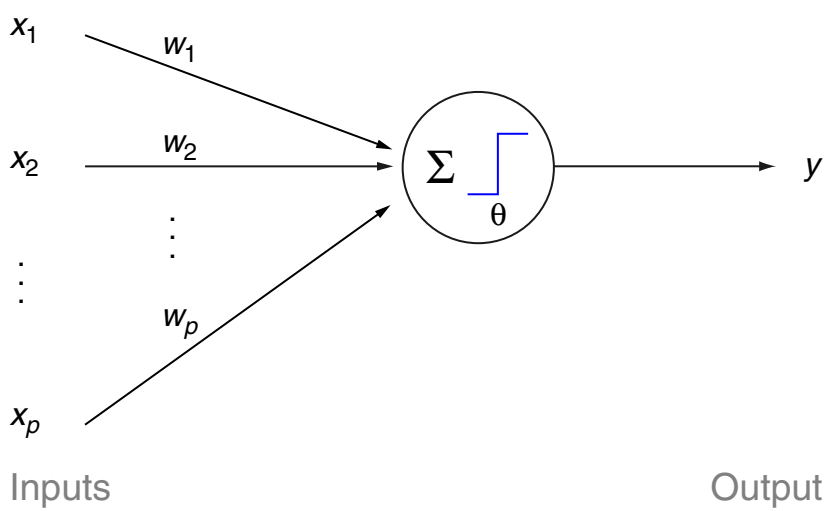


$$x_j, w_j \in \mathbf{R}, \quad j = 1, \ldots, p$$

© STEIN/VÖLSKE 2021

Perceptron Learning

The Perceptron of Rosenblatt [1958]



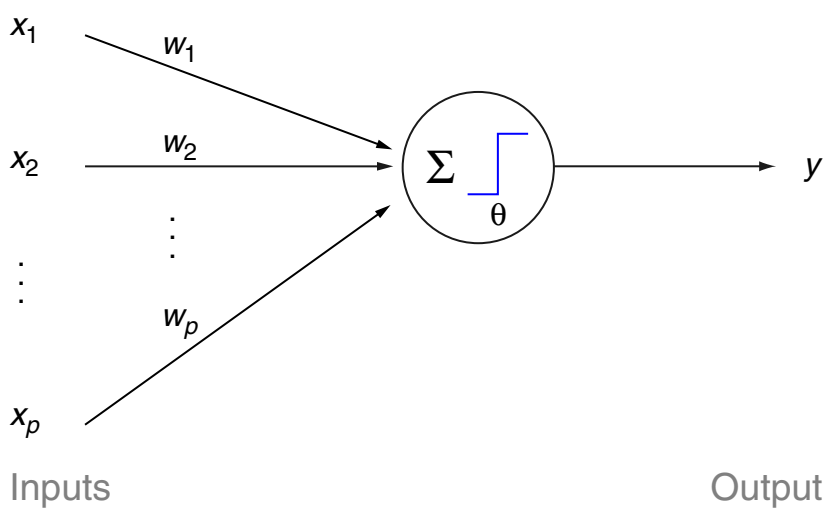
$$x_j, w_j \in \mathbf{R}, \quad j = 1, \ldots, p$$

- Rosenblatt's perceptron relies on the neuron model of McCulloch/Pitts [1943].
- The weights w_j model the reinforcement factor.
- The threshold function models a decision rule.
- The perceptron is a "feed forward system".

© STEIN/VÖLSKE 2021

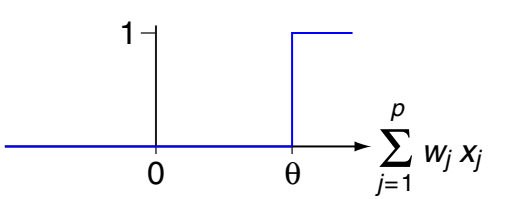
Perceptron Learning

The Perceptron of Rosenblatt [1958] (continued)



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

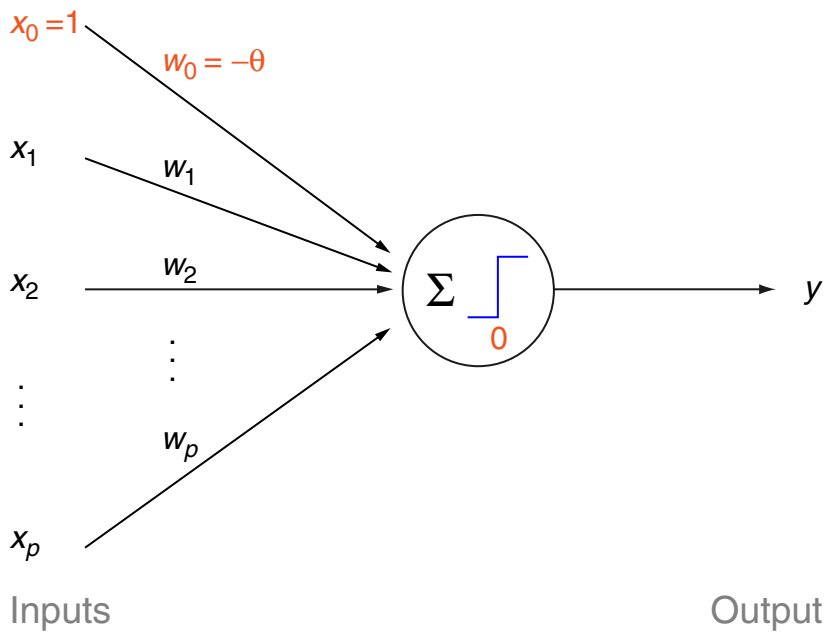
$$y(\mathbf{x}) = \begin{cases} 1, & \text{if } \sum_{j=1}^{p} w_j x_j \geq \theta \\ 0, & \text{if } \sum_{j=1}^{p} w_j x_j < \theta \end{cases}$$



© STEIN/VÖLSKE 2021

Perceptron Learning

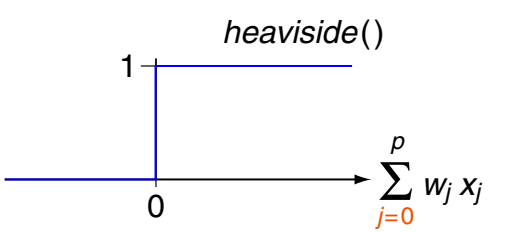
The Perceptron of Rosenblatt [1958] (continued)



$$x_j, w_j \in \mathbf{R}, \quad j = 0, \ldots, p$$

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

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



© STEIN/VÖLSKE 2021

Remarks:

- By extending both the weight vector by $w_0 = -\theta$ and the feature vectors by the constant feature $x_0 = 1$, the learning algorithm gets a canonical form.
- A single hypothesis is determined by $(w_0, w_1, \ldots, w_p)$.
- Since the range of the model function $y(\mathbf{x})$ is $\{0; 1\}$ the *heaviside* function is used for discretization (= class assignment). In section Linear Regression of part Machine Learning Basics the range of the model function is $\{-1; 1\}$ and the sign function does the discretization, which is no difference in principle.
- The *heaviside* function is named after the mathematician Oliver Heaviside.
  [Wikipedia: step function, Oliver Heaviside]

© STEIN/VÖLSKE 2021

Perceptron Learning

Binary Classification Problems

Setting:

- X is a set of feature vectors from an inner product space $\mathbf{X}$, $\mathbf{X} \subseteq \mathbf{R}^p$.
- $C = \{0, 1\}$ is a set of two classes. Similarly: $\{-1, 1\}$, $\{\ominus, \oplus\}$, $\{\text{no}, \text{yes}\}$, etc.
- $c : X \to C$ is the (unknown) 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.

Todo:

- Approximate $c(\mathbf{x})$, which is implicitly given via D, with the perceptron.

© STEIN/VÖLSKE 2021

Perceptron Learning

The PT Algorithm

[Algorithms: LMS BGD$_\sigma$ PT BGD$_2$ IGD]

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

PT(D, η)

1. *initialize_random_weights*$(\mathbf{w})$, $t = 0$
2. **REPEAT**
3. $\quad t = t + 1$
4. $\quad (\mathbf{x}, c(\mathbf{x})) =$ *random_select*(D)
5. $\quad \delta = c(\mathbf{x}) - y(\mathbf{x})$ // $y(\mathbf{x}) \overset{(\star)}{=}$ *heaviside*$(\mathbf{w}^T\mathbf{x}) \in \{0, 1\}$, $c(\mathbf{x}) \in \{0, 1\} \rightsquigarrow \delta \in \{0, 1, -1\}$
6. $\quad \Delta\mathbf{w} \overset{(\star)}{=} \eta \cdot \delta \cdot \mathbf{x}$
7. $\quad \mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$
8. **UNTIL**(*convergence*$(c(D), y(D))$ **OR** $t > t_{\text{max}}$)
9. *return*$(\mathbf{w})$

© STEIN/VÖLSKE 2021

Remarks:

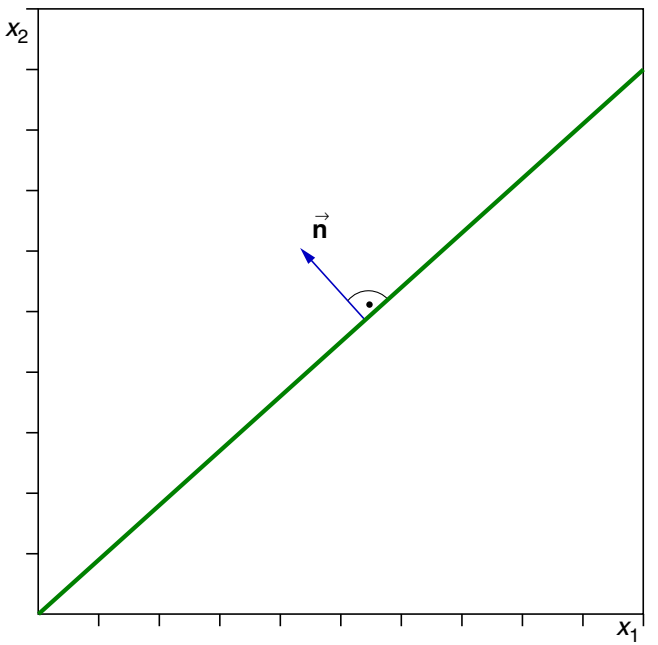
($\star$) Recap. When used as operand in a scalar product with the weight vector, $\mathbf{w}^T\mathbf{x}$, we consider the feature vector $\mathbf{x}$ in its extended form. I.e., $x_0 = 1$, and $\mathbf{x} = (1, x_1, \ldots, x_p)^T \in \mathbf{R}^{p+1}$.

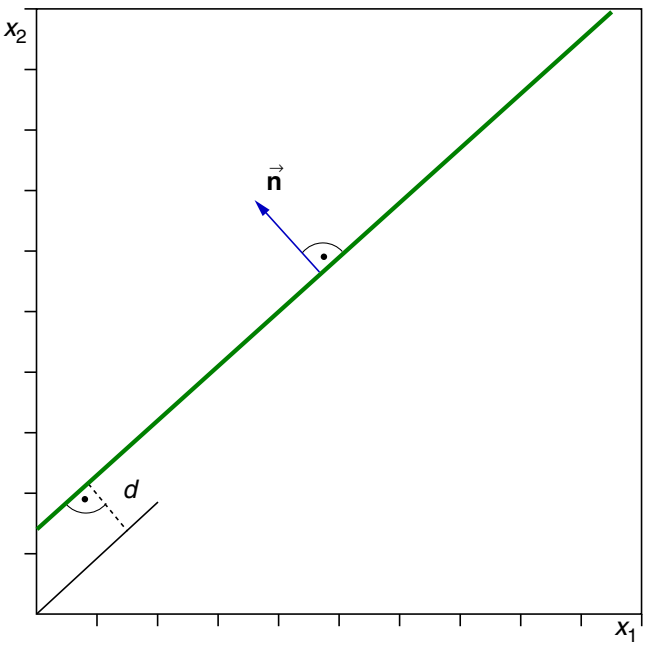
- The variable t denotes the time. The learning algorithm gets an example presented at each point in time and may adapt the weight vector.
- The weight adaptation rule compares the target class $c(\mathbf{x})$ (the ground truth) to the class computed by the perceptron. In case of a wrong classification of a feature vector $\mathbf{x}$, δ is either -1 or $+1$, regardless of the exact numeric difference between $c(\mathbf{x})$ and $\mathbf{w}^T\mathbf{x}$.
- The *convergence* function checks for misclassified examples (= analyzes the $0/1$ loss). In this regard $c(D)$ and $y(D)$ denote the vectors of target classes and computed values respectively.

© STEIN/VÖLSKE 2021

Perceptron Learning

Weight Adaptation: Illustration in Input Space





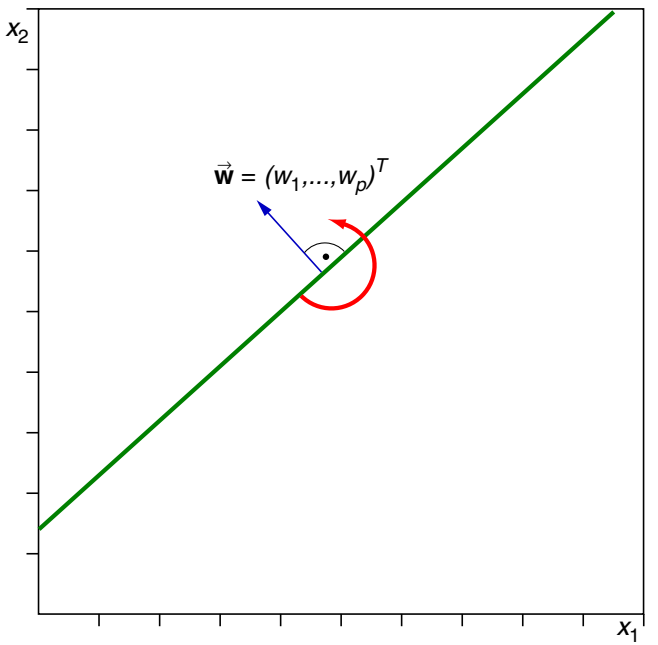
Definition of an (affine) hyperplane: $N = \left\{ \mathbf{x} \mid \vec{\mathbf{n}}^T \mathbf{x} = d \right\}$ [Wikipedia]

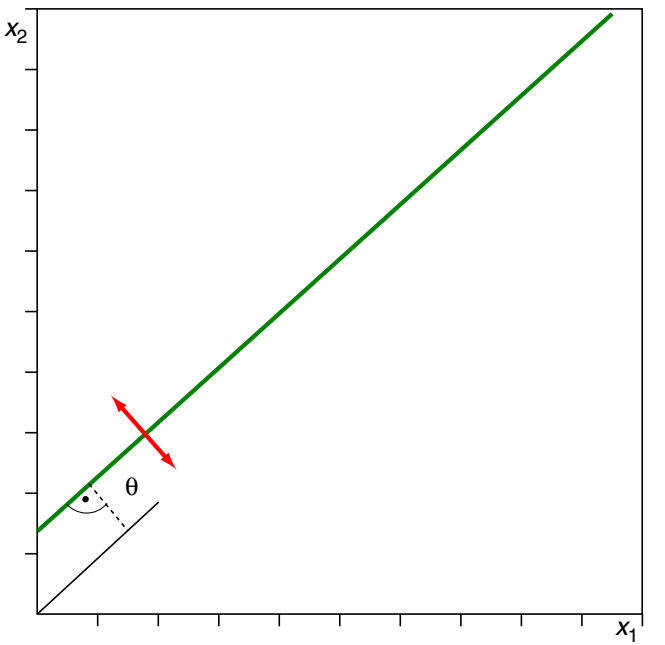
- $\vec{\mathbf{n}}$ is a normal vector of (a vector perpendicular to) the hyperplane N.
- If $||\vec{\mathbf{n}}|| = 1$ then $|\vec{\mathbf{n}}^T \mathbf{x} - d|$ gives the distance of a point $\mathbf{x}$ to N.
- If $\text{sgn}(\vec{\mathbf{n}}^T \mathbf{x}_1 - d) = \text{sgn}(\vec{\mathbf{n}}^T \mathbf{x}_2 - d)$, then $\mathbf{x}_1$ and $\mathbf{x}_2$ are on the same side of the hyperplane.

© STEIN/VÖLSKE 2021

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.

© STEIN/VÖLSKE 2021

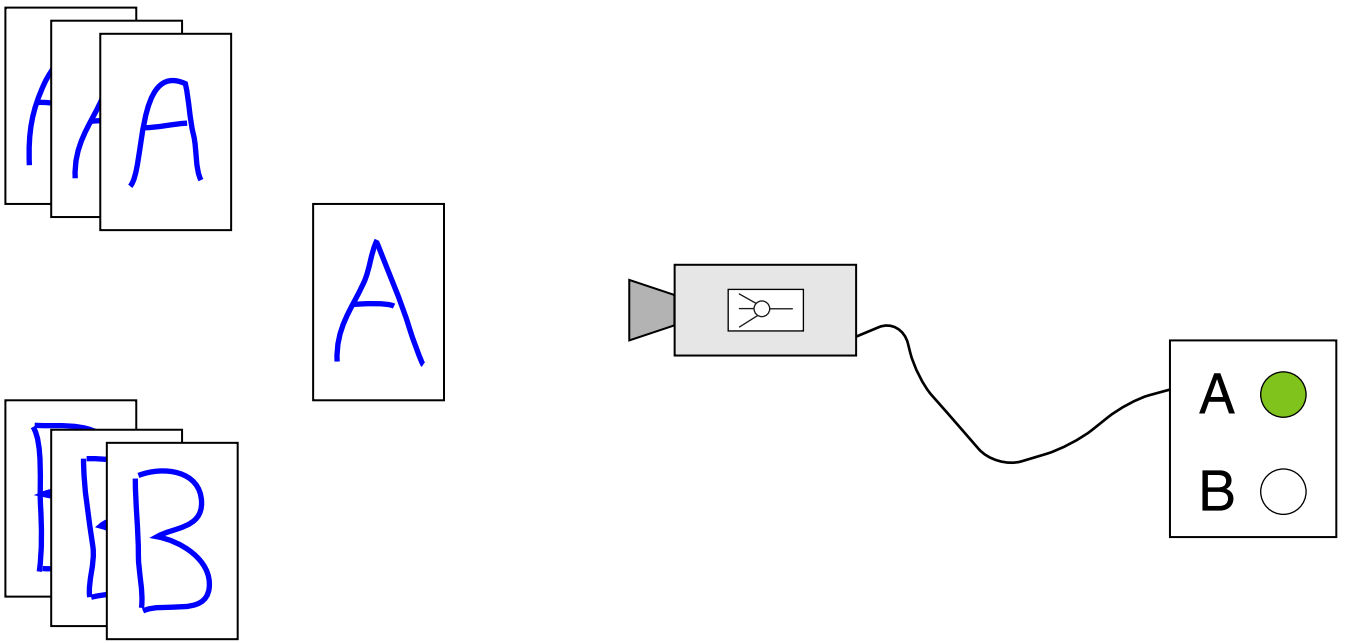
Remarks:

- Recap. Distinguish between the p-dimensional direction vector $\vec{\mathbf{w}} = (w_1, \ldots, w_p)^T$, and the $(p+1)$-dimensional hypothesis $\mathbf{w} = (w_0, w_1, \ldots, w_p)^T$.
- A perceptron defines a hyperplane that is perpendicular (= normal) to $\vec{\mathbf{w}} = (w_1, \ldots, w_p)^T$.
- θ or $-w_0$ defines the offset of the hyperplane from the origin, along $\vec{\mathbf{w}}$ and as multiple of $1/||\vec{\mathbf{w}}||$.
- 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. Consider in this regard the lines 6–7 of the PT algorithm: If some x_j is zero, Δw_j will be zero as well. Keyword: Hebbian learning [Wikipedia: Hebbian theory, Donald Hebb]
- 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.

© STEIN/VÖLSKE 2021

Perceptron Learning

Example



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

© STEIN/VÖLSKE 2021

Perceptron Learning

Example (continued)

Encoding:

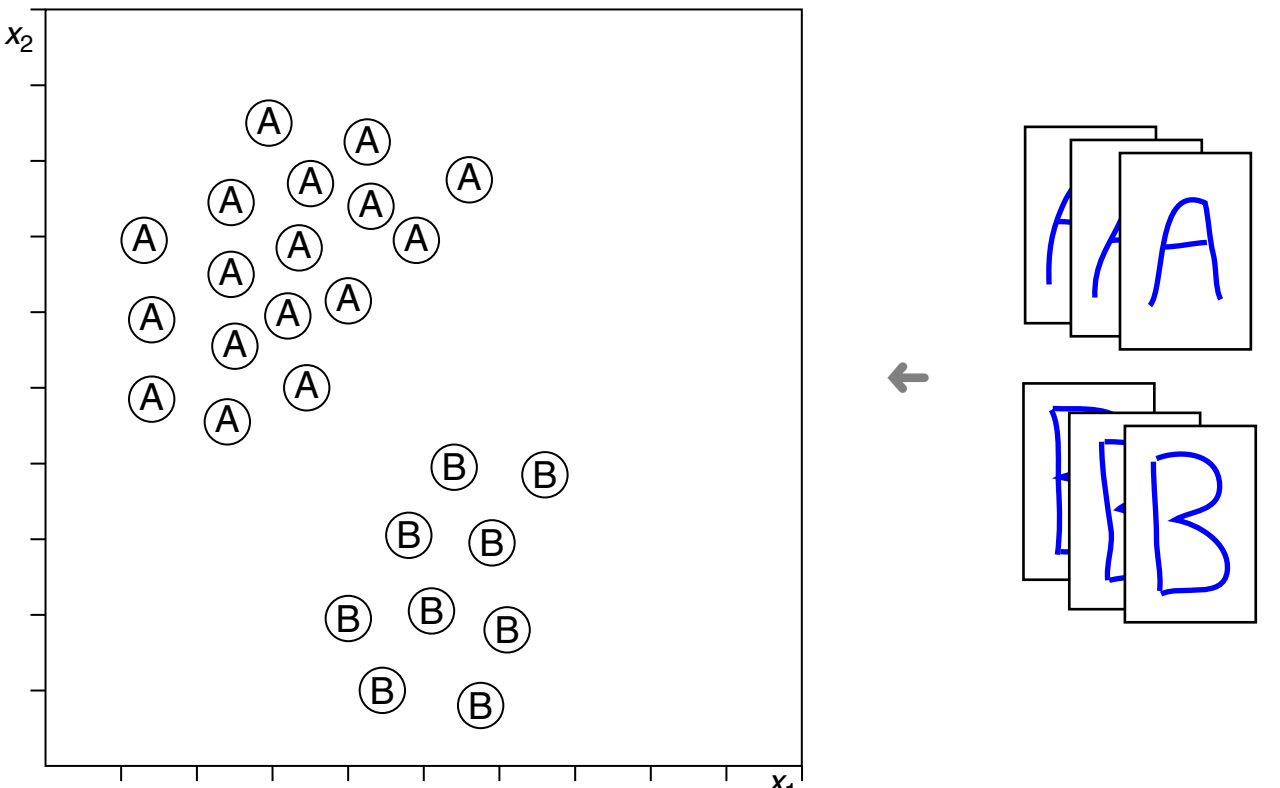
- The encoding of the examples is based on features such as the number of line crossings, most acute angle, longest line, etc.
- The class label, $c(\mathbf{x})$, is encoded as a number: examples from A are labeled with 1, examples from B are labeled with 0.

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

© STEIN/VÖLSKE 2021

Perceptron Learning

Example: Illustration in Input Space [PT algorithm]

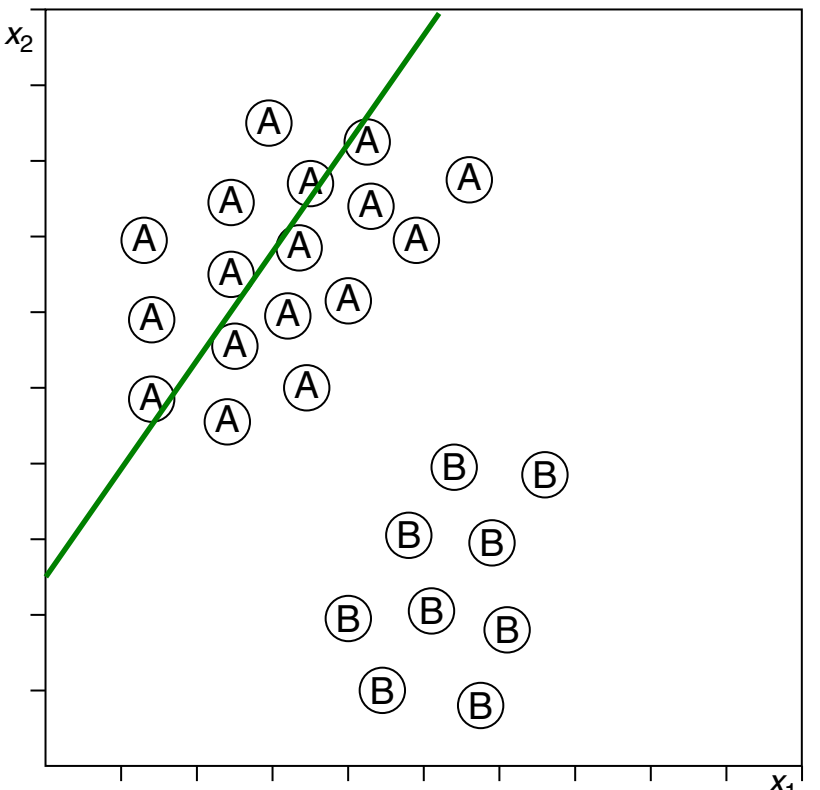


A possible configuration of encoded objects (feature vectors) in the input space $\mathbf{X}$.

