Machine Learning:
Perceptrons

Prof. Dr. Martin Riedmiller

Albert-Ludwigs-University Freiburg

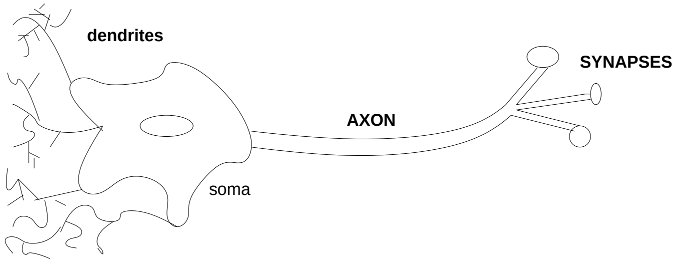
AG Maschinelles Lernen

Neural Networks

- The human brain has approximately 10^{11} neurons
- Switching time $0.001s$ (computer $\approx 10^{-10}s$)
- Connections per neuron: $10^4 - 10^5$
- $0.1s$ 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
- computational learning theory
- support vector machines
- Boosting

1950 1960 1970 1980 1990 2000

Perceptrons: adaptive neurons

- 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: adaptive neurons

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

$$(x_1, \dots, x_n)^T \mapsto y = \begin{cases} 1 & \text{if } x_1 w_1 + x_2 w_2 + \dots + x_n w_n \geq \theta \\ 0 & \text{if } x_1 w_1 + x_2 w_2 + \dots + x_n w_n < \theta \end{cases}$$

Perceptrons: adaptive neurons (cont.)

- 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:

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

with

$$f_{step}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

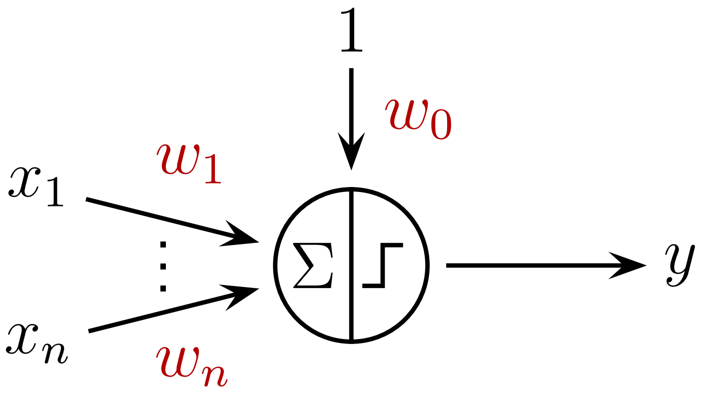
Perceptrons: adaptive neurons (cont.)

- 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:

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

with

$$f_{step}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$



Perceptrons: adaptive neurons (cont.)

geometric interpretation of a perceptron:

- input patterns $(x_1, \ldots, x_n)$ are points in n-dimensional space

Perceptrons: adaptive neurons (cont.)

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 $\vec{w}$

Perceptrons: adaptive neurons (cont.)

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

Perceptrons: adaptive neurons (cont.)

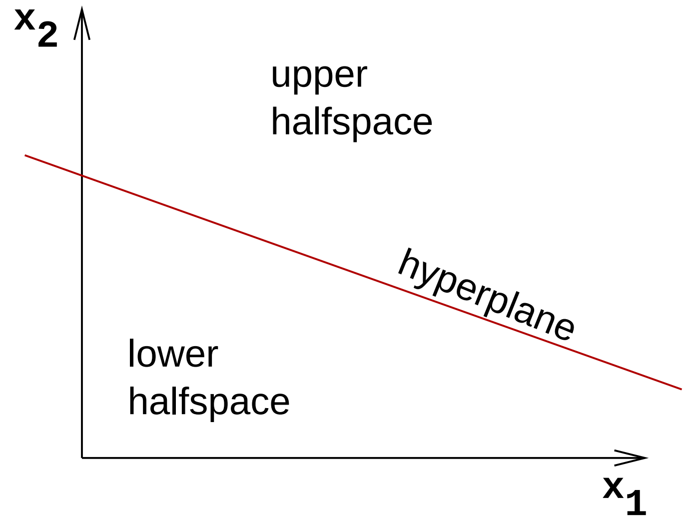
geometric interpretation of a perceptron:

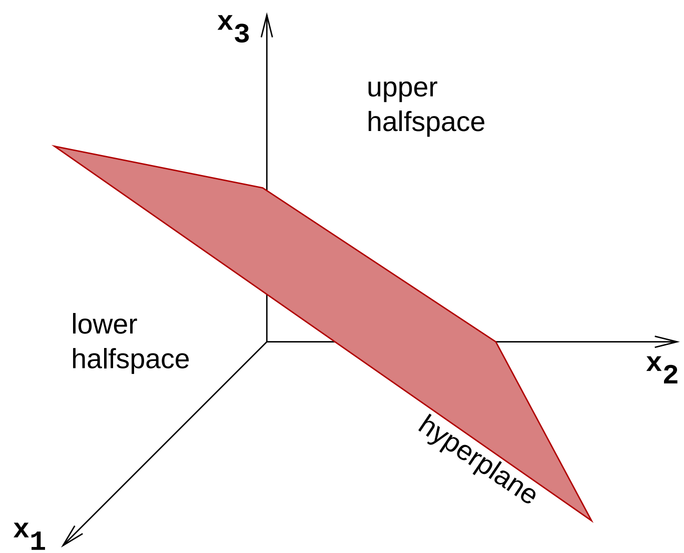
- input patterns $(x_1, \ldots, x_n)$ are points in n-dimensional space
- points with $w_0 + \langle \vec{w}, \vec{x} \rangle = 0$ are on a hyperplane defined by w_0 and $\vec{w}$
- points with $w_0 + \langle \vec{w}, \vec{x} \rangle > 0$ are above the hyperplane
- points with $w_0 + \langle \vec{w},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

Perceptrons: adaptive neurons (cont.)

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 $\Rightarrow$ Supervised Learning: Classification

Perceptron learning problem

- perceptrons can automatically adapt to example data $\Rightarrow$ 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 $\Rightarrow$ 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. $\mathcal{P} \cap \mathcal{N} \neq \emptyset$

Perceptron learning problem (cont.)

- 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.

Perceptron learning problem (cont.)

- 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,

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

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

Perceptron learning problem (cont.)

- 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:

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

  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

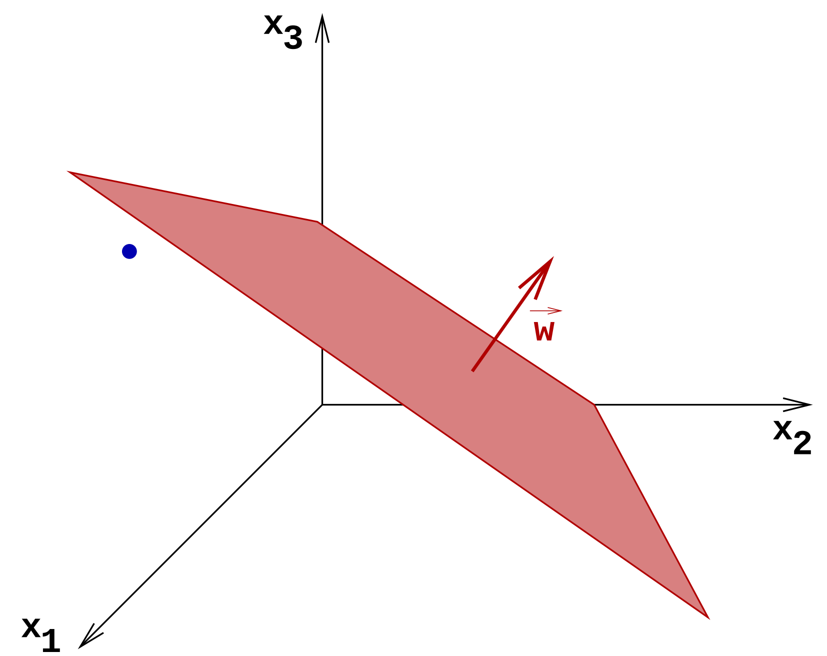
- assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$: $\langle \vec{w}, \vec{x} \rangle + w_0 < 0$
- how can we change $\vec{w}$ and w_0 to avoid this error? – we need to increase $\langle \vec{w}, \vec{x} \rangle + w_0$
  - increase w_0
  - if $x_i > 0$, increase w_i
  - if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

