Neural Networks 1 - Artificial Neurons

18NES1 - Lecture 3, Summer semester 2024/25

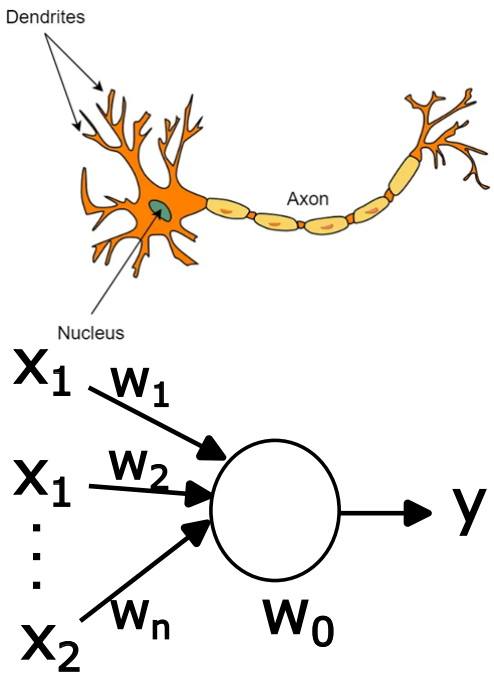
Zuzana Petříčková

March 4, 2025

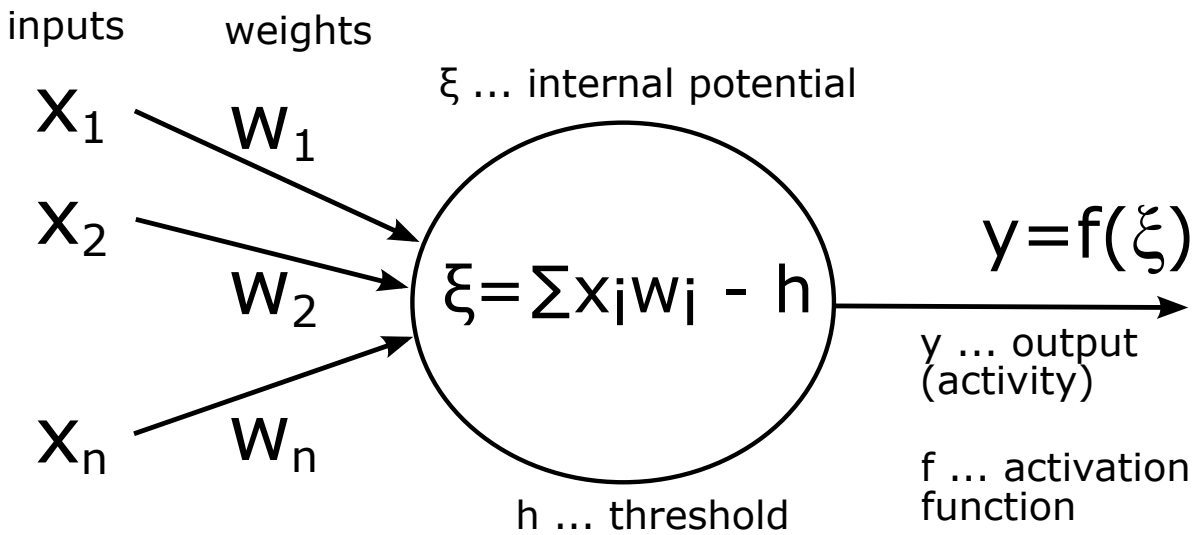
What We Covered Last Time

A n introduction to Single Neuron models

1. From Biological to Artificial Neurons
2. The Earliest Artificial Neuron Models:
   - McCulloch-Pitts Neuron (1943)
   - Perceptron (Rosenblatt, 1955)
3. Perceptron and Logical Function Representation
4. Perceptron Network — Logical Threshold Circuit
5. Geometric Interpretation of the Perceptron and Linear Separability



Mathematical Model of a Neuron - Original Definition



Classical Definition: Threshold h

- Internal potential: $\xi = \sum_{i=1}^{n} w_i x_i - h = \vec{w}\vec{x}^T - h$
- Output: $y = f(\xi)$ ... **Step activation function:**

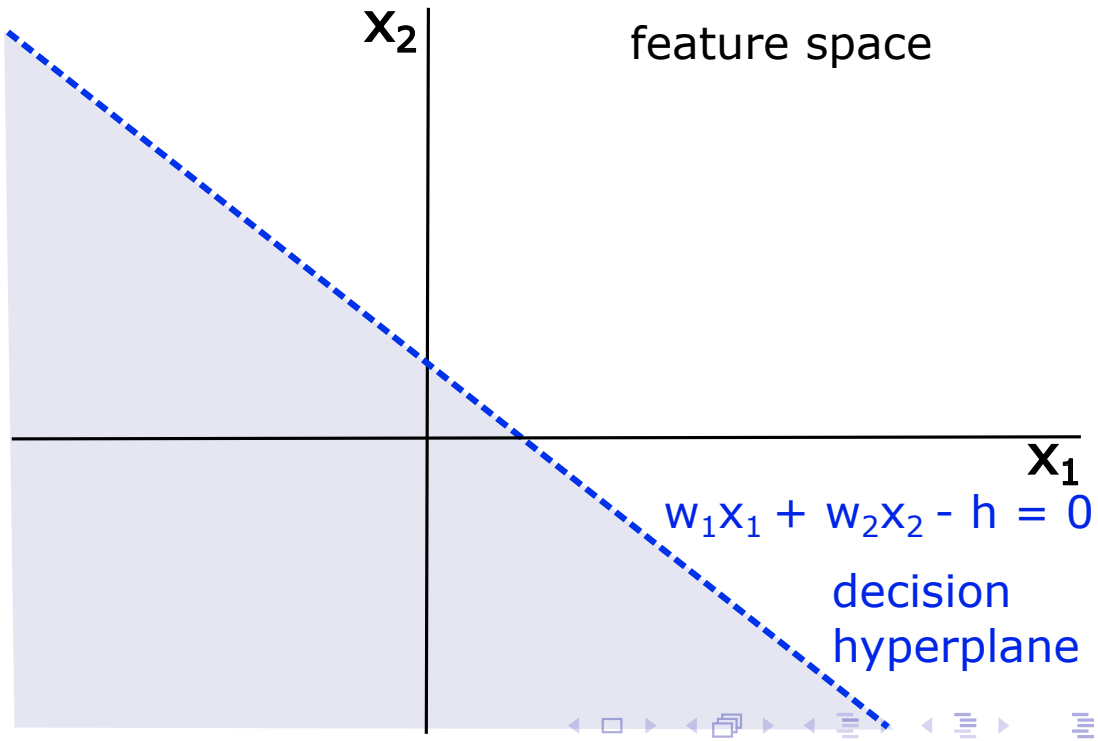
$$f(\xi) = \begin{cases} 1, & \text{if } \xi > 0 \quad \text{(the neuron is active)} \\ 0.5 \text{ (or 0)}, & \text{if } \xi = 0 \quad \text{(the neuron is passive)} \\ 0 \text{ (or -1)}, & \text{if } \xi < 0 \quad \text{(the neuron is neutral)} \end{cases}$$

Geometric Interpretation of an Artificial Neuron

- The neuron's inputs can be represented as points in an n-dimensional Euclidean space (input/feature space).
- Setting the neuron's internal potential to $\xi = 0$ results in the equation of a **decision hyperplane (decision boundary)** .

$$\xi = w_1 x_1 + w_2 x_2 - h = 0$$

$$x_2 = -\frac{w_1}{w_2} x_1 + \frac{h}{w_2}$$



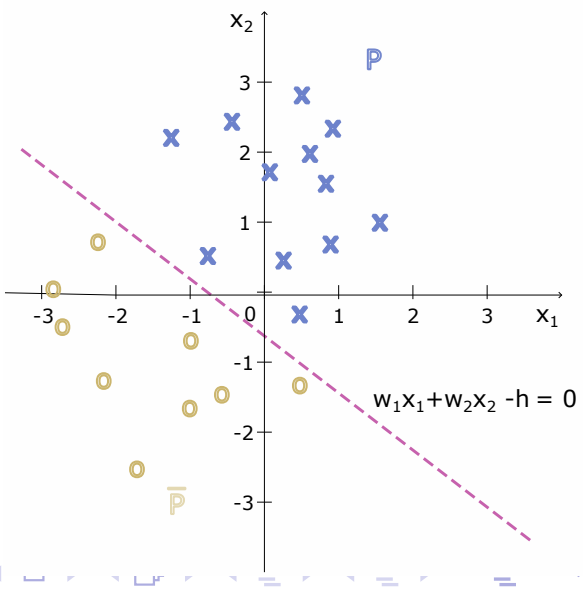
Geometric Interpretation of an Artificial Neuron

The perceptron can be used as a **linear classifier**, separating patterns into two classes (P and $\overline{P}$).