© STEIN/VÖLSKE 2021

Perceptron Learning

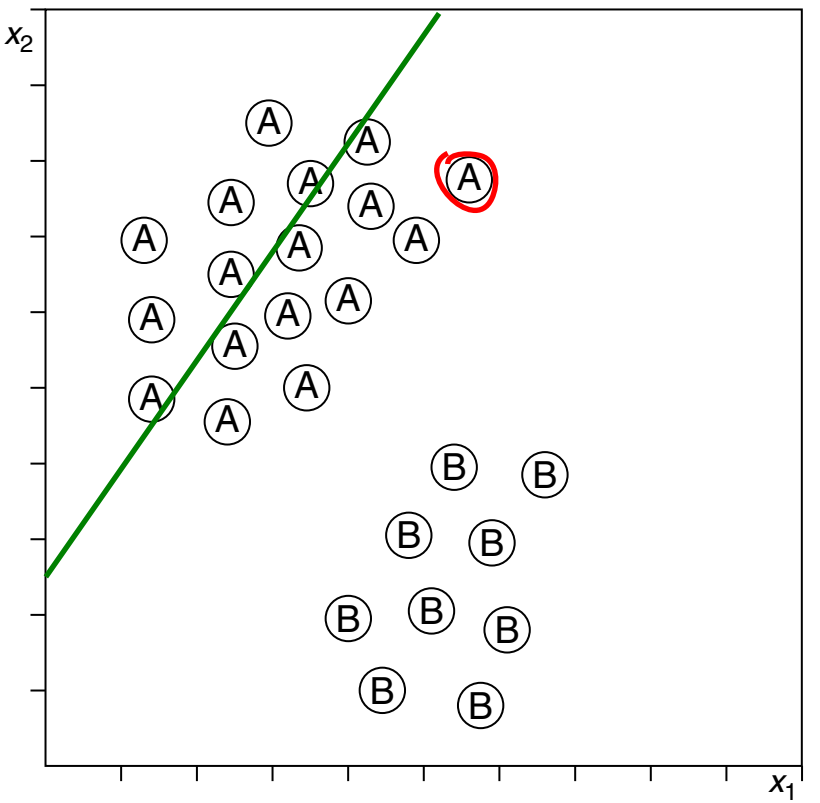
Example: Illustration in Input Space [PT algorithm]



© STEIN/VÖLSKE 2021

Perceptron Learning

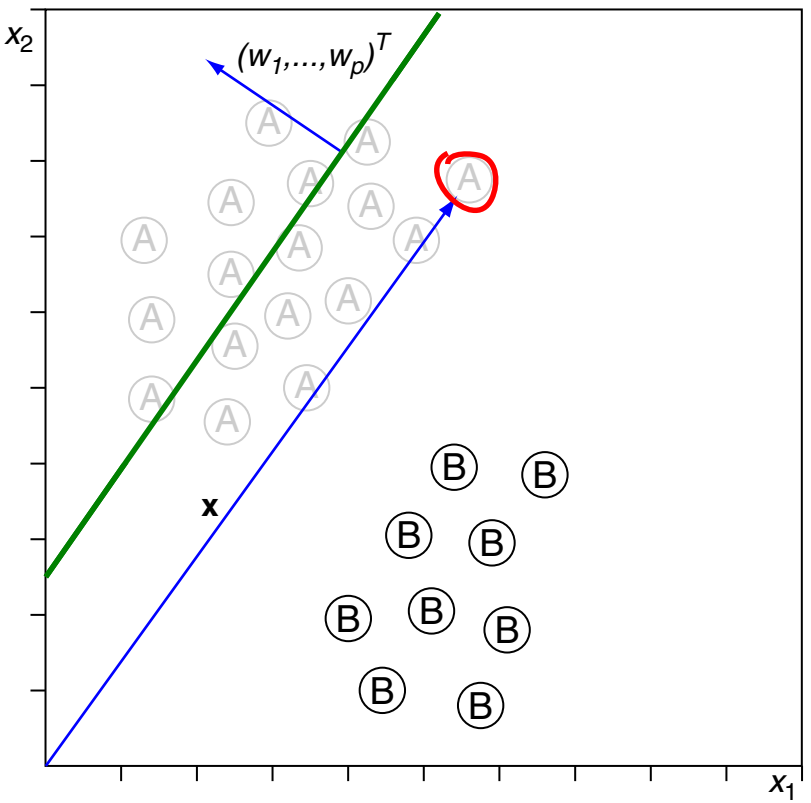
Example: Illustration in Input Space [PT algorithm]



© STEIN/VÖLSKE 2021

Perceptron Learning

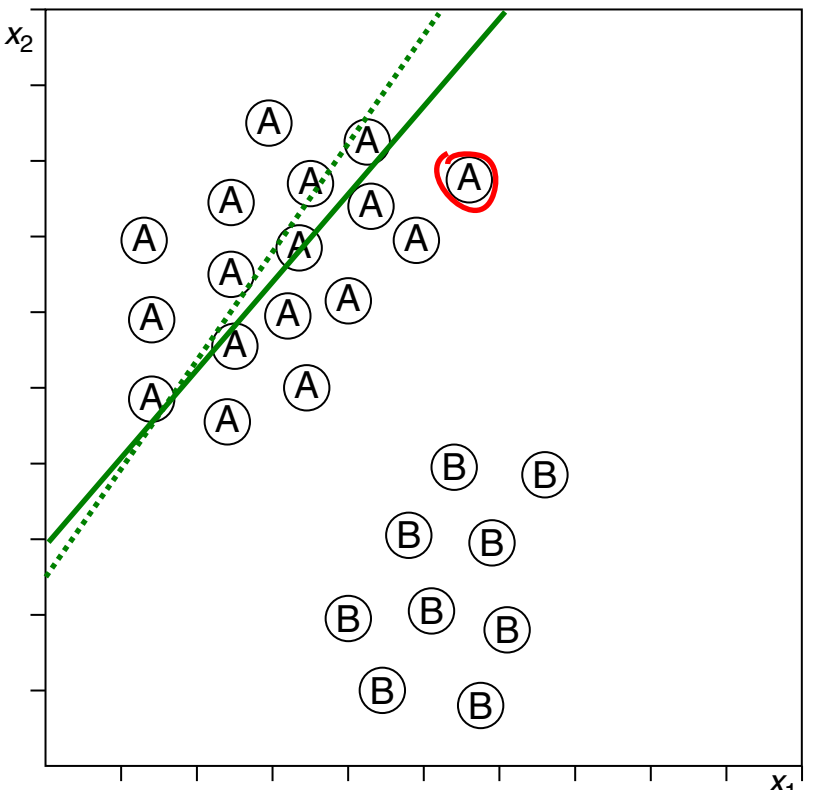
Example: Illustration in Input Space [PT algorithm]



© STEIN/VÖLSKE 2021

Perceptron Learning

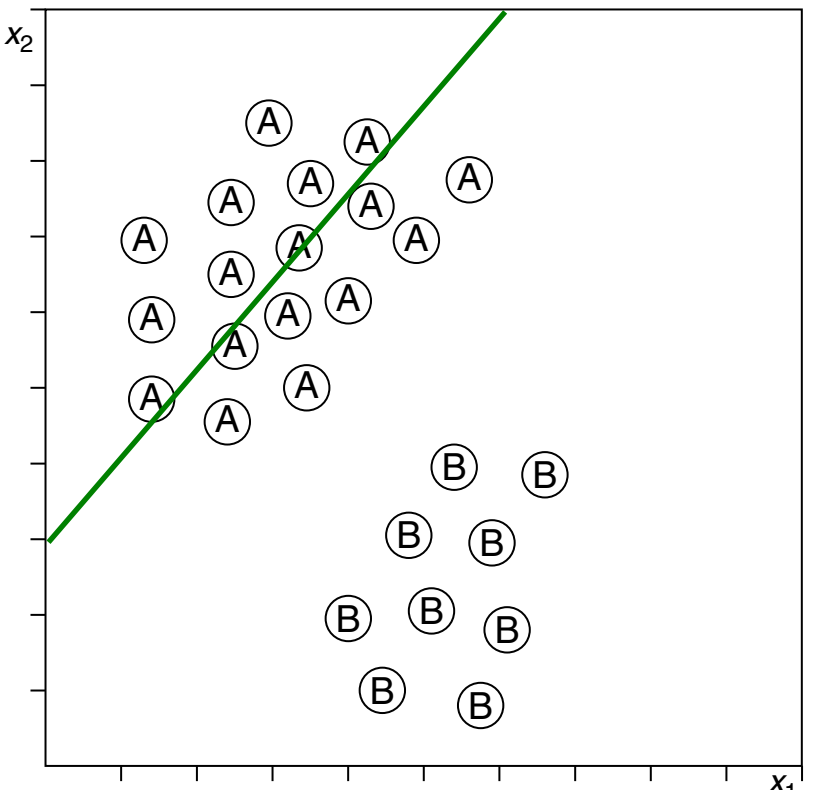
Example: Illustration in Input Space [PT algorithm]



© STEIN/VÖLSKE 2021

Perceptron Learning

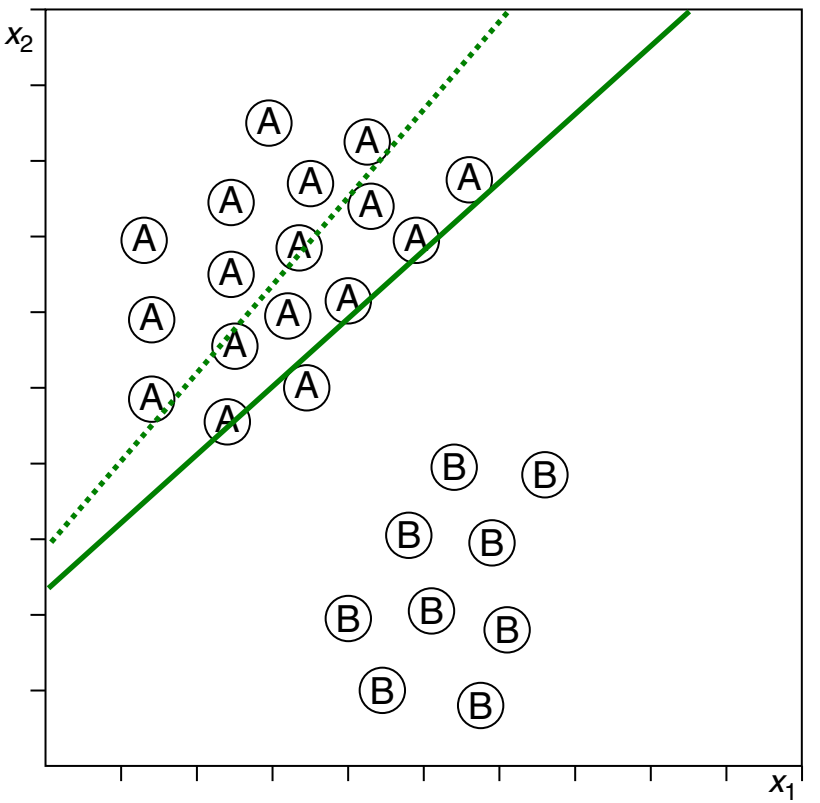
Example: Illustration in Input Space [PT algorithm]



© STEIN/VÖLSKE 2021

Perceptron Learning

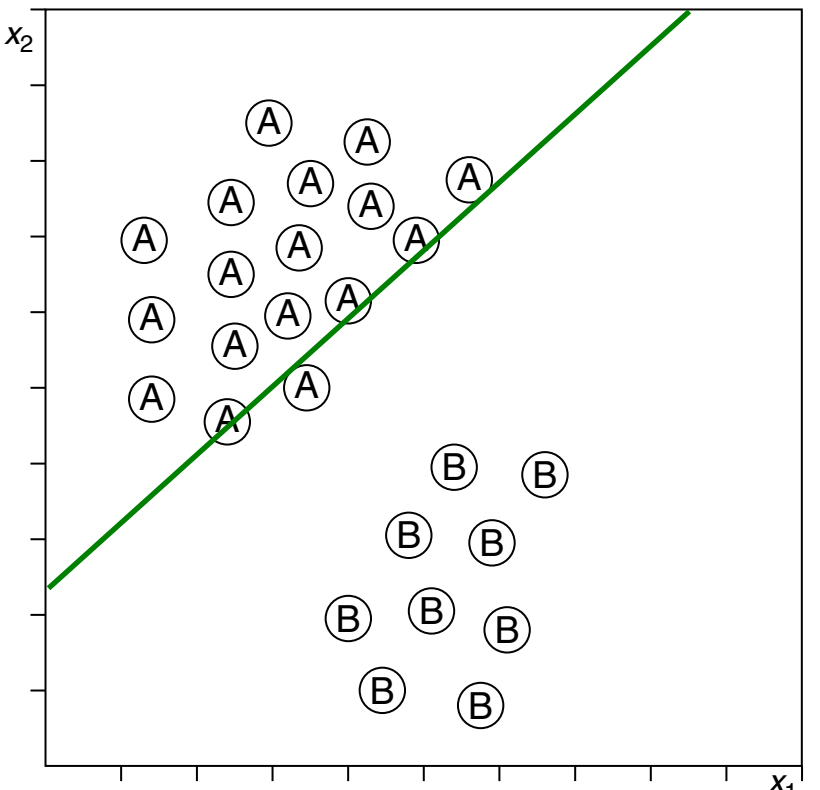
Example: Illustration in Input Space [PT algorithm]



© STEIN/VÖLSKE 2021

Perceptron Learning

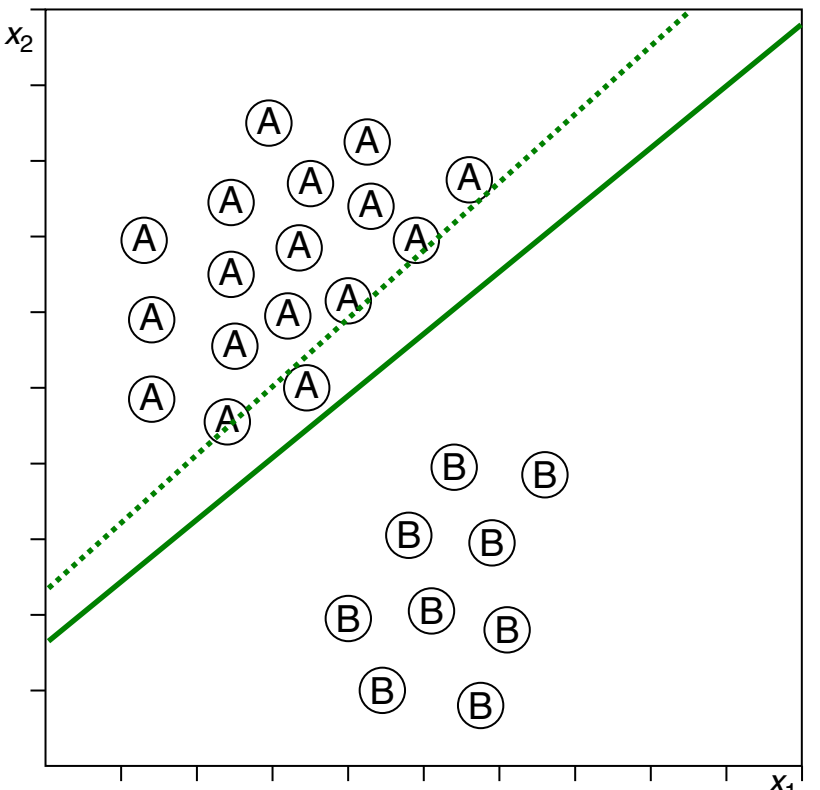
Example: Illustration in Input Space [PT algorithm]



© STEIN/VÖLSKE 2021

Perceptron Learning

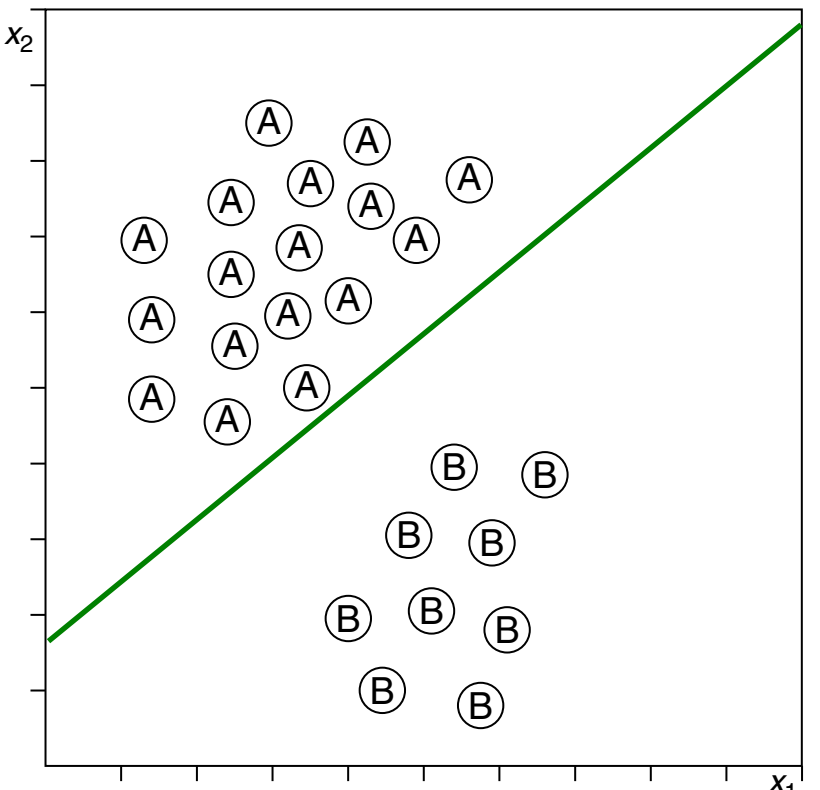
Example: Illustration in Input Space [PT algorithm]



© STEIN/VÖLSKE 2021

Perceptron Learning

Example: Illustration in Input Space [PT algorithm]



© STEIN/VÖLSKE 2021

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?

© STEIN/VÖLSKE 2021

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?

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.

© STEIN/VÖLSKE 2021

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 $\mathbf{x} \in X_1$.

© STEIN/VÖLSKE 2021

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 $\mathbf{x} \in X'$, $\delta \in \{0, 1\}$ (instead of $\{0, 1, -1\}$). [PT algorithm, Line 5]

© STEIN/VÖLSKE 2021

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 $\mathbf{x} \in X'$, $\delta \in \{0, 1\}$ (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 $\mathbf{w}(0)$.

© STEIN/VÖLSKE 2021

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 $\mathbf{x} \in X'$, $\delta \in \{0, 1\}$ (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.

© STEIN/VÖLSKE 2021

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.

© STEIN/VÖLSKE 2021

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].

© STEIN/VÖLSKE 2021

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 $\mathbf{w}(n)$:

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

© STEIN/VÖLSKE 2021

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 $\mathbf{w}(n)$:

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

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.

© STEIN/VÖLSKE 2021

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 $\widehat{\mathbf{w}}$:

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

© STEIN/VÖLSKE 2021

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 $\widehat{\mathbf{w}}$:

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

5. Apply the Cauchy-Schwarz inequality:

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

© STEIN/VÖLSKE 2021

Perceptron Learning

Perceptron Convergence Theorem: Proof (continued)

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

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

© STEIN/VÖLSKE 2021

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 :

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

© STEIN/VÖLSKE 2021

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

© STEIN/VÖLSKE 2021

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

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

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

© STEIN/VÖLSKE 2021

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 $||\mathbf{w}(0)||^2 + n\eta^2\varepsilon \in \Theta(n)$

© STEIN/VÖLSKE 2021

Perceptron Learning

Perceptron Convergence Theorem: Discussion [theorem]

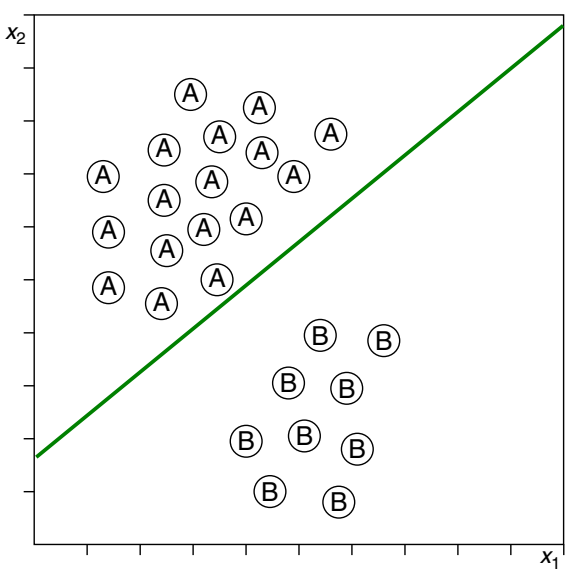
- If a separating hyperplane between X_0 and X_1 exists, the PT algorithm will converge. If no such hyperplane exists, convergence cannot be guaranteed.
- A separating hyperplane can be found in polynomial time with linear programming. The PT Algorithm, however, may require an exponential number of iterations.

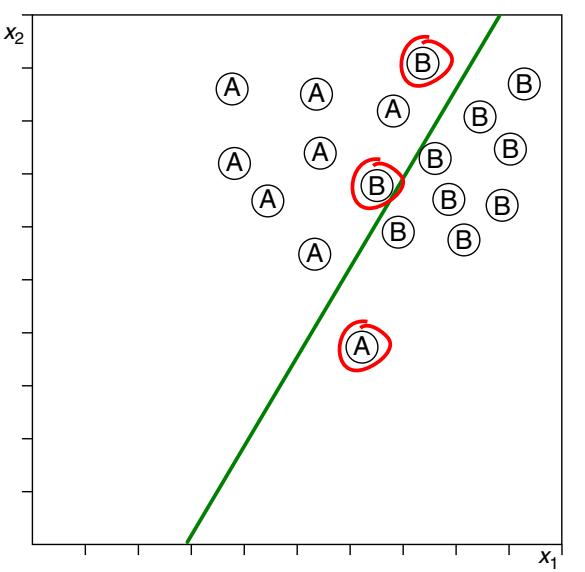
© STEIN/VÖLSKE 2021

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:





© STEIN/VÖLSKE 2021

Chapter ML:IV (continued)

IV. Neural Networks

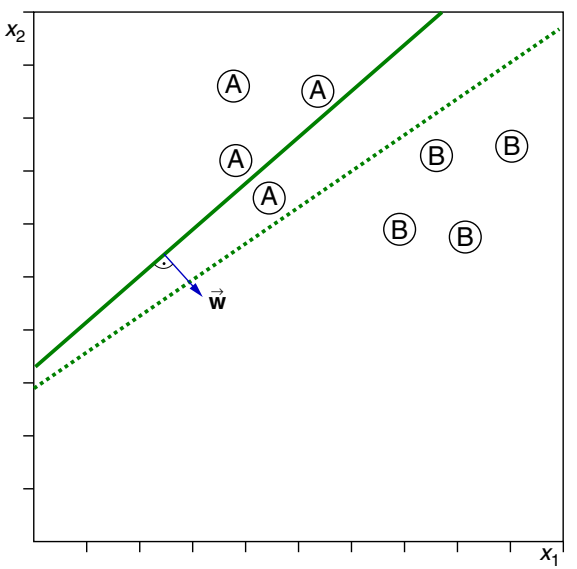
- Perceptron Learning
- Gradient Descent
- Multilayer Perceptron

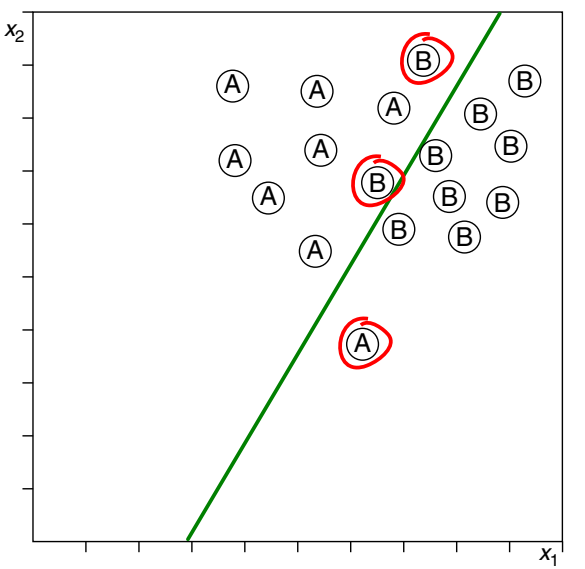
© STEIN/VÖLSKE 2021

Gradient Descent

Motivation

Given some $\mathbf{w}$, the PT algorithm checks if the examples $(\mathbf{x}, c(\mathbf{x})) \in D$ are on the correct *hyperplane side* and possibly adapts $\mathbf{w}$ (left). Goal is to find a separating hyperplane $\mathbf{w}$.





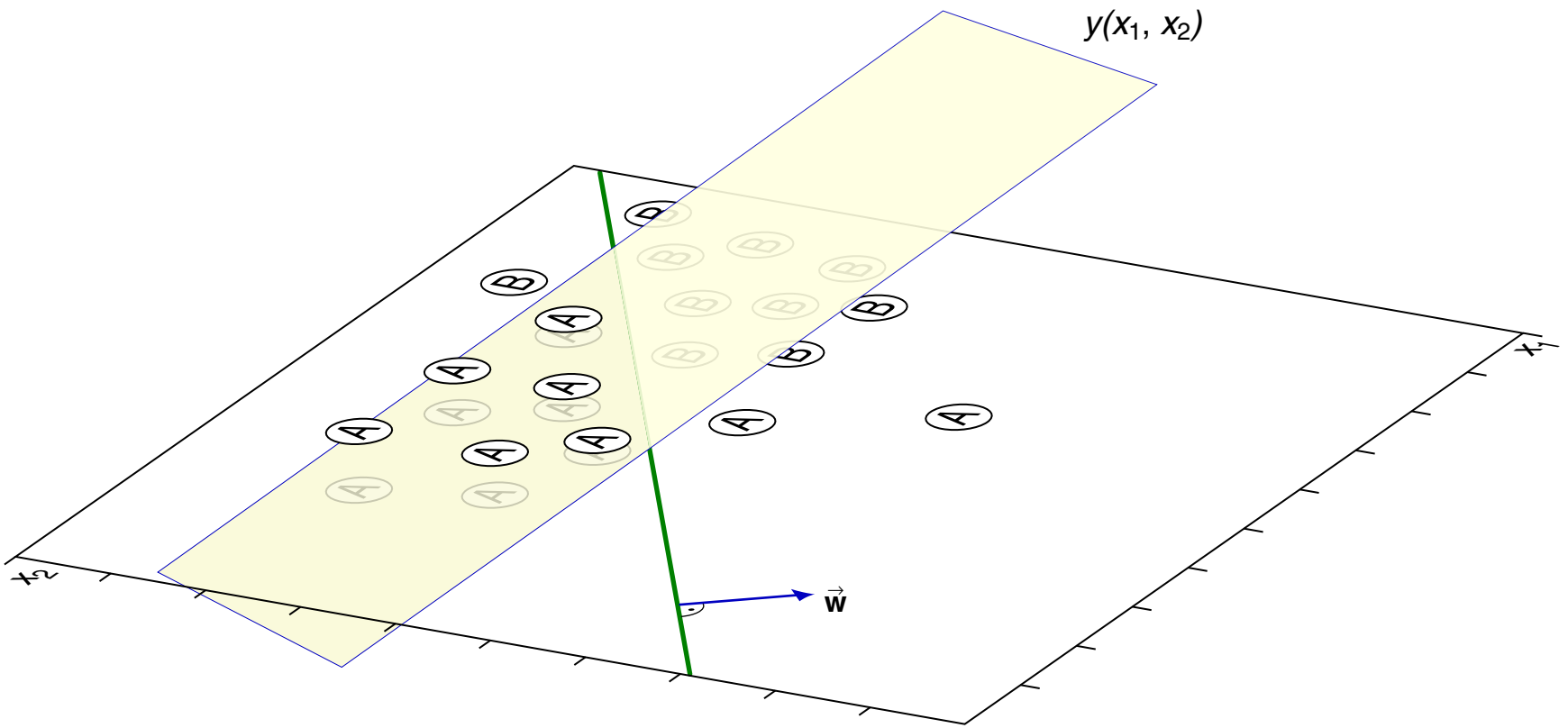
The PT algorithm fails if the classes are not linearly separable (right).

© STEIN/VÖLSKE 2021

Gradient Descent

Motivation (continued)

Given some $\mathbf{w}$, regression methods compute a loss, quantifying the "grade of misclassification", by exploiting both the hyperplane side *and the distance*, given the examples $(\mathbf{x}, c(\mathbf{x})) \in D$. Goal is to find a loss-minimum hyperplane $\mathbf{w}$.