Perceptron learning algorithm: idea

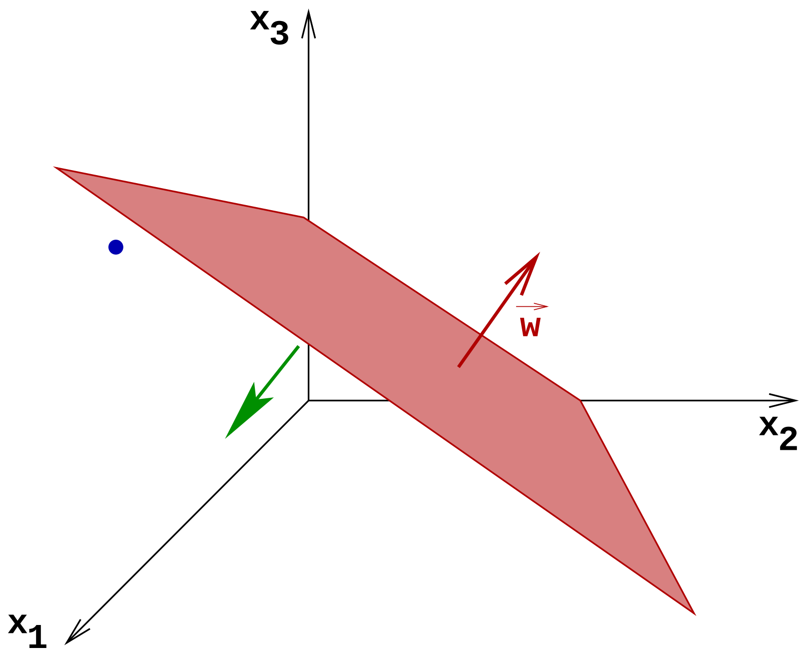
- assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$: $\langle \vec{w}, \vec{x} \rangle + w_0 < 0$
- how can we change $\vec{w}$ and w_0 to avoid this error? – we need to increase $\langle \vec{w}, \vec{x} \rangle + w_0$
  - increase w_0
  - if $x_i > 0$, increase w_i
  - if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

Perceptron learning algorithm: idea

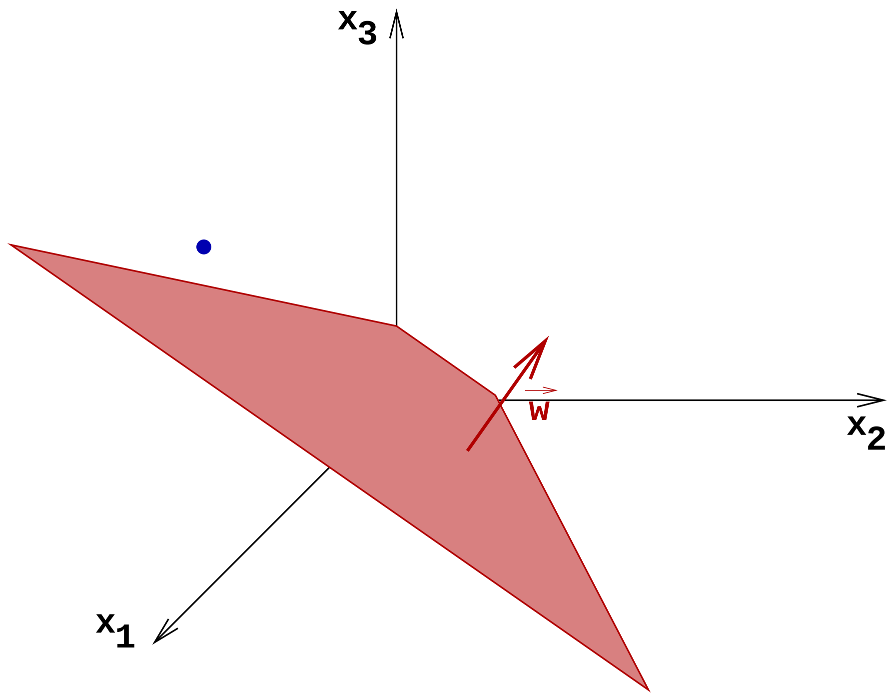
- assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$: $\langle \vec{w}, \vec{x} \rangle + w_0 < 0$
- how can we change $\vec{w}$ and w_0 to avoid this error? – we need to increase $\langle \vec{w}, \vec{x} \rangle + w_0$
  - increase w_0
  - if $x_i > 0$, increase w_i
  - if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



Geometric intepretation: 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.