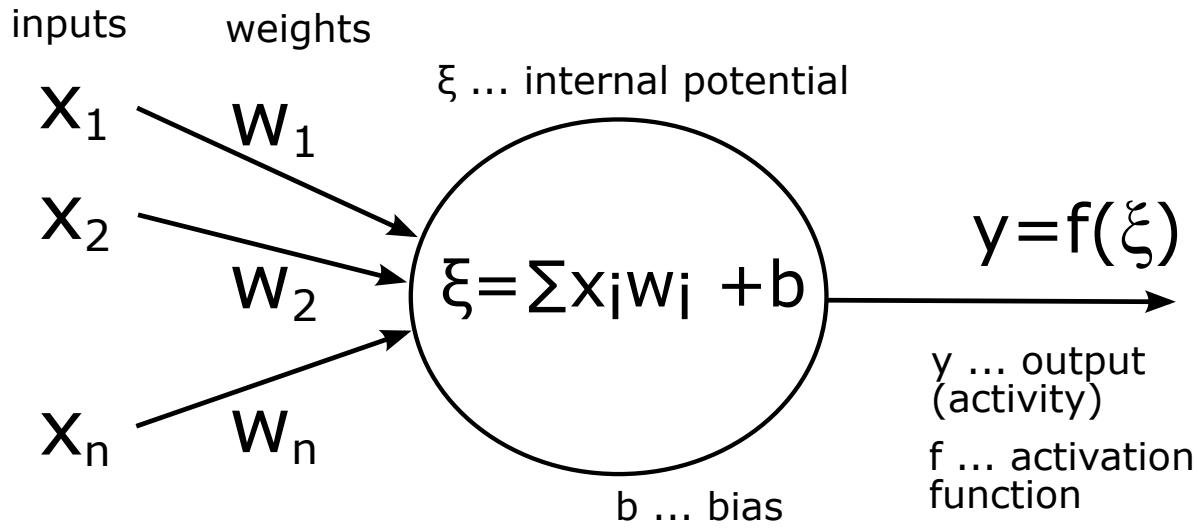
- **why linear?:** The boundary between the two classes is a hyperplane: $w_1x_1 + w_2x_2 + \cdots + w_nx_n - h = 0$
  which represents a point, a line, or a plane depending on the number of input dimensions.

Step Activation Function:

- $f(\xi) = 1$ for $\xi > 0$ ... neuron is **active** (class P)
- $f(\xi) = 0$ (or -1) for $\xi < 0$ ... neuron is **passive** (class $\overline{P}$)
- $f(\xi) = 0.5$ (or 0) for $\xi = 0$ ... neuron is **neutral**, meaning it cannot decide.



Mathematical Model of a Neuron — Modern Definition



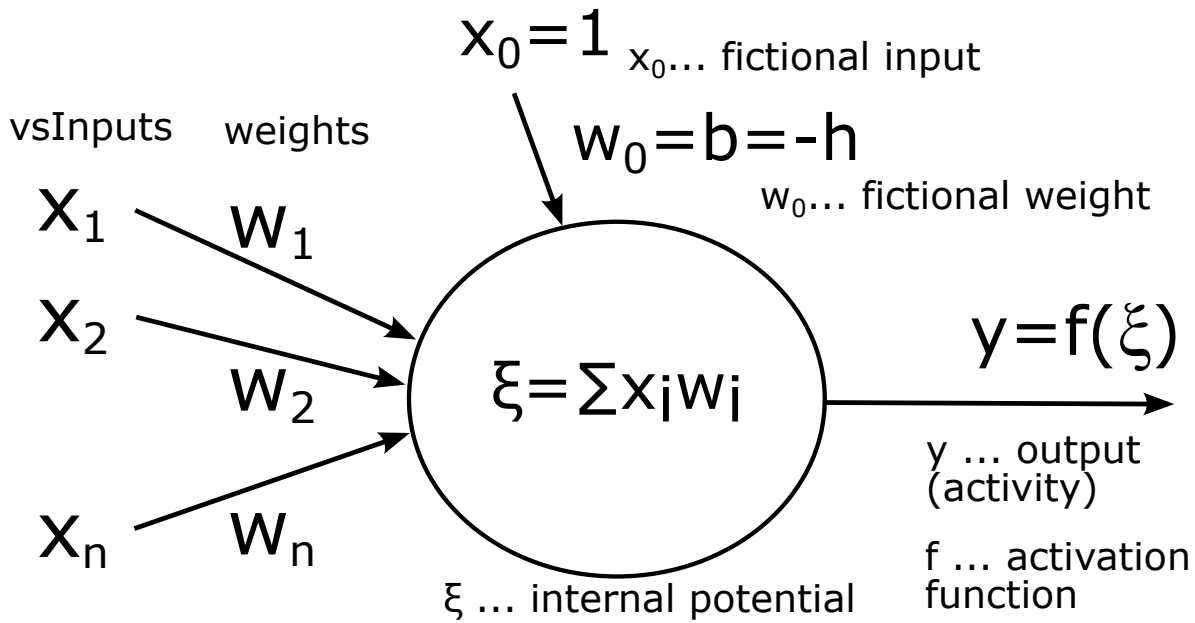
Alternative Definition: Threshold h $\rightarrow$ Bias b

- Internal potential:

$$\xi = \sum_{i=1}^{n} w_i x_i + b = \vec{w} \cdot \vec{x}^T + b$$

- Output: $y = f(\xi)$ (step activation function ... sign)

Mathematical Model of a Neuron — Matrix Definition

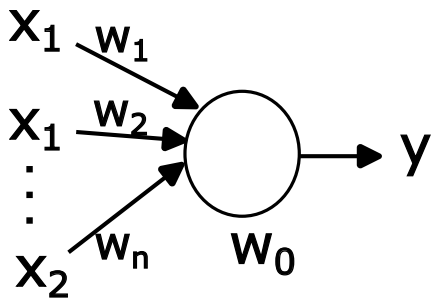


Alternative Definition: Introducing a Fictional Bias Input

- Extended feature space ... $\vec{x} = (x_0 = 1, x_1, ..., x_n)$
- Extended weight vector ... $\vec{w} = (w_0 = b = -h, w_1, ..., w_n)$
- Internal potential ... $\xi = \sum_{i=0}^{n} w_i x_i = \vec{w} \cdot \vec{x}^T$
- Output: $y = f(\xi)$ (step activation function ... sign)

Perceptron (Rosenblatt, 1955) and Logical Function Representation

We consider the following perceptron model:



$$\xi = \vec{w} \cdot \vec{x}^T = \sum_{i=0}^{n} w_i x_i$$

$$= w_0 + \sum_{i=1}^{n} w_i x_i$$

Binary Perceptron

- Inputs: $x_i \in \{0, 1\}$
- Outputs: $y \in \{0, 0.5, 1\}$
- $y = f(\xi) = signum(\xi)$

$$signum(\xi) = \begin{cases} 1, & \text{if } \xi > 0 \\ 0.5, & \text{if } \xi = 0 \\ 0, & \text{if } \xi < 0 \end{cases}$$