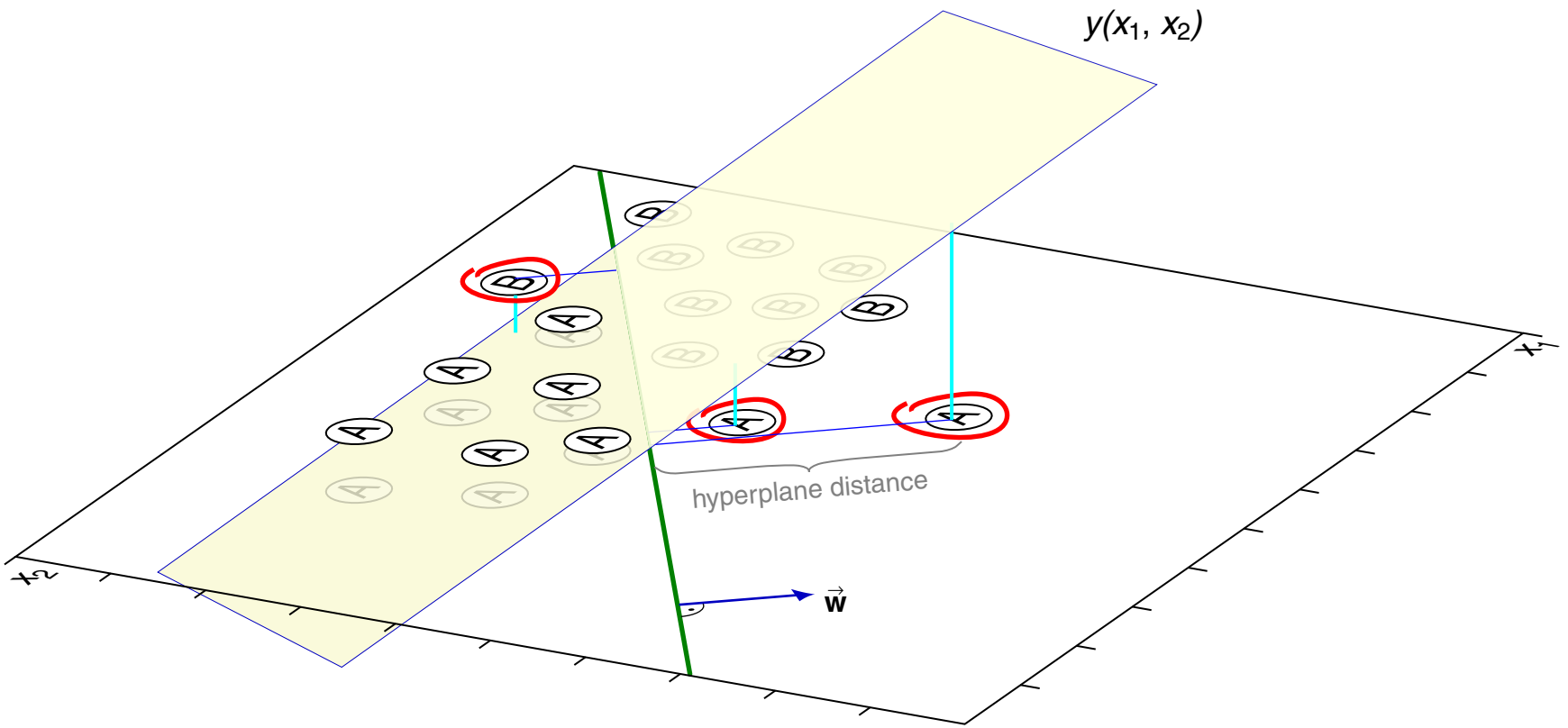


© STEIN/VÖLSKE 2021

Gradient Descent

Motivation (continued)

Given some $\mathbf{w}$, regression methods compute a loss, quantifying the "grade of misclassification", by exploiting both the hyperplane side *and the distance*, given the examples $(\mathbf{x}, c(\mathbf{x})) \in D$. Goal is to find a loss-minimum hyperplane $\mathbf{w}$.



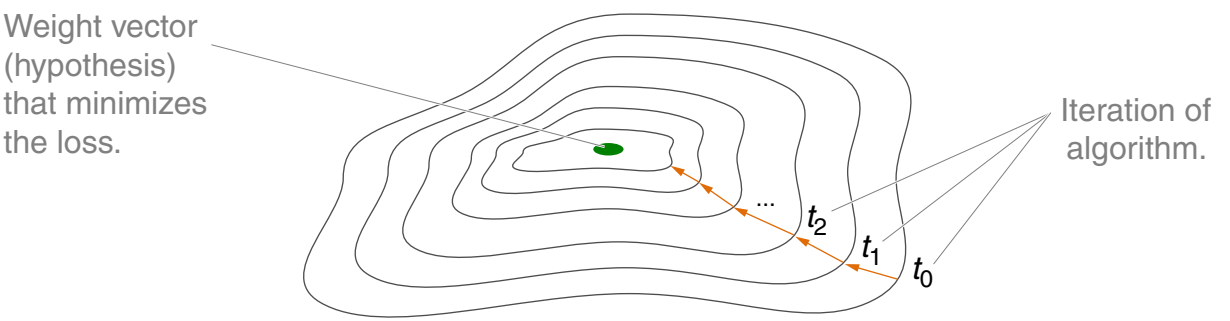
© STEIN/VÖLSKE 2021

Gradient Descent

Principle

Gradient descent, GD, is a first-order iterative optimization algorithm for finding a local extremum of a differentiable function f. [Wikipedia]

In our algorithms f is the global loss function, L, or some objective function, $\mathcal{L}$.



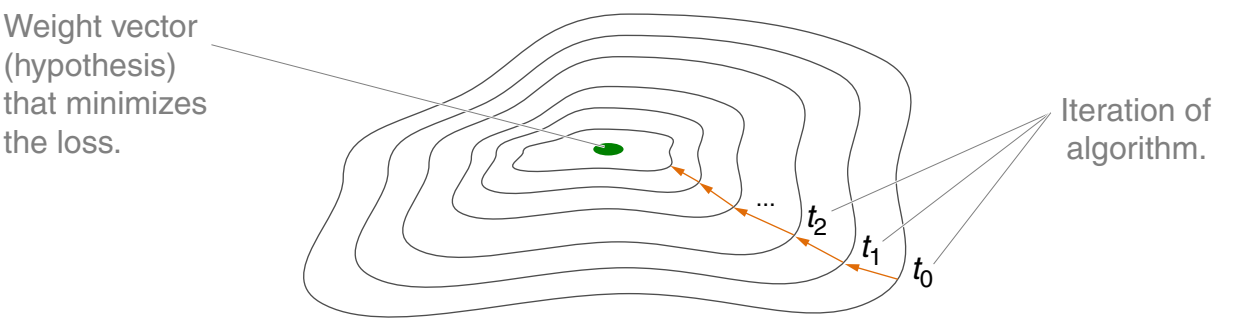
© STEIN/VÖLSKE 2021

Gradient Descent

Principle (continued)

Gradient descent, GD, is a first-order iterative optimization algorithm for finding a local extremum of a differentiable function f. [Wikipedia]

In our algorithms f is the global loss function, L, or some objective function, $\mathcal{L}$.



- The gradient ∇f of a differentiable function f of several variables is a vector whose components are the partial derivatives of f. (simplified definition)
- The gradient of a function is the direction of steepest ascent or descent.
- Gradient *ascent* means stepping in the direction of the gradient.
- Likewise, gradient *descent* means stepping in the opposite direction of the gradient; it will lead to a local minimum of that function.

© STEIN/VÖLSKE 2021

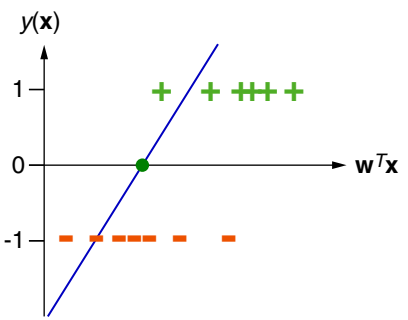
Remarks:

- We will apply the GD principle in three hypothesis search settings in order to take the direction of steepest descent for the respective loss and objective functions:
  (1) linear regression + squared loss
  (2) linear regression + 0/1 loss
  (3) logistic regression + logistic loss + regularization

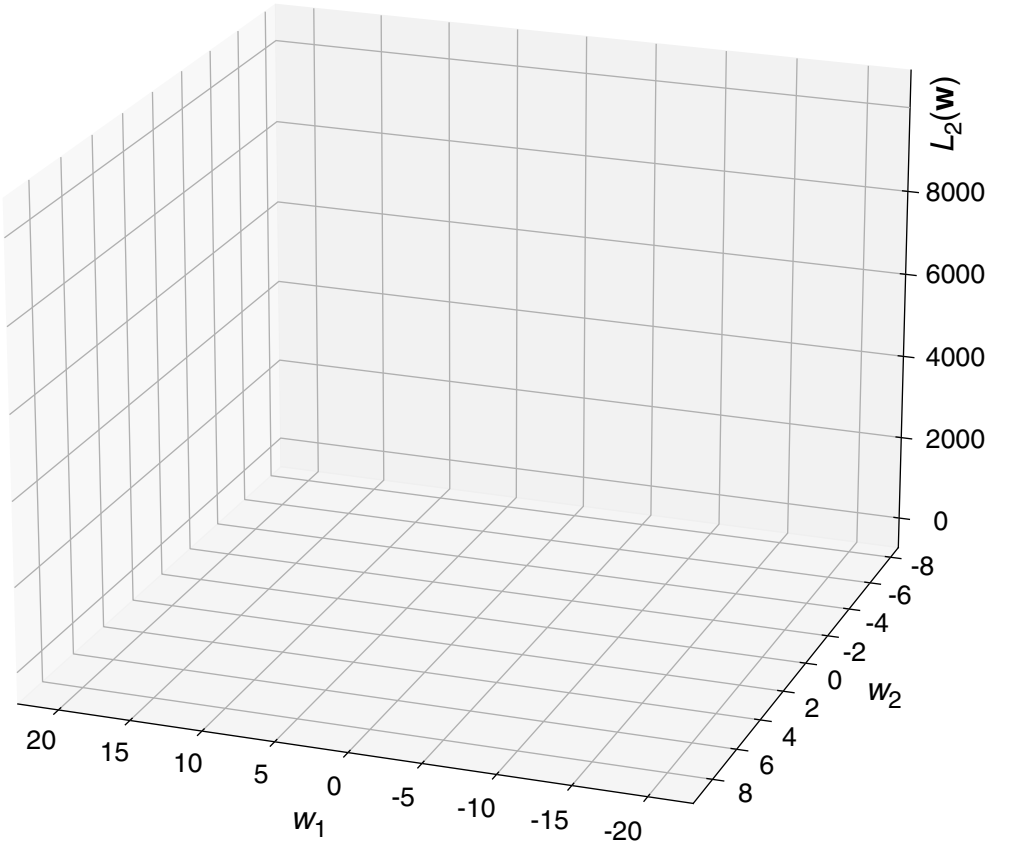
© STEIN/VÖLSKE 2021

Gradient Descent

(1) Linear Regression + Squared Loss



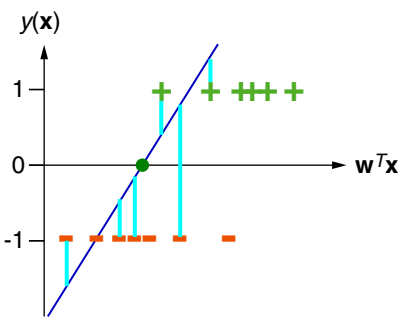
$$y(\mathbf{x}) \overset{(\star)}{=} \mathbf{w}^T \mathbf{x}$$

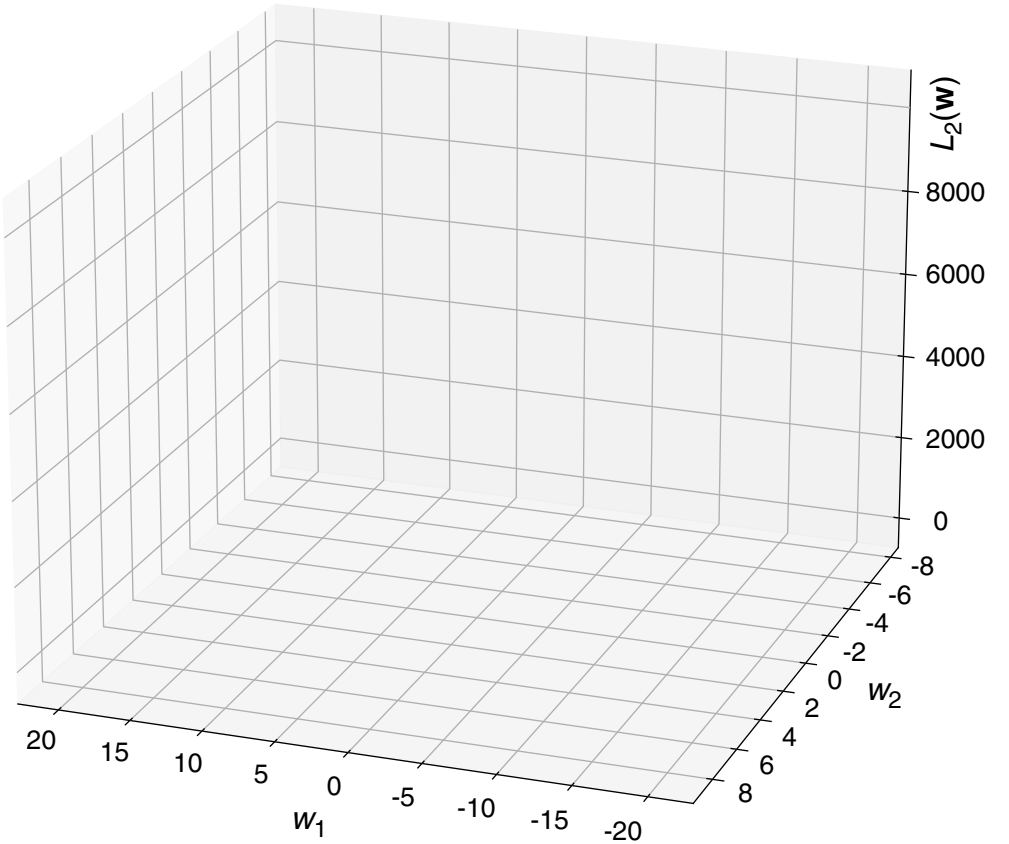


© STEIN/VÖLSKE 2021

Gradient Descent

(1) Linear Regression + Squared Loss (continued)





$$y(\mathbf{x}) \overset{(\star)}{=} \mathbf{w}^T \mathbf{x}$$

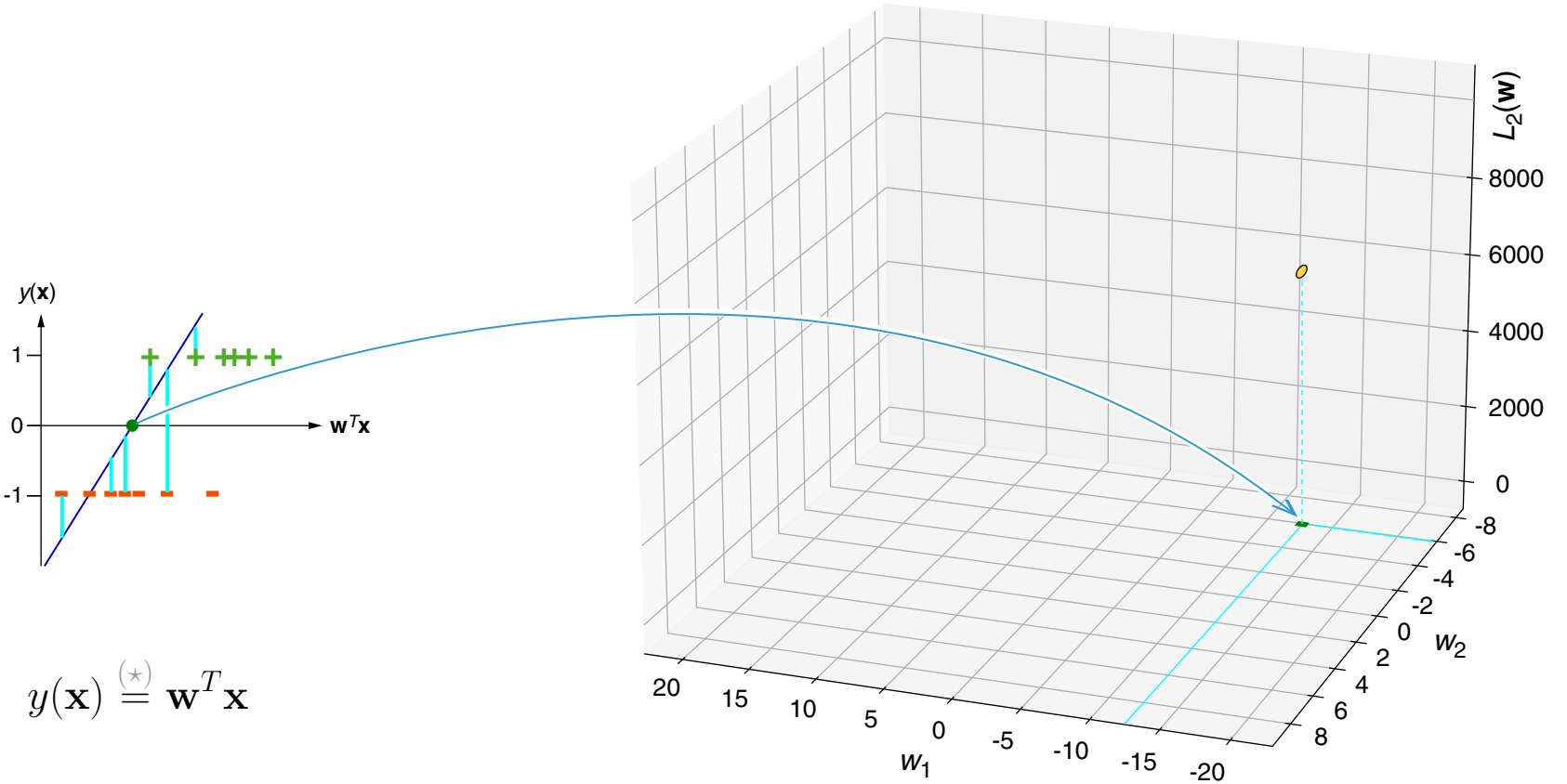
$$L_2(\mathbf{w}) \;=\; \frac{1}{2} \cdot \text{RSS}(\mathbf{w}) \;=\; \frac{1}{2} \cdot \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} \big(c(\mathbf{x}) - y(\mathbf{x})\big)^2$$

[pointwise squared loss]

© STEIN/VÖLSKE 2021

Gradient Descent

(1) Linear Regression + Squared Loss (continued)



$$y(\mathbf{x}) \overset{(\star)}{=} \mathbf{w}^T \mathbf{x}$$

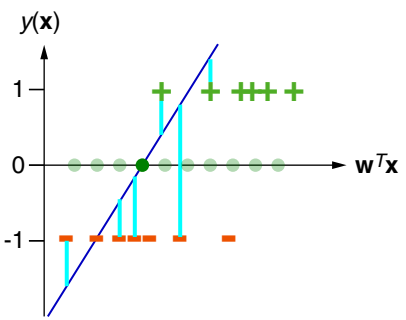
$$L_2(\mathbf{w}) \;=\; \frac{1}{2} \cdot \text{RSS}(\mathbf{w}) \;=\; \frac{1}{2} \cdot \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} \big(c(\mathbf{x}) - y(\mathbf{x})\big)^2$$

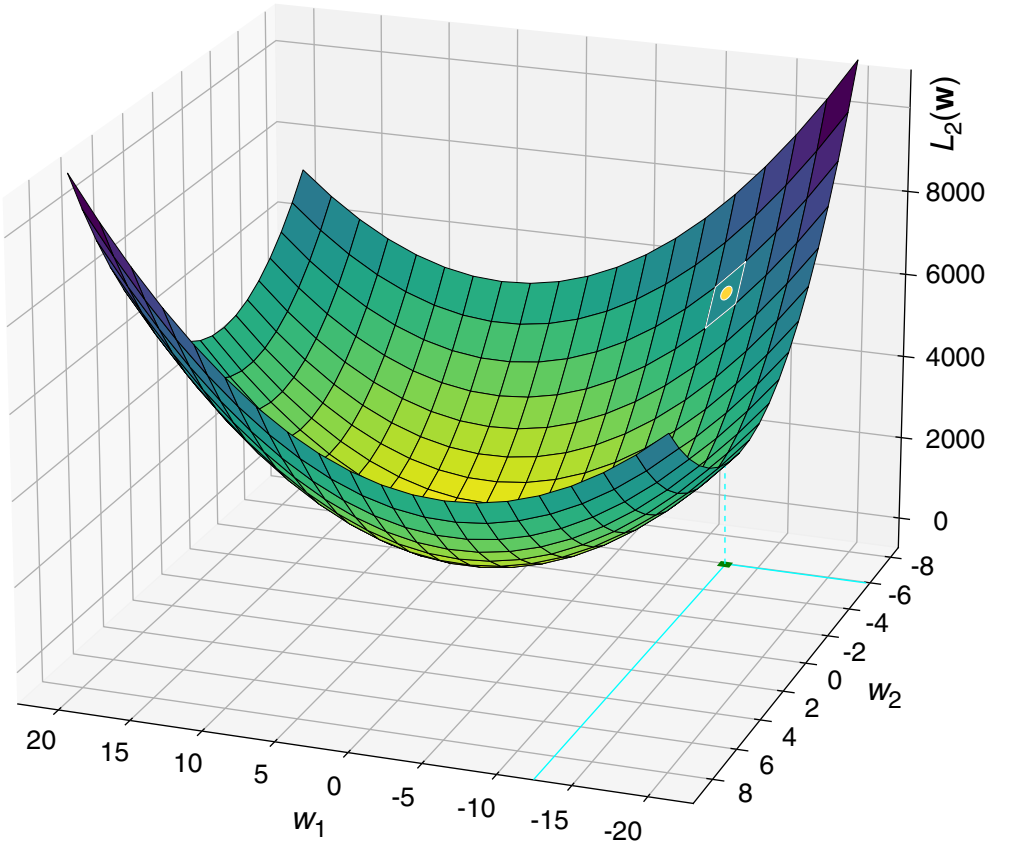
[pointwise squared loss]

© STEIN/VÖLSKE 2021

Gradient Descent

(1) Linear Regression + Squared Loss (continued)





$$y(\mathbf{x}) \overset{(\star)}{=} \mathbf{w}^T\mathbf{x}$$

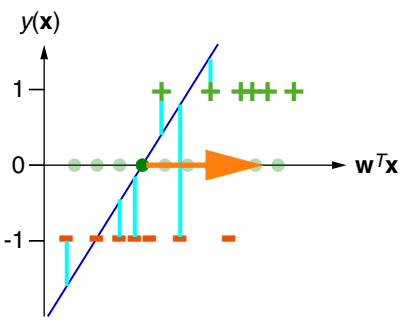
$$L_2(\mathbf{w}) \;=\; \frac{1}{2}\cdot \text{RSS}(\mathbf{w}) \;=\; \frac{1}{2}\cdot \sum_{(\mathbf{x},c(\mathbf{x}))\in D} \big(c(\mathbf{x}) - y(\mathbf{x})\big)^2$$

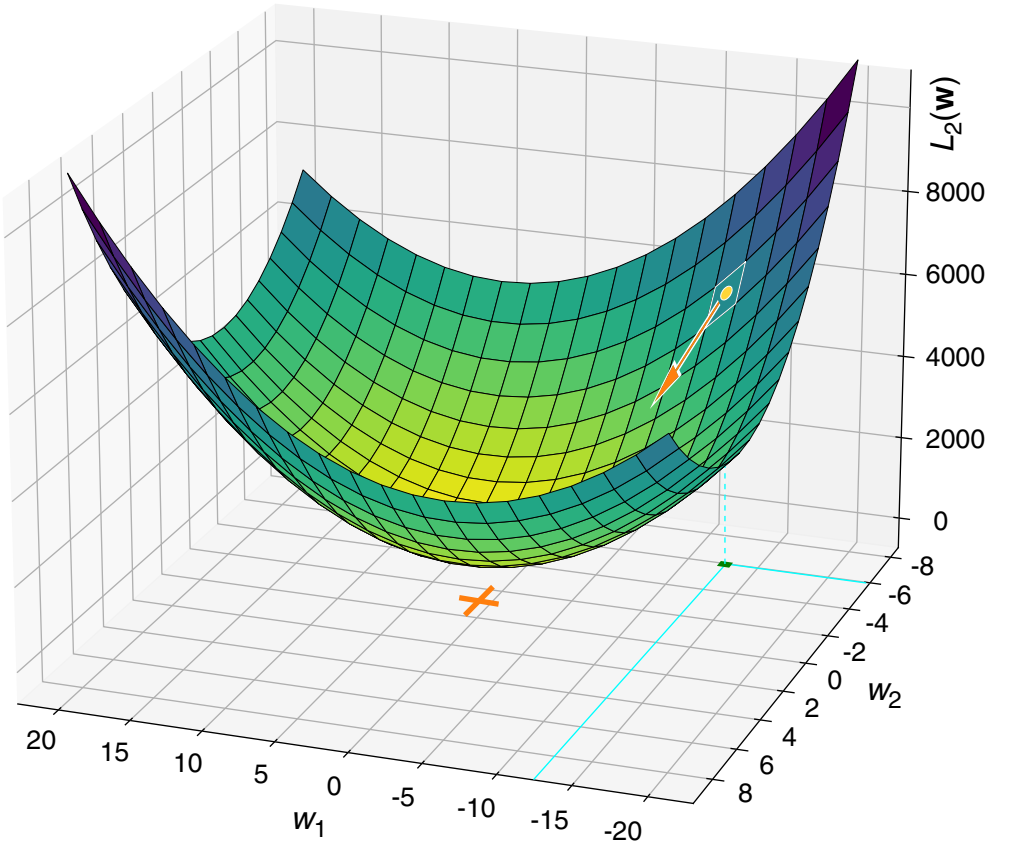
[pointwise squared loss]

© STEIN/VÖLSKE 2021

Gradient Descent

(1) Linear Regression + Squared Loss (continued)





$$y(\mathbf{x}) \overset{(\star)}{=} \mathbf{w}^T\mathbf{x}$$

$$L_2(\mathbf{w}) \;=\; \frac{1}{2}\cdot \mathrm{RSS}(\mathbf{w}) \;=\; \frac{1}{2}\cdot \sum_{(\mathbf{x},c(\mathbf{x}))\in D} \big(c(\mathbf{x}) - y(\mathbf{x})\big)^2$$

[pointwise squared loss]

$$\nabla L_2(\mathbf{w}) \;=\; \left(\frac{\partial L_2(\mathbf{w})}{\partial w_0}, \frac{\partial L_2(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w_p}\right)^T$$

© STEIN/VÖLSKE 2021

Gradient Descent

(1) Linear Regression + Squared Loss (continued)

Update of weight vector $\mathbf{w}$: (BGD$_2$ algorithm, Line 7)

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

using the gradient of the loss function $L_2(\mathbf{w})$ to take steepest descent:

$$\Delta\mathbf{w} \;=\; -\eta \cdot \nabla L_2(\mathbf{w})$$

© STEIN/VÖLSKE 2021

Gradient Descent

(1) Linear Regression + Squared Loss (continued)

Update of weight vector $\mathbf{w}$: (BGD$_2$ algorithm, Line 7)

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

using the gradient of the loss function $L_2(\mathbf{w})$ to take steepest descent:

$$\begin{aligned}
\Delta\mathbf{w} &= -\eta \cdot \nabla L_2(\mathbf{w}) \\
&= -\eta \cdot \left(\frac{\partial L_2(\mathbf{w})}{\partial w_0}, \frac{\partial L_2(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w_p} \right)^T \\
&\vdots \\
&= \eta \cdot \sum_D \left(c(\mathbf{x}) - \mathbf{w}^T\mathbf{x} \right) \cdot \mathbf{x}
\end{aligned}$$

© STEIN/VÖLSKE 2021

Remarks (how to derive $\nabla L_2(\mathbf{w})$) :

- Consider the partial derivative for a parameter $w_j,\ j = 0, \ldots, p$:

$$
\begin{aligned}
\frac{\partial}{\partial w_j} L_2(\mathbf{w}) &= \frac{\partial}{\partial w_j} \frac{1}{2} \cdot \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} (c(\mathbf{x}) - y(\mathbf{x}))^2 \\
&= \frac{1}{2} \cdot \sum_{D} \frac{\partial}{\partial w_j} (c(\mathbf{x}) - y(\mathbf{x}))^2 \\
&= \sum_{D} (c(\mathbf{x}) - y(\mathbf{x})) \cdot \frac{\partial}{\partial w_j} (c(\mathbf{x}) - y(\mathbf{x})) \\
&= \sum_{D} (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) \cdot \frac{\partial}{\partial w_j} (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) \quad \text{// } j\text{-th summand depends on } w_j. \\
&= \sum_{D} (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) \cdot (-x_j) \\
&= -\sum_{D} (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) \cdot x_j
\end{aligned}
$$

- Plugging the results for $\frac{\partial}{\partial w_j} L_2(\mathbf{w})$ into $-\eta \cdot (\ldots)^T$ yields the update formula for $\Delta \mathbf{w}$.