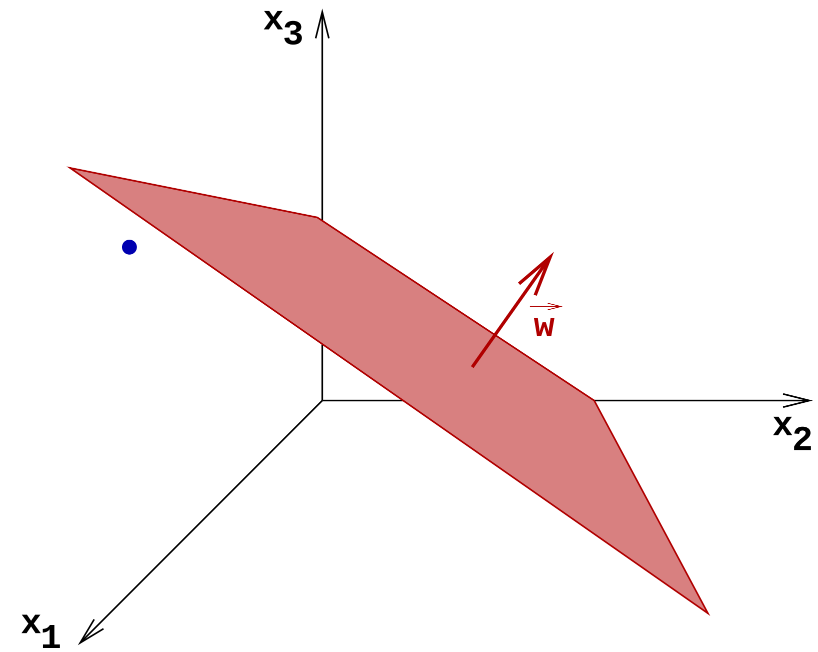
Geometric intepretation: increasing w_0

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

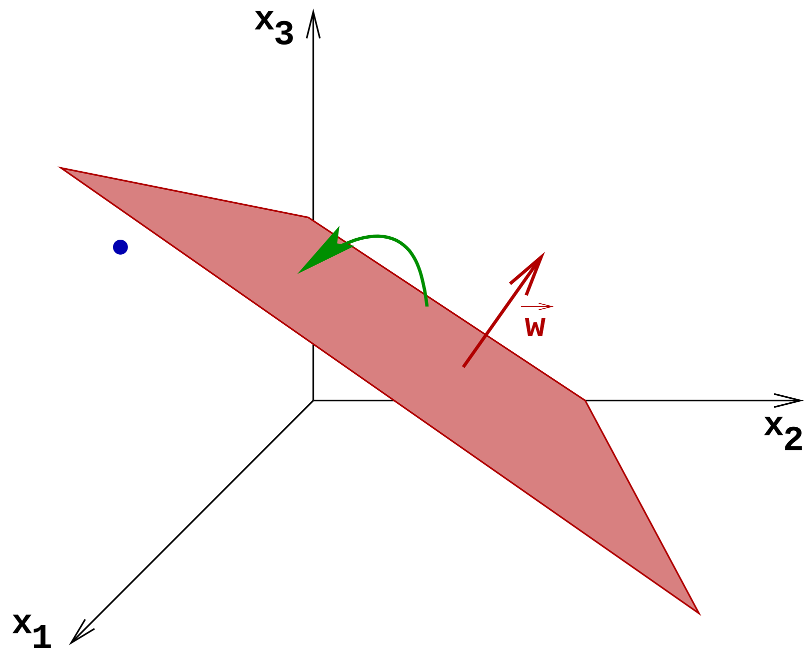
- assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$: $\langle \vec{w}, \vec{x} \rangle + w_0 < 0$
- how can we change $\vec{w}$ and w_0 to avoid this error? – we need to increase $\langle \vec{w}, \vec{x} \rangle + w_0$
  - increase w_0
  - if $x_i > 0$, increase w_i
  - if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



Geometric intepretation: modifying $\vec{w}$

Perceptron learning algorithm: idea

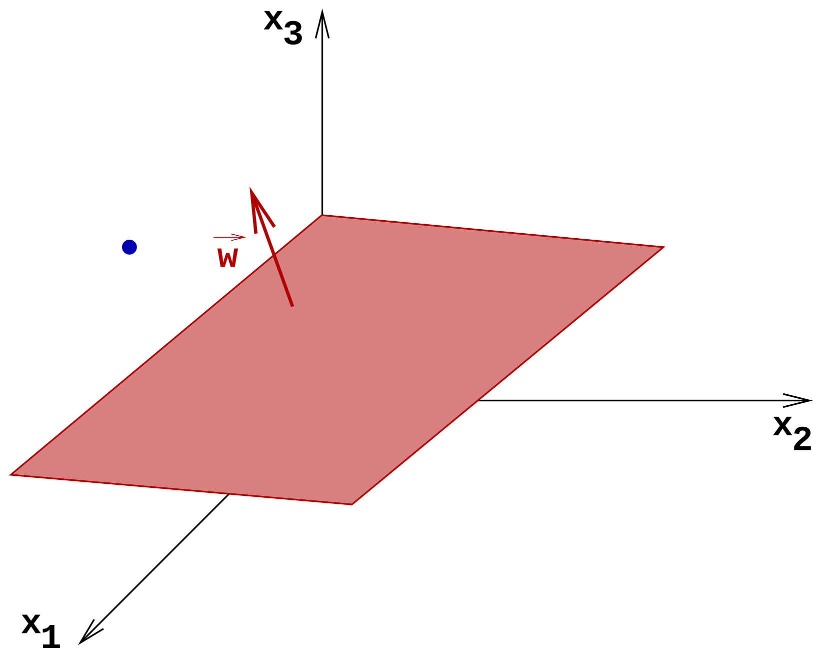
- assume, the perceptron makes an error on a pattern $\vec{x} \in \mathcal{P}$: $\langle \vec{w}, \vec{x} \rangle + w_0 < 0$
- how can we change $\vec{w}$ and w_0 to avoid this error? – we need to increase $\langle \vec{w}, \vec{x} \rangle + w_0$
  - increase w_0
  - if $x_i > 0$, increase w_i
  - if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

initialize weight vector $\vec{w}$ and bias weight w_0 arbitrarily
while exist misclassified pattern $\vec{x} \in \mathcal{P} \cup \mathcal{N}$ **do**
    if $\vec{x} \in \mathcal{P}$ **then**
        $\vec{w} \leftarrow \vec{w} + \vec{x}$
        $w_0 \leftarrow w_0 + 1$
    else
        $\vec{w} \leftarrow \vec{w} - \vec{x}$
        $w_0 \leftarrow w_0 - 1$
    end if
end while
return $\vec{w}$ and w_0

Perceptron learning algorithm: example

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

$\rightarrow$ 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.

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.

  Proof: follows immediately from algorithm.

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.

  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.

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.

  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. $\vec{w} = (w_0, w_1, \dots, w_n)^T$, and 1 to all patterns, i.e. $\vec{x} = (1, x_1, \dots, x_n)^T$.
  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.

Perceptron learning algorithm: Preliminaries

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

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

Perceptron learning algorithm: Preliminaries

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

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

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

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

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:
$\cos \measuredangle(\vec{x}, \vec{y}) = \frac{\langle \vec{x}, \vec{y} \rangle}{||\vec{x}|| \cdot ||\vec{y}||}$

Perceptron learning algorithm: convergence proof (cont.)

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

Perceptron learning algorithm: convergence proof (cont.)

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

$$\begin{aligned}\left\langle \vec{w}^*, \vec{w}^{(t+1)} \right\rangle &= \left\langle \vec{w}^*, \vec{w}^{(t)} \pm \vec{x}^{(t)} \right\rangle \\ &= \left\langle \vec{w}^*, \vec{w}^{(t)} \right\rangle \pm \left\langle \vec{w}^*, \vec{x}^{(t)} \right\rangle \\ &\geq \left\langle \vec{w}^*, \vec{w}^{(t)} \right\rangle + \delta\end{aligned}$$

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

Perceptron learning algorithm: convergence proof (cont.)

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

$$\begin{aligned}\left\langle \vec{w}^*, \vec{w}^{(t+1)} \right\rangle &= \left\langle \vec{w}^*, \vec{w}^{(t)} \pm \vec{x}^{(t)} \right\rangle \\ &= \left\langle \vec{w}^*, \vec{w}^{(t)} \right\rangle \pm \left\langle \vec{w}^*, \vec{x}^{(t)} \right\rangle \\ &\geq \left\langle \vec{w}^*, \vec{w}^{(t)} \right\rangle + \delta\end{aligned}$$

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

Perceptron learning algorithm: convergence proof (cont.)