Bipolar Perceptron

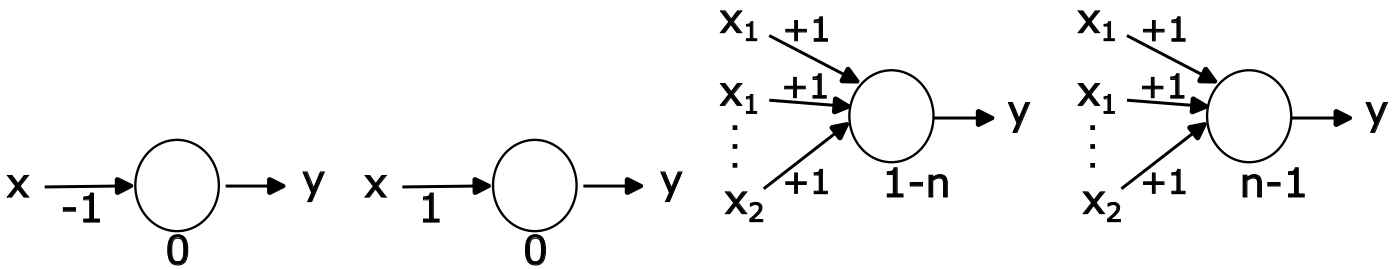
- Inputs: $x_i \in \{-1, +1\}$
- Outputs: $y \in \{-1, 0, +1\}$
- $y = f(\xi) = sign(\xi)$

$$sign(\xi) = \begin{cases} 1, & \text{if } \xi > 0 \\ 0, & \text{if } \xi = 0 \\ -1, & \text{if } \xi < 0 \end{cases}$$

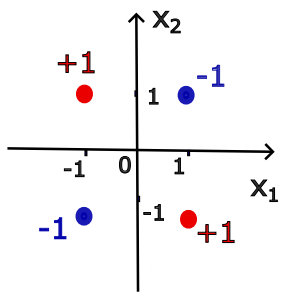
Perceptron and Logical Function Representation

A perceptron can implement basic logical functions:

- NOT (negation), ID (identity)
- AND (conjunction), OR (disjunction)



but it can not represent all logical functions, e.g., XOR (Exclusive OR, A $\otimes$ B):

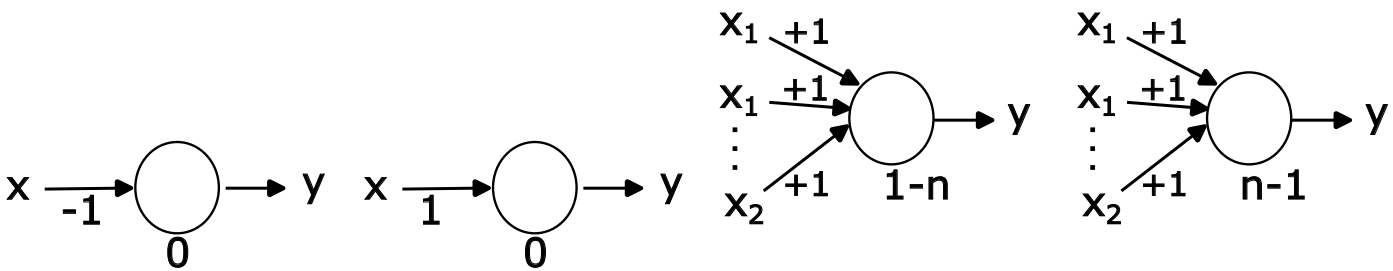


... the problem of **linear separability**

Perceptron and Logical Function Representation

A perceptron can implement basic logical functions:

- NOT (negation), ID (identity)
- AND (conjunction), OR (disjunction)



→ Using perceptrons, we can build a **logical threshold circuit**, allowing the representation of any Boolean function.

- AND represents the **intersection** of convex regions, while OR represents their **union**.

XOR - Solution to the Optional Homework

Example: Exclusive OR (XOR)

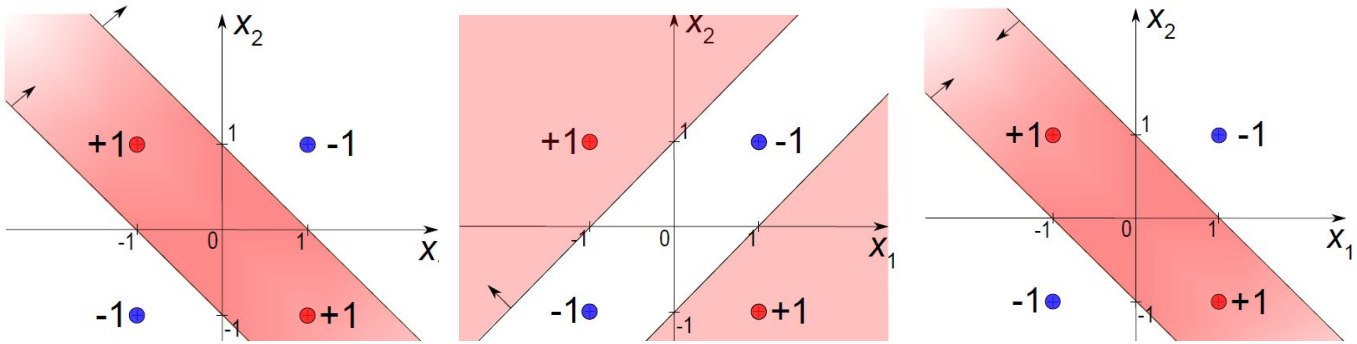
1. XOR can be represented using basic logical operations (AND, OR, NOT) in different ways. Can you design multiple representations?
2. Design the smallest possible neural network that can represent XOR. How many neurons does it contain?

x_1	x_2	$y = x_1 \otimes x_2$
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	-1

XOR - Solution to the Optional Homework

Example: Exclusive OR (XOR)

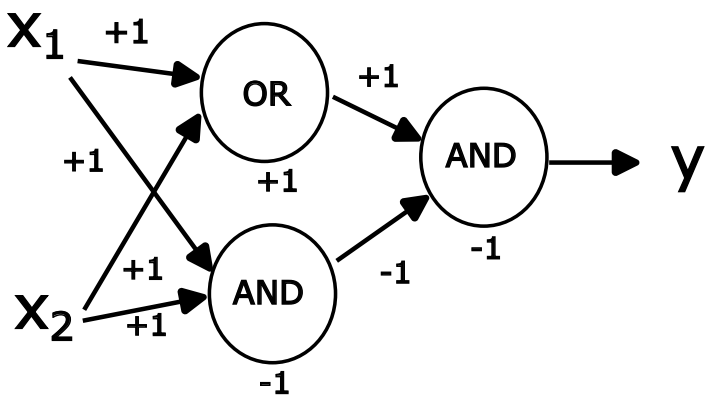
1. XOR can be represented using basic logical operations (AND, OR, NOT) in different ways. Can you design multiple representations?
2. Design the smallest possible neural network that can represent XOR. How many neurons does it contain?

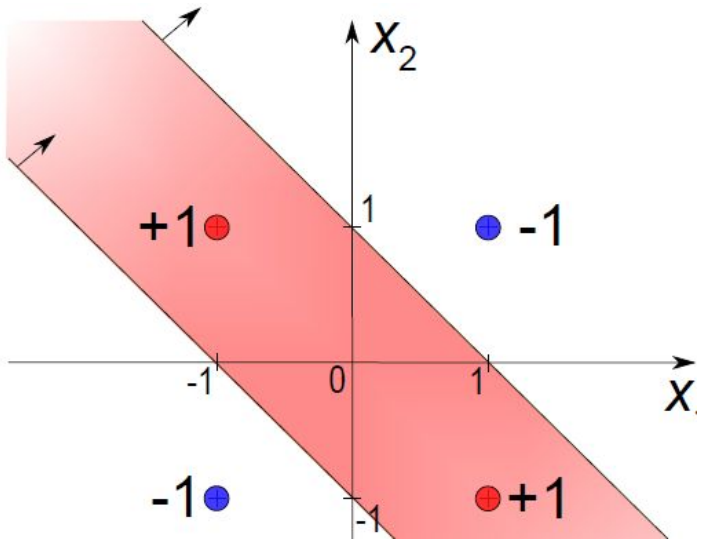


XOR - Solution to the Optional Homework

Example: Exclusive OR (XOR)

