Machine Learning: Perceptrons

Prof. Dr. Martin Riedmiller

Albert-Ludwigs-University Freiburg
AG Maschinelles Lernen
Neural Networks

- The human brain has approximately $10^{11}$ neurons
- Switching time $0.001 \text{s}$ (computer $\approx 10^{-10} \text{s}$)
- Connections per neuron: $10^4 - 10^5$
- $0.1 \text{s}$ for face recognition
- I.e. at most 100 computation steps
- parallelism
- additionally: robustness, distributedness
- ML aspects: use biology as an inspiration for artificial neural models and algorithms; do not try to explain biology: technically imitate and exploit capabilities
Biological Neurons

- Dentrites input information to the cell
- Neuron fires (has action potential) if a certain threshold for the voltage is exceeded
- Output of information by axon
- The axon is connected to dentrites of other cells via synapses
- Learning corresponds to adaptation of the efficiency of synapse, of the synaptical weight
Historical ups and downs

- 1942: artificial neurons (McCulloch/Pitts)
- 1949: Hebbian learning (Hebb)
- 1958: Rosenblatt perceptron (Rosenblatt)
- 1960: Adaline/MAdaline (Widrow/Hoff)
- 1960: Lernmatrix (Steinbuch)
- 1969: “perceptrons” (Minsky/Papert)
- 1970: evolutionary algorithms (Rechenberg)
- 1972: self-organizing maps (Kohonen)
- 1982: Hopfield networks (Hopfield)
- 1986: Backpropagation (orig. 1974)
- 1992: Bayes inference
- 1992: Computational learning theory
- 1992: Support vector machines
- 1992: Boosting
perceptrons (Rosenblatt 1958, Minsky/Papert 1969) are generalized variants of a former, more simple model (McCulloch/Pitts neurons, 1942):

- inputs are weighted
- weights are real numbers (positive and negative)
- no special inhibitory inputs
perceptrons (Rosenblatt 1958, Minsky/Papert 1969) are generalized variants of a former, more simple model (McCulloch/Pitts neurons, 1942):

- inputs are weighted
- weights are real numbers (positive and negative)
- no special inhibitory inputs

A perceptron with $n$ inputs is described by a weight vector $\vec{w} = (w_1, \ldots, w_n)^T \in \mathbb{R}^n$ and a threshold $\theta \in \mathbb{R}$. It calculates the following function:

$$
(x_1, \ldots, x_n)^T \mapsto y = \begin{cases} 
1 & \text{if } x_1 w_1 + x_2 w_2 + \cdots + x_n w_n \geq \theta \\
0 & \text{if } x_1 w_1 + x_2 w_2 + \cdots + x_n w_n < \theta
\end{cases}
$$
for convenience: replacing the threshold by an additional weight (bias weight) $w_0 = -\theta$. A perceptron with weight vector $\vec{w}$ and bias weight $w_0$ performs the following calculation:

$$(x_1, \ldots, x_n)^T \mapsto y = f_{\text{step}}(w_0 + \sum_{i=1}^{n} (w_i x_i)) = f_{\text{step}}(w_0 + \langle \vec{w}, \vec{x} \rangle)$$

with

$$f_{\text{step}}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$
for convenience: replacing the threshold by an additional weight (bias weight) \( w_0 = -\theta \). A perceptron with weight vector \( \vec{w} \) and bias weight \( w_0 \) performs the following calculation:

\[
(x_1, \ldots, x_n)^T \mapsto y = f_{\text{step}}(w_0 + \sum_{i=1}^{n}(w_i x_i)) = f_{\text{step}}(w_0 + \langle \vec{w}, \vec{x} \rangle)
\]

with

\[
f_{\text{step}}(z) = \begin{cases} 
1 & \text{if } z \geq 0 \\
0 & \text{if } z < 0 
\end{cases}
\]
geometric interpretation of a perceptron:

- input patterns \((x_1, \ldots, x_n)\) are points in \(n\)-dimensional space
geometric interpretation of a perceptron:

- input patterns \((x_1, \ldots, x_n)\) are points in \(n\)-dimensional space
- points with \(w_0 + \langle \vec{w}, \vec{x} \rangle = 0\) are on a hyperplane defined by \(w_0\) and \(\vec{w}\)
geometric interpretation of a perceptron:

- input patterns \((x_1, \ldots, x_n)\) are points in \(n\)-dimensional space
- points with \(w_0 + \langle \vec{w}, \vec{x} \rangle = 0\) are on a hyperplane defined by \(w_0\) and \(\vec{w}\)
- points with \(w_0 + \langle \vec{w}, \vec{x} \rangle > 0\) are above the hyperplane
geometric interpretation of a perceptron:

- input patterns \((x_1, \ldots, x_n)\) are points in \(n\)-dimensional space
- points with \(w_0 + \langle \vec{w}, \vec{x} \rangle = 0\) are on a hyperplane defined by \(w_0\) and \(\vec{w}\)
- points with \(w_0 + \langle \vec{w}, \vec{x} \rangle > 0\) are above the hyperplane
- points with \(w_0 + \langle \vec{w}, \vec{x} \rangle < 0\) are below the hyperplane
geometric interpretation of a perceptron:

- input patterns \((x_1, \ldots, x_n)\) are points in \(n\)-dimensional space
- points with \(w_0 + \langle \vec{w}, \vec{x} \rangle = 0\) are on a hyperplane defined by \(w_0\) and \(\vec{w}\)
- points with \(w_0 + \langle \vec{w}, \vec{x} \rangle > 0\) are above the hyperplane
- points with \(w_0 + \langle \vec{w}, \vec{x} \rangle < 0\) are below the hyperplane
- perceptrons partition the input space into two halfspaces along a hyperplane
Perceptron learning problem

- perceptrons can automatically adapt to example data => Supervised Learning: Classification
Perceptron learning problem

- perceptrons can automatically adapt to example data ⇒ Supervised Learning: Classification

- perceptron learning problem:
  
  given:
  
  - a set of input patterns $\mathcal{P} \subseteq \mathbb{R}^n$, called the set of positive examples
  - another set of input patterns $\mathcal{N} \subseteq \mathbb{R}^n$, called the set of negative examples

  task:
  
  - generate a perceptron that yields $1$ for all patterns from $\mathcal{P}$ and $0$ for all patterns from $\mathcal{N}$
Perceptron learning problem

- Perceptrons can automatically adapt to example data ⇒ Supervised Learning: Classification

- Perceptron learning problem:
  - given:
    - a set of input patterns $\mathcal{P} \subseteq \mathbb{R}^n$, called the set of positive examples
    - another set of input patterns $\mathcal{N} \subseteq \mathbb{R}^n$, called the set of negative examples
  - task:
    - generate a perceptron that yields 1 for all patterns from $\mathcal{P}$ and 0 for all patterns from $\mathcal{N}$

- Obviously, there are cases in which the learning task is unsolvable, e.g. $\mathcal{P} \cap \mathcal{N} \neq \emptyset$
Lemma (strict separability):
Whenever exist a perceptron that classifies all training patterns accurately, there is also a perceptron that classifies all training patterns accurately and no training pattern is located on the decision boundary, i.e. 
\[ \vec{w}_0 + \langle \vec{w}, \vec{x} \rangle \neq 0 \] for all training patterns.
Lemma (strict separability):
Whenever exist a perceptron that classifies all training patterns accurately, there is also a perceptron that classifies all training patterns accurately and no training pattern is located on the decision boundary, i.e.
\[ \vec{w}_0 + \langle \vec{w}, \vec{x} \rangle \neq 0 \text{ for all training patterns.} \]

Proof:
Let \((\vec{w}, w_0)\) be a perceptron that classifies all patterns accurately. Hence,
\[
\langle \vec{w}, \vec{x} \rangle + w_0 \begin{cases} 
\geq 0 & \text{for all } \vec{x} \in \mathcal{P} \\
< 0 & \text{for all } \vec{x} \in \mathcal{N}
\end{cases}
\]
Lemma (strict separability):
Whenever exist a perceptron that classifies all training patterns accurately, there is also a perceptron that classifies all training patterns accurately and no training pattern is located on the decision boundary, i.e. \( \vec{w}_0 + \langle \vec{w}, \vec{x} \rangle \neq 0 \) for all training patterns.

Proof:
Let \((\vec{w}, w_0)\) be a perceptron that classifies all patterns accurately. Hence,

\[
\langle \vec{w}, \vec{x} \rangle + w_0 \begin{cases} 
\geq 0 & \text{for all } \vec{x} \in \mathcal{P} \\
< 0 & \text{for all } \vec{x} \in \mathcal{N}
\end{cases}
\]

Define \( \varepsilon = \min \{- (\langle \vec{w}, \vec{x} \rangle + w_0) | \vec{x} \in \mathcal{N} \} \). Then:

\[
\langle \vec{w}, \vec{x} \rangle + w_0 + \frac{\varepsilon}{2} \begin{cases} 
\geq \frac{\varepsilon}{2} > 0 & \text{for all } \vec{x} \in \mathcal{P} \\
\leq -\frac{\varepsilon}{2} < 0 & \text{for all } \vec{x} \in \mathcal{N}
\end{cases}
\]
Lemma (strict separability):
Whenever exist a perceptron that classifies all training patterns accurately, there is also a perceptron that classifies all training patterns accurately and no training pattern is located on the decision boundary, i.e. $\vec{w}_0 + \langle \vec{w}, \vec{x} \rangle \neq 0$ for all training patterns.