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

$$\begin{aligned}\left\langle\vec{w}^*,\vec{w}^{(t+1)}\right\rangle &= \left\langle\vec{w}^*,\vec{w}^{(t)}\pm\vec{x}^{(t)}\right\rangle \\ &= \left\langle\vec{w}^*,\vec{w}^{(t)}\right\rangle \pm \left\langle\vec{w}^*,\vec{x}^{(t)}\right\rangle \\ &\geq \left\langle\vec{w}^*,\vec{w}^{(t)}\right\rangle + \delta\end{aligned}$$

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
Hence,

$$\left\langle\vec{w}^*,\vec{w}^{(t+1)}\right\rangle \geq \left\langle\vec{w}^*,\vec{w}^{(0)}\right\rangle + (t+1)\delta$$

Perceptron learning algorithm: convergence proof (cont.)

$$
\begin{aligned}
||\vec{w}^{(t+1)}||^2 &= \left\langle \vec{w}^{(t+1)}, \vec{w}^{(t+1)} \right\rangle \\
&= \left\langle \vec{w}^{(t)} \pm \vec{x}^{(t)}, \vec{w}^{(t)} \pm \vec{x}^{(t)} \right\rangle \\
&= ||\vec{w}^{(t)}||^2 \pm 2 \left\langle \vec{x}^{(t)}, \vec{w}^{(t)} \right\rangle + ||\vec{x}^{(t)}||^2
\end{aligned}
$$

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

Perceptron learning algorithm: convergence proof (cont.)

$$
\begin{aligned}
||\vec{w}^{(t+1)}||^2 &= \left\langle \vec{w}^{(t+1)}, \vec{w}^{(t+1)} \right\rangle \\
&= \left\langle \vec{w}^{(t)} \pm \vec{x}^{(t)}, \vec{w}^{(t)} \pm \vec{x}^{(t)} \right\rangle \\
&= ||\vec{w}^{(t)}||^2 \pm 2 \left\langle \vec{x}^{(t)}, \vec{w}^{(t)} \right\rangle + ||\vec{x}^{(t)}||^2
\end{aligned}
$$

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

if we go from t to t+1, then $x(t)$ was not correctly classified. Hence,

Perceptron learning algorithm: convergence proof (cont.)

$$\begin{aligned}||\vec{w}^{(t+1)}||^2 &= \left\langle \vec{w}^{(t+1)}, \vec{w}^{(t+1)} \right\rangle \\ &= \left\langle \vec{w}^{(t)} \pm \vec{x}^{(t)}, \vec{w}^{(t)} \pm \vec{x}^{(t)} \right\rangle \\ &= ||\vec{w}^{(t)}||^2 \pm 2 \left\langle \vec{x}^{(t)}, \vec{w}^{(t)} \right\rangle + ||\vec{x}^{(t)}||^2\end{aligned}$$

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

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

Perceptron learning algorithm: convergence proof (cont.)

$$\begin{aligned}
||\vec{w}^{(t+1)}||^2 &= \left\langle \vec{w}^{(t+1)}, \vec{w}^{(t+1)} \right\rangle \\
&= \left\langle \vec{w}^{(t)} \pm \vec{x}^{(t)}, \vec{w}^{(t)} \pm \vec{x}^{(t)} \right\rangle \\
&= ||\vec{w}^{(t)}||^2 \pm 2 \left\langle \vec{w}^{(t)}, \vec{x}^{(t)} \right\rangle + ||\vec{x}^{(t)}||^2 \\
&\leq ||\vec{w}^{(t)}||^2 + \varepsilon
\end{aligned}$$

with $\varepsilon := \max\{||\vec{x}||^2 | \vec{x} \in \mathcal{P} \cup \mathcal{N}\}$

Perceptron learning algorithm: convergence proof (cont.)

$$
\begin{aligned}
||\vec{w}^{(t+1)}||^2 &= \left\langle \vec{w}^{(t+1)}, \vec{w}^{(t+1)} \right\rangle \\
&= \left\langle \vec{w}^{(t)} \pm \vec{x}^{(t)}, \vec{w}^{(t)} \pm \vec{x}^{(t)} \right\rangle \\
&= ||\vec{w}^{(t)}||^2 \pm 2 \left\langle \vec{w}^{(t)}, \vec{x}^{(t)} \right\rangle + ||\vec{x}^{(t)}||^2 \\
&\leq ||\vec{w}^{(t)}||^2 + \varepsilon
\end{aligned}
$$

with $\varepsilon := \max\{||\vec{x}||^2 | \vec{x} \in \mathcal{P} \cup \mathcal{N}\}$

Hence,

$$||\vec{w}^{(t+1)}||^2 \leq ||\vec{w}^{(0)}||^2 + (t+1)\varepsilon$$

Perceptron learning algorithm: convergence proof (cont.)

$$\cos \measuredangle(\vec{w}^*, \vec{w}^{(t+1)}) = \frac{\left\langle \vec{w}^*, \vec{w}^{(t+1)} \right\rangle}{||\vec{w}^*|| \ \cdot \ ||\vec{w}^{(t+1)}||}$$

Perceptron learning algorithm: convergence proof (cont.)

$$\cos \measuredangle(\vec{w}^*, \vec{w}^{(t+1)}) = \frac{\left\langle \vec{w}^*, \vec{w}^{(t+1)} \right\rangle}{||\vec{w}^*|| \cdot ||\vec{w}^{(t+1)}||}$$

$$\geq \frac{\left\langle \vec{w}^*, \vec{w}^{(0)} \right\rangle + (t+1)\delta}{||\vec{w}^*|| \cdot \sqrt{||\vec{w}^{(0)}||^2 + (t+1)\varepsilon}}$$

Perceptron learning algorithm: convergence proof (cont.)

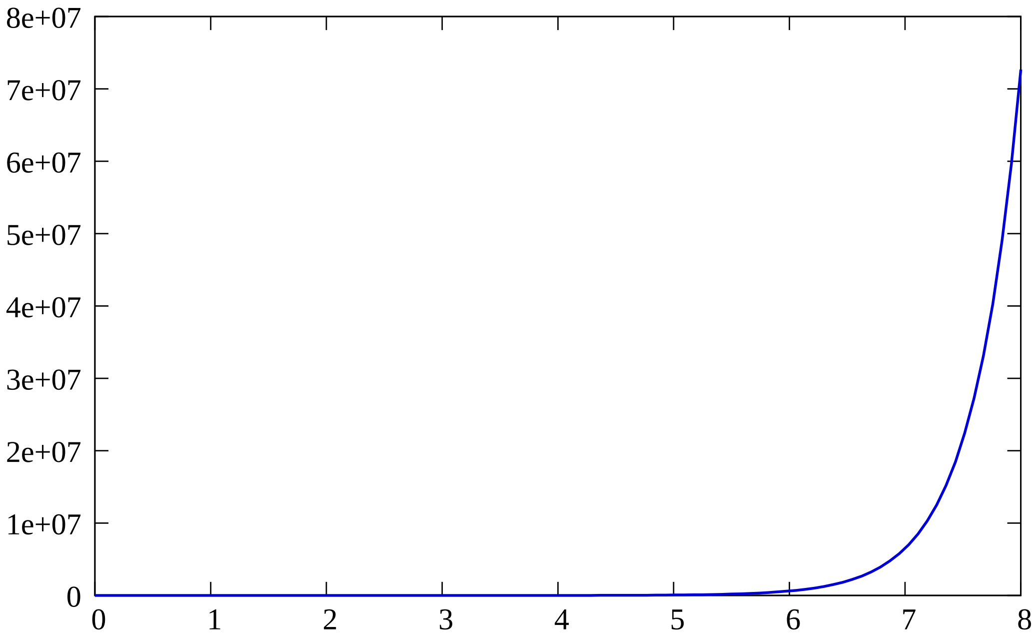
$$\cos\measuredangle(\vec{w}^*, \vec{w}^{(t+1)}) = \frac{\left\langle \vec{w}^*, \vec{w}^{(t+1)} \right\rangle}{||\vec{w}^*|| \cdot ||\vec{w}^{(t+1)}||}$$

$$\geq \frac{\left\langle \vec{w}^*, \vec{w}^{(0)} \right\rangle + (t+1)\delta}{||\vec{w}^*|| \cdot \sqrt{||\vec{w}^{(0)}||^2 + (t+1)\varepsilon}} \quad \underset{t\to\infty}{\longrightarrow} \quad \infty$$

Since $\cos\measuredangle(\vec{w}^*, \vec{w}^{(t+1)}) \leq 1$, t must be bounded above. ■

Perceptron learning algorithm: convergence

- Lemma (worst case running time):
  If the given problem is solvable, perceptron learning terminates after at most $(n+1)^2 2^{(n+1)\log(n+1)}$ iterations.