- 1st Solution: $x_1 \otimes x_2 = (x_1 \vee x_2) \wedge \neg(x_1 \wedge x_2)$

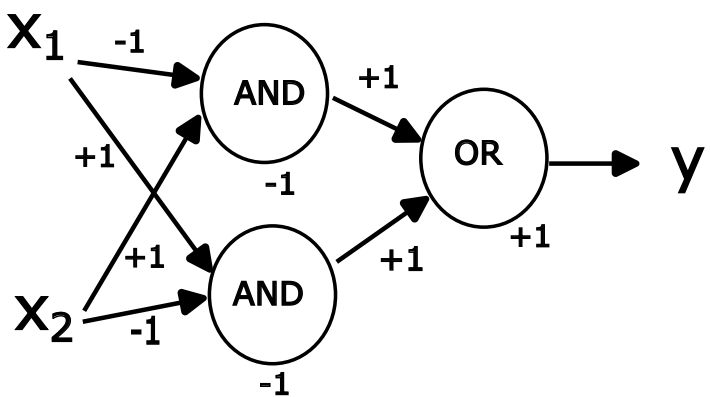


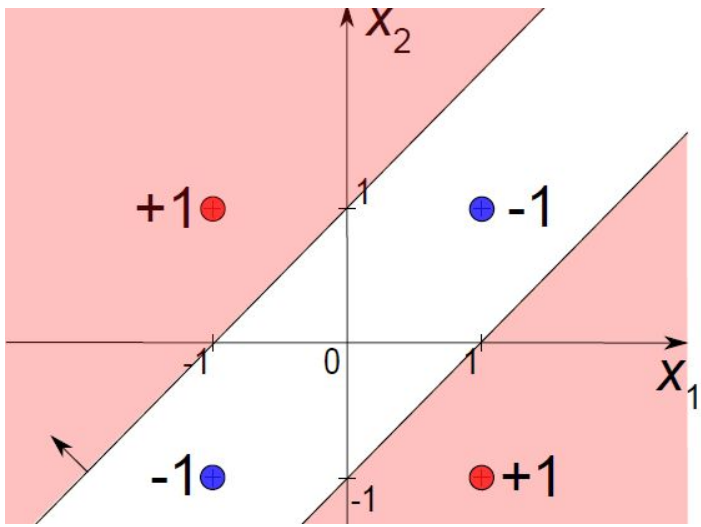


XOR - Solution to the Optional Homework

Example: Exclusive OR (XOR)

- 2nd Solution: $x_1 \otimes x_2 = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$

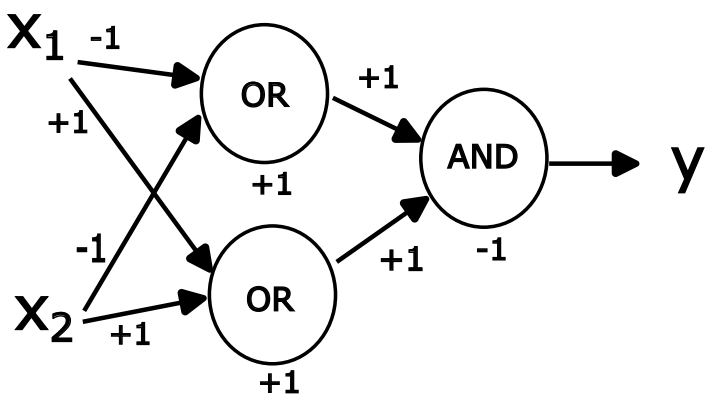


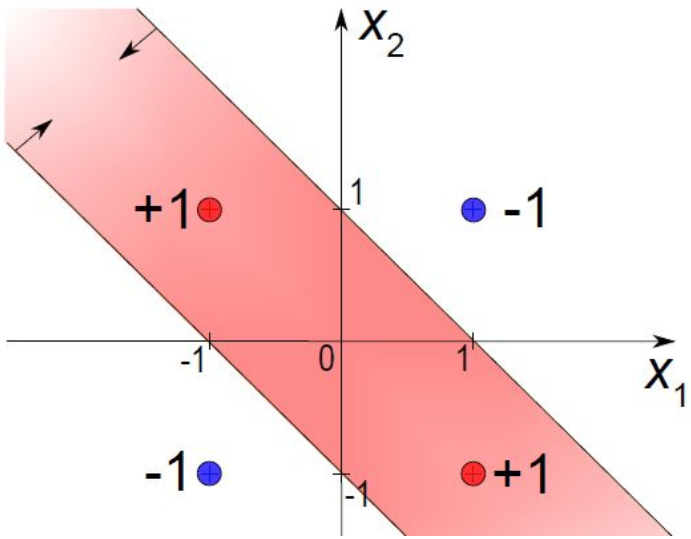


XOR - Solution to the Optional Homework

Example: Exclusive OR (XOR)

- 3rd Solution:
  $x_1 \otimes x_2 = (x_1 \vee x_2) \wedge \neg(x_1 \wedge x_2) = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$





Neural Networks 1 - Lecture 3: Artificial Neuron

Perceptron - Learning Algorithm

We already know:

A perceptron (with a step activation function) can be used as a **linear classifier** to separate input patterns into two sets/classes (P and $\overline{P}$).

Separating Hyperplane

Defined by an (n+1)-dimensional weight vector $\vec{w}$, it is the set of all points $\vec{x} \in R^n$ for which $\vec{w} \cdot \vec{x} + w_0 = 0$.

Problem:

Find the appropriate weights and threshold/bias that allow for the correct classification of input patterns into sets P and $\overline{P}$ using a separating hyperplane.

Possible Solution:

The Perceptron (Rosenblatt's) Learning Algorithm (Rosenblatt, 1959)

Perceptron - Learning Algorithm

Data for Training the Model:

- Training (data)set T
  - A set of N training patterns $T = \{(\vec{x_1}, d_1), ..., (\vec{x_N}, d_N)\}$
- Training pattern $(\vec{x}, d)$:
  - $\vec{x} = (x_1, ..., x_n)$ ... input pattern with n features
  - $d \in \{-1, 1\}$ ... desired (expected) output
- T can be divided into two sets P and $\overline{P}$:
  - P ... positive patterns $(d = 1)$
  - $\overline{P}$ ... negative patterns $(d = -1)$
  - $T = P \cup \overline{P}$

Perceptron - Learning Algorithm

Data used for training the model:

- Training set $T = \{(\vec{x_1}, d_1), ..., (\vec{x_N}, d_N)\}$:
- Matrix representation $T = (X|\vec{d})$:

x_{11}	x_{12}	...	x_{1n}	d_1	... 1st training pattern
x_{21}	x_{22}	...	x_{2n}	d_2	... 2nd training pattern
...	...		...	...	
x_{N1}	x_{N2}	...	x_{Nn}	d_N	... N-th training pattern

- For the extended feature space:

$x_{10} = 1$	x_{11}	...	x_{1n}	d_1	... 1st training pattern
$x_{20} = 1$	x_{21}	...	x_{2n}	d_2	... 2nd training pattern
...	...		...	...	
$x_{N0} = 1$	x_{N1}	...	x_{Nn}	d_N	... N-th training pattern

Perceptron - Learning Algorithm

Learning Objective:

- Set the weights (and bias) of the neuron so that it correctly classifies all training patterns, i.e.:
  - $y_p = d_p$ ... for all training patterns in T
    y_p ... actual response (output) of the neuron for the input pattern $\vec{x}_p$
- Perceptron with a step activation function:

$$y_p = sign(\xi_p)$$

$$\xi_p = \sum_{i=1}^{n} w_i x_{pi} + w_0 = \sum_{i=0}^{n} w_i x_{pi} = \vec{w} \cdot \vec{x}_p$$

$\vec{x}_p = (1, x_{p1}, ..., x_{pn})$ ... extended input pattern

$\rightarrow$ Our goal:

- $\vec{w} \cdot \vec{x_p} < 0$ ... for extended training patterns from $\overline{P}$
- $\vec{w} \cdot \vec{x_p} > 0$ ... for extended training patterns from P

Perceptron - Learning Algorithm

Our Goal:

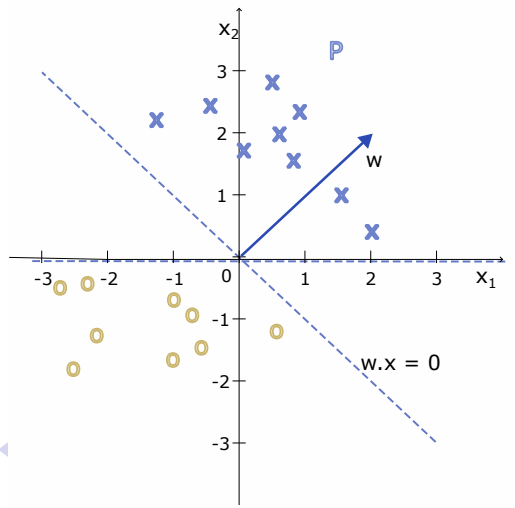
- $\vec{w} \cdot \vec{x_p} < 0$ ... for extended training patterns from $\overline{P}$
- $\vec{w} \cdot \vec{x_p} > 0$ ... for extended training patterns from P

Possibilities:

1. The system of inequalities has no solution $\rightarrow$ we seek an imperfect solution.
2. At least one perfect solution $\vec{w}$ exists in R^{n+1} $\rightarrow$ an infinite number of solutions exist in R^{n+1} (even in Z^{n+1}).