© STEIN/VÖLSKE 2021

Gradient Descent

The BGD Algorithm

[Algorithms: LMS BGD$_\sigma$ PT BGD$_2$ IGD]

Algorithm:	BGD$_2$	Batch Gradient Descent
Input:	D	Training examples $(\mathbf{x}, c(\mathbf{x}))$ with $\mathbf{x} \in \mathbf{R}^p$, $c(\mathbf{x}) \in \{-1, 1\}$.
	η	Learning rate, a small positive constant.
Output:	$\mathbf{w}$	Weight vector from $\mathbf{R}^{p+1}$ (= hypothesis).

BGD(D, η)

```
initialize_random_weights(w),  t = 0
REPEAT
  t = t + 1
  Δw = 0
  FOREACH (x, c(x)) ∈ D DO
    δ = c(x) − y(x)      // y(x) ⁽ˣ⁾= wᵀx,  δ ∈ R.
    Δw ⁽ˣ⁾= Δw + η · δ · x    // δ · x is the derivative of ℓ₂(c(x), y(x)).
  ENDDO
  w = w + Δw     // Δw is −η · ∇L₂(w) here.
UNTIL(convergence(c(D), y(D)) OR t > t_max)
return(w)
```


© STEIN/VÖLSKE 2021

Remarks:

($\star$) Recap. When used as operand in a scalar product with the weight vector, $\mathbf{w}^T\mathbf{x}$, we consider the feature vector $\mathbf{x}$ in its extended form. I.e., $x_0 = 1$, and $\mathbf{x} = (1, x_1, \ldots, x_p)^T \in \mathbf{R}^{p+1}$.

- Recap. Each BGD iteration "`REPEAT ... UNTIL`"
  1. computes the direction of steepest loss descent as $-\nabla L_2(\mathbf{w}_t) = \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} (c(\mathbf{x}) - y_t(\mathbf{x})) \cdot \mathbf{x}$, and
  2. updates $\mathbf{w}_t$ by taking a step of length η in this direction.

- Recap. The *convergence* function may analyze the global squared loss, $L_2(\mathbf{w}_t)$, or the norm of the loss gradient, $||\nabla L_2(\mathbf{w}_t)||$, and compare it to some small positive bound ε. In this regard $c(D)$ and $y(D)$ denote the vectors of target classes and computed values respectively.

- The basic gradient descent is a first-order optimization method, and its speed of convergence may be considered unsatisfactory. Even when taking the optimal step size η at each iteration, it has merely a linear rate of convergence. [Meza 2010]

  More advanced numerical algorithms to tackle the optimization (search for minimum L_2) are listed below but are not treated in detail here:

  - Newton-Raphson algorithm
  - BFGS algorithm (Broyden-Fletcher-Goldfarb-Shanno)
  - L-BFGS algorithm (limited memory BFGS)
  - conjugate gradient method

© STEIN/VÖLSKE 2021

Gradient Descent

Global Loss versus Pointwise Loss

The weight adaptation of the BGD algorithm computes in each iteration the global loss, i.e., the loss of *all* examples in D ("batch gradient descent").

The (squared) loss with regard to a *single* example $(\mathbf{x}, c(\mathbf{x})) \in D$, also called pointwise loss is given as:

$$\ell_2(c(\mathbf{x}), y(\mathbf{x})) \;=\; \frac{1}{2}\,(c(\mathbf{x}) - \mathbf{w}^T\mathbf{x})^2 \qquad \text{[global squared loss]}$$

The respective weight adaptation computes canonically as follows:

$$\Delta\mathbf{w} \;=\; \eta \cdot (c(\mathbf{x}) - \mathbf{w}^T\mathbf{x}) \cdot \mathbf{x} \qquad \text{[IGD algorithm]}$$

© STEIN/VÖLSKE 2021

Remarks:

- The adaptation rule for single example, $\Delta \mathbf{w} = \eta \cdot (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) \cdot \mathbf{x}$, is known under different names:
  - delta rule
  - Widrow-Hoff rule
  - adaline rule
  - least mean squares (LMS) rule

- The delta rule gives rise to the IGD algorithm (incremental gradient descent), which is introduced in the following. Moreover, the delta rule forms the basis of the backpropagation algorithm.

© STEIN/VÖLSKE 2021

Gradient Descent

The IGD Algorithm

[Algorithms: LMS BGD$_\sigma$ PT BGD$_2$ IGD]

Algorithm:	IGD	Incremental Gradient Descent
Input:	D	Training examples $(\mathbf{x}, c(\mathbf{x}))$ with $\mathbf{x} \in \mathbf{R}^p$, $c(\mathbf{x}) \in \{-1, 1\}$.
	η	Learning rate, a small positive constant.
Output:	$\mathbf{w}$	Weight vector from $\mathbf{R}^{p+1}$ (= hypothesis).

IGD(D, η)

```
initialize_random_weights(w),  t = 0
REPEAT
  t = t + 1
  FOREACH (x, c(x)) ∈ D DO
    δ = c(x) − y(x)      // y(x) ⁽ˢ⁾= wᵀx,  δ ∈ R.
    Δw ⁽ˢ⁾= η · δ · x    // δ · x is the derivative of ℓ₂(c(x), y(x)).
    w = w + Δw
  ENDDO
UNTIL(convergence(c(D), y(D)) OR t > t_max)
return(w)
```


© STEIN/VÖLSKE 2021

Remarks (IGD) :

- The sequence of incremental weight adaptations approximates the gradient descent of the batch approach. If η is chosen sufficiently small this approximation can be done at arbitrary precision.
- The computation of the global loss, $L_2(\mathbf{w})$, of batch gradient descent enables larger weight adaptation steps.
- 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 loss function.
- When, as is done here, the residual sum of squares, RSS, is chosen as loss function, the incremental gradient descent algorithm is very similar to the LMS algorithm.
- A related method to incremental gradient descent is *stochastic* gradient descent, which estimates the gradient from a randomly selected subset of the data.

© STEIN/VÖLSKE 2021

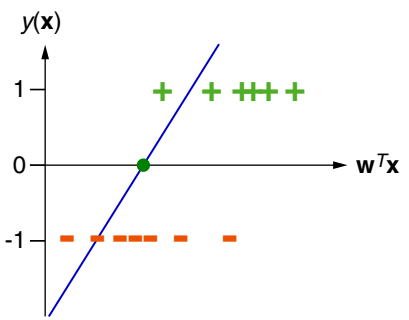
Remarks (GD versus PT) :

- The incremental gradient descent algorithm looks similar to the PT algorithm, since these algorithms differ only in the computation of δ (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 which exploits the residua provided by a loss function of choice, whose differential is evaluated to guide hyperplane search.
  - 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. [theorem]
  - Gradient descent may converge even if the data is not linearly separable. However, this convergence process is of an asymptotic nature and no finite iteration bound can be stated.
- 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).

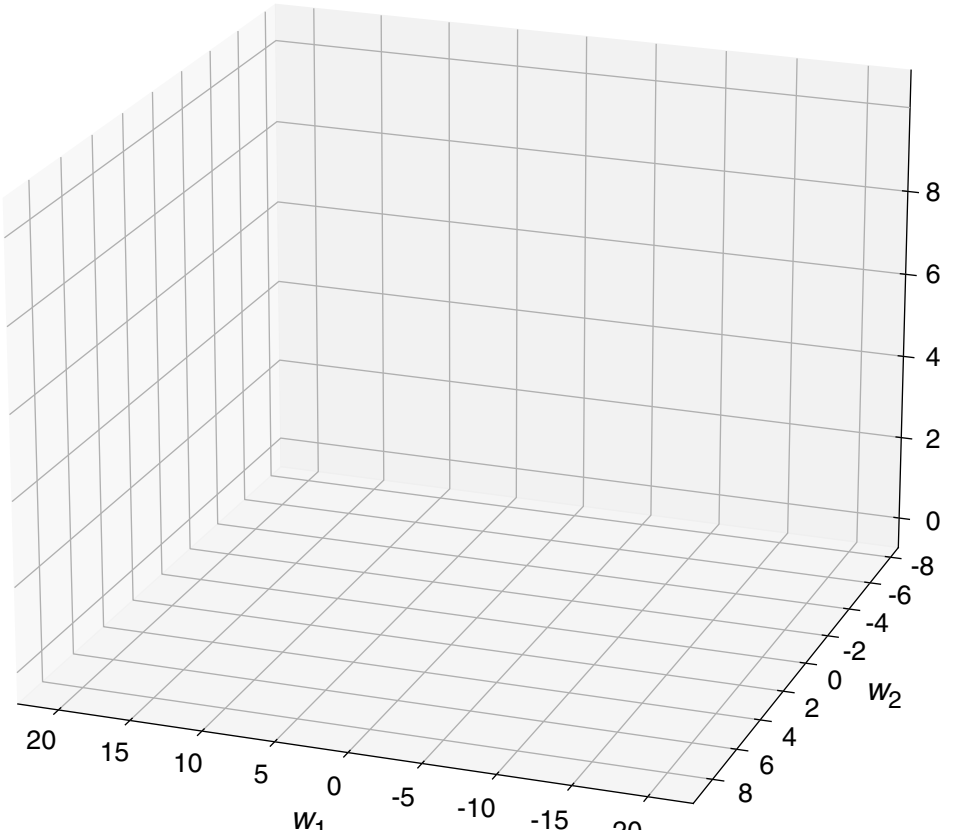
© STEIN/VÖLSKE 2021

Gradient Descent

(2) Linear Regression + 0/1 Loss



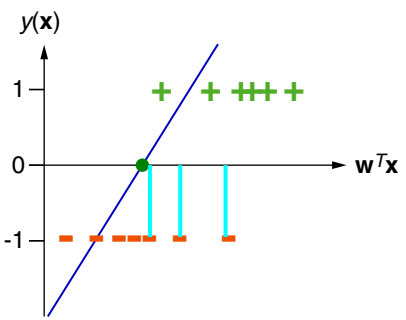
$$y(\mathbf{x}) \overset{(\star)}{=} \mathbf{w}^T \mathbf{x}$$

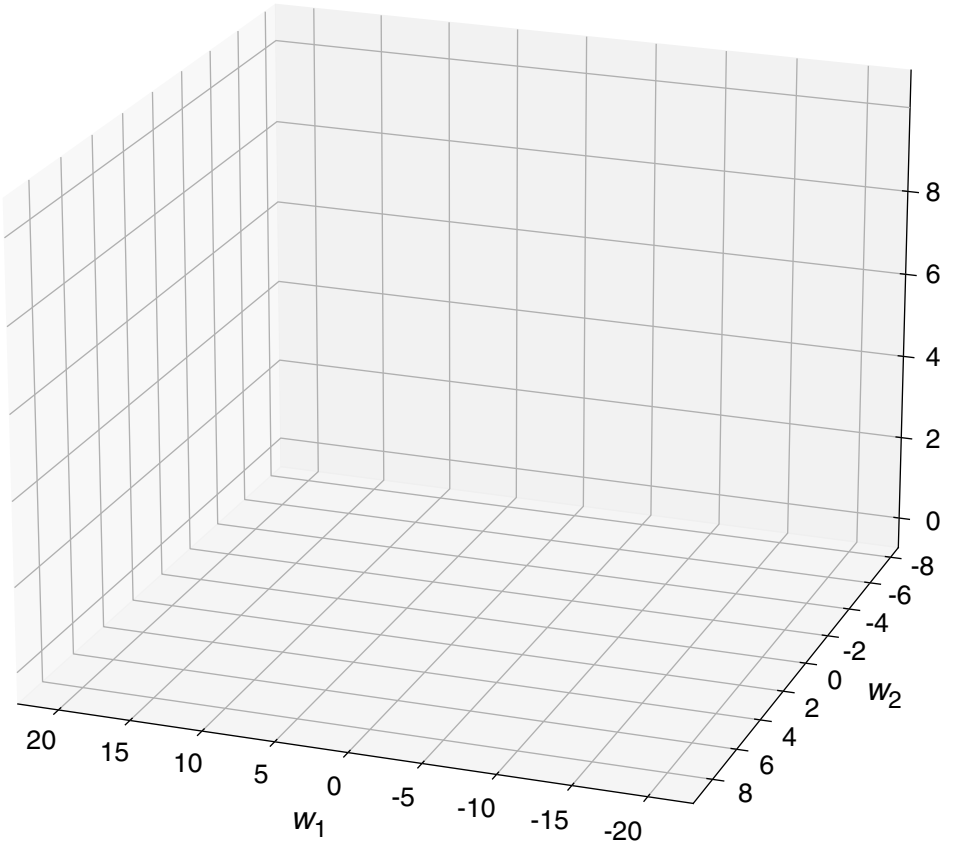


© STEIN/VÖLSKE 2021

Gradient Descent

(2) Linear Regression + 0/1 Loss (continued)





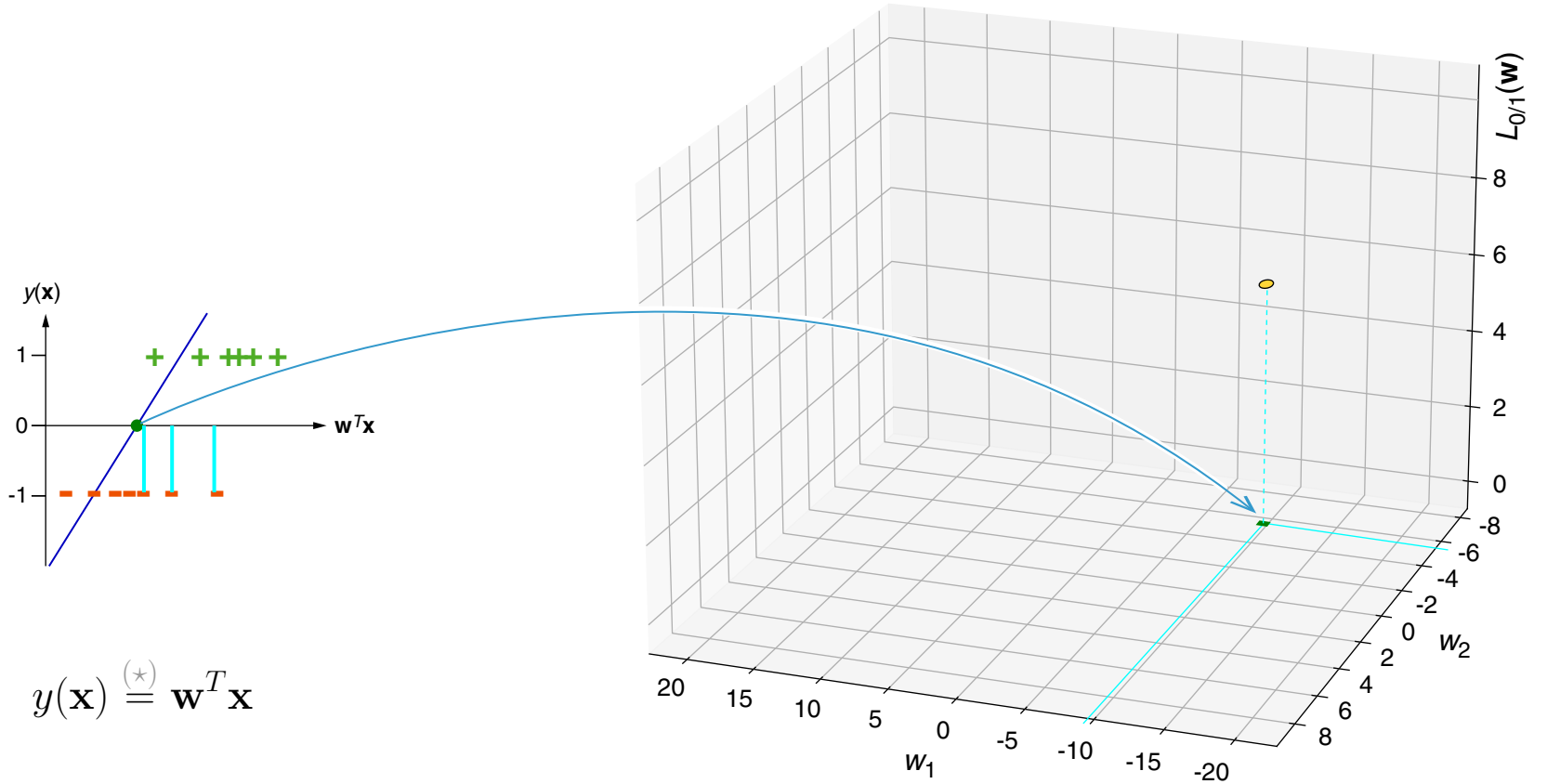
$$y(\mathbf{x}) \overset{(\star)}{=} \mathbf{w}^T\mathbf{x}$$

$$L_{0/1}(\mathbf{w}) \;=\; \sum_{(\mathbf{x},c(\mathbf{x}))\in D} I(c(\mathbf{x}), \text{sign}(y(\mathbf{x}))) \;=\; \sum_{D} \frac{1}{2}\cdot(c(\mathbf{x}) - \text{sign}(\mathbf{w}^T\mathbf{x}))$$

© STEIN/VÖLSKE 2021

Gradient Descent

(2) Linear Regression + 0/1 Loss (continued)



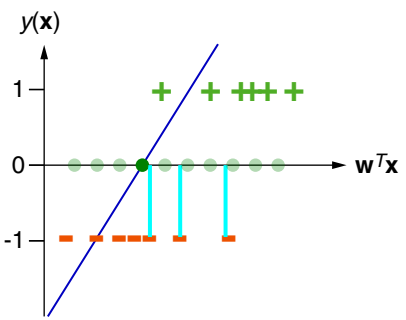
$$y(\mathbf{x}) \overset{(\star)}{=} \mathbf{w}^T\mathbf{x}$$

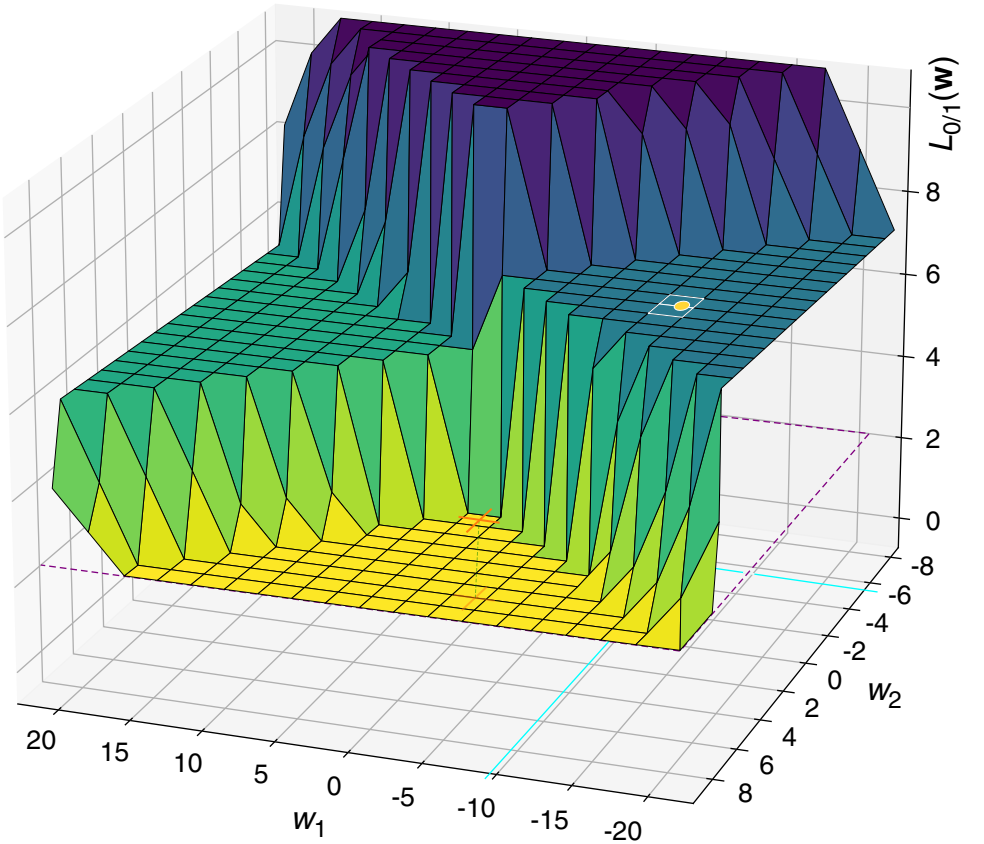
$$L_{0/1}(\mathbf{w}) \;=\; \sum_{(\mathbf{x},c(\mathbf{x}))\in D} I(c(\mathbf{x}), \text{sign}(y(\mathbf{x}))) \;=\; \sum_{D} \frac{1}{2}\cdot(c(\mathbf{x}) - \text{sign}(\mathbf{w}^T\mathbf{x}))$$

© STEIN/VÖLSKE 2021

Gradient Descent

(2) Linear Regression + 0/1 Loss (continued)





$$y(\mathbf{x}) \overset{(\star)}{=} \mathbf{w}^T\mathbf{x}$$

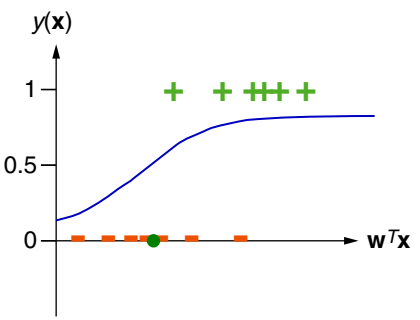
$$L_{0/1}(\mathbf{w}) \;=\; \sum_{(\mathbf{x},c(\mathbf{x}))\in D} I(c(\mathbf{x}), \text{sign}(y(\mathbf{x}))) \;=\; \sum_{D} \frac{1}{2}\cdot(c(\mathbf{x}) - \text{sign}(\mathbf{w}^T\mathbf{x}))$$

$\nabla L_{0/1}(\mathbf{w})$ cannot be expressed as a differentiable function.

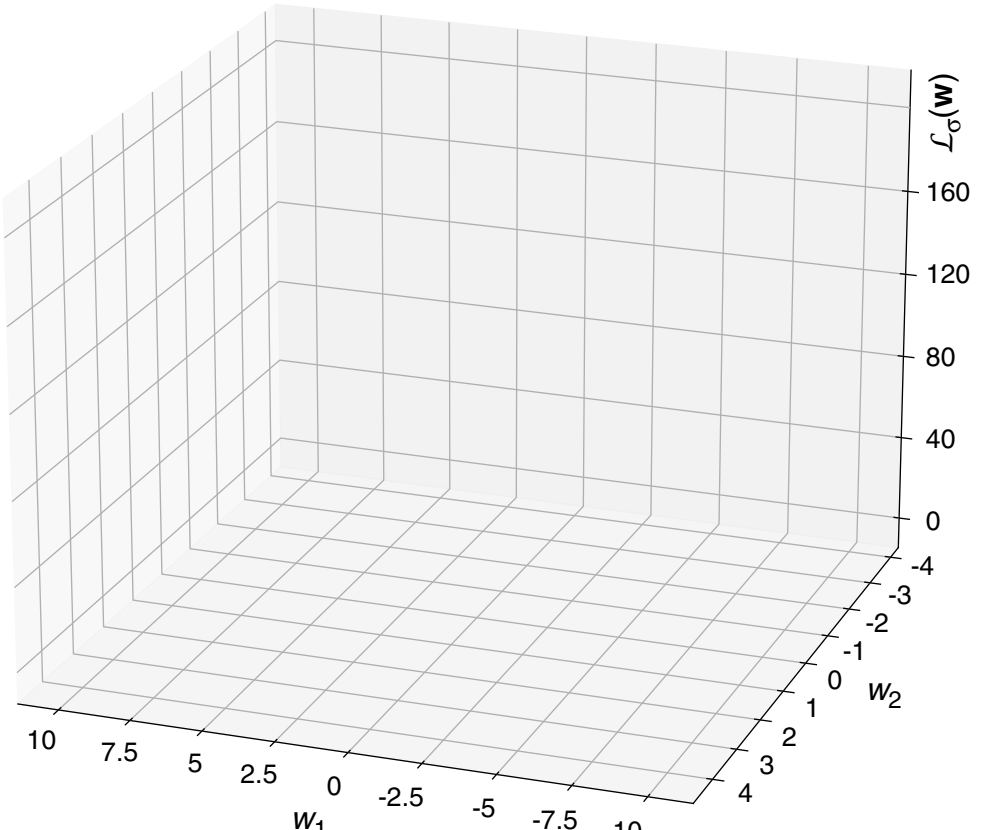
© STEIN/VÖLSKE 2021

Gradient Descent

(3) Logistic Regression + Logistic Loss + Regularization



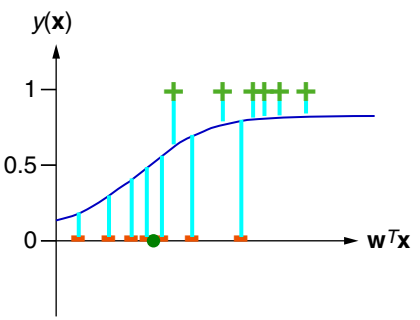
$$y(\mathbf{x}) \overset{(\star)}{=} \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}$$