**Proof:**
Let $(\vec{w}, w_0)$ be a perceptron that classifies all patterns accurately. Hence,

$$\langle \vec{w}, \vec{x} \rangle + w_0 \begin{cases} \geq 0 & \text{for all } \vec{x} \in \mathcal{P} \\ < 0 & \text{for all } \vec{x} \in \mathcal{N} \end{cases}$$

Define $\varepsilon = \min \{-\left(\langle \vec{w}, \vec{x} \rangle + w_0 \right) | \vec{x} \in \mathcal{N}\}$. Then:

$$\langle \vec{w}, \vec{x} \rangle + w_0 + \frac{\varepsilon}{2} \begin{cases} \geq \frac{\varepsilon}{2} > 0 & \text{for all } \vec{x} \in \mathcal{P} \\ \leq -\frac{\varepsilon}{2} < 0 & \text{for all } \vec{x} \in \mathcal{N} \end{cases}$$

Thus, the perceptron $(\vec{w}, w_0 + \frac{\varepsilon}{2})$ proves the lemma.
**Perceptron learning algorithm:**

**idea**

- Assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$:
  $$\langle \vec{w}, \vec{x} \rangle + w_0 < 0$$

- How can we change $\vec{w}$ and $w_0$ to avoid this error?
Perceptron learning algorithm: 

**idea**

- assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$:
  $$\langle \vec{w}, \vec{x} \rangle + w_0 < 0$$

- how can we change $\vec{w}$ and $w_0$ to avoid this error? – we need to increase $\langle \vec{w}, \vec{x} \rangle + w_0$
Perceptron learning algorithm: idea

- Assume, the perceptron makes an error on a pattern \( \vec{x} \in \mathcal{P} \):
  \[ \langle \vec{w}, \vec{x} \rangle + w_0 < 0 \]

- How can we change \( \vec{w} \) and \( w_0 \) to avoid this error? We need to increase \( \langle \vec{w}, \vec{x} \rangle + w_0 \)
  - Increase \( w_0 \)
  - If \( x_i > 0 \), increase \( w_i \)
  - If \( x_i < 0 \) (‘negative influence’), decrease \( w_i \)

- Perceptron learning algorithm: add \( \vec{x} \) to \( \vec{w} \), add 1 to \( w_0 \) in this case. Errors on negative patterns: analogously.
Perceptron learning algorithm:

**idea**

- assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$:
  $$\langle \vec{w}, \vec{x} \rangle + w_0 < 0$$

- how can we change $\vec{w}$ and $w_0$ to avoid this error? – we need to increase $\langle \vec{w}, \vec{x} \rangle + w_0$
  - increase $w_0$
  - if $x_i > 0$, increase $w_i$
  - if $x_i < 0$ ('negative influence'), decrease $w_i$

- perceptron learning algorithm: add $\vec{x}$ to $\vec{w}$, add 1 to $w_0$ in this case. Errors on negative patterns: analogously.
assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$:
$$\langle \vec{w}, \vec{x} \rangle + w_0 < 0$$

how can we change $\vec{w}$ and $w_0$ to avoid this error? – we need to increase $\langle \vec{w}, \vec{x} \rangle + w_0$

- increase $w_0$
- if $x_i > 0$, increase $w_i$
- if $x_i < 0$ ('negative influence'), decrease $w_i$

perceptron learning algorithm: add $\vec{x}$ to $\vec{w}$, add 1 to $w_0$ in this case. Errors on negative patterns: analogously.
Perceptron learning algorithm: idea

- assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$:
  \[
  \langle \vec{w}, \vec{x} \rangle + w_0 < 0
  \]

- how can we change $\vec{w}$ and $w_0$ to avoid this error? – we need to increase $\langle \vec{w}, \vec{x} \rangle + w_0$
  - increase $w_0$
  - if $x_i > 0$, increase $w_i$
  - if $x_i < 0$ (‘negative influence’), decrease $w_i$

- perceptron learning algorithm: add $\vec{x}$ to $\vec{w}$, add 1 to $w_0$ in this case. Errors on negative patterns: analogously.
**Perceptron learning algorithm:**

**idea**

- assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$: 
  $\langle \vec{w}, \vec{x} \rangle + w_0 < 0$

- how can we change $\vec{w}$ and $w_0$ to avoid this error? – we need to increase $\langle \vec{w}, \vec{x} \rangle + w_0$
  - increase $w_0$
  - if $x_i > 0$, increase $w_i$
  - if $x_i < 0$ (‘negative influence’), decrease $w_i$

- perceptron learning algorithm: add $\vec{x}$ to $\vec{w}$, add 1 to $w_0$ in this case. Errors on negative patterns: analogously.

Geometric interpretation: increasing $w_0$
Perceptron learning algorithm: idea

- assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$:
  $\langle \vec{w}, \vec{x} \rangle + w_0 < 0$

- how can we change $\vec{w}$ and $w_0$ to avoid this error? – we need to increase $\langle \vec{w}, \vec{x} \rangle + w_0$
  - increase $w_0$
  - if $x_i > 0$, increase $w_i$
  - if $x_i < 0$ (’negative influence’), decrease $w_i$

- perceptron learning algorithm: add $\vec{x}$ to $\vec{w}$, add 1 to $w_0$ in this case. Errors on negative patterns: analogously.
Perceptron learning algorithm:

**idea**

- assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$:
  \[ \langle \vec{w}, \vec{x} \rangle + w_0 < 0 \]