Possible Additional Conditions:

- $|w_0| + |w_1| ... + |w_n| = \min$
- $\sqrt{{w_0}^2 + \cdots {w_n}^2} = \min$
- $\max(|w_0|, ..., |w_n|) = \min$



Perceptron - Learning Algorithm

Notation:

- $y = f(\vec{x})$ ... actual response (output) of the neuron for the input pattern $\vec{x}$ (where d is the desired response)

Target (Error) Function

- Number of misclassified patterns: $E = \sum_{(\vec{x},d)\in T} [d \neq f(\vec{x})]$
- For a bipolar model:

$$E = \sum_{x\in P} \frac{1}{2}(1 - f(\vec{x})) + \sum_{x\in \overline{P}} \frac{1}{2}(1 + f(\vec{x}))$$

- For a binary model:

$$E = \sum_{x\in P} (1 - f(\vec{x})) + \sum_{x\in \overline{P}} f(\vec{x})$$

Learning Objective:

- Minimize E in the weight space. Ideally, $E = 0$.

Perceptron - Rosenblatt's Learning Algorithm

Concept and Derivation:

- We will work in the extended feature and weight space.

Our goal:

- $\vec{w} \cdot \vec{x} < 0$ for $(\vec{x}, d) \in \overline{P}$
- $\vec{w} \cdot \vec{x} > 0$ for $(\vec{x}, d) \in P$

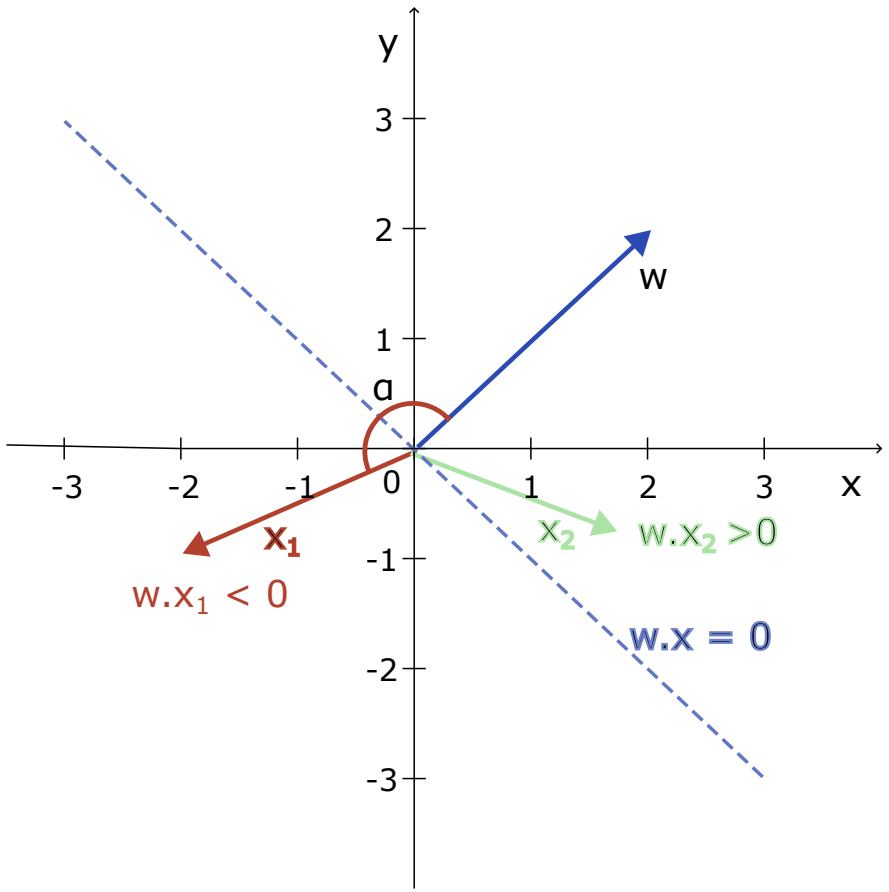
From the definition of the dot product:

- $\vec{w} \cdot \vec{x} = |\vec{w}||\vec{x}| \cos(\alpha)$

$\rightarrow$

- $\vec{w} \cdot \vec{x} = 0 \ldots |\alpha| = 90°$
- $\vec{w} \cdot \vec{x} > 0 \ldots |\alpha| < 90°$
- $\vec{w} \cdot \vec{x} < 0 \ldots 180 \geq |\alpha| > 90°$

Geometric Interpretation:



Perceptron - Rosenblatt's Learning Algorithm

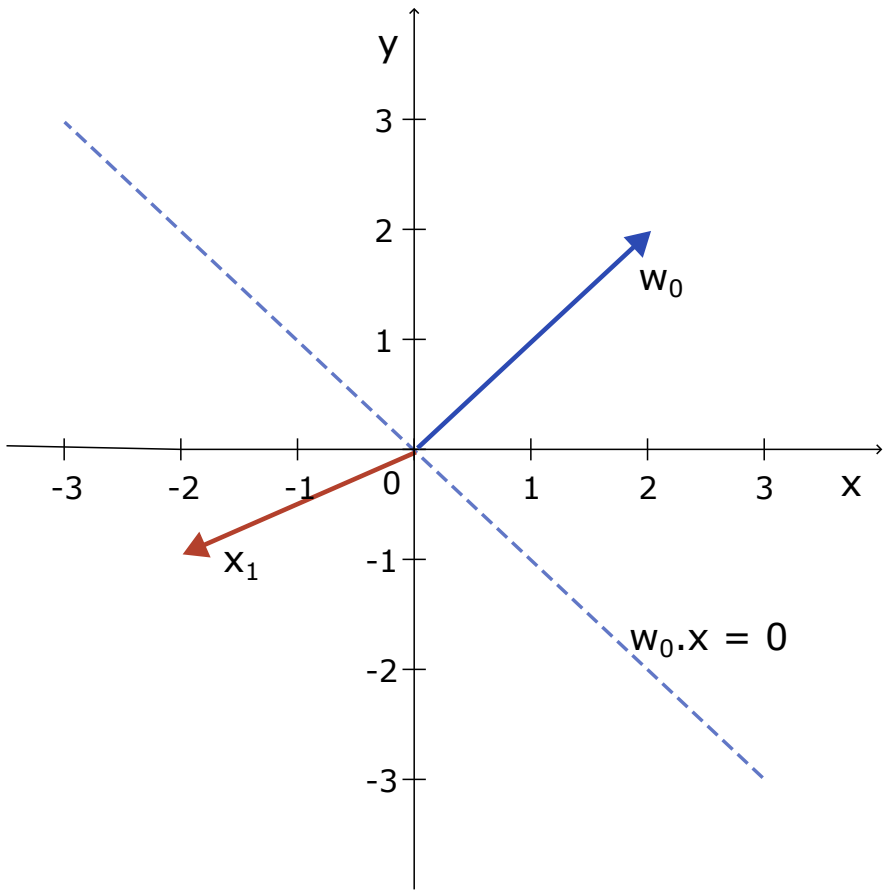
Concept and Derivation:

- $\vec{w_0}$ ... initial (current) weight vector
- $(\vec{x_1}, d_1 = 1) \in P$ ... training pattern where the model produces an incorrect output:
  $\vec{w_0} \cdot \vec{x_1} \leq 0$

What we do:

- Adjust $\vec{w_0}$ to decrease the angle between $\vec{w_0}$ and $\vec{x_1}$
  $\rightarrow$ ideally $|\alpha| < 90$

Geometric Interpretation:



Perceptron - Rosenblatt's Learning Algorithm

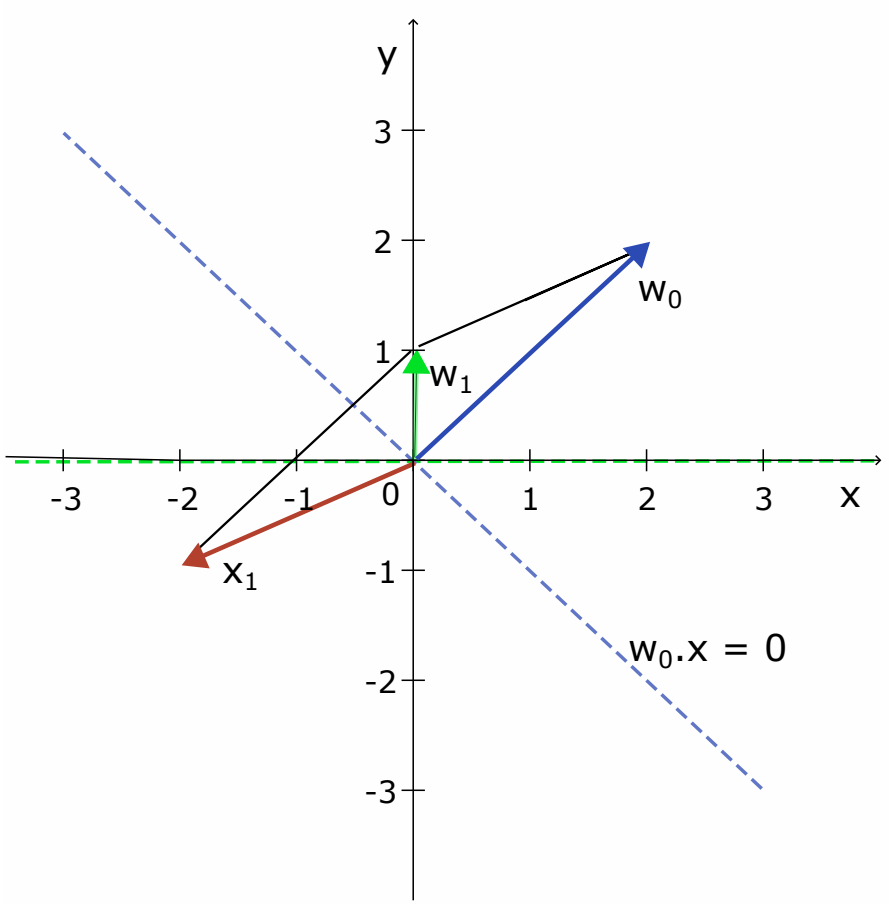
Concept and Derivation:

- $\vec{w_0}$ ... initial (current) weight vector
- $(\vec{x_1}, d_1 = 1) \in P$ ... training pattern where the model produces an incorrect output:
  $\vec{w_0} \cdot \vec{x_1} \leq 0$

What we do:

- Adjust $\vec{w_0}$ to decrease the angle between $\vec{w_0}$ and $\vec{x_1}$
  $\rightarrow$ ideally $|\alpha| < 90$
  $\rightarrow$ Add $\vec{x_1}$ to $\vec{w_0}$
  $\vec{w_1} = \vec{w_0} + \vec{x_1}$

Geometric Interpretation:



Perceptron - Rosenblatt's Learning Algorithm

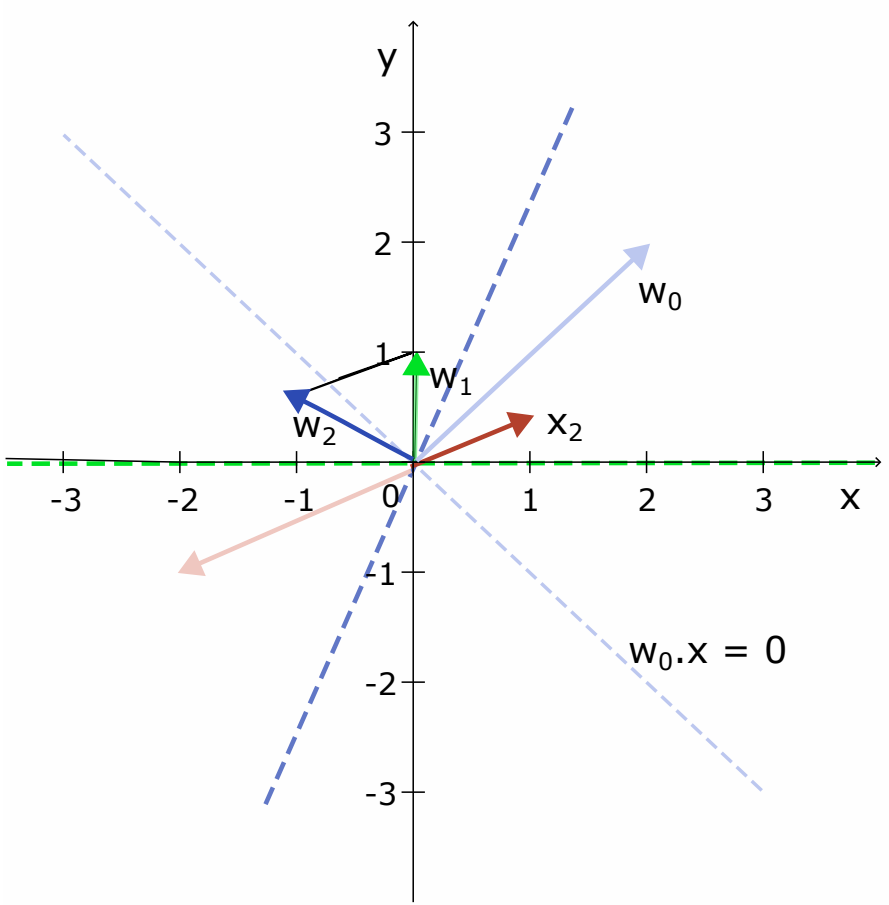
Concept and Derivation:

- $\vec{w}_1$ ... current weight vector
- Training pattern $(\vec{x}_2, d_2 = -1) \in \overline{P}$, for which: $\vec{w}_1 \cdot \vec{x}_2 \geq 0$

What we do:

- Adjust $\vec{w}_1$ to increase the angle between $\vec{w}_1$ and $\vec{x}_2$
  $\rightarrow$ ideally $|\alpha| > 90$
  $\rightarrow$ Subtract $\vec{x}_2$ from $\vec{w}_1$
  $\vec{w}_2 = \vec{w}_1 - \vec{x}_2$

Geometric Interpretation:



Perceptron - Rosenblatt's Learning Algorithm (1959)

1. Initialize weights:
   $\vec{w}(0) = (w_0, w_1, ..., w_n)^T$ ... weight vector at time 0 (including threshold/bias) $w_0 = b = -h$
2. Present the next training pattern $(\vec{x}_t, d_t)$:
   $\vec{x}_t = (x_{t0} = 1, x_{t1}, ..., x_{tn})$ ... input pattern
   d_t ... desired output
3. Compute the actual output (network response):
   $y_t = sign(\vec{x}_t \vec{w})$
4. Update weights:

$$\vec{w}(t+1) = \begin{cases} \vec{w}(t) & \text{if } y(t) = d(t) \\ \vec{w}(t) + \vec{x}_t^T & \text{if } y_t \neq 1, d_t = 1 \\ \vec{w}(t) - \vec{x}_t^T & \text{if } y_t \neq -1, d_t = -1 \end{cases}$$

   or equivalently: $\vec{w}(t+1) = \vec{w}(t) + \vec{x}_t^T sign(d_t - y_t)$
5. If t has not reached the maximum value, return to step 2.

Perceptron - Rosenblatt's Learning Algorithm (1959)

How to present training patterns? ... different strategies:

1. **Iteratively**: one by one
   - **By epochs**: during one epoch, each pattern is presented exactly once
   - Number of epochs ... how many times the entire training set is presented
2. **Randomly** ... in each iteration, a random training pattern is selected
3. A suitable **combination of previous strategies**:
   1. Shuffle patterns randomly within each epoch
   2. Start with random presentation, then systematically go through all patterns at the end of training

Perceptron - Rosenblatt's Learning Algorithm (1959)

How to initialize weights? ... different strategies:

- $\vec{w}(0) = \vec{0}$
- Randomly: small random values
- Use a heuristic: e.g., the average of patterns from P minus the average of patterns from $\overline{P}$
- ...

When to stop training?