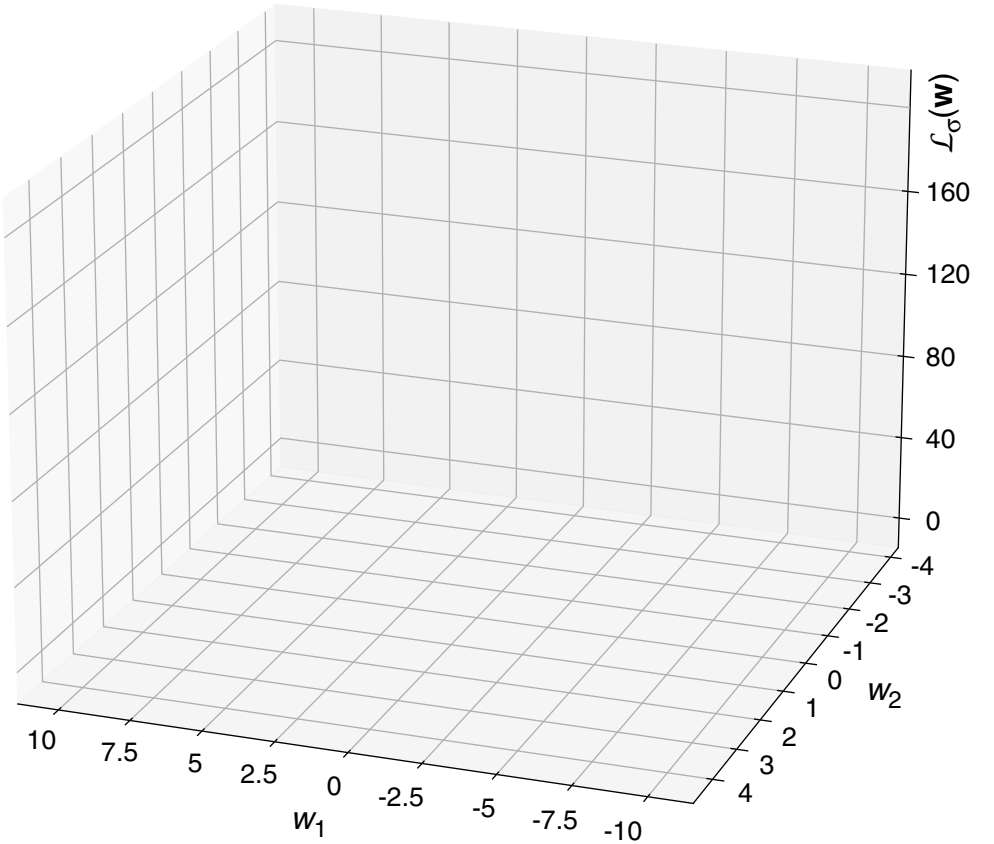


© STEIN/VÖLSKE 2021

Gradient Descent

(3) Logistic Regression + Logistic Loss + Regularization (continued)





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

$$\mathcal{L}_\sigma(\mathbf{w}) = L_\sigma + \lambda \cdot R_{\|\vec{\mathbf{w}}\|_2^2} = \sum_{(\mathbf{x},c(\mathbf{x}))\in D} \ell_\sigma(c(\mathbf{x}), y(\mathbf{x})) + \lambda \cdot \vec{\mathbf{w}}^T \vec{\mathbf{w}}$$

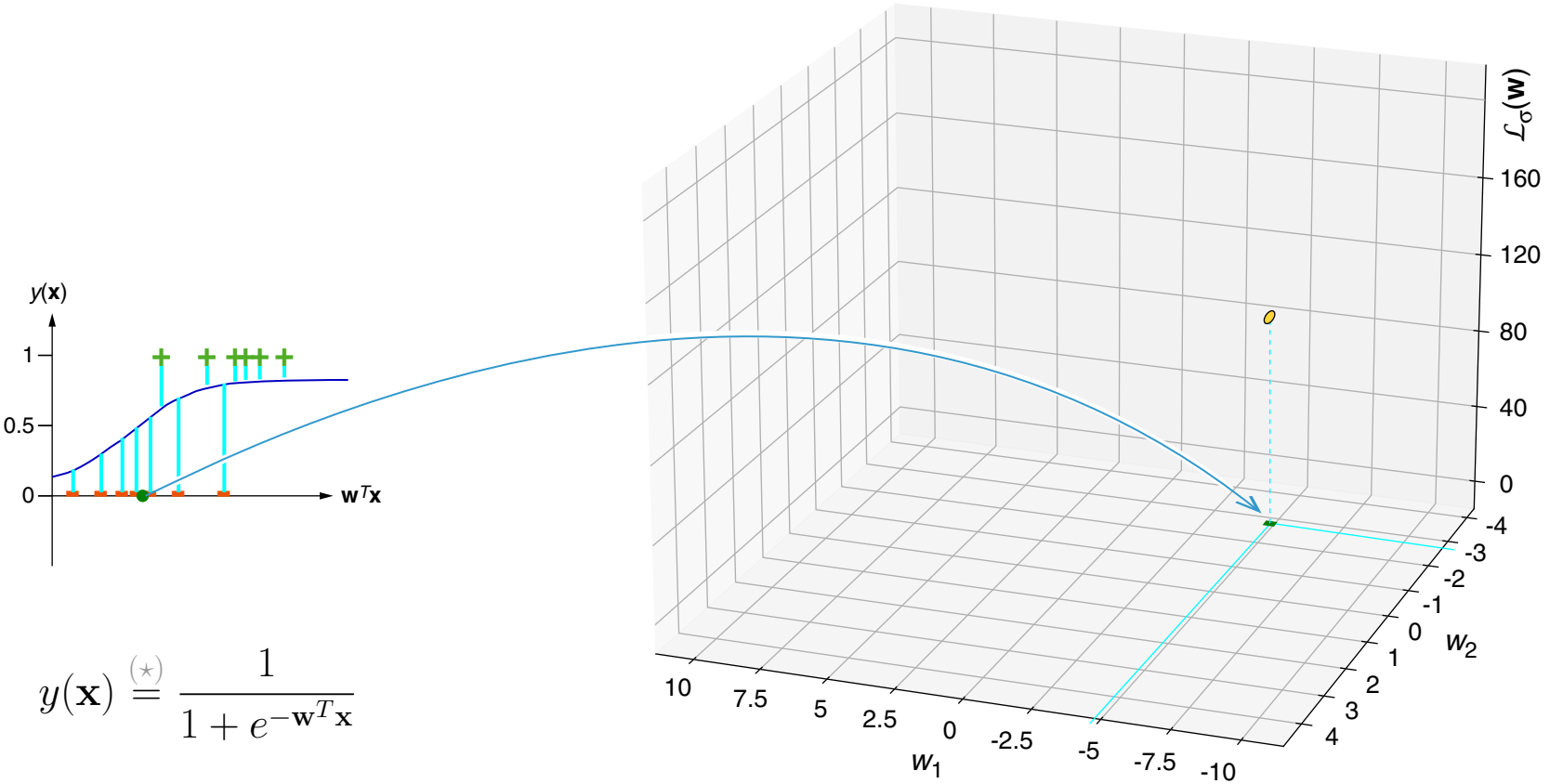
[Definitions: L, ℓ_σ, $R_{\|\vec{\mathbf{w}}\|_2^2}$]

$$= \sum_{D} -c(\mathbf{x}) \cdot \log(y(\mathbf{x})) - (1-c(\mathbf{x})) \cdot \log(1-y(\mathbf{x})) + \lambda \cdot \vec{\mathbf{w}}^T \vec{\mathbf{w}}$$

© STEIN/VÖLSKE 2021

Gradient Descent

(3) Logistic Regression + Logistic Loss + Regularization (continued)



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

$$\mathcal{L}_\sigma(\mathbf{w}) = L_\sigma + \lambda \cdot R_{\|\vec{\mathbf{w}}\|_2^2} = \sum_{(\mathbf{x},c(\mathbf{x}))\in D} \ell_\sigma(c(\mathbf{x}), y(\mathbf{x})) + \lambda \cdot \vec{\mathbf{w}}^T\vec{\mathbf{w}}$$

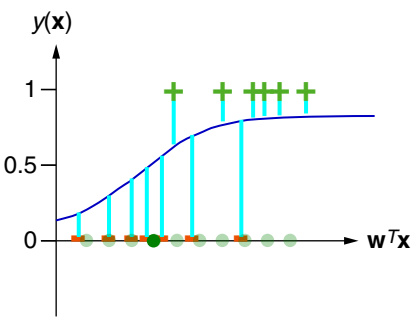
[Definitions: L, ℓ_σ, $R_{\|\vec{\mathbf{w}}\|_2^2}$]

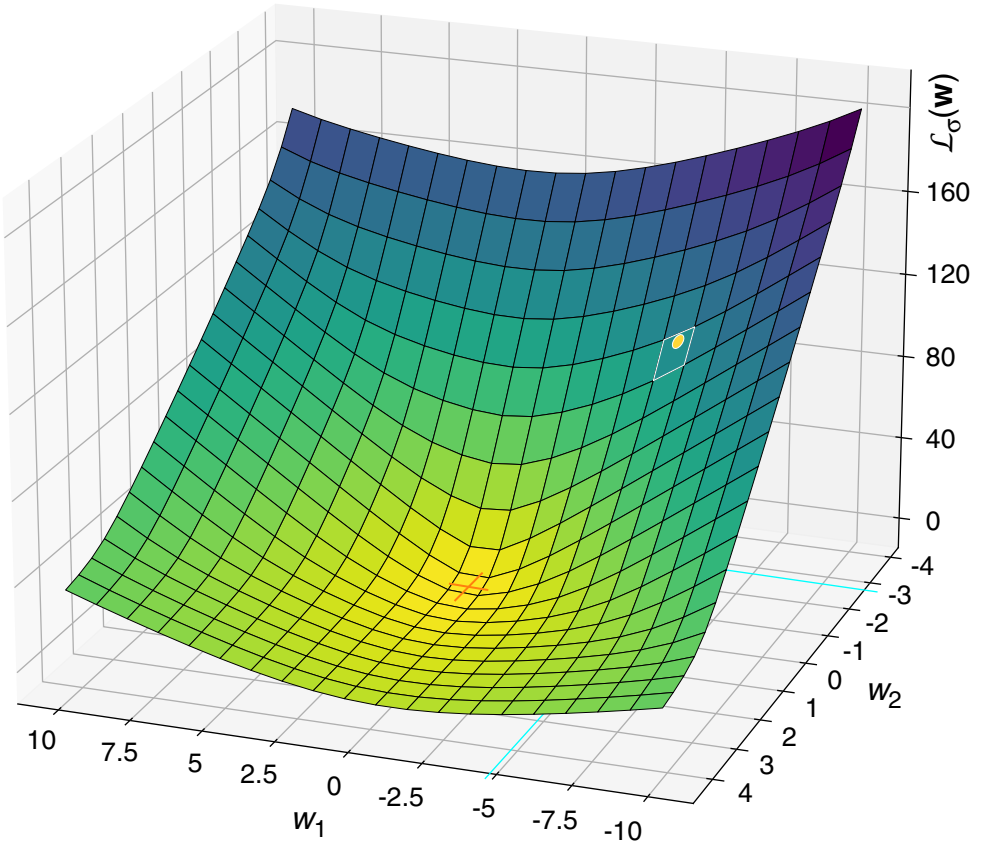
$$= \sum_{D} -c(\mathbf{x}) \cdot \log(y(\mathbf{x})) - (1-c(\mathbf{x})) \cdot \log(1-y(\mathbf{x})) + \lambda \cdot \vec{\mathbf{w}}^T\vec{\mathbf{w}}$$

© STEIN/VÖLSKE 2021

Gradient Descent

(3) Logistic Regression + Logistic Loss + Regularization (continued)





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

$$\mathcal{L}_\sigma(\mathbf{w}) = L_\sigma + \lambda \cdot R_{\|\vec{\mathbf{w}}\|_2^2} = \sum_{(\mathbf{x},c(\mathbf{x}))\in D} \ell_\sigma(c(\mathbf{x}), y(\mathbf{x})) + \lambda \cdot \vec{\mathbf{w}}^T \vec{\mathbf{w}}$$

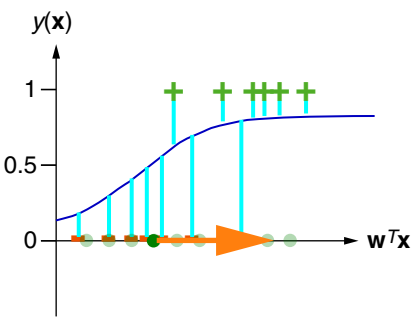
[Definitions: L, ℓ_σ, $R_{\|\vec{\mathbf{w}}\|_2^2}$]

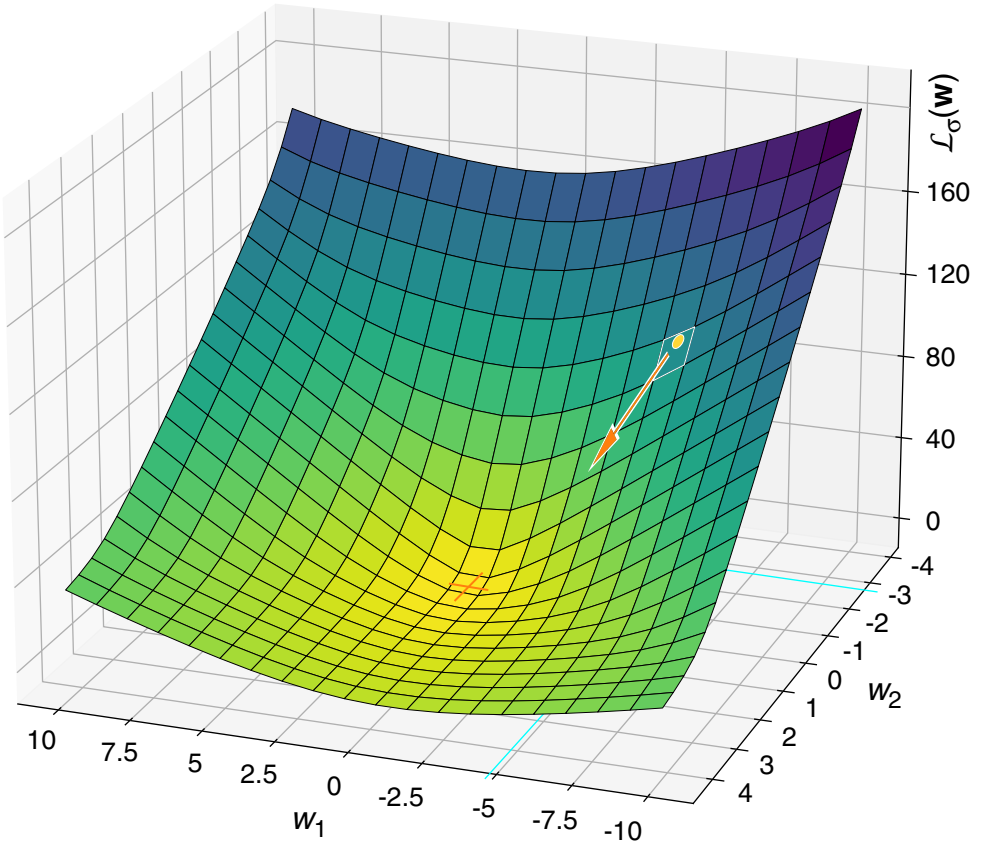
$$= \sum_{D} -c(\mathbf{x}) \cdot \log(y(\mathbf{x})) - (1-c(\mathbf{x})) \cdot \log(1-y(\mathbf{x})) + \lambda \cdot \vec{\mathbf{w}}^T \vec{\mathbf{w}}$$

© STEIN/VÖLSKE 2021

Gradient Descent

(3) Logistic Regression + Logistic Loss + Regularization (continued)





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

$$\mathcal{L}_\sigma(\mathbf{w}) \;=\; L_\sigma + \lambda\cdot R_{\|\vec{\mathbf{w}}\|_2^2} \;=\; \sum_{(\mathbf{x},c(\mathbf{x}))\in D} \ell_\sigma(c(\mathbf{x}), y(\mathbf{x})) + \lambda\cdot\vec{\mathbf{w}}^T\vec{\mathbf{w}}$$

[Definitions: L, ℓ_σ, $R_{\|\vec{\mathbf{w}}\|_2^2}$]

$$\nabla\mathcal{L}_\sigma(\mathbf{w}) \;=\; \left(\frac{\partial\mathcal{L}_\sigma(\mathbf{w})}{\partial w_0}, \frac{\partial\mathcal{L}_\sigma(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial\mathcal{L}_\sigma(\mathbf{w})}{\partial w_p}\right)^T$$

© STEIN/VÖLSKE 2021

Gradient Descent

(3) Logistic Regression + Logistic Loss + Regularization (continued)

Update of weight vector $\mathbf{w}$: (BGD$_\sigma$ algorithm, Line 7)

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

using the gradient of the objective function $\mathcal{L}_\sigma(\mathbf{w})$ to take steepest descent:

$$\Delta\mathbf{w} \;=\; -\eta \cdot \nabla\mathcal{L}_\sigma(\mathbf{w})$$

© STEIN/VÖLSKE 2021

Gradient Descent

(3) Logistic Regression + Logistic Loss + Regularization (continued)

Update of weight vector $\mathbf{w}$: (BGD$_\sigma$ algorithm, Line 7)

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

using the gradient of the objective function $\mathcal{L}_\sigma(\mathbf{w})$ to take steepest descent:

$$\begin{aligned}
\Delta\mathbf{w} &= -\eta \cdot \nabla\mathcal{L}_\sigma(\mathbf{w}) \\
&= -\eta \cdot \left(\frac{\partial\mathcal{L}_\sigma(\mathbf{w})}{\partial w_0}, \frac{\partial\mathcal{L}_\sigma(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial\mathcal{L}_\sigma(\mathbf{w})}{\partial w_p} \right)^T \\
&\vdots \\
&= \eta \cdot \sum_D \left(c(\mathbf{x}) - \sigma(\mathbf{w}^T\mathbf{x})\right) \cdot \mathbf{x} \;-\; \eta \cdot 2\lambda \cdot \binom{0}{\vec{\mathbf{w}}}
\end{aligned}$$

© STEIN/VÖLSKE 2021

Remarks:

- Recap. Distinguish between the p-dimensional direction vector $\vec{\mathbf{w}} = (w_1, \ldots, w_p)^T$, and the $(p+1)$-dimensional hypothesis $\mathbf{w} = (w_0, w_1, \ldots, w_p)^T$.

- The respective BGD variant, BGD$_\sigma$, has already been introduced in section Loss Computation of part Linear Models. However, the presented algorithm does not consider the update term $\eta \cdot 2\lambda \cdot \binom{0}{\vec{\mathbf{w}}}$ for the ridge regression regularization constraint, $\lambda \cdot \vec{\mathbf{w}}^T \vec{\mathbf{w}}$, which has been derived now and which may be added to Line 9:

  [BGD$_\sigma$] Line 9: $\mathbf{w} = \mathbf{w} + \Delta\mathbf{w} + \eta \cdot 2\lambda \cdot \binom{0}{\vec{\mathbf{w}}}$

© STEIN/VÖLSKE 2021

Remarks (how to derive $\nabla \mathcal{L}_\sigma(\mathbf{w})$) :

- Consider the partial derivative for a parameter $w_j,\ j = 0, \ldots, p$:

$$
\begin{aligned}
\frac{\partial}{\partial w_j} \mathcal{L}_\sigma(\mathbf{w}) \;&=\; \frac{\partial}{\partial w_j} L_\sigma(\mathbf{w}) \;+\; \frac{\partial}{\partial w_j} \lambda \cdot R_{\|\vec{\mathbf{w}}\|_2^2}(\mathbf{w}) \\
&= \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} \frac{\partial}{\partial w_j} \ell_\sigma(c(\mathbf{x}), y(\mathbf{x})) \;+\; \lambda \cdot \frac{\partial}{\partial w_j} \vec{\mathbf{w}}^T \vec{\mathbf{w}} \\
&= \sum_{D} \frac{\partial}{\partial w_j} \left[-c(\mathbf{x}) \cdot \mathsf{log}(\sigma(\mathbf{w}^T \mathbf{x})) - (1 - c(\mathbf{x})) \cdot \mathsf{log}(1 - \sigma(\mathbf{w}^T \mathbf{x})) \right] \;+\; \lambda \cdot \frac{\partial}{\partial w_j} \sum_{i=1}^{p} w_i^2 \\
&= \sum_{D} \left[-c(\mathbf{x}) \cdot \frac{\partial}{\partial w_j} \mathsf{log}(\sigma(\mathbf{w}^T \mathbf{x})) - (1 - c(\mathbf{x})) \cdot \frac{\partial}{\partial w_j} \mathsf{log}(1 - \sigma(\mathbf{w}^T \mathbf{x})) \right] \overset{(1)}{+} \lambda \cdot \frac{\partial}{\partial w_j} w_j^2 \\
&\overset{(2)}{=} \sum_{D} \Bigg[-c(\mathbf{x}) \cdot \frac{1}{\sigma(\mathbf{w}^T \mathbf{x})} \cdot \frac{\partial}{\partial w_j} \sigma(\mathbf{w}^T \mathbf{x}) \\
&\qquad - (1 - c(\mathbf{x})) \cdot \frac{1}{1 - \sigma(\mathbf{w}^T \mathbf{x})} \cdot \left(- \frac{\partial}{\partial w_j} \sigma(\mathbf{w}^T \mathbf{x}) \right) \Bigg] \overset{(1)}{+} 2\lambda \cdot w_j \\
&= \hookrightarrow
\end{aligned}
$$

p. 83

© STEIN/VÖLSKE 2021

Remarks (how to derive $\nabla \mathcal{L}_\sigma(\mathbf{w})$) (continued) :

$$\overset{(3)}{=} \sum_D \Big[-c(\mathbf{x}) \cdot \frac{1}{\sigma(\mathbf{w}^T\mathbf{x})} \cdot \sigma(\mathbf{w}^T\mathbf{x}) \cdot (1-\sigma(\mathbf{w}^T\mathbf{x})) \cdot \frac{\partial}{\partial w_j} \mathbf{w}^T\mathbf{x}$$

$$- (1-c(\mathbf{x})) \cdot \frac{1}{1-\sigma(\mathbf{w}^T\mathbf{x})} \cdot (-1) \cdot \sigma(\mathbf{w}^T\mathbf{x}) \cdot (1-\sigma(\mathbf{w}^T\mathbf{x})) \cdot \frac{\partial}{\partial w_j} \mathbf{w}^T\mathbf{x} \Big] \overset{(1)}{+} 2\lambda \cdot w_j$$

$$= \sum_D -c(\mathbf{x}) \cdot (1-\sigma(\mathbf{w}^T\mathbf{x})) \cdot x_j + (1-c(\mathbf{x})) \cdot \sigma(\mathbf{w}^T\mathbf{x}) \cdot x_j \overset{(1)}{+} 2\lambda \cdot w_j$$

$$= -\sum_D (c(\mathbf{x}) - \sigma(\mathbf{w}^T\mathbf{x})) \cdot x_j \overset{(1)}{+} 2\lambda \cdot w_j$$

- Plugging the results for $\frac{\partial}{\partial w_j} \mathcal{L}_\sigma(\mathbf{w})$ into $-\eta \cdot (\ldots)^T$ yields the update formula for $\Delta \mathbf{w}$.
- Hints:
  - (1) Since $\vec{\mathbf{w}} \equiv (w_1, \ldots, w_p)^T$ the right summand is defined as 0 for $w_j = w_0$.
  - (2) Chain rule with ' $\frac{d}{dz} \log(z) = \frac{1}{z}$ '
  - (3) Chain rule with ' $\frac{d}{dz} \sigma(z) = \sigma(z) \cdot (1 - \sigma(z))$ ', where $\sigma(z) = \frac{1}{1+e^{-z}}$

© STEIN/VÖLSKE 2021

Chapter ML:IV (continued)

IV. Neural Networks

- Perceptron Learning
- Gradient Descent
- Multilayer Perceptron

© STEIN/VÖLSKE 2021

Multilayer Perceptron

Definition 1 (Linear Separability)

Two sets of feature vectors, X_0, X_1, sampled from a p-dimensional feature space $\mathbf{X}$, are called linearly separable if $p+1$ real numbers, $w_0, w_1, \ldots, w_p$, exist such that the following conditions holds:

1. $\forall \mathbf{x} \in X_0$: $\sum_{j=0}^{p} w_j x_j < 0$
2. $\forall \mathbf{x} \in X_1$: $\sum_{j=0}^{p} w_j x_j \geq 0$

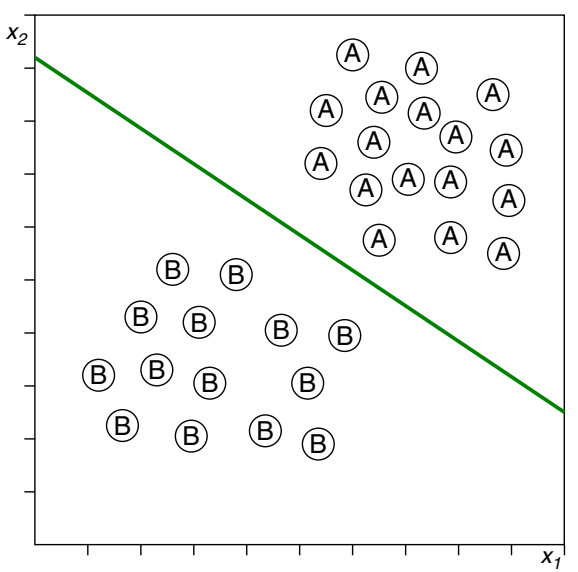
© STEIN/VÖLSKE 2021

Multilayer Perceptron

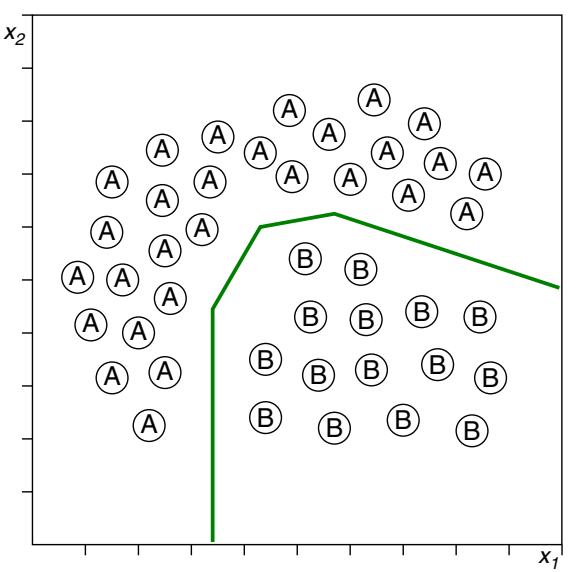
Definition 1 (Linear Separability)

Two sets of feature vectors, X_0, X_1, sampled from a p-dimensional feature space $\mathbf{X}$, are called linearly separable if $p+1$ real numbers, $w_0, w_1, \ldots, w_p$, exist such that the following conditions holds:

1. $\forall \mathbf{x} \in X_0$: $\sum_{j=0}^{p} w_j x_j < 0$
2. $\forall \mathbf{x} \in X_1$: $\sum_{j=0}^{p} w_j x_j \geq 0$



linearly separable



not linearly separable

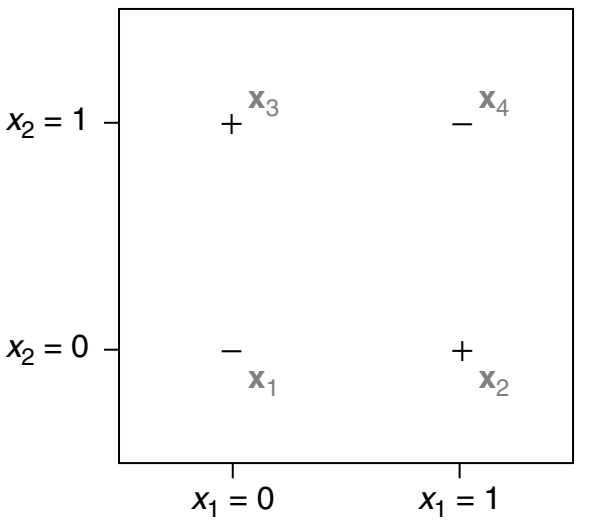
© STEIN/VÖLSKE 2021

Multilayer Perceptron

Linear Separability (continued)

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

	x_1	x_2	XOR	$c(\mathbf{x})$
$\mathbf{x}_1$	0	0	0	$-$
$\mathbf{x}_2$	1	0	1	$+$
$\mathbf{x}_3$	0	1	1	$+$
$\mathbf{x}_4$	1	1	0	$-$



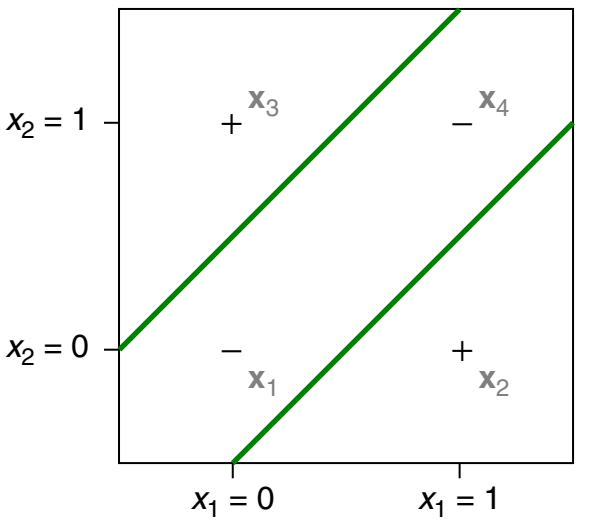
© STEIN/VÖLSKE 2021

Multilayer Perceptron

Linear Separability (continued)

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

	x_1	x_2	XOR	$c(\mathbf{x})$
$\mathbf{x}_1$	0	0	0	$-$
$\mathbf{x}_2$	1	0	1	$+$
$\mathbf{x}_3$	0	1	1	$+$
$\mathbf{x}_4$	1	1	0	$-$



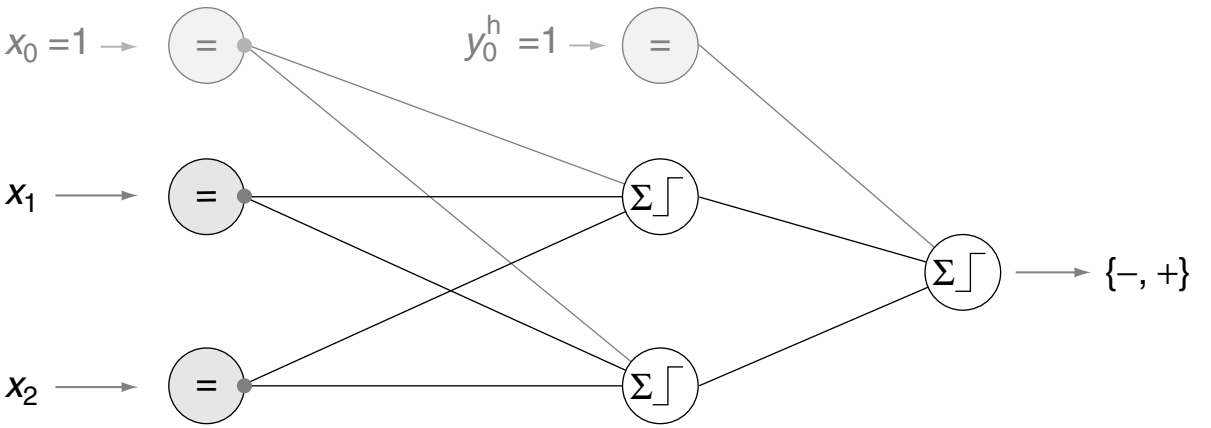
- ➜ Specification of several hyperplanes.
- ➜ Layered combination of several perceptrons: the multilayer perceptron.