- how can we change $\vec{w}$ and $w_0$ to avoid this error? – we need to increase $\langle \vec{w}, \vec{x} \rangle + w_0$
  - increase $w_0$
  - if $x_i > 0$, increase $w_i$
  - if $x_i < 0$ ('negative influence'), decrease $w_i$

- perceptron learning algorithm: add $\vec{x}$ to $\vec{w}$, add 1 to $w_0$ in this case. Errors on negative patterns: analogously.
Perceptron learning algorithm: idea

- assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$:
  $\langle \vec{w}, \vec{x} \rangle + w_0 < 0$
  
- how can we change $\vec{w}$ and $w_0$ to avoid this error? – we need to increase $\langle \vec{w}, \vec{x} \rangle + w_0$
  
  - increase $w_0$
  
  - if $x_i > 0$, increase $w_i$
  
  - if $x_i < 0$ ('negative influence'), decrease $w_i$

- perceptron learning algorithm: add $\vec{x}$ to $\vec{w}$, add 1 to $w_0$ in this case. Errors on negative patterns: analogously.

Geometric interpretation: modifying $\vec{w}$
**Perceptron learning algorithm:**

*idea*

- Assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$:
  $$\langle \vec{w}, \vec{x} \rangle + w_0 < 0$$

- How can we change $\vec{w}$ and $w_0$ to avoid this error? – We need to increase $\langle \vec{w}, \vec{x} \rangle + w_0$
  - Increase $w_0$
  - If $x_i > 0$, increase $w_i$
  - If $x_i < 0$ ('negative influence'), decrease $w_i$

- Perceptron learning algorithm: Add $\vec{x}$ to $\vec{w}$, add 1 to $w_0$ in this case. Errors on negative patterns: analogously.

*Geometric interpretation: modifying $\vec{w}$*
**Perceptron learning algorithm**

**Require:** positive training patterns $\mathcal{P}$ and a negative training examples $\mathcal{N}$

**Ensure:** if exists, a perceptron is learned that classifies all patterns accurately

1: initialize weight vector $\vec{w}$ and bias weight $w_0$ arbitrarily

2: while exist misclassified pattern $\vec{x} \in \mathcal{P} \cup \mathcal{N}$ do

3: if $\vec{x} \in \mathcal{P}$ then

4: $\vec{w} \leftarrow \vec{w} + \vec{x}$

5: $w_0 \leftarrow w_0 + 1$

6: else

7: $\vec{w} \leftarrow \vec{w} - \vec{x}$

8: $w_0 \leftarrow w_0 - 1$

9: end if

10: end while

11: return $\vec{w}$ and $w_0$
Perceptron learning algorithm:

example

\[ \mathcal{N} = \{(1, 0)^T, (1, 1)^T\}, \mathcal{P} = \{(0, 1)^T\} \]

→ exercise
Perceptron learning algorithm: convergence

Lemma (correctness of perceptron learning):
Whenever the perceptron learning algorithm terminates, the perceptron given by \((\vec{w}, w_0)\) classifies all patterns accurately.
Lemma (correctness of perceptron learning):
Whenever the perceptron learning algorithm terminates, the perceptron given by \((\vec{w}, w_0)\) classifies all patterns accurately.

Proof: follows immediately from algorithm.
Lemma (correctness of perceptron learning):
Whenever the perceptron learning algorithm terminates, the perceptron given by \((\vec{w}, w_0)\) classifies all patterns accurately.

*Proof:* follows immediately from algorithm.

Theorem (termination of perceptron learning):
Whenever exists a perceptron that classifies all training patterns correctly, the perceptron learning algorithm terminates.
Lemma (correctness of perceptron learning):
Whenever the perceptron learning algorithm terminates, the perceptron given by \((\vec{w}, w_0)\) classifies all patterns accurately.

Proof: follows immediately from algorithm.

Theorem (termination of perceptron learning):
Whenever exists a perceptron that classifies all training patterns correctly, the perceptron learning algorithm terminates.

Proof:
for simplification we will add the bias weight to the weight vector, i.e. \(\vec{w} = (w_0, w_1, \ldots, w_n)^T\), and 1 to all patterns, i.e. \(\vec{x} = (1, x_1, \ldots, x_n)^T\).

We will denote with \(\vec{w}^{(t)}\) the weight vector in the \(t\)-th iteration of perceptron learning and with \(\vec{x}^{(t)}\) the pattern used in the \(t\)-th iteration.
Inner product (dot product of two vectors $\vec{w}, \vec{x}$)

$$\langle \vec{w}, \vec{x} \rangle = \vec{w}^T \vec{x} = \sum_{i=1}^{n} w_i x_i$$
Inner product (dot product of two vectors $\vec{w}, \vec{x}$)

$$\langle \vec{w}, \vec{x} \rangle = \vec{w}^T \vec{x} = \sum_{i=1}^{n} w_{i} x_{i}$$

$$\langle \vec{w}, \vec{x} \rangle + \langle \vec{w}, \vec{y} \rangle = \langle \vec{w}, \vec{x} + \vec{y} \rangle$$
Inner product (dot product of two vectors $\vec{w}, \vec{x}$)

$$\langle \vec{w}, \vec{x} \rangle = \vec{w}^T \vec{x} = \sum_{i=1}^{n} w_i x_i$$

$$\langle \vec{w}, \vec{x} \rangle + \langle \vec{w}, \vec{y} \rangle = \langle \vec{w}, \vec{x} + \vec{y} \rangle$$