1. Predefined number of iterations or epochs
2. When $E = 0$, or when the error is sufficiently small ... $E < E_{min}$

Perceptron - Rosenblatt's Learning Algorithm (1959)

Example 1

	x_0	x_1	x_2	d
a	+1	-1	-1	+1
b	+1	-1	+1	+1
c	+1	+1	-1	+1
d	+1	+1	+1	-1

- $\vec{w}(0) = \vec{0}$
- Patterns are presented iteratively (randomly within each epoch)
  c, b, d, a; b, a, c, d; ...

Perceptron - Rosenblatt's Learning Algorithm (1959)

Example 1

	x_0	x_1	x_2	d
a	+1	-1	-1	+1
b	+1	-1	+1	+1
c	+1	+1	-1	+1
d	+1	+1	+1	-1

Solution c, b, d, a; b, a, c, d; ...

	w_0	w_1	w_2	d	ξ	y	operation
$\vec{w}(0)$	0	0	0				
c	+1	+1	-1	+1	0	0	+
$\vec{w}(1)$	+1	+1	-1				
b	+1	-1	+1	+1	-1	-1	+
$\vec{w}(2)$	+2	0	0				

Perceptron - Rosenblatt's Learning Algorithm (1959)

Example 1

	x_0	x_1	x_2	d
a	+1	-1	-1	+1
b	+1	-1	+1	+1
c	+1	+1	-1	+1
d	+1	+1	+1	-1

Solution c, b, d, a; b, a, c, d; ...

	w_0	w_1	w_2	d	ξ	y	operation
$\vec{w}(2)$	+2	0	0				
d	+1	+1	+1	-1	+2	+1	-
$\vec{w}(3)$	+1	-1	-1				
a	+1	-1	-1	+1	+3	+1	none

Perceptron - Rosenblatt's Learning Algorithm (1959)

Example 1

	x_0	x_1	x_2	d
a	+1	-1	-1	+1
b	+1	-1	+1	+1
c	+1	+1	-1	+1
d	+1	+1	+1	-1

Solution c, b, d, a; b, a, c, d; ...

	w_0	w_1	w_2	d	ξ	y	operation
$\vec{w}(4)$	+1	-1	-1				
b	+1	-1	+1	+1	+1	+1	none
a	+1	-1	-1	+1	+3	+1	none
c	+1	+1	-1	+1	+1	+1	none
d	+1	+1	+1	-1	-1	-1	none, end

$$\rightarrow \vec{w} = (+1, -1, -1)^T$$

Perceptron - Rosenblatt's Learning Algorithm (1959)

What if input patterns have different magnitudes?

- Or what if one or several patterns are outliers?
  $\rightarrow$ This may significantly slow down training
- **Solution:** normalize input vectors to the same magnitude:

$$\vec{x}_{new} = \frac{\vec{x}}{|\vec{x}|} = \frac{\vec{x}}{\sqrt{x_0^2 + x_1^2 + ... + x_n^2}}$$

  $\rightarrow$ Normalized Rosenblatt learning algorithm

Perceptron - Rosenblatt's Learning Algorithm (1959)

Advantages

- Simple algorithm
- For linearly separable sets, the algorithm converges, i.e., it finds the correct solution in a finite number of steps *(Rosenblatt, 1959)*

Disadvantages

- Very slow algorithm
  - The actual number of steps grows exponentially with the number of inputs
  - Sensitive to outliers (if no normalization is applied)
- Can only classify linearly separable sets
- It cannot be extended to work for neural networks with multiple layers
- Poor sgeneralization ... may not find the "optimal"decision hyperplane

Perceptron - Other Learning Algorithms

Other Variants of the Perceptron Learning Algorithm

1. Rosenblatt's Batch Learning Algorithm
   - The entire training set is presented at once:

$$\vec{w}(t+1) = \vec{w}(t) + \sum_{p=1}^{N} \vec{x}_p^T \, sign(d_p - y_p)$$

2. Rosenblatt's Algorithm with a Learning Parameter
   - When adding/subtracting patterns, we use a weighting factor α: $\vec{w}(t+1) = \vec{w}(t) + \alpha \vec{x}_t^T \, sign(d_t - y_t)$
3. Pocket Algorithm
   - Is able to find an optimal solution even for non-linearly separable sets

Hebbian Learning

- Each training pattern is presented exactly once:

$$\vec{w}(t+1) = \vec{w}(t) + d_t \vec{x}_t^T$$

Perceptron - Rosenblatt's Batch Learning Algorithm

- The entire training set is presented at once:

$$\vec{w}(t+1) = \vec{w}(t) + \sum_{p=1}^{N} \vec{x}_p^T \, sign(d_p - y_p)$$

- For matrix representation:
  $w(t) = (w_0, w_1, ..., w_n)^T$
  $T = (X, \vec{d})$

$x_{10} = 1$	x_{11}	...	x_{1n}	d_1
...	...		...	...
$x_{N0} = 1$	x_{N1}	...	x_{Nn}	d_N

$$\vec{w}(t+1) = \vec{w}(t) + X^T \, sign(\vec{d} - \vec{y})$$

Perceptron - Rosenblatt's Batch Learning Algorithm

1. Initialize weights:
   $\vec{w}(0) = (w_0, w_1, ..., w_n)^T$ ... weight vector at time (epoch) 0
2. Present the entire training set $(X, \vec{d})$

$x_{10} = 1$	x_{11}	...	x_{1n}	d_1
...	...		...	
$x_{N0} = 1$	x_{N1}	...	x_{Nn}	d_N

3. Compute the actual output (network response) $\vec{y}$:

$$\vec{y} = sign(X\vec{w})$$

4. Update weights:

$$\vec{w}(t+1) = \vec{w}(t) + X^T sign(\vec{d} - \vec{y})$$

5. If the number of epochs has not reached the maximum, return to step 2.

Perceptron - Rosenblatt's Batch Learning Algorithm

Advantages and Disadvantages

- Matrix representation, possibility of parallel computation
- Often faster and more stable convergence than Rosenblatt's algorithm (but not always)
- Learning outcome does not depend on how we order the patterns
- Higher memory requirements
- No known proof of convergence, even for linearly separable sets
- For non-linearly separable sets, just like the original algorithm, the algorithm may oscillate indefinitely

Bipolar Perceptron - Hebbian Learning (1949)

- Each training pattern is presented exactly once:
  1. Initialize weights:
     $\vec{w}(0) = \vec{0}^T$
  2. For each training pattern $\vec{x}_t$ $(t = 1, \cdots, N)$, update the weights:
     $$\vec{w}(t+1) = \vec{w}(t) + d_t \vec{x}_t^T$$
     In matrix form:
     $$\vec{w} = X^T \vec{d}$$

Advantages and Disadvantages

- Simpler than Rosenblatt's algorithm
- No need to compute the actual output during training
- Learning outcome does not depend on the order of patterns
- Weights can be easily interpreted
- Does not guarantee finding a perfect solution

Bipolar Perceptron - Hebbian Learning (1949)

Example 1

	x_0	x_1	x_2	d
a	+1	-1	-1	+1
b	+1	-1	+1	+1
c	+1	+1	-1	+1
d	+1	+1	+1	-1

Bipolar Perceptron - Hebbian Learning (1949)

Example 1

	x_0	x_1	x_2	d
a	+1	-1	-1	+1
b	+1	-1	+1	+1
c	+1	+1	-1	+1
d	+1	+1	+1	-1

Solution:

	w_0	w_1	w_2
$\vec{w}(0)$	0	0	0
$d_a\vec{x}_a$	+1	-1	-1
$\vec{w}(1)$	+1	-1	-1
$d_b\vec{x}_b$	+1	-1	+1
$\vec{w}(2)$	+2	-2	0

Bipolar Perceptron - Hebbian Learning (1949)

Example 1

	x_0	x_1	x_2	d
a	+1	-1	-1	+1
b	+1	-1	+1	+1
c	+1	+1	-1	+1
d	+1	+1	+1	-1

Solution:

	w_0	w_1	w_2
$\vec{w}(2)$	+2	-2	0
$d_c\vec{x}_c$	+1	+1	-1
$\vec{w}(3)$	+3	-1	-1
$d_d\vec{x}_d$	-1	-1	-1
$\vec{w}(4)$	+2	-2	-2

$\rightarrow \vec{w} = (+2, -2, -2)^T$

Bipolar Perceptron - Hebbian Learning (1949)

Example 1

	x_0	x_1	x_2	d
a	+1	-1	-1	+1
b	+1	-1	+1	+1
c	+1	+1	-1	+1
d	+1	+1	+1	-1

Solution in Matrix Form: $\vec{w} = X^T \vec{d}$

$$\begin{pmatrix} +1 & +1 & +1 & +1 \\ -1 & -1 & +1 & +1 \\ -1 & +1 & -1 & +1 \end{pmatrix} \begin{pmatrix} +1 \\ +1 \\ +1 \\ -1 \end{pmatrix} = \begin{pmatrix} +2 \\ -2 \\ -2 \end{pmatrix}$$

$\rightarrow \vec{w} = (+2, -2, -2)^T$ ... Did the perceptron learn correctly?