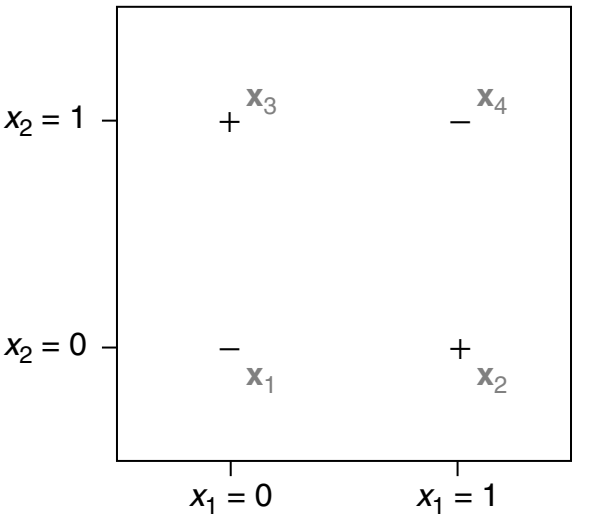
© STEIN/VÖLSKE 2021

Multilayer Perceptron

Overcoming the Linear Separability Restriction

A minimum multilayer perceptron $y(\mathbf{x})$ that can handle the *XOR* problem:



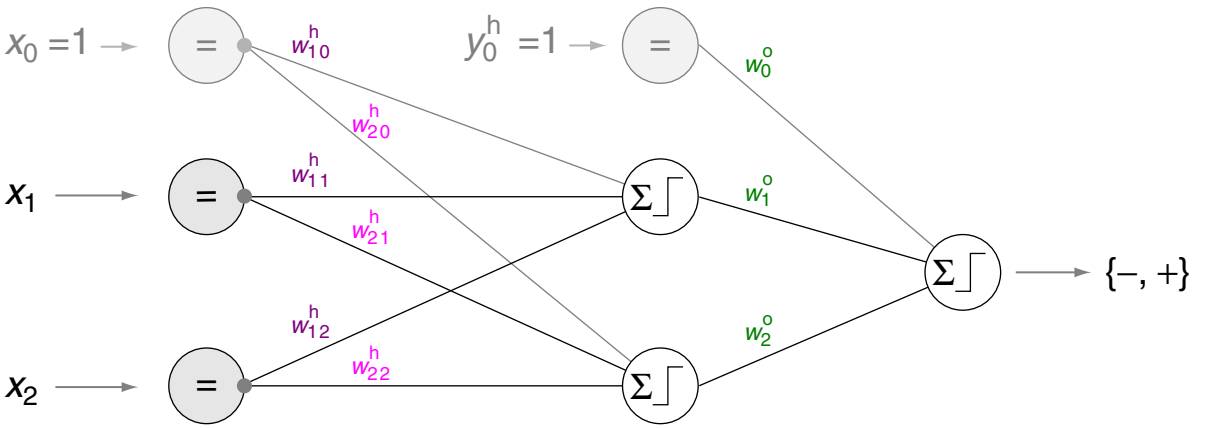


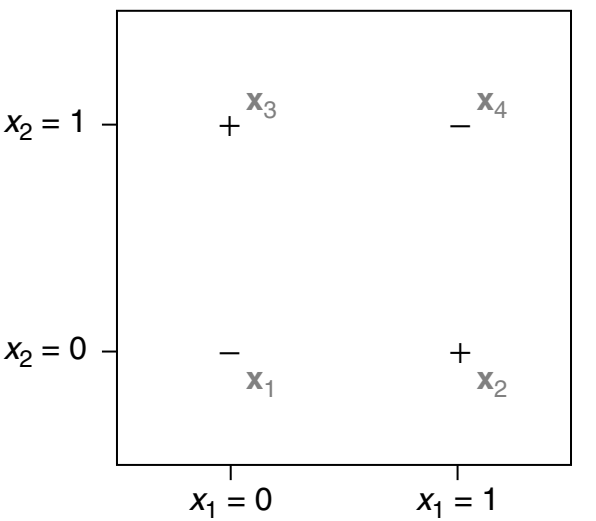
© STEIN/VÖLSKE 2021

Multilayer Perceptron

Overcoming the Linear Separability Restriction (continued)

A minimum multilayer perceptron $y(\mathbf{x})$ that can handle the *XOR* problem:



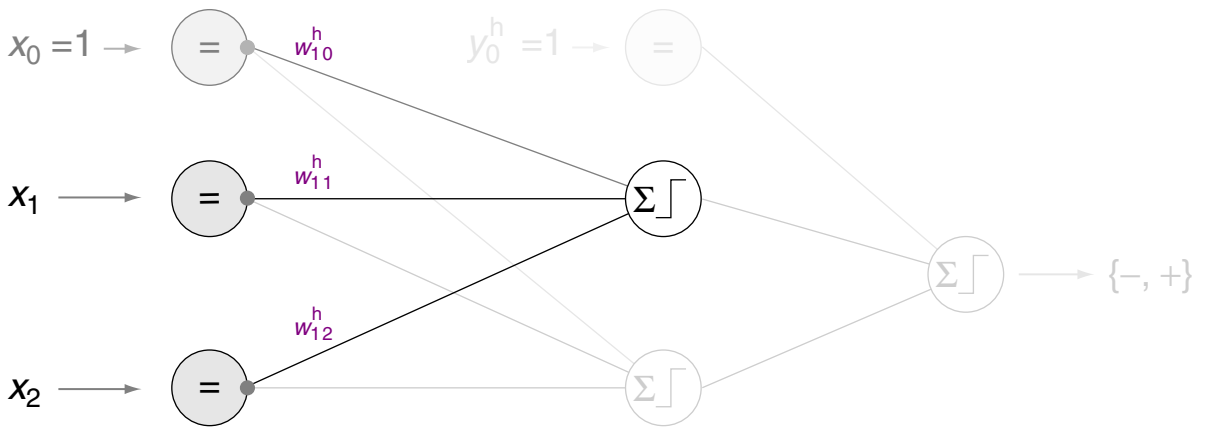


© STEIN/VÖLSKE 2021

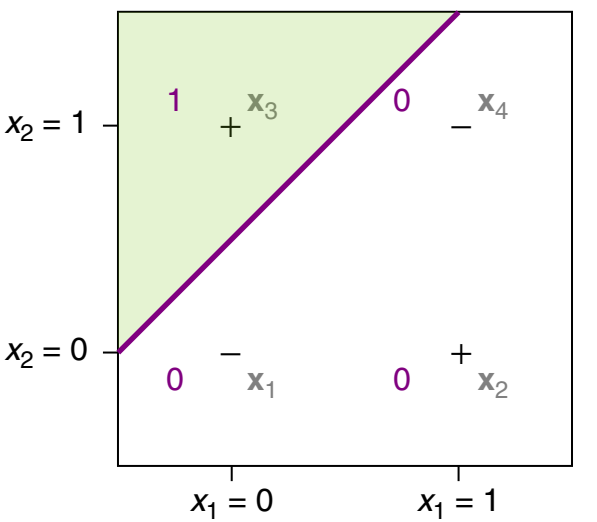
Multilayer Perceptron

Overcoming the Linear Separability Restriction (continued)

A minimum multilayer perceptron $y(\mathbf{x})$ that can handle the *XOR* problem:



$$W^h = \begin{bmatrix} -0.5 & -1 & 1 \\ 0.5 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

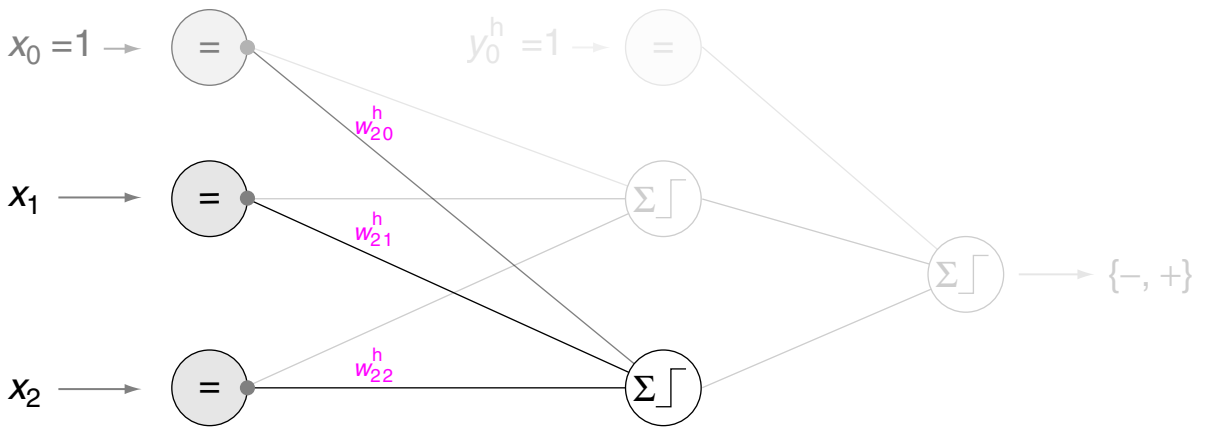


© STEIN/VÖLSKE 2021

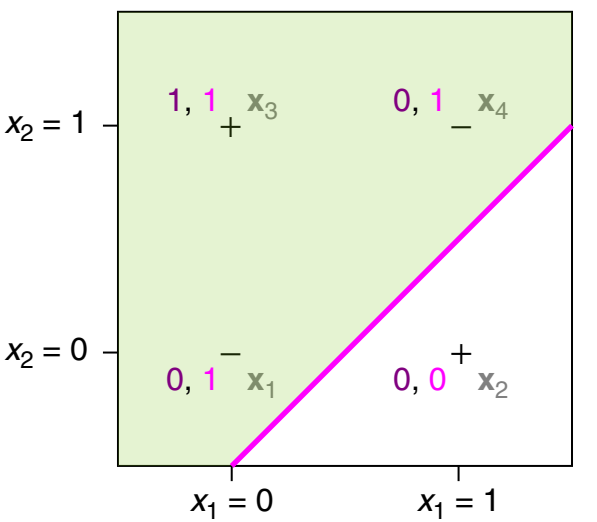
Multilayer Perceptron

Overcoming the Linear Separability Restriction (continued)

A minimum multilayer perceptron $y(\mathbf{x})$ that can handle the *XOR* problem:



$$W^h = \begin{bmatrix} -0.5 & -1 & 1 \\ 0.5 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

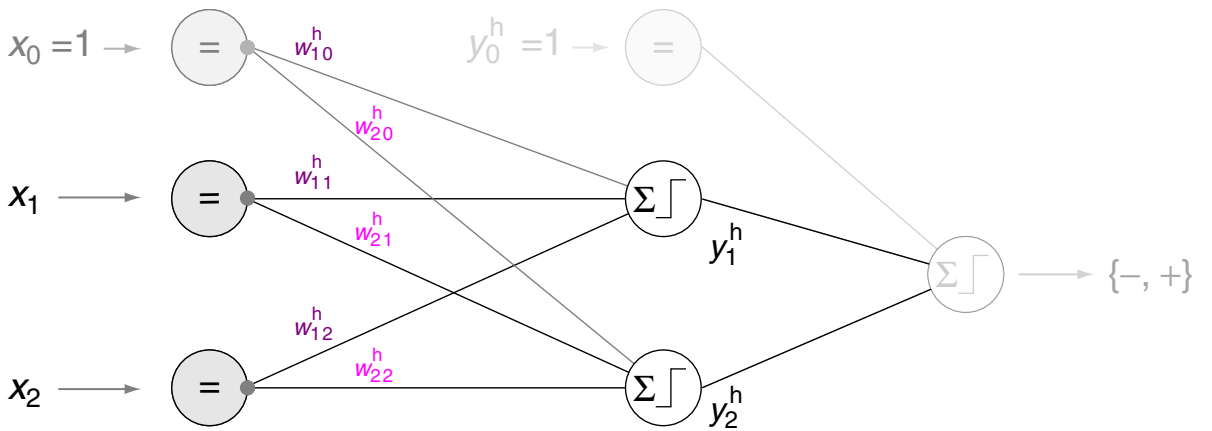


© STEIN/VÖLSKE 2021

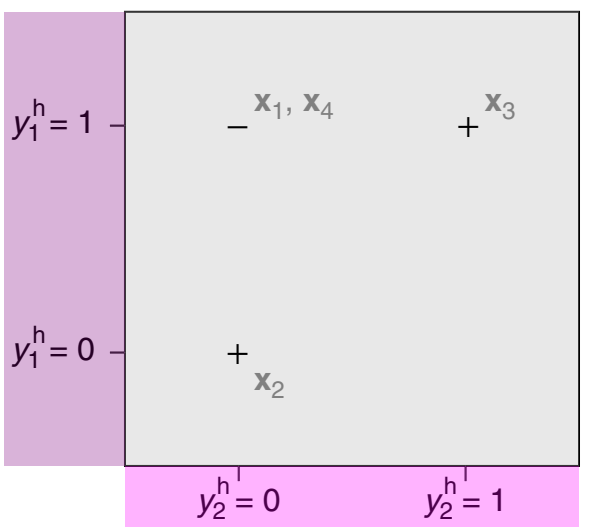
Multilayer Perceptron

Overcoming the Linear Separability Restriction (continued)

A minimum multilayer perceptron $y(\mathbf{x})$ that can handle the *XOR* problem:



$$W^h = \begin{bmatrix} -0.5 & -1 & 1 \\ 0.5 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

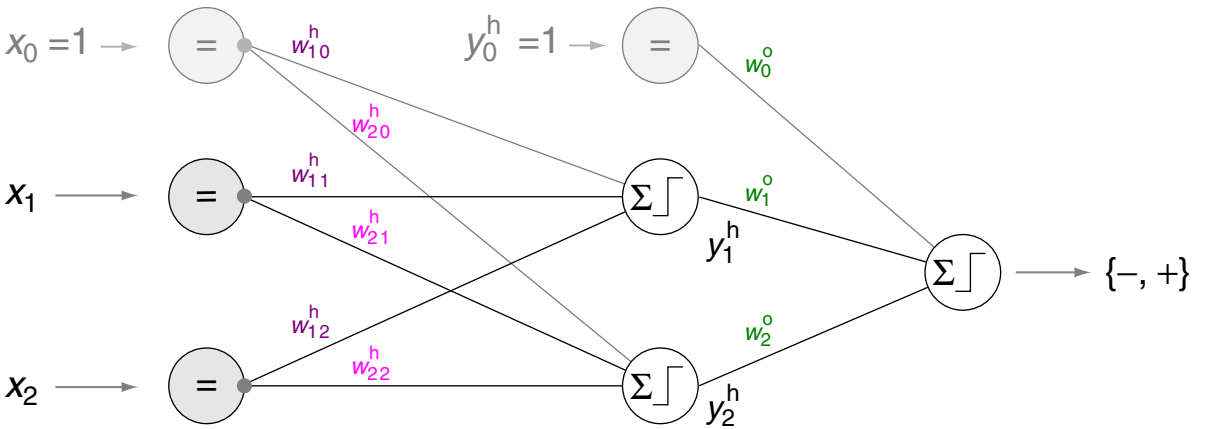


© STEIN/VÖLSKE 2021

Multilayer Perceptron

Overcoming the Linear Separability Restriction (continued)

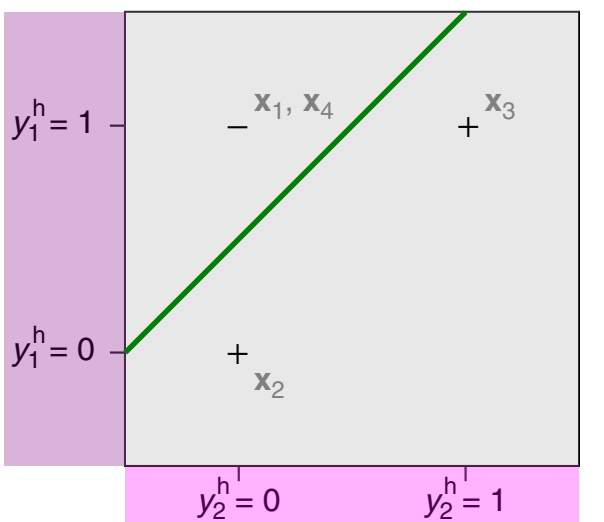
A minimum multilayer perceptron $y(\mathbf{x})$ that can handle the *XOR* problem:



$$y(\mathbf{x}) = \textit{heaviside}\left(W^{\text{o}}\left(\overset{1}{\textit{Heaviside}}(W^{\text{h}}\,\mathbf{x})\right)\right)$$

$$W^{\text{h}} = \begin{bmatrix} -0.5 & -1 & 1 \\ 0.5 & -1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

$$W^{\text{o}} = \begin{bmatrix} 0.5 & 1 & -1 \end{bmatrix}$$



© STEIN/VÖLSKE 2021

Remarks:

- The first, second, and third layer of the shown multilayer perceptron are called input, hidden, and output layer respectively. Here, in the example, the input layer is comprised of $p{+}1{=}3$ units, the hidden layer contains $l{+}1{=}3$ units, and the output layer consists of $k{=}1$ unit.

- Each input unit is connected via a weighted edge to all hidden units (except to the topmost hidden unit, which has a constant input $y_0^{\mathsf{h}} = 1$), resulting in six weights, organized as 2$\times$3-matrix W^{h}. Each hidden unit is connected via a weighted edge to the output unit, resulting in three weights, organized as 1$\times$3-matrix W^{o}.

- The input units perform no computation but only distribute the values x_0, x_1, x_2 to the next layer. The hidden units (again except the topmost unit) and the output unit apply the *heaviside* function to the sum of their weighted inputs and propagate the result.
  I.e., the nine weights $\mathbf{w} = (w_{10}^{\mathsf{h}}, \ldots, w_{22}^{\mathsf{h}}, w_1^{\mathsf{o}}, w_2^{\mathsf{o}}, w_3^{\mathsf{o}})$, organized as W^{h} and W^{o}, specify the multilayer perceptron (model function) $y(\mathbf{x})$ completely: $y(\mathbf{x}) = \textit{heaviside}(W^{\mathsf{o}} \begin{pmatrix} 1 \\ \textit{Heaviside}(W^{\mathsf{h}}\,\mathbf{x}) \end{pmatrix})$

- The function *Heaviside* denotes the extension of the scalar *heaviside* function to vectors. For $\mathbf{z} \in \mathbf{R}^d$ the function *Heaviside*$(\mathbf{z})$ is defined as $(\textit{heaviside}(z_1), \ldots, \textit{heaviside}(z_d))^T$.

© STEIN/VÖLSKE 2021

Remarks (continued) :

- 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 (and non-linear) threshold functions along with 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: MIT Media Lab, Wikipedia]

© STEIN/VÖLSKE 2021

Multilayer Perceptron

Unrestricted Classification Problems

Setting:

- X is a set of feature vectors from an inner product space $\mathbf{X}$, $\mathbf{X} \subseteq \mathbf{R}^p$.
- $C = \{0, 1\}^k$ is the set of all multiclass labelings for k classes.
- $\mathbf{c} : X \to C$ is the (unknown) ideal classifier for X.
- $D = \{(\mathbf{x}_1, \mathbf{c}(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, \mathbf{c}(\mathbf{x}_n))\} \subseteq X \times C$ is a set of examples.

Todo:

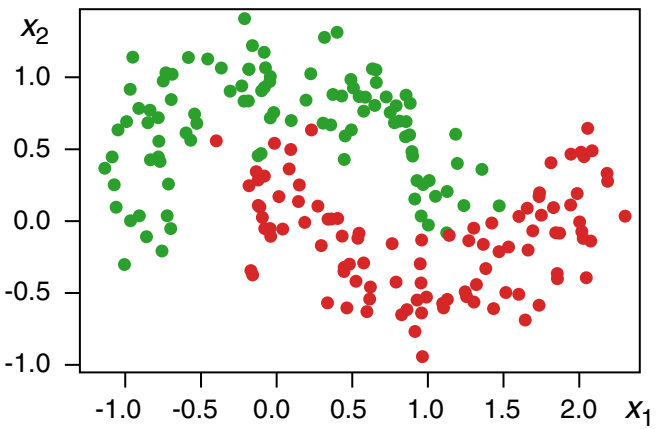
- Approximate $\mathbf{c}(\mathbf{x})$, which is implicitly given via D, with the multilayer perceptron.

