

# Artificial Neural Networks in a Nutshell

## Lecture “CML: Control, Machine Learning, and Numerics”

---

Leon Bungert, Daniel Tenbrinck

May 31, 2022

Friedrich-Alexander-Universität Erlangen-Nürnberg

Summer semester 2022

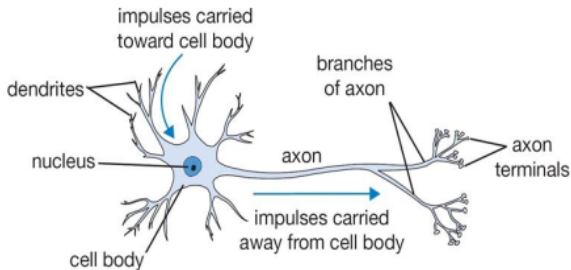
### **Acknowledgement:**

We thank Jonas Adler and Ozan Öktem from KTH Stockholm for allowing us to use parts of their lecture series on Deep Learning.

# Motivation

## Historic aim (McCulloch & Pitts, 1943):

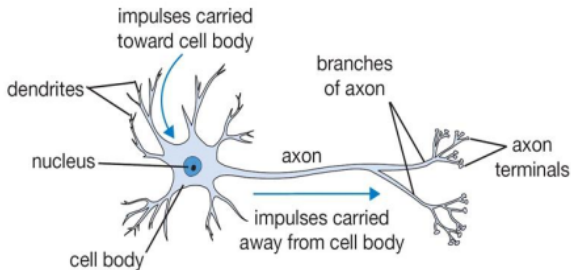
Mimic the biological processes of real neurons for Machine Learning.



# Motivation

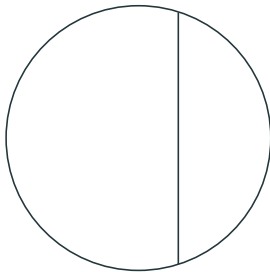
## Historic aim (McCulloch& Pitts, 1943):

Mimic the biological processes of real neurons for Machine Learning.



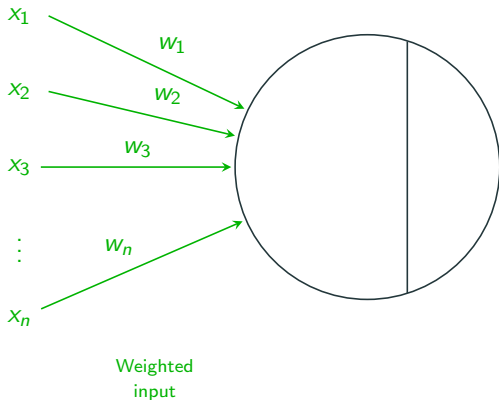
**Key observation:** Biological neurons transmit signals **only** if the **required activation energy** is reached by all incoming signals.

## (Simple) Perceptron - an artificial neuron



The simple perceptron (Rosenblatt, 1958) is an **artificial neuron** that is able to compute mathematical functions of the form:

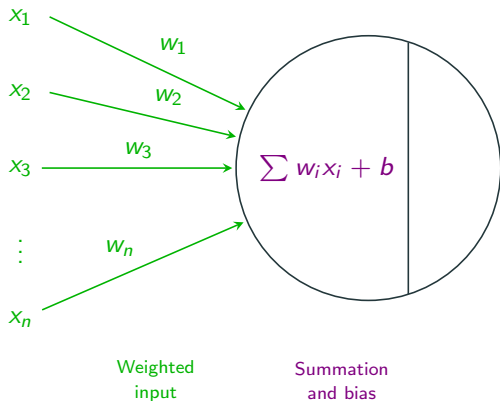
## (Simple) Perceptron - an artificial neuron



The simple perceptron (Rosenblatt, 1958) is an **artificial neuron** that is able to compute mathematical functions of the form:

$$w_i x_i$$

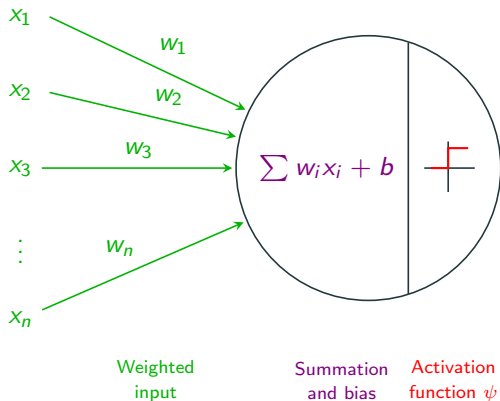
# (Simple) Perceptron - an artificial neuron



The simple perceptron (Rosenblatt, 1958) is an **artificial neuron** that is able to compute mathematical functions of the form:

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

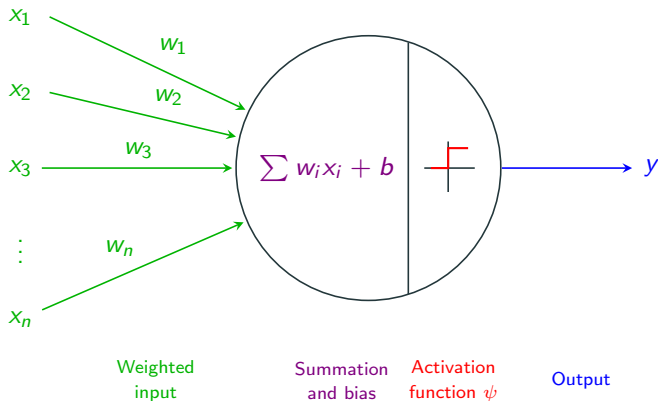
# (Simple) Perceptron - an artificial neuron