- Exponential running time is a problem of the perceptron learning algorithm. There are algorithms that solve the problem with complexity $O(n^{\frac{7}{2}})$

Perceptron learning algorithm: cycle theorem

- 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

Perceptron learning algorithm: cycle theorem

- 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*

Perceptron learning algorithm: cycle theorem

Proof:

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

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

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

Perceptron learning algorithm: cycle theorem

Proof:

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

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

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

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

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

Perceptron learning algorithm: cycle theorem

Proof:

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

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

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

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

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

Hence,

$$\left\langle \vec{w}^*, \vec{x}^{(t+1)} + \dots + \vec{x}^{(t+q)} \right\rangle \geq 0$$

$$\left\langle \vec{w}^*, \vec{x}^{(t+q+1)} + \dots + \vec{x}^{(t+k)} \right\rangle < 0 \qquad \text{contradiction!}$$

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 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 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 occured so far!
  $\Rightarrow$ Pocket algorithm

Perceptron networks

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

Perceptron networks

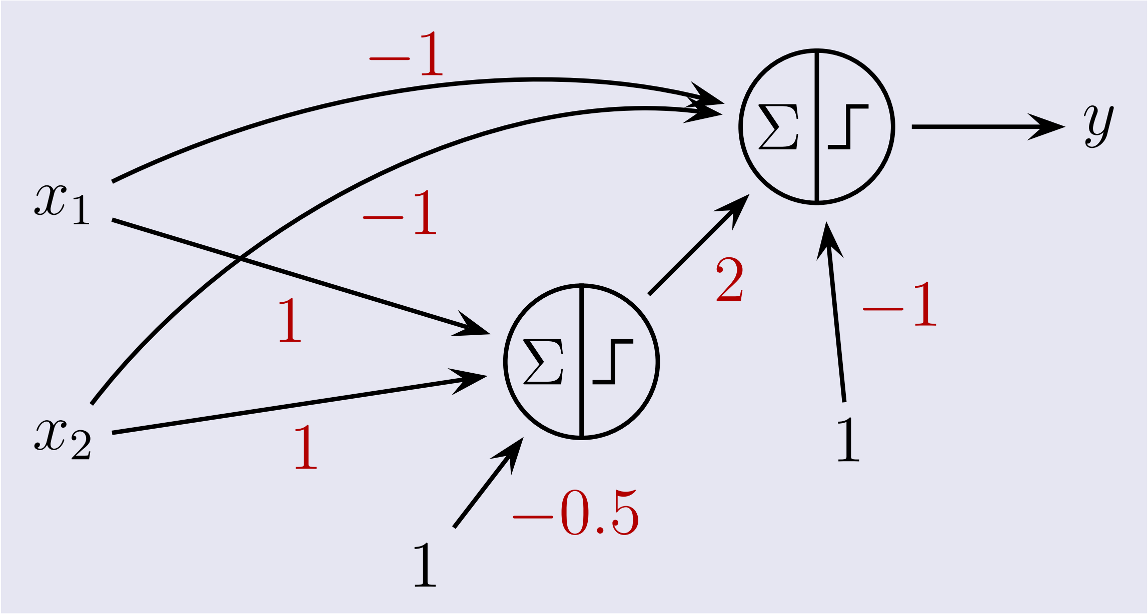
- perceptrons can only learn linearly separable problems.
- famous counterexample:
  $XOR(x_1, x_2)$:
  $\mathcal{P} = \{(0,1)^T, (1,0)^T\}$,
  $\mathcal{N} = \{(0,0)^T, (1,1)^T\}$
- networks with several perceptrons are computationally more powerful (cf. McCullough/Pitts neurons)
- let's try to find 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 $(1,1)^T$

Perceptron networks

- 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: $v_0 = -1, v_1 = v_2 = -1$, $v_3 = 2$

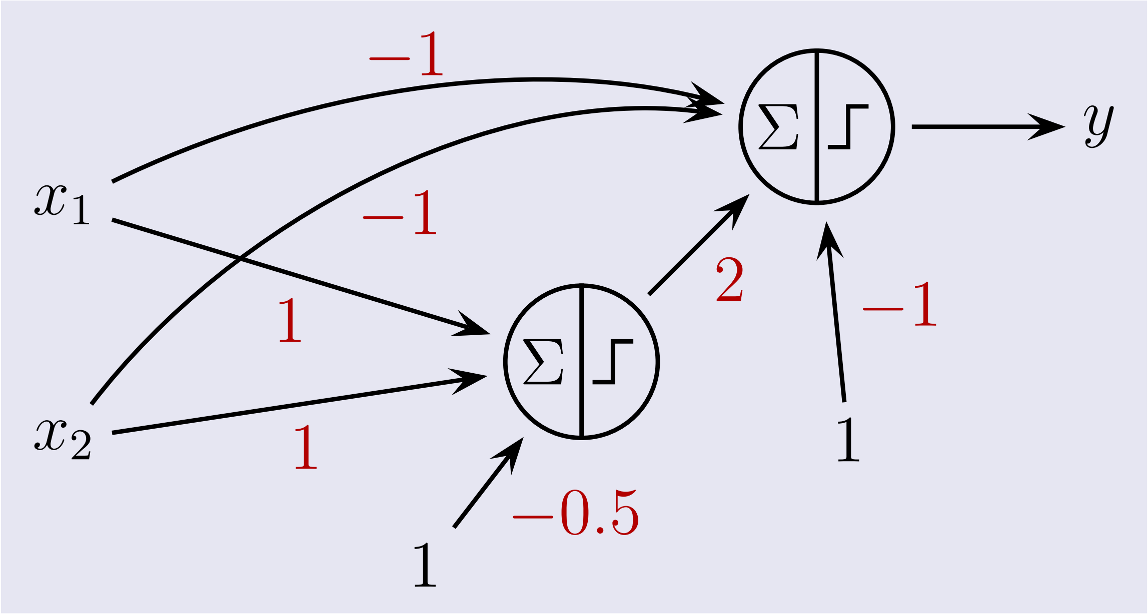
Perceptron networks (cont.)

XOR-network:

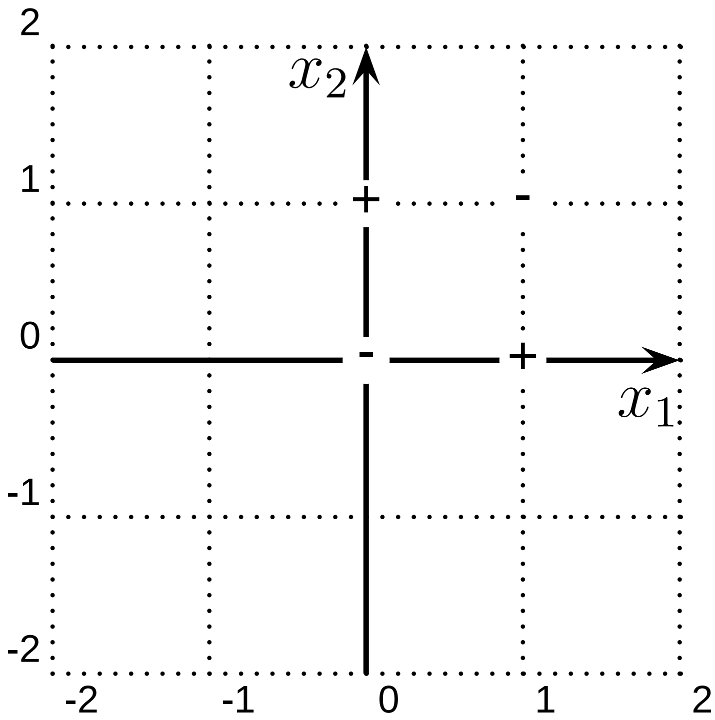


Perceptron networks (cont.)