Bipolar Perceptron - Hebbian Learning (1949)

Example 1

Solution: $\vec{w} = (+2, -2, -2)^T$ ... Did the perceptron learn correctly?

	x_0	x_1	x_2	d	ξ	y	
a	+1	-1	-1	+1	+6	+1	ok
b	+1	-1	+1	+1	+2	+1	ok
c	+1	+1	-1	+1	+2	+1	ok
d	+1	+1	+1	-1	-2	-1	ok

$\rightarrow$ Yes (but this is not always the case)

Note

- The first step of the batch learning algorithm would be the same for $\vec{w}_0 = \vec{0}$
- Hebbian learning can be used to initialize weights for Rosenblatt's algorithm

Perceptron - Rosenblatt's Algorithm with a Learning Parameter

1. Initialize weights and bias with small random values:
   $\vec{w}(0) = (w_0, w_1, ..., w_n)$ ... weight vector at time 0 (including threshold/bias) $w_0 = b = -h$
2. Present the next training pattern $(\vec{x}_t, d_t)$:
   $\vec{x}_t = (x_{t0} = 1, x_{t1}, ..., x_{tn})$ ... input pattern
   d_t ... desired output
3. Compute the actual output (network response):
   $y_t = sign(\vec{w} \cdot \vec{x}_t)$
4. Update weights:

$$\vec{w}(t+1) = \begin{cases} \vec{w}(t) & \text{if } y(t) = d(t) \\ \vec{w}(t) + \alpha \vec{x}_t^T & \text{if } y_t \neq 1, d_t = 1 \\ \vec{w}(t) - \alpha \vec{x}_t^T & \text{if } y_t \neq -1, d_t = -1 \end{cases}$$

or equivalently: $\vec{w}(t+1) = \vec{w}(t) + \alpha \vec{x}_t^T sign(d_t - y_t)$
α ... learning parameter

Perceptron - Rosenblatt's Algorithm with a Learning Parameter

How does the choice of the learning parameter affect the result?

- Significant acceleration of training
- Recommendation: $\alpha \in (0, 1]$
- Best practice: initially, α is large and gradually decreases towards 0, $\alpha \to 0$

Advantages and Disadvantages

- Usually faster than Rosenblatt's original algorithm
- For linearly separable sets, the algorithm converges, i.e., it finds the correct solution in a finite number of steps *(Rosenblatt, 1959)*
- The actual number of steps grows exponentially with the number of inputs
- A major issue arises if the pattern sets are not linearly separable $\rightarrow$ the algorithm diverges

Perceptron - Pocket Algorithm (Gallant, 1990)

Idea

- Uses the (iterative) Rosenblatt learning algorithm
- The best weight vector found so far is stored in a "pocket"
- If a better weight vector (i.e., one with a smaller error) is found, it is saved in the pocket

Advantages

- Even for non-linearly separable sets, the algorithm finds the best possible solution.
  - If the training set is finite and the components of the weight vector and input vectors are rational, it can be shown that the pocket algorithm converges to the optimal solution with probability 1 (Gallant, 1990).

Neural Networks 1 - Lecture 3: Artificial Neuron

Examples - 1. Demonstrations of Learning Algorithms in Python

rosenblatt_perceptron.ipynb

- Rosenblatt's learning algorithm - selected variants (iterative, batch, with learning parameter), Hebbian learning
- Visualization of data and decision boundary
- Examples: learning of simple logical functions:

Negated AND

x_1	x_2	d
-1	-1	+1
-1	+1	+1
+1	-1	+1
+1	+1	-1

Demonstration: How the perceptron learns using different algorithm variants and how the decision boundary shifts over time.

Examples - 1. Demonstrations of Learning Algorithms in Python

rosenblatt_perceptron.ipynb

- Rosenblatt's learning algorithm - selected variants (iterative, batch, with learning parameter), Hebbian learning
- Visualization of data and decision boundary
- Examples: learning of simple logical functions:

XOR (Exclusive OR)

x_1	x_2	$d = x_1 \otimes x_2$
-1	-1	-1
-1	+1	+1
+1	-1	+1
+1	+1	-1

Demonstration: How perceptron learning behaves when the data is not linearly separable.

Examples - 2. Various Practical Tasks

perceptron_examples.ipynb

- Learning simple logical functions
- Data with outliers
- Randomly generated data
- Letters
- Handwritten digits

Learning Simple Logical Functions

Example 1 – Gallbladder Attack

Salad	Pork belly	Medication	Will I feel sick?
+1	-1	-1	+1
+1	-1	+1	-1
+1	+1	-1	+1
-1	+1	+1	-1
+1	+1	-1	+1
+1	+1	+1	+1

- How long will the perceptron take to learn? Will it be able to perfectly represent the given logical function?
- Are there any differences between learning algorithms?

Learning Simple Logical Functions

Example 2 - The Pub (variation on a majority circuit)

Pavel	Pepa	Honza	Are we going for a beer?
+1	-1	-1	-1
+1	-1	+1	+1
-1	+1	-1	-1
-1	+1	+1	+1
+1	+1	-1	+1
+1	+1	+1	+1
-1	-1	-1	-1
-1	-1	+1	-1

- How long will the perceptron take to learn? Will it be able to perfectly represent the given logical function?
- Are there any differences between learning algorithms?

Examples - Various Practical Tasks

Comparison of Different Machine Learning Models, Specifically Variants of Rosenblatt's Learning Algorithm

- Typically, we are interested in error (how well the model learned the task) and efficiency (number of epochs / training time)
- Since Rosenblatt's algorithm is partially stochastic, the learning process and outcome can vary each time
- Therefore, it is better to repeat the experiment multiple times (e.g., 100 times) and compare average values (as well as variance)

For different tasks, a different model/algorithm variant may be the best.

Examples - Various Practical Tasks

Example 3 - Data with Outliers

- We have data where input vectors have different magnitudes (here, the 3rd sample is an outlier)

x_1	x_2	y
-0.5	-0.5	1
0.3	-0.5	1
-40	50	-1
-0.5	0.5	-1
-0.1	1.0	1

- How long will the perceptron take to learn? Will it learn the given task?
- How long will it take if we normalize all input vectors to length 1?

Examples - Various Practical Tasks

Example 4 - Randomly Generated Data

- When comparing different models or algorithms, it is useful to start with artificially, e.g., randomly, generated data
- In this case, the data is not perfectly linearly separable (random noise is added)
- Generate data multiple times and observe how perceptrons learn. Which variant of Rosenblatt's algorithm is best for this task?

Example 5 - Randomly Generated Clusters

- Next, we generate data containing two clusters of patterns
- Generate data multiple times and observe how perceptrons learn. Which variant of Rosenblatt's algorithm is best for this task?

Examples - Various Practical Tasks

Example 5 - Letters letters_example.ipynb

- We use the prepared dataset **letters.csv**
- Letters were segmented from **letters.png**
- Explore the dataset and visualize some letters
- Create a test set: data with added noise or subsequently smoothed
- Train the perceptron using different algorithms (and variants) to recognize individual letters
- Measure classification error on the training and test sets (also track number of epochs / training time)
- How much noise in the data can the perceptron handle?
- Identify which letters the perceptron struggled with the most
- Which learning algorithm performed the best?

Examples - Various Practical Tasks

Example 6 - Handwritten Digits

- We use the prepared dataset **OcrData.csv** containing handwritten digits.
- Explore the dataset and visualize some digits (use the provided script).
- Train a perceptron using different learning algorithms (and their variants) to recognize individual digits.
- Determine (and compare) the classification error on the training set (and optionally, the number of epochs/training time).
- Identify which digits the perceptron struggled with the most.
- Which learning algorithm performed the best?