Euclidean norm:

$$||\vec{w}||^2 = \langle \vec{w}, \vec{w} \rangle = \sum_{i=1}^{n} w_i w_i$$
Perceptron learning algorithm: Preliminaries

Inner product (dot product of two vectors $\vec{w}, \vec{x}$)

$$\langle \vec{w}, \vec{x} \rangle = \vec{w}^T \vec{x} = \sum_{i=1}^{n} w_i x_i$$

$$\langle \vec{w}, \vec{x} \rangle + \langle \vec{w}, \vec{y} \rangle = \langle \vec{w}, \vec{x} + \vec{y} \rangle$$

Euclidean norm:

$$||\vec{w}||^2 = \langle \vec{w}, \vec{w} \rangle = \sum_{i=1}^{n} w_i w_i$$

Angle between two vectors:

$$\cos \angle (\vec{x}, \vec{y}) = \frac{\langle \vec{x}, \vec{y} \rangle}{||\vec{x}|| \cdot ||\vec{y}||}$$
Let be $\vec{w}^*$ a weight vector that strictly classifies all training patterns.
Let be $\vec{w}^*$ a weight vector that strictly classifies all training patterns.

$$\langle \vec{w}^*, \vec{w}^{(t+1)} \rangle = \langle \vec{w}^*, \vec{w}^{(t)} \pm \vec{x}^{(t)} \rangle$$

$$= \langle \vec{w}^*, \vec{w}^{(t)} \rangle \pm \langle \vec{w}^*, \vec{x}^{(t)} \rangle$$

$$\geq \langle \vec{w}^*, \vec{w}^{(t)} \rangle + \delta$$

with $\delta := \min \left(\{ \langle \vec{w}^*, \vec{x} \rangle | \vec{x} \in \mathcal{P} \} \cup \{- \langle \vec{w}^*, \vec{x} \rangle | \vec{x} \in \mathcal{N} \} \right)$
Let be $\vec{w}^*$ a weight vector that strictly classifies all training patterns.

$$\langle \vec{w}^*, \vec{w}^{(t+1)} \rangle = \langle \vec{w}^*, \vec{w}^{(t)} \pm \vec{x}^{(t)} \rangle$$

$$= \langle \vec{w}^*, \vec{w}^{(t)} \rangle \pm \langle \vec{w}^*, \vec{x}^{(t)} \rangle$$

$$\geq \langle \vec{w}^*, \vec{w}^{(t)} \rangle + \delta$$

with $\delta := \min \left( \{ \langle \vec{w}^*, \vec{x} \rangle | \vec{x} \in \mathcal{P} \} \cup \{-\langle \vec{w}^*, \vec{x} \rangle | \vec{x} \in \mathcal{N} \} \right)$

$\delta > 0$ since $\vec{w}^*$ strictly classifies all patterns
Let be $\vec{w}^*$ a weight vector that strictly classifies all training patterns.

$$
\langle \vec{w}^*, \vec{w}^{(t+1)} \rangle = \langle \vec{w}^*, \vec{w}^{(t)} \pm \vec{x}^{(t)} \rangle \\
= \langle \vec{w}^*, \vec{w}^{(t)} \rangle \pm \langle \vec{w}^*, \vec{x}^{(t)} \rangle \\
\geq \langle \vec{w}^*, \vec{w}^{(t)} \rangle + \delta
$$

with $\delta := \min (\{ \langle \vec{w}^*, \vec{x} \rangle | \vec{x} \in \mathcal{P} \} \cup \{- \langle \vec{w}^*, \vec{x} \rangle | \vec{x} \in \mathcal{N} \})$

$\delta > 0$ since $\vec{w}^*$ strictly classifies all patterns.

Hence,

$$
\langle \vec{w}^*, \vec{w}^{(t+1)} \rangle \geq \langle \vec{w}^*, \vec{w}^{(0)} \rangle + (t + 1) \delta
$$
Perceptron learning algorithm: convergence proof (cont.)

\[ ||\vec{w}^{(t+1)}||^2 = \langle \vec{w}^{(t+1)}, \vec{w}^{(t+1)} \rangle \]
\[ = \langle \vec{w}^{(t)} \pm \vec{x}^{(t)}, \vec{w}^{(t)} \pm \vec{x}^{(t)} \rangle \]
\[ = ||\vec{w}^{(t)}||^2 \pm 2 \langle \vec{x}^{(t)}, \vec{w}^{(t)} \rangle + ||\vec{x}^{(t)}||^2 \]

consider \( \langle \vec{x}^{(t)}, \vec{w}^{(t)} \rangle \):

Perceptron learning algorithm: convergence proof (cont.)