XOR-network:

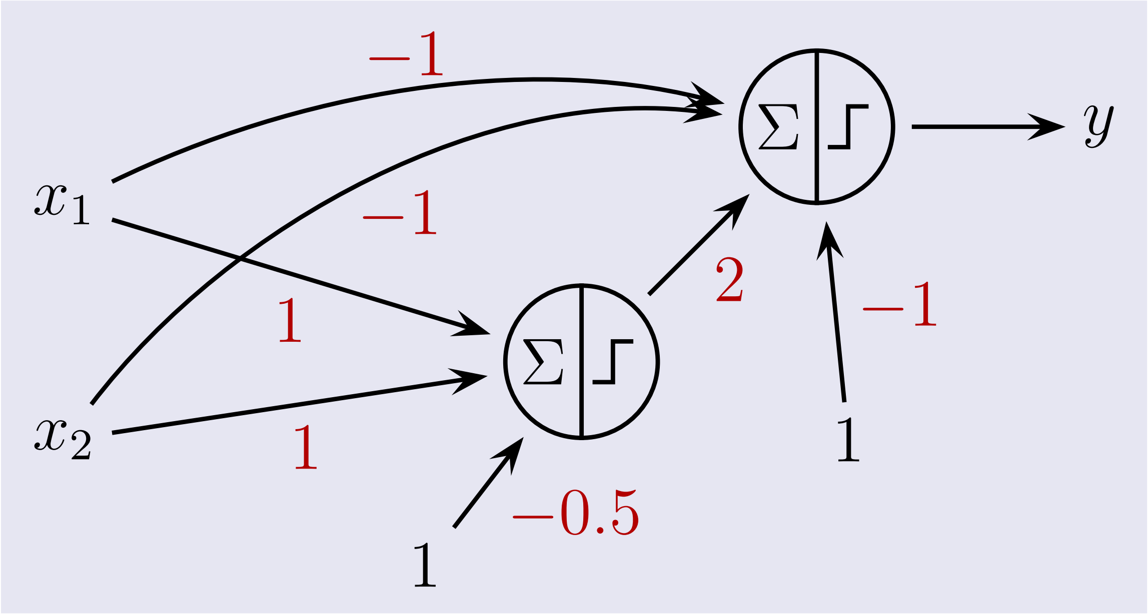


Geometric interpretation:



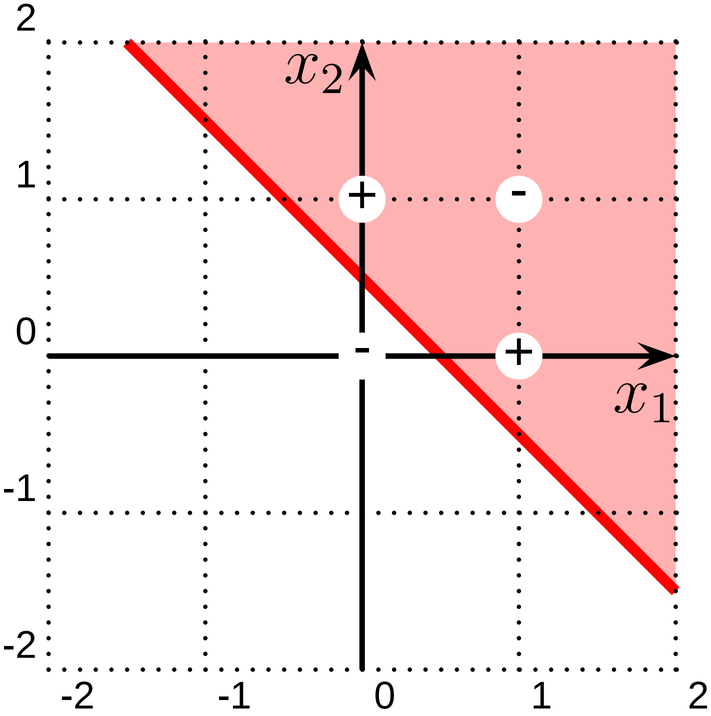
Perceptron networks (cont.)

XOR-network:



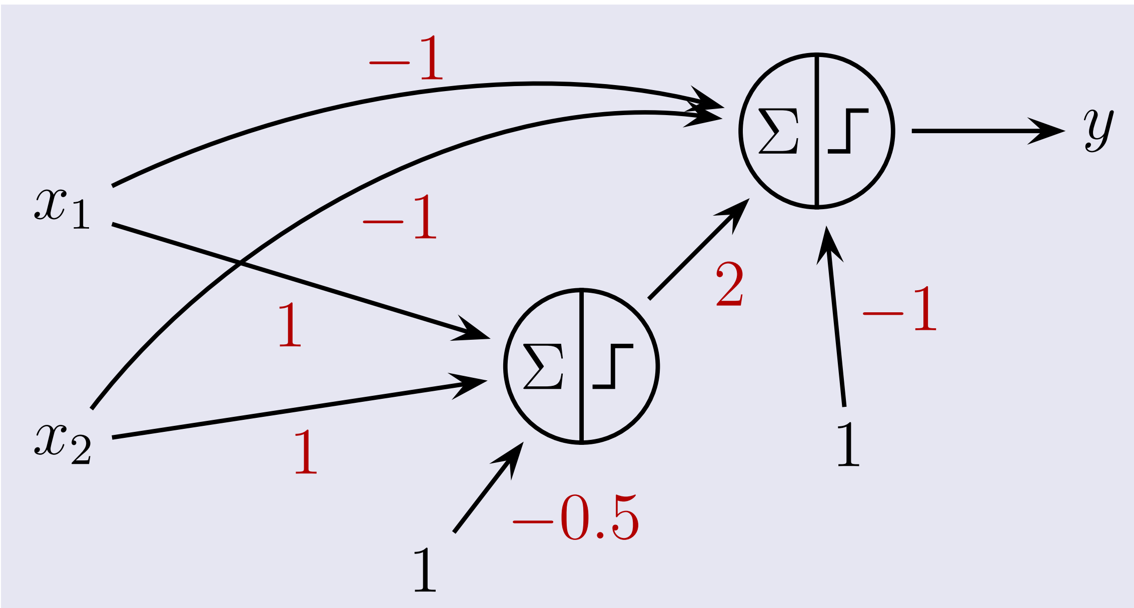
Geometric interpretation:

partitioning of first perceptron



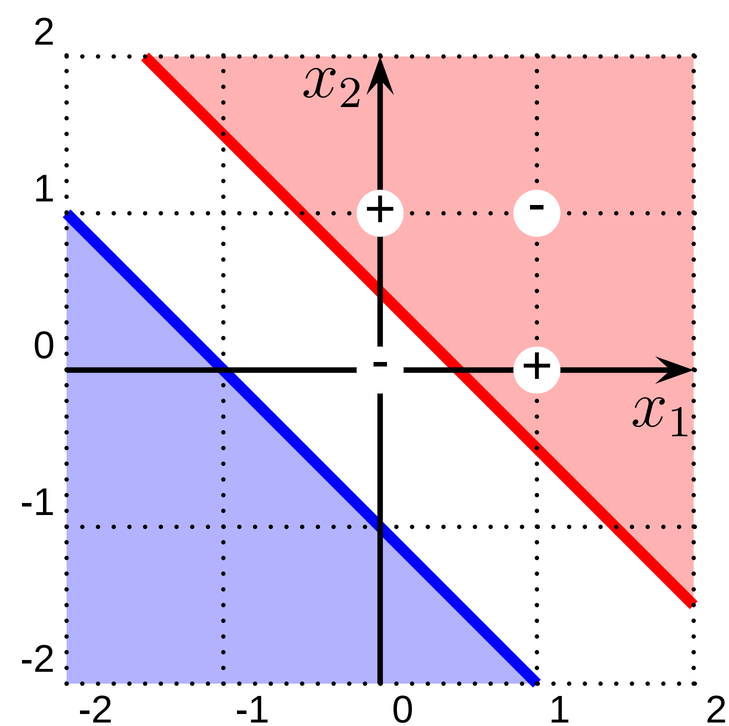
Perceptron networks (cont.)