© STEIN/VÖLSKE 2021

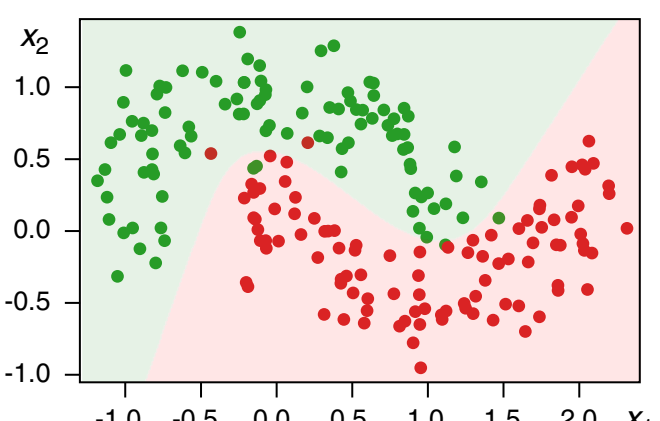
Multilayer Perceptron

Unrestricted Classification Problems: Example

Two-class classification problem:



Separated classes:

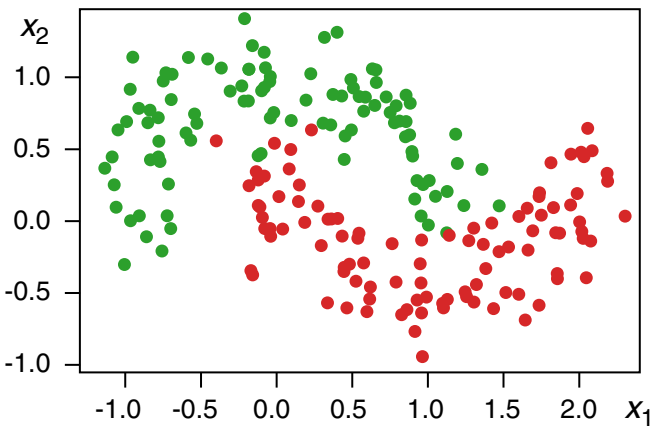


© STEIN/VÖLSKE 2021

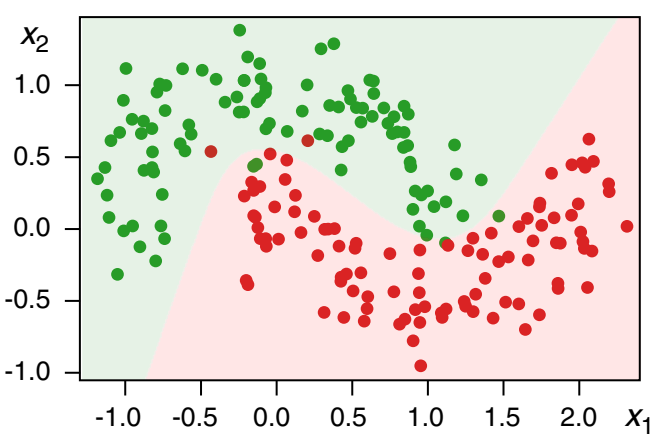
Multilayer Perceptron

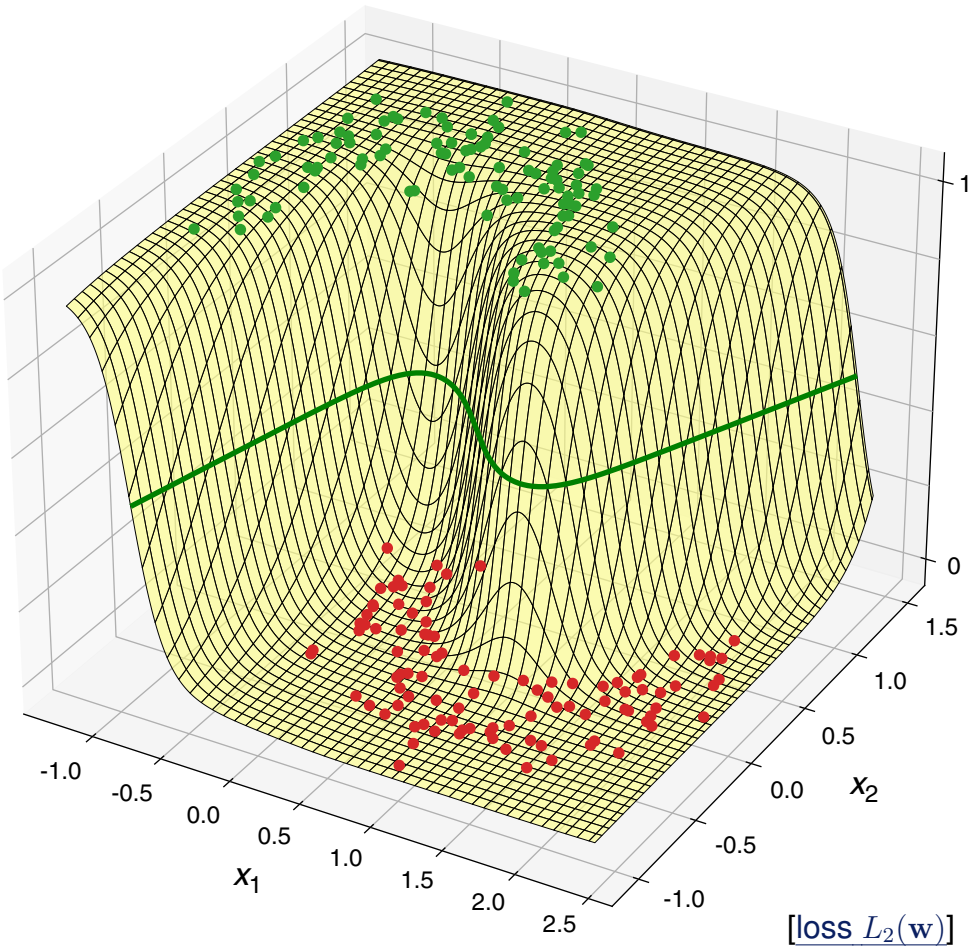
Unrestricted Classification Problems: Example

Two-class classification problem:



Separated classes:





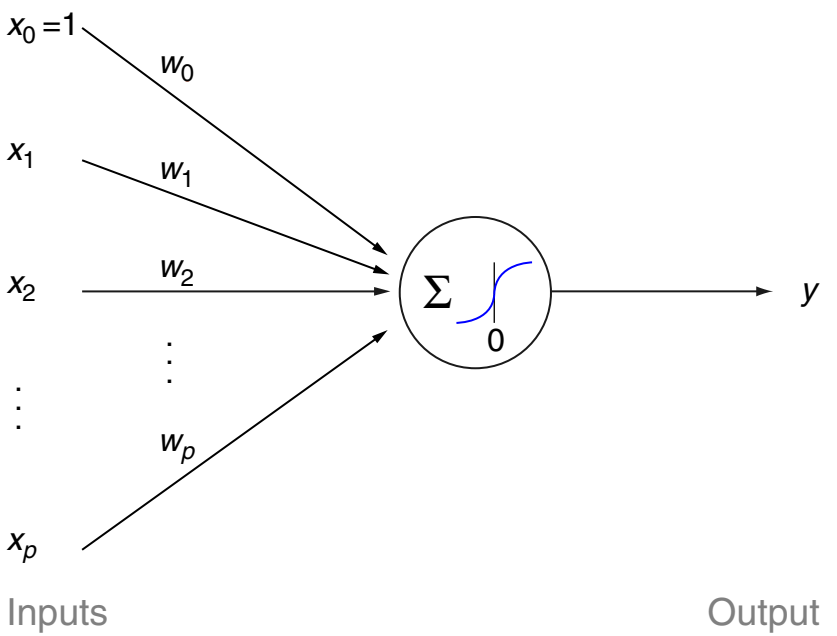
[loss $L_2(\mathbf{w})$]

© STEIN/VÖLSKE 2021

Multilayer Perceptron

Sigmoid Function [Heaviside]

A perceptron with a continuous *and* non-linear threshold function:



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

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \qquad \frac{d\,\sigma(z)}{dz} = \sigma(z) \cdot (1 - \sigma(z))$$

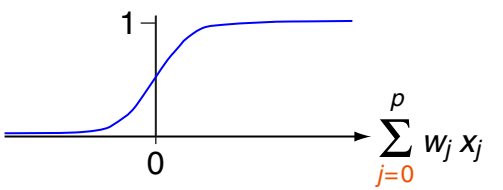
© STEIN/VÖLSKE 2021

Multilayer Perceptron

Sigmoid Function (continued)

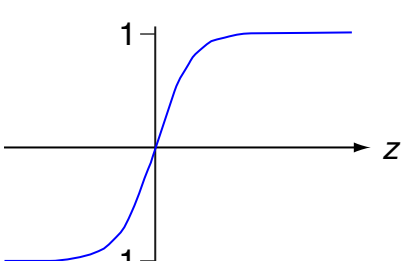
Computation of the perceptron output $y(\mathbf{x})$ via the sigmoid function σ:

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



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

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



© STEIN/VÖLSKE 2021

Remarks:

- How to derive the derivation of $\sigma(z)$:

$$
\begin{aligned}
\frac{d\,\sigma(z)}{dz} &= \frac{d}{dz}\,\frac{1}{1+e^{-z}} = \frac{d}{dz}\,(1+e^{-z})^{-1} \\
&= -1 \cdot (1+e^{-z})^{-2} \cdot (-1) \cdot e^{-z} \\
&= \sigma(z) \cdot \sigma(z) \cdot e^{-z} \\
&= \sigma(z) \cdot \sigma(z) \cdot (1+e^{-z}-1) \\
&= \sigma(z) \cdot \sigma(z) \cdot (\sigma(z)^{-1}-1) \\
&= \sigma(z) \cdot (1-\sigma(z))
\end{aligned}
$$

© STEIN/VÖLSKE 2021

Remarks (continued):

- Employing a nonlinear function as threshold function in the perceptron, such as sigmoid or *heaviside*, is necessary to synthesize complex nonlinear functions via layered composition. A "multilayer" perceptron with linear threshold functions can be expressed as a single linear function and hence is equivalent to the power of a single perceptron only.

- Consider the following exemplary composition of three linear functions as a "multilayer" perceptron with p input units, two hidden units, and one output unit: $y(\mathbf{x}) = W^{\mathsf{o}}\,[W^{\mathsf{h}}\,\mathbf{x}]$
  The respective weight matrices are as follows:

$$W^{\mathsf{h}} = \begin{bmatrix} w_{11}^{\mathsf{h}} & \ldots & w_{1p}^{\mathsf{h}} \\ w_{21}^{\mathsf{h}} & \ldots & w_{1p}^{\mathsf{h}} \end{bmatrix}, \qquad W^{\mathsf{o}} = \begin{bmatrix} w_1^{\mathsf{o}} & w_2^{\mathsf{o}} \end{bmatrix}$$

  Obviously holds:

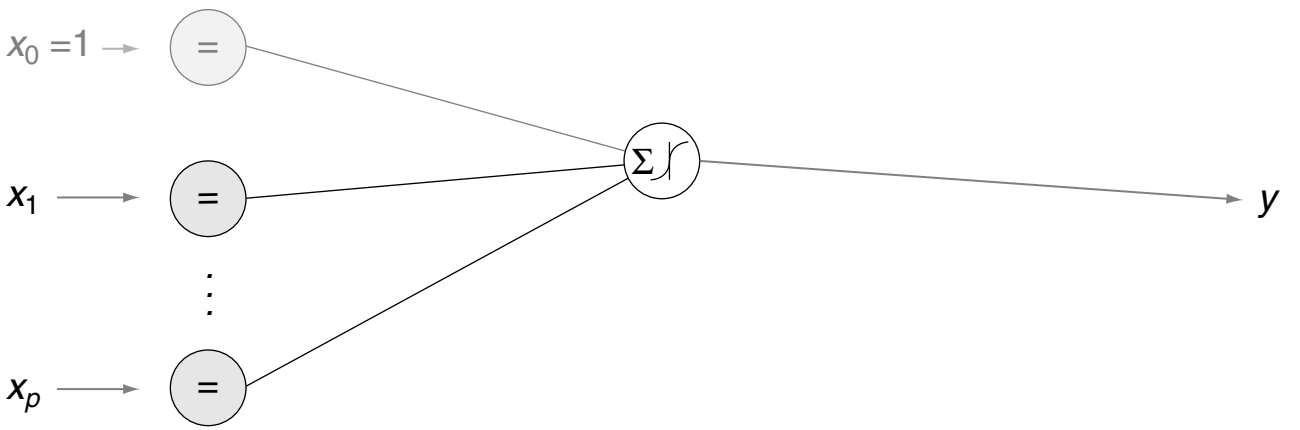
$$\begin{aligned} y(\mathbf{x}) = W^{\mathsf{o}}\,[W^{\mathsf{h}}\,\mathbf{x}] &= W^{\mathsf{o}} \begin{bmatrix} w_{11}^{\mathsf{h}} x_1 & + \ \ldots \ + & w_{1p}^{\mathsf{h}} x_p \\ w_{21}^{\mathsf{h}} x_1 & + \ \ldots \ + & w_{1p}^{\mathsf{h}} x_p \end{bmatrix} \\ &= w_1^{\mathsf{o}} w_{11}^{\mathsf{h}} x_1 + \ \ldots \ + w_1^{\mathsf{o}} w_{1p}^{\mathsf{h}} x_p + w_2^{\mathsf{o}} w_{21}^{\mathsf{h}} x_1 + \ \ldots \ + w_2^{\mathsf{o}} w_{1p}^{\mathsf{h}} x_p \\ &= (w_1^{\mathsf{o}} w_{11}^{\mathsf{h}} + w_2^{\mathsf{o}} w_{21}^{\mathsf{h}}) x_1 + \ \ldots \ + (w_1^{\mathsf{o}} w_{1p}^{\mathsf{h}} + w_2^{\mathsf{o}} w_{1p}^{\mathsf{h}}) x_p \\ &= w_1 x_1 + \ \ldots \ + w_p x_p \ = \ \mathbf{w}^T \mathbf{x} \end{aligned}$$

© STEIN/VÖLSKE 2021

Multilayer Perceptron

Network Architecture

A single perceptron $y(\mathbf{x})$:

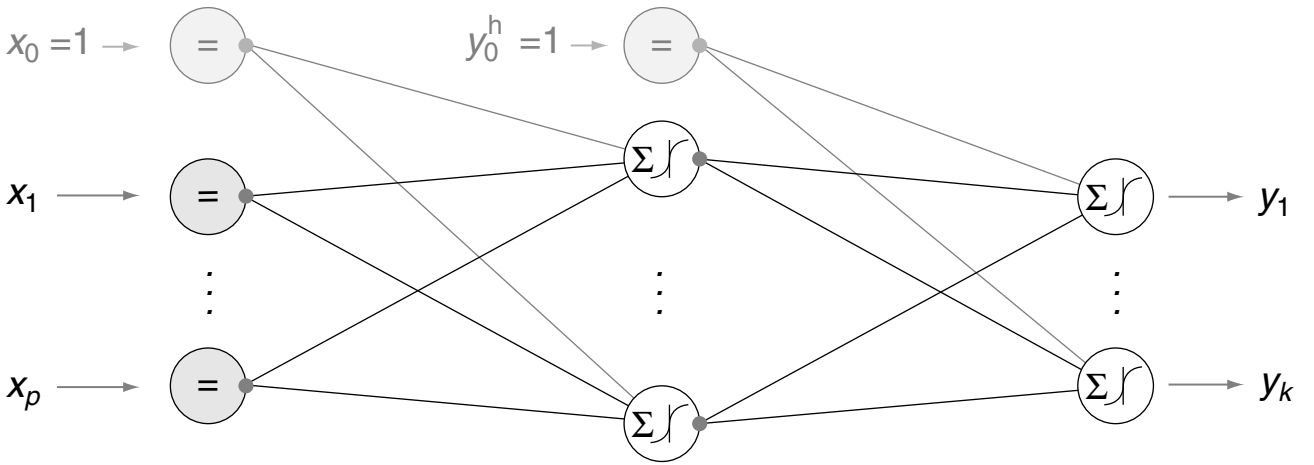


© STEIN/VÖLSKE 2021

Multilayer Perceptron

Network Architecture (continued)

A multilayer perceptron $\mathbf{y}(\mathbf{x})$ with one hidden layer and k-dimensional output layer:

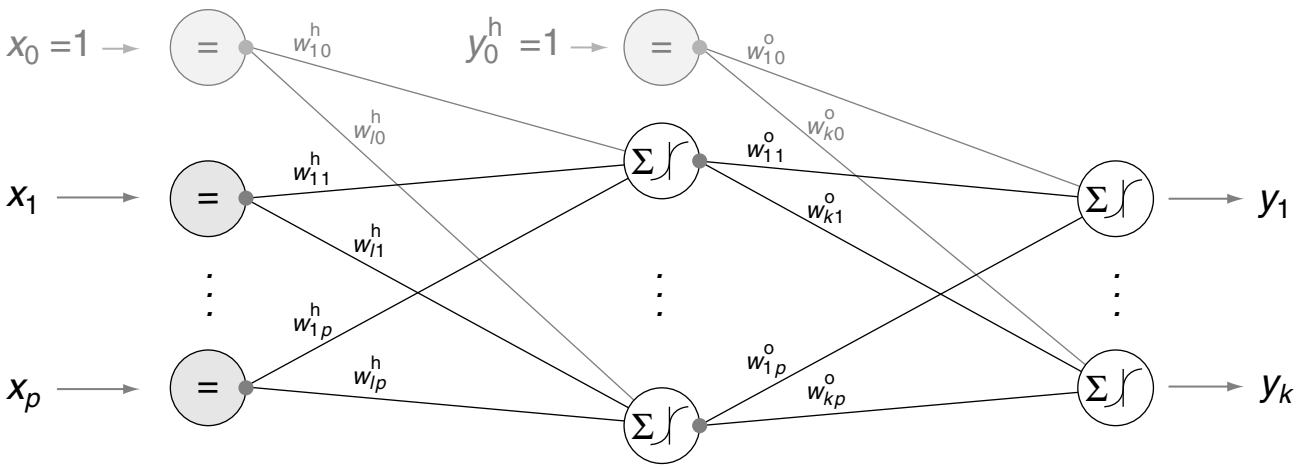


© STEIN/VÖLSKE 2021

Multilayer Perceptron

Network Architecture (continued)

A multilayer perceptron $\mathbf{y}(\mathbf{x})$ with one hidden layer and k-dimensional output layer:



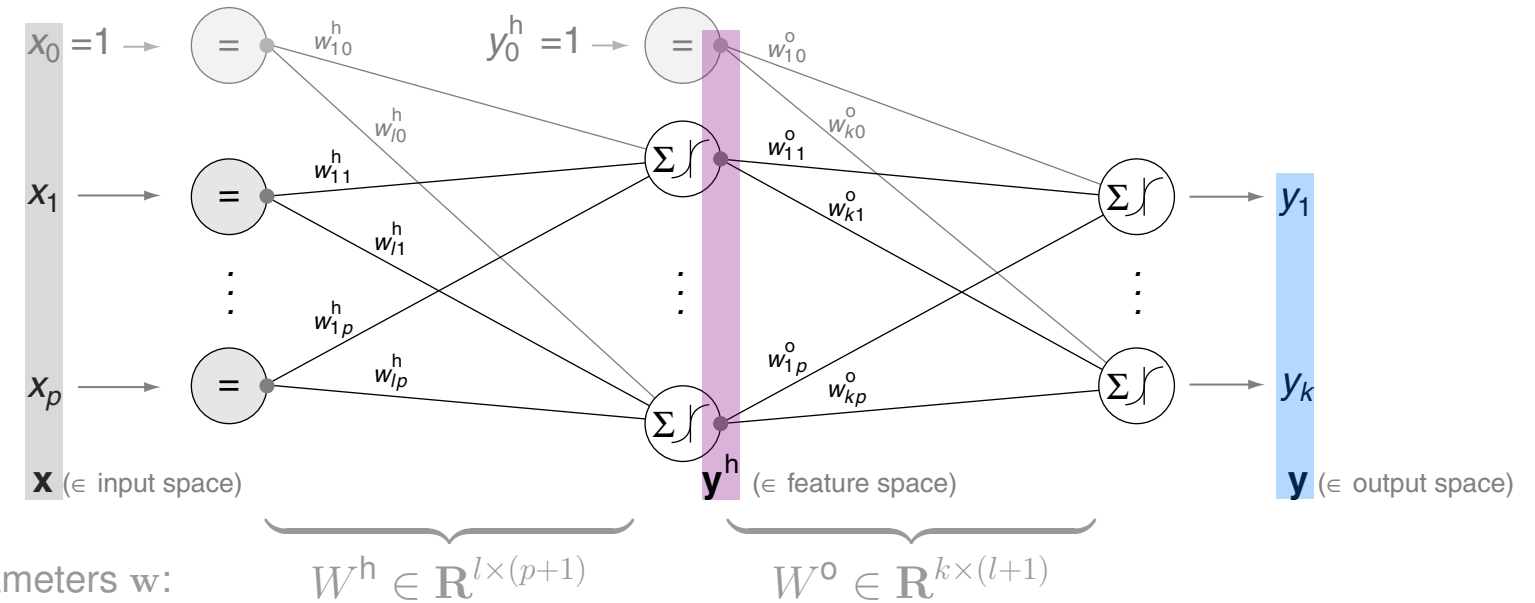
Parameters $\mathbf{w}$: $W^{\mathrm{h}} \in \mathbf{R}^{l \times (p+1)}$ $W^{\mathrm{o}} \in \mathbf{R}^{k \times (l+1)}$

© STEIN/VÖLSKE 2021

Multilayer Perceptron

Network Architecture (continued)

A multilayer perceptron $\mathbf{y}(\mathbf{x})$ with one hidden layer and k-dimensional output layer:

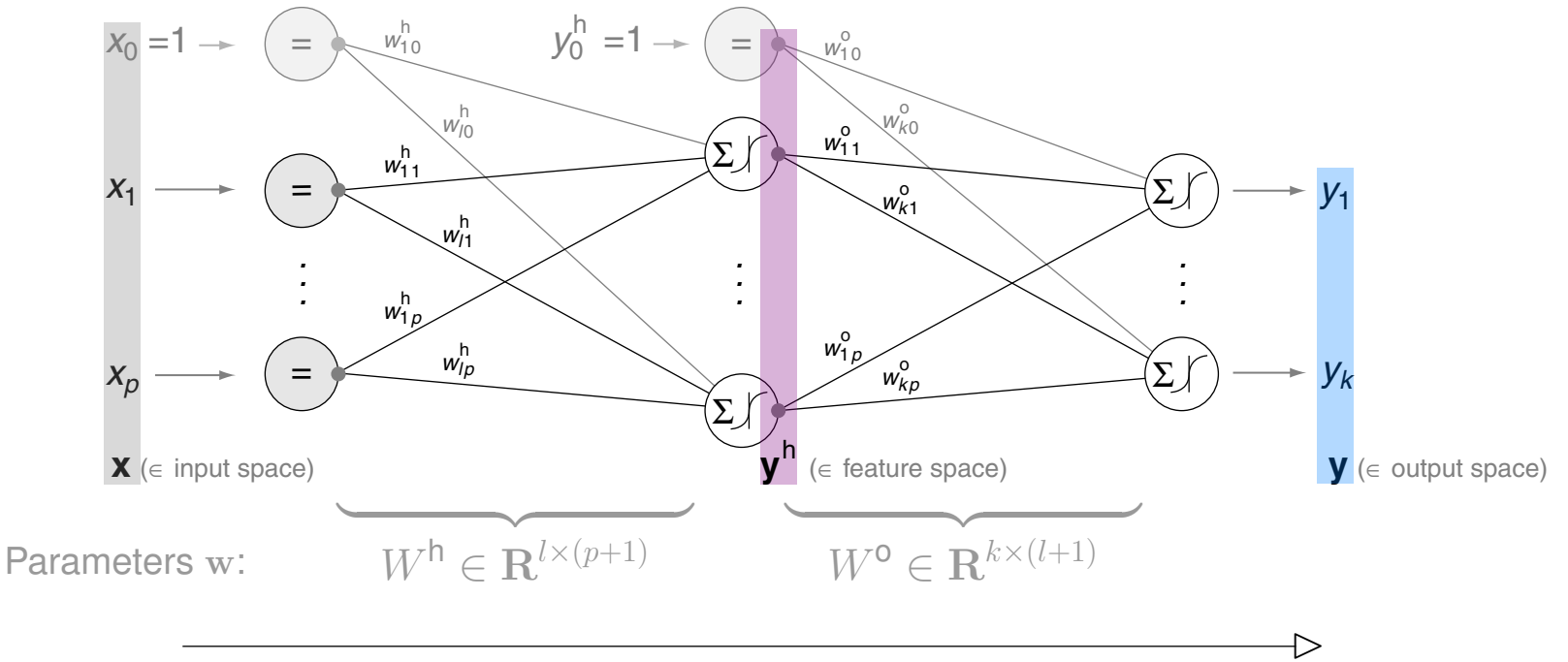


© STEIN/VÖLSKE 2021

Multilayer Perceptron

(1) Forward Propagation

A multilayer perceptron $\mathbf{y}(\mathbf{x})$ with one hidden layer and k-dimensional output layer:



Forward pass computation (aka. forward propagation) :

$$\mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}\left(W^{\mathrm{o}}\, \mathbf{y}^{\mathrm{h}}(\mathbf{x})\right) = \boldsymbol{\sigma}\left(W^{\mathrm{o}} \begin{pmatrix} 1 \\ \boldsymbol{\sigma}\left(W^{\mathrm{h}}\, \mathbf{x}\right) \end{pmatrix}\right)$$

© STEIN/VÖLSKE 2021

Remarks:

- Each input unit is connected to the hidden units $1, \ldots, l$, resulting in $l \cdot (p+1)$ weights, organized as matrix $W^{\mathsf{h}} \in \mathbf{R}^{l \times (p+1)}$. Each hidden unit is connected to the output units $1, \ldots, k$, resulting in $k \cdot (l+1)$ weights, organized as matrix $W^{\mathsf{o}} \in \mathbf{R}^{k \times (l+1)}$.

- The hidden units and the output unit(s) apply the (vectorial) sigmoid function, $\boldsymbol{\sigma}$, to the sum of their weighted inputs and propagate the result as $\mathbf{y}^{\mathsf{h}}$ and $\mathbf{y}$ respectively. For $\mathbf{z} \in \mathbf{R}^d$ the vectorial sigmoid function $\boldsymbol{\sigma}(\mathbf{z})$ is defined as $(\sigma(z_1), \ldots, \sigma(z_d))^T$.
  The parameter vector $\mathbf{w} = (w^{\mathsf{h}}_{10}, \ldots, w^{\mathsf{h}}_{lp}, w^{\mathsf{o}}_{10}, \ldots, w^{\mathsf{o}}_{kl})$, organized as matrices W^{h} and W^{o}, specify the multilayer perceptron (model function) $\mathbf{y}(\mathbf{x})$ completely: $\mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}(W^{\mathsf{o}} \begin{pmatrix} 1 \\ \boldsymbol{\sigma}(W^{\mathsf{h}} \mathbf{x}) \end{pmatrix})$

- The shown architecture with k output units allows for the distinction of k classes, either within an exclusive class assignment setting or within a multi-label setting. In the former setting a so-called "soft-max" layer can be added subsequent to the output layer to directly return the class label $1, \ldots, k$.

- The non-linear characteristic of the sigmoid function allows for networks that approximate every (computable) function. For this capability only three "active" layers are required, i.e., two layers with hidden units and one layer with output units. Keyword: universal approximator [Kolmogorov theorem, 1957]

- Multilayer perceptrons are also called multilayer networks or (artificial) neural networks, ANN for short.