\[ \|\vec{w}^{(t+1)}\|^2 = \langle \vec{w}^{(t+1)}, \vec{w}^{(t+1)} \rangle \]

\[ = \langle \vec{w}^{(t)} \pm \vec{x}^{(t)}, \vec{w}^{(t)} \pm \vec{x}^{(t)} \rangle \]

\[ = \|\vec{w}^{(t)}\|^2 \pm 2 \langle \vec{x}^{(t)}, \vec{w}^{(t)} \rangle + \|\vec{x}^{(t)}\|^2 \]

consider \( \langle \vec{x}^{(t)}, \vec{w}^{(t)} \rangle \):

if we go from \( t \) to \( t+1 \), then \( x(t) \) was not correctly classified. Hence,
\[ \||\vec{w}^{(t+1)}||^2 = \langle \vec{w}^{(t+1)}, \vec{w}^{(t+1)} \rangle \]
\[ = \langle \vec{w}^{(t)} \pm \vec{x}^{(t)}, \vec{w}^{(t)} \pm \vec{x}^{(t)} \rangle \]
\[ = ||\vec{w}^{(t)}||^2 \pm 2 \langle \vec{x}^{(t)}, \vec{w}^{(t)} \rangle + ||\vec{x}^{(t)}||^2 \]

consider \( \langle \vec{x}^{(t)}, \vec{w}^{(t)} \rangle \):

if we go from \( t \) to \( t+1 \), then \( x^{(t)} \) was not correctly classified. Hence, \( x^{(t)} \) not correctly classified, then if \( \vec{x}^{(t)} \in \mathcal{P} : \langle \vec{w}^{(t)}, \vec{x}^{(t)} \rangle < 0 \), if \( \vec{x}^{(t)} \in \mathcal{N} : \langle \vec{w}^{(t)}, \vec{x}^{(t)} \rangle \geq 0 \). Therefore: \( \pm \langle \vec{w}^{(t)}, \vec{x}^{(t)} \rangle \leq 0 \). Dropping it makes expression larger.
Perceptron learning algorithm: convergence proof (cont.)

\[ \|\vec{w}^{(t+1)}\|^2 = \langle \vec{w}^{(t+1)}, \vec{w}^{(t+1)} \rangle \]
\[ = \langle \vec{w}^{(t)} \pm \vec{x}^{(t)}, \vec{w}^{(t)} \pm \vec{x}^{(t)} \rangle \]
\[ = \|\vec{w}^{(t)}\|^2 \pm 2 \langle \vec{w}^{(t)}, \vec{x}^{(t)} \rangle + \|\vec{x}^{(t)}\|^2 \]
\[ \leq \|\vec{w}^{(t)}\|^2 + \varepsilon \]

with \( \varepsilon := \max\{\|\vec{x}\|^2 | \vec{x} \in \mathcal{P} \cup \mathcal{N} \} \)
\[
\| \mathbf{w}^{(t+1)} \|^2 = \langle \mathbf{w}^{(t+1)}, \mathbf{w}^{(t+1)} \rangle \\
= \langle \mathbf{w}^{(t)} \pm \mathbf{x}^{(t)}, \mathbf{w}^{(t)} \pm \mathbf{x}^{(t)} \rangle \\
= \| \mathbf{w}^{(t)} \|^2 \pm 2 \langle \mathbf{w}^{(t)}, \mathbf{x}^{(t)} \rangle + \| \mathbf{x}^{(t)} \|^2 \\
\leq \| \mathbf{w}^{(t)} \|^2 + \varepsilon
\]

with \( \varepsilon := \max\{\| \mathbf{x} \|^2 | \mathbf{x} \in \mathcal{P} \cup \mathcal{N} \} \)

Hence,
\[
\| \mathbf{w}^{(t+1)} \|^2 \leq \| \mathbf{w}^{(0)} \|^2 + (t + 1)\varepsilon
\]
Perceptron learning algorithm: convergence proof (cont.)

$$\cos \angle (\vec{w}^*, \vec{w}^{(t+1)}) = \frac{\langle \vec{w}^*, \vec{w}^{(t+1)} \rangle}{||\vec{w}^*|| \cdot ||\vec{w}^{(t+1)}||}$$
\[
\cos \angle(\vec{w}^*, \vec{w}^{(t+1)}) = \frac{\langle \vec{w}^*, \vec{w}^{(t+1)} \rangle}{||\vec{w}^*|| \cdot ||\vec{w}^{(t+1)}||} \\
\geq \frac{\langle \vec{w}^*, \vec{w}^{(0)} \rangle + (t + 1)\delta}{||\vec{w}^*|| \cdot \sqrt{||\vec{w}^{(0)}||^2 + (t + 1)\varepsilon}}
\]
Perceptron learning algorithm:
convergence proof (cont.)

\[
\cos \angle (\vec{w}^*, \vec{w}^{(t+1)}) = \frac{\langle \vec{w}^*, \vec{w}^{(t+1)} \rangle}{\|\vec{w}^*\| \cdot \|\vec{w}^{(t+1)}\|}
\]

\[
\geq \frac{\langle \vec{w}^*, \vec{w}^{(0)} \rangle + (t + 1)\delta}{\|\vec{w}^*\| \cdot \sqrt{\|\vec{w}^{(0)}\|^2 + (t + 1)\varepsilon}} \quad \longrightarrow \quad \infty
\]

Since \( \cos \angle (\vec{w}^*, \vec{w}^{(t+1)}) \leq 1 \), \( t \) must be bounded above. \[\blacksquare\]
**Lemma (worst case running time):**

If the given problem is solvable, perceptron learning terminates after at most \((n + 1)^22^{(n+1)\log(n+1)}\) iterations.