XOR-network:

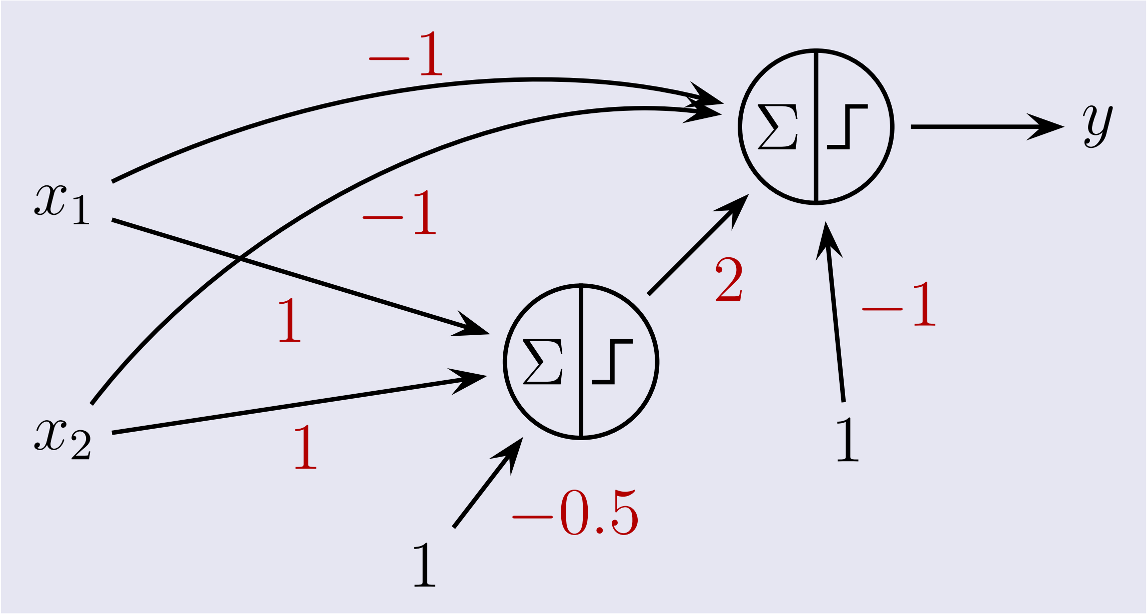


Geometric interpretation:

partitioning of second perceptron, assuming first perceptron yields 0

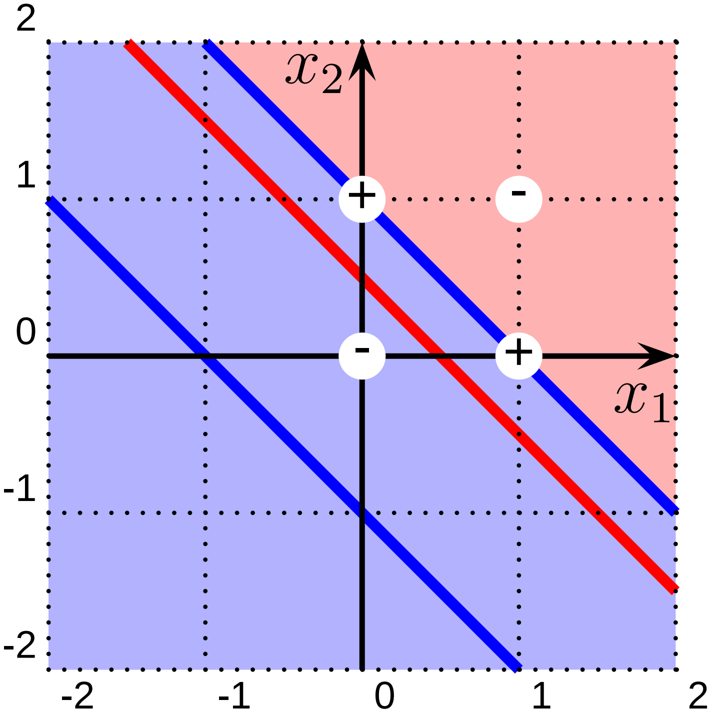


XOR-network:



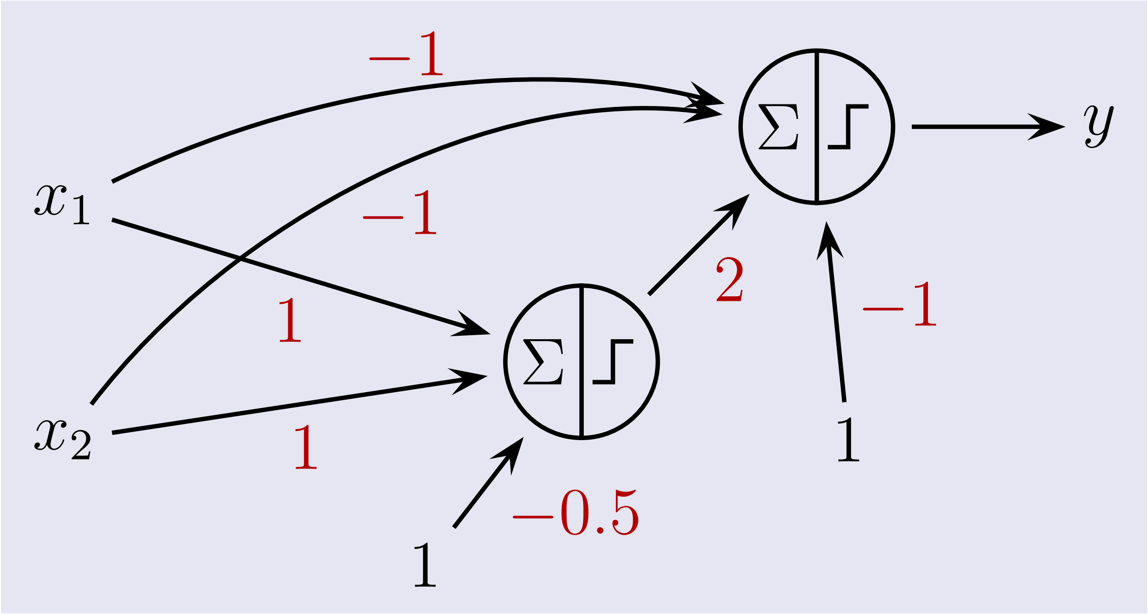
Geometric interpretation:

partitioning of second perceptron, assuming first perceptron yields 1



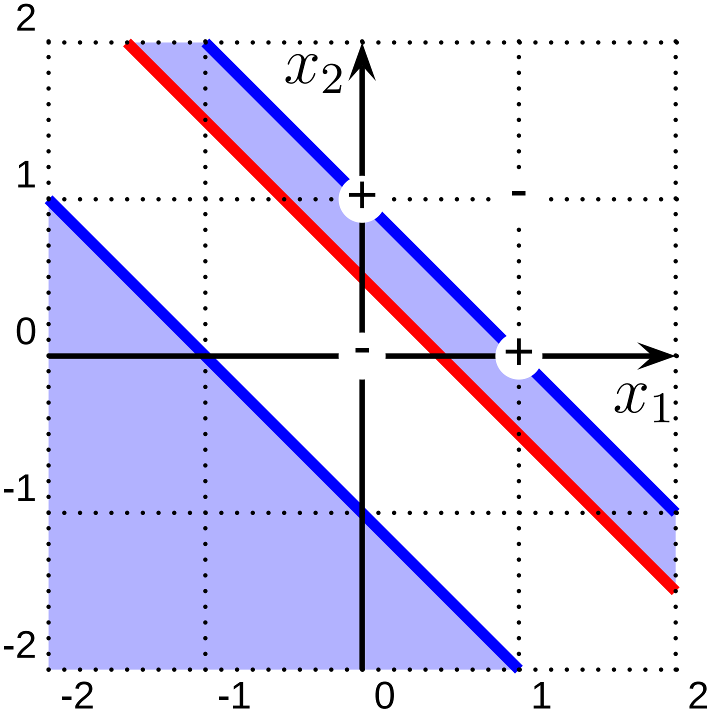
Perceptron networks (cont.)

XOR-network:



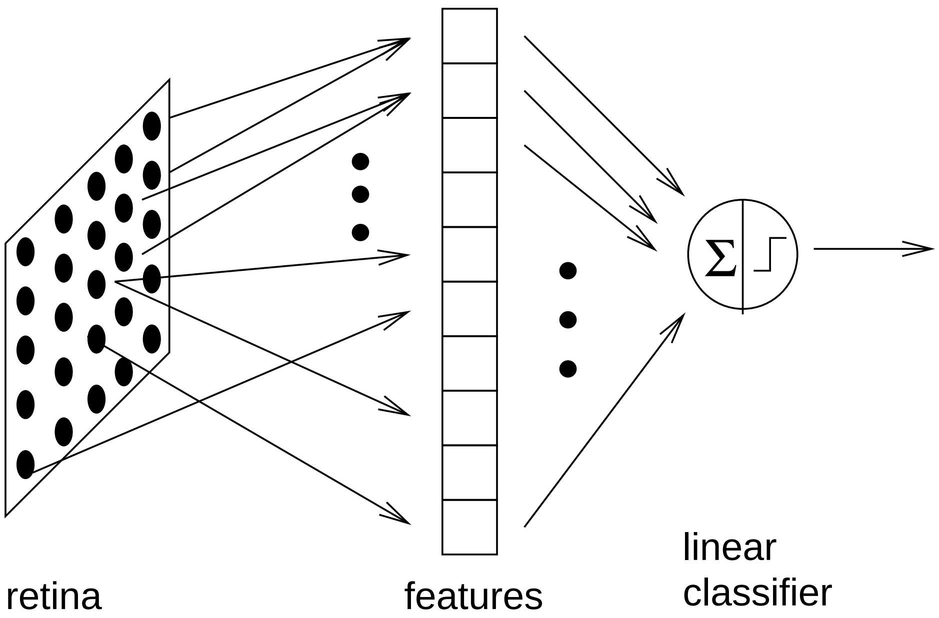
Geometric interpretation:

combining both



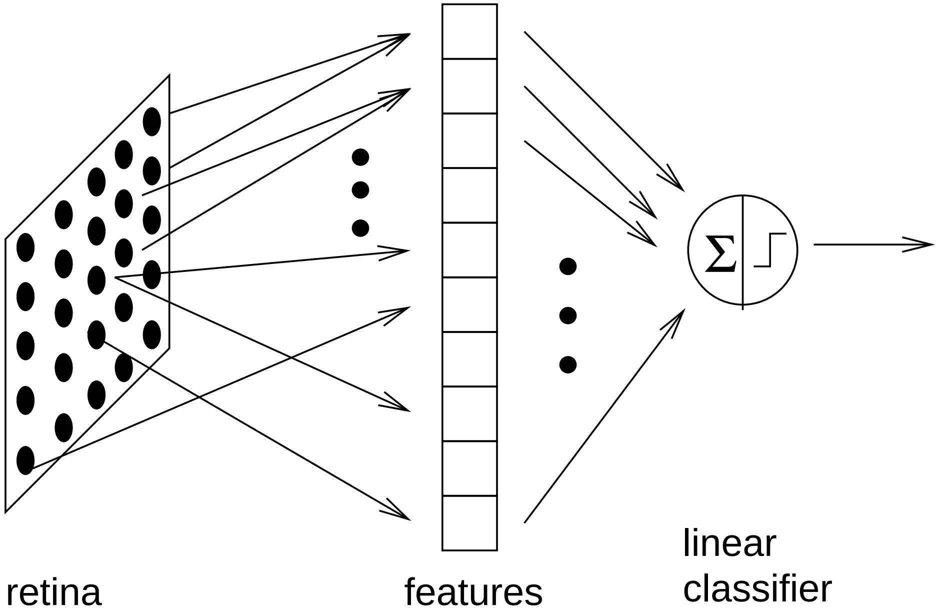
Historical remarks

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



Historical remarks

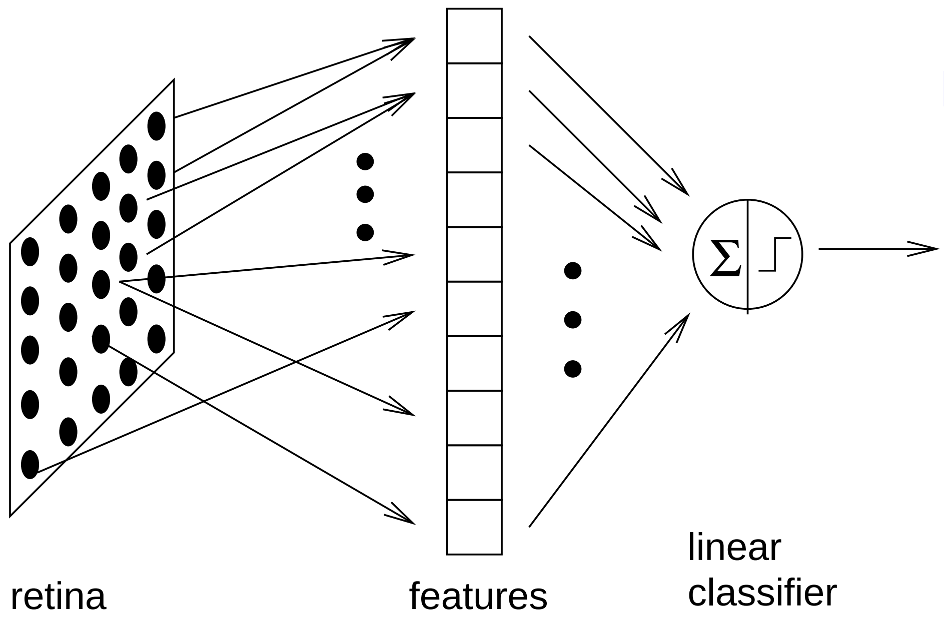
- Rosenblatt perceptron (1958):
  - retinal input (array of pixels)
  - preprocessing level, calculation of features
  - adaptive linear classifier
  - inspired by human vision



- 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)

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

Summary

- Perceptrons are simple neurons with limited representation capabilites: linear seperable 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