© STEIN/VÖLSKE 2021

Multilayer Perceptron

(1) Forward Propagation [network architecture] (continued)

(a) Propagate $\mathbf{x}$ from input to hidden layer: (IGD$_{\text{MLP}}$ algorithm, Line 5)

$$W^{\mathsf{h}} \in \mathbf{R}^{l \times (p+1)} \qquad \mathbf{x} \in \mathbf{R}^{p+1}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w_{10}^{\mathsf{h}} & \dots & w_{1p}^{\mathsf{h}} \\ & \vdots & \\ w_{l0}^{\mathsf{h}} & \dots & w_{lp}^{\mathsf{h}} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}\right) = \begin{bmatrix} y_1^{\mathsf{h}} \\ \vdots \\ y_l^{\mathsf{h}} \end{bmatrix}$$

© STEIN/VÖLSKE 2021

Multilayer Perceptron

(1) Forward Propagation [network architecture] (continued)

(a) Propagate $\mathbf{x}$ from input to hidden layer: (IGD$_{\text{MLP}}$ algorithm, Line 5)

$$W^{\mathsf{h}} \in \mathbf{R}^{l \times (p+1)} \qquad \mathbf{x} \in \mathbf{R}^{p+1}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w_{10}^{\mathsf{h}} & \dots & w_{1p}^{\mathsf{h}} \\ & \vdots & \\ w_{l0}^{\mathsf{h}} & \dots & w_{lp}^{\mathsf{h}} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}\right) = \begin{bmatrix} y_1^{\mathsf{h}} \\ \vdots \\ y_l^{\mathsf{h}} \end{bmatrix}$$

(b) Propagate $\mathbf{y}^{\mathsf{h}}$ from hidden to output layer: (IGD$_{\text{MLP}}$ algorithm, Line 6)

$$W^{\mathsf{o}} \in \mathbf{R}^{k \times (l+1)} \qquad \mathbf{y}^{\mathsf{h}} \in \mathbf{R}^{l+1} \qquad \mathbf{y} \in \mathbf{R}^{k}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w_{10}^{\mathsf{o}} & \dots & w_{1l}^{\mathsf{o}} \\ & \vdots & \\ w_{k0}^{\mathsf{o}} & \dots & w_{kl}^{\mathsf{o}} \end{bmatrix} \begin{bmatrix} 1 \\ y_1^{\mathsf{h}} \\ \vdots \\ y_p^{\mathsf{h}} \end{bmatrix}\right) = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}$$

© STEIN/VÖLSKE 2021

Multilayer Perceptron

(1) Forward Propagation: Batch Mode [network architecture]

(a) Propagate $\mathbf{x}$ from input to hidden layer: (IGD$_{\text{MLP}}$ algorithm, Line 5)

$$W^{\mathsf{h}} \in \mathbf{R}^{l \times (p+1)} \qquad D \subset \mathbf{R}^{p+1}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w_{10}^{\mathsf{h}} & \dots & w_{1p}^{\mathsf{h}} \\ & \vdots & \\ w_{l0}^{\mathsf{h}} & \dots & w_{lp}^{\mathsf{h}} \end{bmatrix} \begin{bmatrix} 1 & \dots & 1 \\ x_{11} & \dots & x_{1n} \\ & \vdots & \\ x_{p1} & \dots & x_{pn} \end{bmatrix}\right) = \begin{bmatrix} y_{11}^{\mathsf{h}} & \dots & y_{1n}^{\mathsf{h}} \\ & \vdots & \\ y_{l1}^{\mathsf{h}} & \dots & y_{ln}^{\mathsf{h}} \end{bmatrix}$$

(b) Propagate $\mathbf{y}^{\mathsf{h}}$ from hidden to output layer: (IGD$_{\text{MLP}}$ algorithm, Line 6)

$$W^{\mathsf{o}} \in \mathbf{R}^{k \times (l+1)}$$

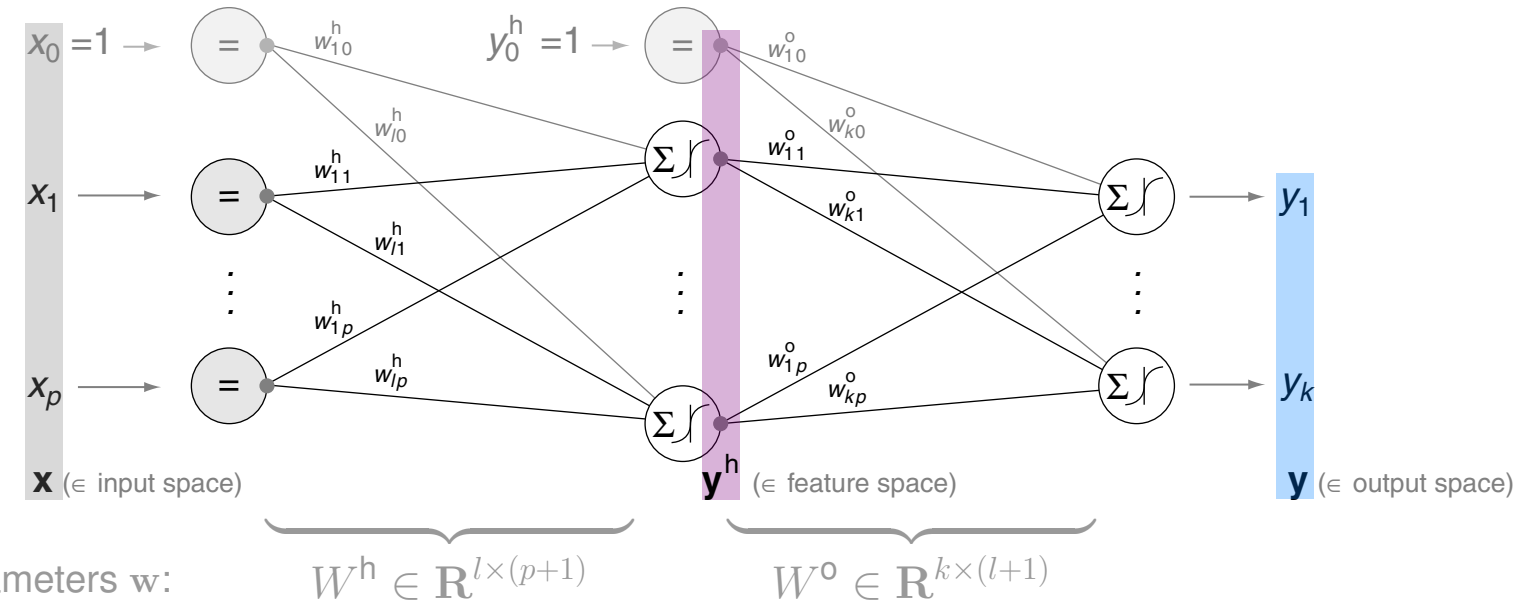
$$\boldsymbol{\sigma}\left(\begin{bmatrix} w_{10}^{\mathsf{o}} & \dots & w_{1l}^{\mathsf{o}} \\ & \vdots & \\ w_{k0}^{\mathsf{o}} & \dots & w_{kl}^{\mathsf{o}} \end{bmatrix} \begin{bmatrix} 1 & \dots & 1 \\ y_{11}^{\mathsf{h}} & \dots & y_{1n}^{\mathsf{h}} \\ & \vdots & \\ y_{l1}^{\mathsf{h}} & \dots & y_{ln}^{\mathsf{h}} \end{bmatrix}\right) = \begin{bmatrix} y_{11} & \dots & y_{1n} \\ & \vdots & \\ y_{k1} & \dots & y_{kn} \end{bmatrix}$$

© STEIN/VÖLSKE 2021

Multilayer Perceptron

(2) Backward Propagation

The considered multilayer perceptron $\mathbf{y}(\mathbf{x})$:

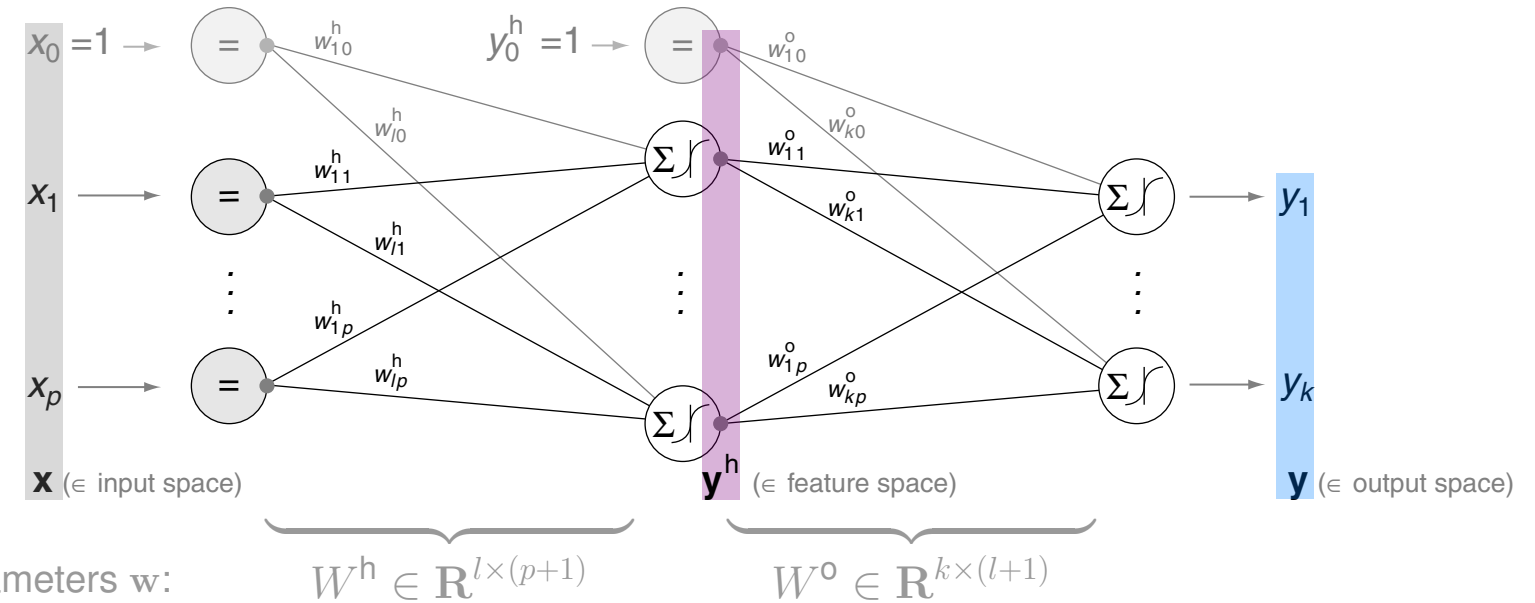


© STEIN/VÖLSKE 2021

Multilayer Perceptron

(2) Backward Propagation (continued)

The considered multilayer perceptron $\mathbf{y}(\mathbf{x})$:



Weight update (aka. backward propagation) wrt. the global squared loss:

$$L_2(\mathbf{w}) = \frac{1}{2} \cdot \mathrm{RSS}(\mathbf{w}) = \frac{1}{2} \cdot \sum_{(\mathbf{x},\mathbf{c}(\mathbf{x})) \in D} \sum_{o=1}^{k} (c_o(\mathbf{x}) - y_o)^2$$

© STEIN/VÖLSKE 2021

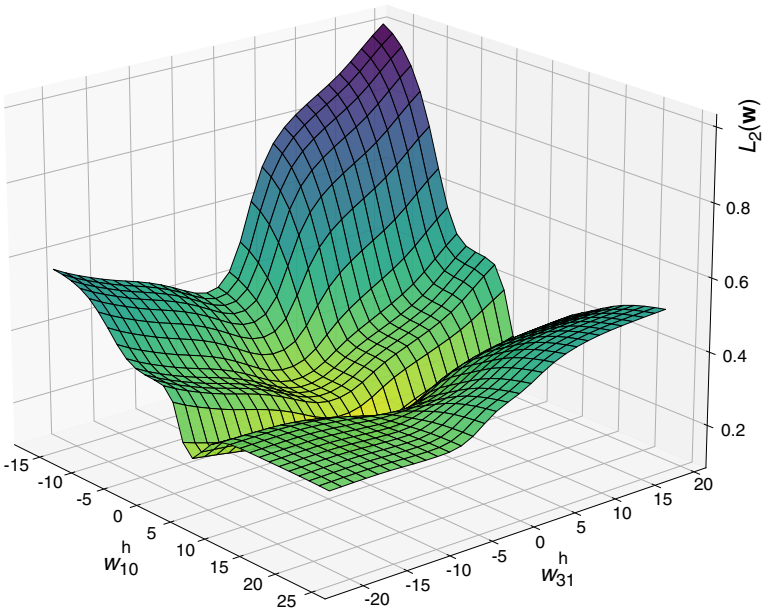
Multilayer Perceptron

(2) Backward Propagation (continued)

$L_2(\mathbf{w})$ usually contains various local minima:

$$\mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}\left(W^{\mathsf{o}}\left(\overset{1}{\boldsymbol{\sigma}}(W^{\mathsf{h}}\mathbf{x})\right)\right)$$

$$L_2(\mathbf{w}) = \frac{1}{2} \cdot \sum_{(\mathbf{x},\mathbf{c}(\mathbf{x}))\in D} \sum_{o=1}^{k} (c_o(\mathbf{x}) - y_o)^2$$



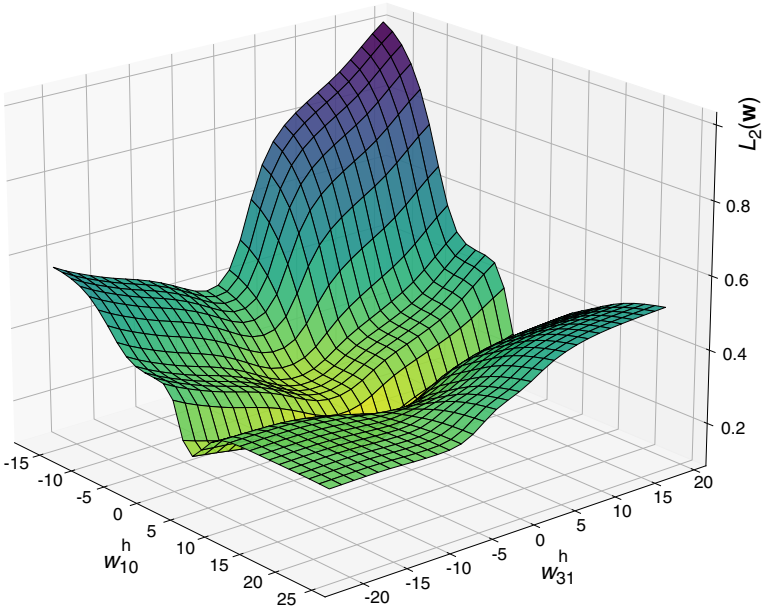
[model function $y(\mathbf{x})$]

© STEIN/VÖLSKE 2021

Multilayer Perceptron

(2) Backward Propagation (continued)

$L_2(\mathbf{w})$ usually contains various local minima:



[model function $y(\mathbf{x})$]

$$\mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}\left(W^{\mathsf{o}}\left(\overset{1}{\boldsymbol{\sigma}}_{(W^{\mathsf{h}}\mathbf{x})}\right)\right)$$

$$L_2(\mathbf{w}) = \frac{1}{2} \cdot \sum_{(\mathbf{x},\mathbf{c}(\mathbf{x}))\in D} \sum_{o=1}^{k} (c_o(\mathbf{x}) - y_o)^2$$

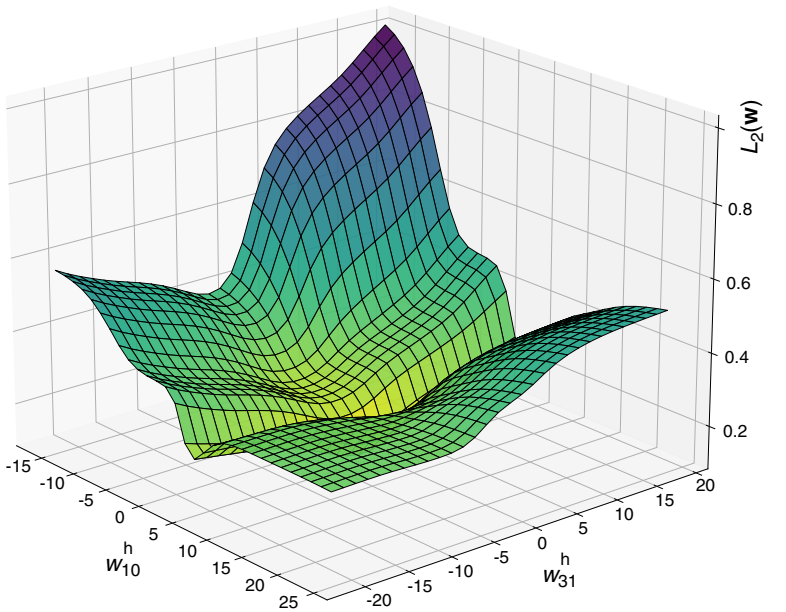
$$\nabla L_2(\mathbf{w}) = \left(\frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{10}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{kl}}, \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{10}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{lp}}\right)$$

© STEIN/VÖLSKE 2021

Multilayer Perceptron

(2) Backward Propagation (continued)

$L_2(\mathbf{w})$ usually contains various local minima:



[model function $y(\mathbf{x})$]

$$\mathbf{y}(\mathbf{x}) = \boldsymbol{\sigma}\left(W^{\mathsf{o}}\left(\overset{1}{\boldsymbol{\sigma}(W^{\mathsf{h}}\mathbf{x})}\right)\right)$$

$$L_2(\mathbf{w}) = \frac{1}{2} \cdot \sum_{(\mathbf{x},\mathbf{c}(\mathbf{x}))\in D} \sum_{o=1}^{k} (c_o(\mathbf{x}) - y_o)^2$$

$$\nabla L_2(\mathbf{w}) = \left(\frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{10}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{kl}}, \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{10}}, \ldots, \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{lp}}\right)$$

(a) Gradient in direction of W^{o}, written as matrix:

$$\begin{bmatrix} \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{10}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{1l}} \\ & \vdots & \\ \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{k0}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{kl}} \end{bmatrix} \equiv \nabla^{\mathsf{o}} L_2(\mathbf{w})$$

(b) Gradient in direction of W^{h}:

$$\begin{bmatrix} \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{10}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{1p}} \\ & \vdots & \\ \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{l0}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{h}}_{lp}} \end{bmatrix} \equiv \nabla^{\mathsf{h}} L_2(\mathbf{w})$$

© STEIN/VÖLSKE 2021

Remarks:

- Basically, the computation of the gradient $\nabla L_2(\mathbf{w})$ is independent of the organization of the weights in matrices W^{h} and W^{o} of a network $\mathbf{y}(\mathbf{x})$. Adopt the following view instead:

  To compute $\nabla L_2(\mathbf{w})$ one has to compute each of its components $\partial L_2(\mathbf{w})/\partial w$, $w \in \mathbf{w}$, since each weight (parameter) has a certain impact on the global loss $L_2(\mathbf{w})$ of the network. This impact—as well as the computation of this impact—is different for different weights, but it is canonical for all weights of the same layer though: note that each weight w influences "only" its direct and indirect successor nodes, and that the structure of the influenced successor graph (in fact a tree) is identical for all weights of the same layer.

  Hence it is convenient, but not necessary, to compute the components of the gradient layer-wise (matrix-wise). Even more, when deriving the partial derivation $\partial L_2(\mathbf{w})/\partial w$ of an arbitrary weight $w \in \mathbf{w}$ only two cases need to be distinguished: (a) w belongs to the output layer, or (b) w belongs to some hidden layer. Both derivations are shown in the following.

© STEIN/VÖLSKE 2021

Multilayer Perceptron

(2) Backward Propagation (continued)

(a) Update of weight matrix W^{o} : (IGD$_{\text{MLP}}$ algorithm, Line 8)

$$W^{\mathrm{o}} = W^{\mathrm{o}} + \Delta W^{\mathrm{o}},$$

using the ∇^{o}-gradient of the loss function $L_2(\mathbf{w})$ to take steepest descent:

$$\Delta W^{\mathrm{o}} \;=\; -\eta \cdot \nabla^{\mathrm{o}} L_2(\mathbf{w})$$

© STEIN/VÖLSKE 2021

Multilayer Perceptron

(2) Backward Propagation (continued)

(a) Update of weight matrix W^{o} : (IGD$_{\text{MLP}}$ algorithm, Line 8)

$$W^{\mathsf{o}} = W^{\mathsf{o}} + \Delta W^{\mathsf{o}},$$

using the ∇^{o}-gradient of the loss function $L_2(\mathbf{w})$ to take steepest descent:

$$\Delta W^{\mathsf{o}} = -\eta \cdot \nabla^{\mathsf{o}} L_2(\mathbf{w})$$

$$= -\eta \cdot \begin{bmatrix} \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{10}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{1l}} \\ & \vdots & \\ \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{k0}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w^{\mathsf{o}}_{kl}} \end{bmatrix}$$

$$\vdots$$

$$= \eta \cdot \sum_{D} \underbrace{\left[(\mathbf{c}(\mathbf{x}) - \mathbf{y}(\mathbf{x})) \odot \mathbf{y}(\mathbf{x}) \odot (1 - \mathbf{y}(\mathbf{x})) \right]}_{\boldsymbol{\delta}^{\mathsf{o}}} \otimes \mathbf{y}^{\mathsf{h}}\big|_{1,\ldots,l}$$

© STEIN/VÖLSKE 2021

Multilayer Perceptron

(2) Backward Propagation (continued)

(b) Update of weight matrix $W^{\mathbf{h}}$: (IGD$_{\text{MLP}}$ algorithm, Line 10)

$$W^{\mathsf{h}} = W^{\mathsf{h}} + \Delta W^{\mathsf{h}},$$

using the ∇^{h}-gradient of the loss function $L_2(\mathbf{w})$ to take steepest descent:

$$\Delta W^{\mathsf{h}} = -\eta \cdot \nabla^{\mathsf{h}} L_2(\mathbf{w})$$

$$= -\eta \cdot \begin{bmatrix} \frac{\partial L_2(\mathbf{w})}{\partial w_{10}^{\mathsf{h}}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w_{1p}^{\mathsf{h}}} \\ & \vdots & \\ \frac{\partial L_2(\mathbf{w})}{\partial w_{l0}^{\mathsf{h}}} & \cdots & \frac{\partial L_2(\mathbf{w})}{\partial w_{lp}^{\mathsf{h}}} \end{bmatrix}$$

$$\vdots$$

$$= \eta \cdot \sum_{D} \underbrace{\left[(W^{\mathsf{o}T} \boldsymbol{\delta}^{\mathsf{o}}) \odot \mathbf{y}^{\mathsf{h}}(\mathbf{x}) \odot (1 - \mathbf{y}^{\mathsf{h}}(\mathbf{x})) \right]_{1,\ldots,l}}_{\boldsymbol{\delta}^{\mathsf{h}}} \otimes \mathbf{x}$$

© STEIN/VÖLSKE 2021

Multilayer Perceptron

The IGD Algorithm for Multilayer Perceptrons

Algorithm:	IGD_{MLP}	Incremental Gradient Descent for two-layer MLP
Input:	D	Training examples $(\mathbf{x}, \mathbf{c}(\mathbf{x}))$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c}(\mathbf{x}) \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\text{h}}, W^{\text{o}}$	Weights of $l \cdot (p+1)$ hidden and $k \cdot (l+1)$ output layer units (= hypothesis).

```
initialize_random_weights(W^h, W^o),  t = 0
REPEAT
  t = t + 1
  FOREACH (x, c(x)) ∈ D DO
  ENDDO
UNTIL(convergence(c(D), y(D)) OR t > t_max)
return(W^h, W^o)
```


© STEIN/VÖLSKE 2021

Multilayer Perceptron

The IGD Algorithm for Multilayer Perceptrons

Algorithm:	IGD_{MLP}	Incremental Gradient Descent for two-layer MLP
Input:	D	Training examples $(\mathbf{x}, \mathbf{c}(\mathbf{x}))$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c}(\mathbf{x}) \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\text{h}}, W^{\text{o}}$	Weights of $l \cdot (p+1)$ hidden and $k \cdot (l+1)$ output layer units (= hypothesis).

```
initialize_random_weights(W^h, W^o),  t = 0
REPEAT
  t = t + 1
  FOREACH (x, c(x)) ∈ D DO
    y^h = (1, σ(W^h x))  // forward propagation, x is extended by x_0 = 1
    y = σ(W^o y^h)
  ENDDO
UNTIL(convergence(c(D), y(D)) OR t > t_max)
return(W^h, W^o)
```


© STEIN/VÖLSKE 2021

Multilayer Perceptron

The IGD Algorithm for Multilayer Perceptrons

Algorithm:	IGD_{MLP}	Incremental Gradient Descent for two-layer MLP
Input:	D	Training examples $(\mathbf{x}, \mathbf{c}(\mathbf{x}))$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c}(\mathbf{x}) \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\text{h}}, W^{\text{o}}$	Weights of $l \cdot (p+1)$ hidden and $k \cdot (l+1)$ output layer units (= hypothesis).

```
initialize_random_weights(W^h, W^o),  t = 0
REPEAT
  t = t + 1
  FOREACH (x, c(x)) ∈ D DO
    y^h = (1, σ(W^h x))  // forward propagation, x is extended by x_0 = 1
    y = σ(W^o y^h)

    δ^o = (c(x) − y) ⊙ y ⊙ (1 − y)  // backward propagation
    ΔW^o = η · (δ^o ⊗ y^h|_{1,...,l})
    δ^h = [(W^o^T δ^o) ⊙ y^h ⊙ (1 − y^h)]_{1,...,l}
    ΔW^h = η · (δ^h ⊗ x)



  ENDDO
UNTIL(convergence(c(D), y(D)) OR t > t_max)
return(W^h, W^o)
```


© STEIN/VÖLSKE 2021

Multilayer Perceptron

The IGD Algorithm for Multilayer Perceptrons

Algorithm:	IGD_{MLP}	Incremental Gradient Descent for two-layer MLP
Input:	D	Training examples $(\mathbf{x}, \mathbf{c}(\mathbf{x}))$ with $\mathbf{x} \in \mathbf{R}^p$, $\mathbf{c}(\mathbf{x}) \in \{0,1\}^k$.
	η	Learning rate, a small positive constant.
Output:	$W^{\text{h}}, W^{\text{o}}$	Weights of $l \cdot (p+1)$ hidden and $k \cdot (l+1)$ output layer units (= hypothesis).

```
initialize_random_weights(W^h, W^o),  t = 0
REPEAT
  t = t + 1
  FOREACH (x, c(x)) ∈ D DO
    y^h = (1, σ(W^h x))  // forward propagation, x is extended by x_0 = 1
    y = σ(W^o y^h)

    δ^o = (c(x) − y) ⊙ y ⊙ (1 − y)  // backward propagation
    ΔW^o = η · (δ^o ⊗ y^h|_{1,...,l})
    δ^h = [(W^o^T δ^o) ⊙ y^h ⊙ (1 − y^h)]_{1,...,l}
    ΔW^h = η · (δ^h ⊗ x)

    W^o = W^o + ΔW^o  // weight update
    W^h = W^h + ΔW^h
  ENDDO
UNTIL(convergence(c(D), y(D)) OR t > t_max)
return(W^h, W^o)
```


© STEIN/VÖLSKE 2021

Remarks:

- The symbol »$\odot$« denotes the Hadamard product, also known as the element-wise or the Schur product. It is a binary operation that takes two matrices of the same dimensions and produces another matrix of the same dimension as the operands, where each element is the product of the respective elements of the two original matrices. [Wikipedia]

- The symbol »$\otimes$« denotes the dyadic product, also called outer product or tensor product. The dyadic product takes two vectors and returns a second order tensor called a dyadic in this context: $\mathbf{v} \otimes \mathbf{w} \equiv \mathbf{v}\mathbf{w}^T$. [Wikipedia]

© STEIN/VÖLSKE 2021

Remarks (how to derive $\nabla^{\mathsf{o}} L_2(\mathbf{w})$) :

- Consider the partial derivative for a weight of the output layer:

$$
\begin{aligned}
\frac{\partial}{\partial w_{ij}^{\mathsf{o}}} L_2(\mathbf{w}) &= \frac{\partial}{\partial w_{ij}^{\mathsf{o}}} \frac{1}{2} \cdot \sum_{(\mathbf{x},c(\mathbf{x}))\in D} \sum_{o=1}^{k} (c_o(\mathbf{x}) - y_o(\mathbf{x}))^2 \\
&= \frac{1}{2} \cdot \sum_{D} \frac{\partial}{\partial w_{ij}^{\mathsf{o}}} \sum_{o=1}^{k} (c_o(\mathbf{x}) - y_o(\mathbf{x}))^2 \quad \text{// } i\text{-th summand depends on } w_{ij}^{\mathsf{h}}. \\
&= \frac{1}{2} \cdot \sum_{D} \frac{\partial}{\partial w_{ij}^{\mathsf{o}}} (c_i(\mathbf{x}) - y_i(\mathbf{x}))^2 \\
&= \sum_{D} (c_i(\mathbf{x}) - y_i(\mathbf{x})) \cdot \frac{\partial}{\partial w_{ij}^{\mathsf{o}}} (c_i(\mathbf{x}) - y_i(\mathbf{x})) \quad \text{// } y_i(\mathbf{x}) \text{ depends on } w_{ij}^{\mathsf{h}}. \\
&\overset{(1)}{=} -\sum_{D} (c_i(\mathbf{x}) - y_i(\mathbf{x})) \cdot \frac{\partial}{\partial w_{ij}^{\mathsf{o}}} \sigma\left(W_{i*}^{\mathsf{o}} \mathbf{y}^{\mathsf{h}}(\mathbf{x})\right) \quad \text{// } j\text{-th summand depends on } w_{ij}^{\mathsf{h}}. \\
&\overset{(1,2)}{=} -\sum_{D} \underbrace{(c_i(\mathbf{x}) - y_i(\mathbf{x})) \cdot y_i(\mathbf{x}) \cdot (1 - y_i(\mathbf{x}))}_{\delta^{\mathsf{o}}} \cdot y_j^{\mathsf{h}}(\mathbf{x})
\end{aligned}
$$

- Plugging the results for $\frac{\partial}{\partial w_{ij}^{\mathsf{o}}} L_2(\mathbf{w})$ into $-\eta \cdot (\cdots)$ yields the update formula for ΔW^{o}.
- Hints:

  (1) $y_i(\mathbf{x}) = \boldsymbol{\sigma}_i\left(W^{\mathsf{o}} \mathbf{y}^{\mathsf{h}}(\mathbf{x})\right) = \sigma\left(W_{i*}^{\mathsf{o}} \mathbf{y}^{\mathsf{h}}(\mathbf{x})\right)$

  (2) $\frac{d}{dz}\sigma(z) = \sigma(z) \cdot (1 - \sigma(z))$

© STEIN/VÖLSKE 2021