Exponential running time is a problem of the perceptron learning algorithm. There are algorithms that solve the problem with complexity \(O(n^{7/2})\).
Lemma:
If a weight vector occurs twice during perceptron learning, the given task is not solvable. (Remark: here, we mean with weight vector the extended variant containing also $w_0$)

Proof: next slide
Lemma:
If a weight vector occurs twice during perceptron learning, the given task is not solvable. (Remark: here, we mean with weight vector the extended variant containing also $w_0$)

Proof: next slide

Lemma:
Starting the perceptron learning algorithm with weight vector $\vec{0}$ on an unsolvable problem, at least one weight vector will occur twice.

Proof: omitted, see Minsky/Papert, *Perceptrons*
Proof:
Assume \( \vec{w}^{(t+k)} = \vec{w}^{(t)} \). Meanwhile, the patterns \( \vec{x}^{(t+1)}, \ldots, \vec{x}^{(t+k)} \) have been applied. Without loss of generality, assume \( \vec{x}^{(t+1)}, \ldots, \vec{x}^{(t+q)} \in \mathcal{P} \) and \( \vec{x}^{(t+q+1)}, \ldots, \vec{x}^{(t+k)} \in \mathcal{N} \). Hence:

\[
\vec{w}^{(t)} = \vec{w}^{(t+k)} = \vec{w}^{(t)} + \vec{x}^{(t+1)} + \cdots + \vec{x}^{(t+q)} - (\vec{x}^{(t+q+1)} + \cdots + \vec{x}^{(t+k)})
\]

\[
\Rightarrow \quad \vec{x}^{(t+1)} + \cdots + \vec{x}^{(t+q)} = \vec{x}^{(t+q+1)} + \cdots + \vec{x}^{(t+k)}
\]
Proof:
Assume $\vec{w}^{(t+k)} = \vec{w}^{(t)}$. Meanwhile, the patterns $\vec{x}^{(t+1)}, \ldots, \vec{x}^{(t+k)}$ have been applied. Without loss of generality, assume $\vec{x}^{(t+1)}, \ldots, \vec{x}^{(t+q)} \in \mathcal{P}$ and $\vec{x}^{(t+q+1)}, \ldots, \vec{x}^{(t+k)} \in \mathcal{N}$. Hence:

$$\vec{w}^{(t)} = \vec{w}^{(t+k)} = \vec{w}^{(t)} + \vec{x}^{(t+1)} + \cdots + \vec{x}^{(t+q)} - \left( \vec{x}^{(t+q+1)} + \cdots + \vec{x}^{(t+k)} \right)$$

$$\Rightarrow \quad \vec{x}^{(t+1)} + \cdots + \vec{x}^{(t+q)} = \vec{x}^{(t+q+1)} + \cdots + \vec{x}^{(t+k)}$$

Assume, a solution $\vec{w}^*$ exists. Then:

$$\langle \vec{w}^*, \vec{x}^{(t+i)} \rangle \begin{cases} 
\geq 0 & \text{if } i \in \{1, \ldots, q\} \\
< 0 & \text{if } i \in \{q + 1, \ldots, k\}
\end{cases}$$
**Perceptron learning algorithm: cycle theorem**

**Proof:**
Assume $\mathbf{w}^{(t+k)} = \mathbf{w}^{(t)}$. Meanwhile, the patterns $\mathbf{x}^{(t+1)}, \ldots, \mathbf{x}^{(t+k)}$ have been applied. Without loss of generality, assume $\mathbf{x}^{(t+1)}, \ldots, \mathbf{x}^{(t+q)} \in \mathcal{P}$ and $\mathbf{x}^{(t+q+1)}, \ldots, \mathbf{x}^{(t+k)} \in \mathcal{N}$. Hence:

$$
\mathbf{w}^{(t)} = \mathbf{w}^{(t+k)} = \mathbf{w}^{(t)} + \mathbf{x}^{(t+1)} + \cdots + \mathbf{x}^{(t+q)} - (\mathbf{x}^{(t+q+1)} + \cdots + \mathbf{x}^{(t+k)})
$$

$$
\Rightarrow \quad \mathbf{x}^{(t+1)} + \cdots + \mathbf{x}^{(t+q)} = \mathbf{x}^{(t+q+1)} + \cdots + \mathbf{x}^{(t+k)}
$$

Assume, a solution $\mathbf{w}^*$ exists. Then:

$$
\langle \mathbf{w}^*, \mathbf{x}^{(t+i)} \rangle \begin{cases} 
\geq 0 & \text{if } i \in \{1, \ldots, q\} \\
< 0 & \text{if } i \in \{q + 1, \ldots, k\}
\end{cases}
$$

Hence,

$$
\langle \mathbf{w}^*, \mathbf{x}^{(t+1)} + \cdots + \mathbf{x}^{(t+q)} \rangle \geq 0
$$

$$
\langle \mathbf{w}^*, \mathbf{x}^{(t+q+1)} + \cdots + \mathbf{x}^{(t+k)} \rangle < 0 \quad \text{contradiction!}
$$
how can we determine a “good” perceptron if the given task cannot be solved perfectly?

“good” in the sense of: perceptron makes minimal number of errors
how can we determine a “good” perceptron if the given task cannot be solved perfectly?

“good” in the sense of: perceptron makes minimal number of errors
Perceptron learning algorithm: Pocket algorithm

- how can we determine a “good” perceptron if the given task cannot be solved perfectly?
- “good” in the sense of: perceptron makes minimal number of errors
- Perceptron learning: the number of errors does not decrease monotonically during learning
- Idea: memorise the best weight vector that has occurred so far!

⇒ Pocket algorithm
perceptrons can only learn linearly separable problems.

famous counterexample:
\[
XOR(x_1, x_2):
\]
\[
P = \{(0, 1)^T, (1, 0)^T\},
\]
\[
N = \{(0, 0)^T, (1, 1)^T\}
\]
perceptrons can only learn linearly separable problems.

famous counterexample: \( \text{XOR}(x_1, x_2) \):
\[
\mathcal{P} = \{(0, 1)^T, (1, 0)^T\},
\]
\[
\mathcal{N} = \{(0, 0)^T, (1, 1)^T\}
\]

networks with several perceptrons are computationally more powerful (cf. McCullough/Pitts neurons)

let's try to find a network with two perceptrons that can solve the XOR problem:
  - first step: find a perceptron that classifies three patterns accurately, e.g. \( w_0 = -0.5, w_1 = w_2 = 1 \) classifies \((0, 0)^T, (0, 1)^T, (1, 0)^T\) but fails on \((1, 1)^T\)
perceptrons can only learn linearly separable problems.

famous counterexample: $XOR(x_1, x_2)$:
\[\mathcal{P} = \{(0, 1)^T, (1, 0)^T\},\]
\[\mathcal{N} = \{(0, 0)^T, (1, 1)^T\}\]

networks with several perceptrons are computationally more powerful (cf. McCullough/Pitts neurons)

let’s try to find a network with two perceptrons that can solve the XOR problem:
  • first step: find a perceptron that classifies three patterns accurately, e.g. $w_0 = -0.5$, $w_1 = w_2 = 1$ classifies $(0, 0)^T, (0, 1)^T, (1, 0)^T$ but fails on $(1, 1)^T$
  • second step: find a perceptron that uses the output of the first perceptron as additional input.

Hence, training patterns are:
\[\mathcal{N'} = \{(0, 0, 0), (1, 1, 1)\},\]
\[\mathcal{P'} = \{(0, 1, 1), (1, 0, 1)\}.
perceptron learning yields:
\[v_0 = -1, v_1 = v_2 = -1, v_3 = 2\]
XOR-network:
Perceptron networks
(cont.)

XOR-network:

Geometric interpretation:
Perceptron networks (cont.)

XOR-network:

\[ y = \sum x_i \gamma_i + \theta \]

Geometric interpretation: partitioning of first perceptron
XOR-network:

Geometric interpretation:

partitioning of second perceptron, assuming first perceptron yields 0
XOR-network:

\[ \begin{align*}
\sum & \Gamma \\
1 & \rightarrow y \\
1 & \rightarrow y \\
1 & \rightarrow -0.5 \\
1 & \rightarrow -1 \\
-1 & \rightarrow \Sigma \\
-1 & \rightarrow \Sigma \\
\end{align*} \]

Geometric interpretation:

partitioning of second perceptron, assuming first perceptron yields 1
XOR-network:

Geometric interpretation:

combining both
Rosenblatt perceptron (1958):

- retinal input (array of pixels)
- preprocessing level, calculation of features
- adaptive linear classifier
- inspired by human vision
Rosenblatt perceptron (1958):
- retinal input (array of pixels)
- preprocessing level, calculation of features
- adaptive linear classifier
- inspired by human vision

- if features are complex enough, everything can be classified
- if features are restricted (only parts of the retinal pixels available to features), some interesting tasks cannot be learned (Minsky/Papert, 1969)
Historical remarks

- Rosenblatt perceptron (1958):
  - retinal input (array of pixels)
  - preprocessing level, calculation of features
  - adaptive linear classifier
  - inspired by human vision

- if features are complex enough, everything can be classified
- if features are restricted (only parts of the retinal pixels available to features), some interesting tasks cannot be learned (Minsky/Papert, 1969)

- important idea: create features instead of learning from raw data
Perceptrons are simple neurons with limited representation capabilities: linear separable functions only.

Simple but provably working learning algorithm.

Networks of perceptrons can overcome limitations.

Working in feature space may help to overcome limited representation capability.