The simple perceptron (Rosenblatt, 1958) is an **artificial neuron** that is able to compute mathematical functions of the form:

$$\psi\left(\sum_{i=1}^n w_i x_i + b\right)$$

# (Simple) Perceptron - an artificial neuron

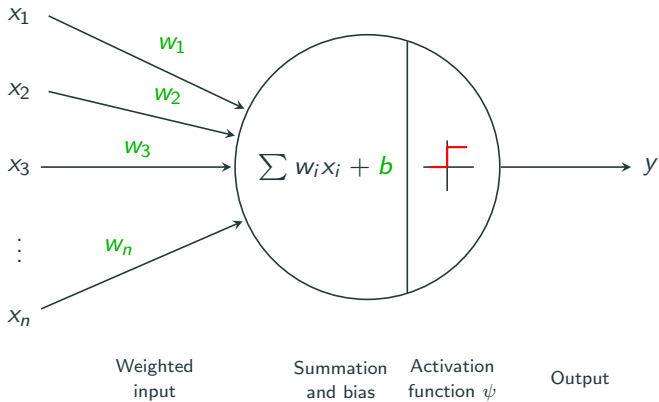


The simple perceptron (Rosenblatt, 1958) is an **artificial neuron** that is able to compute mathematical functions of the form:

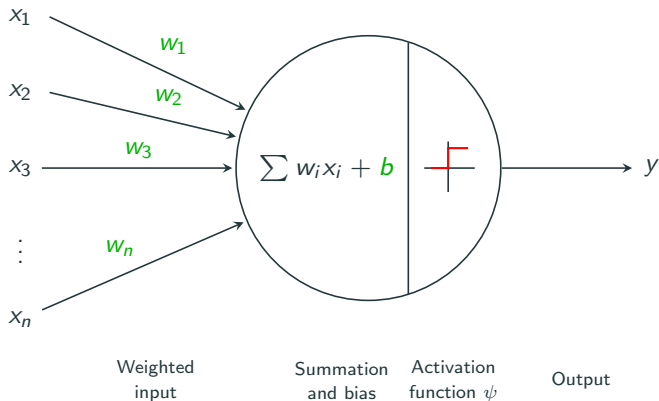
$$y = \psi\left(\sum_{i=1}^n w_i x_i + b\right)$$



# (Simple) Perceptron - an artificial neuron



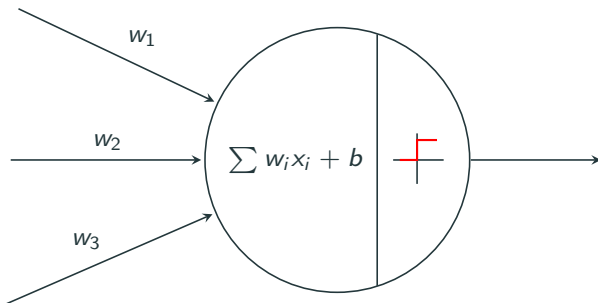
# (Simple) Perceptron - an artificial neuron



For a **fixed activation function**  $\psi: \mathbb{R} \rightarrow \mathbb{R}$  the behaviour of the perceptron is defined by the free parameters  $(w_1, \dots, w_n, b) = (\vec{w}, b) =: \theta \in \mathbb{R}^{n+1}$ . Thus, the perceptron realizes a parametrized map  $f_\theta: \mathbb{R}^n \rightarrow \mathbb{R}$  with  $f_\theta(\vec{x}) := f(\vec{x}; \theta) = f(x_1, \dots, x_n; \theta)$ .

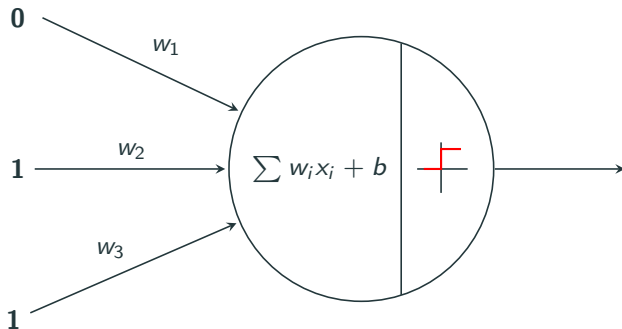
## An example perceptron

We analyze a perceptron with 3 fixed input signals  $(x_1, x_2, x_3) = (0, 1, 1)$ . We use the **Heavyside function**  $H: \mathbb{R} \rightarrow \{0, 1\}$  as activation function and set the free parameters as  $\theta = (w_1, w_2, w_3, b) = (1, 0, 1, -1)$ .



## An example perceptron

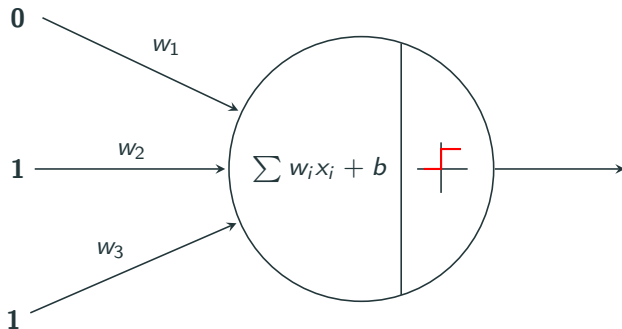
We analyze a perceptron with 3 fixed input signals  $(x_1, x_2, x_3) = (0, 1, 1)$ . We use the **Heavyside function**  $H: \mathbb{R} \rightarrow \{0, 1\}$  as activation function and set the free parameters as  $\theta = (w_1, w_2, w_3, b) = (1, 0, 1, -1)$ .



Thus, we get  $f_{\theta}(\vec{x}) = [0, 1, 1]^T$

# An example perceptron

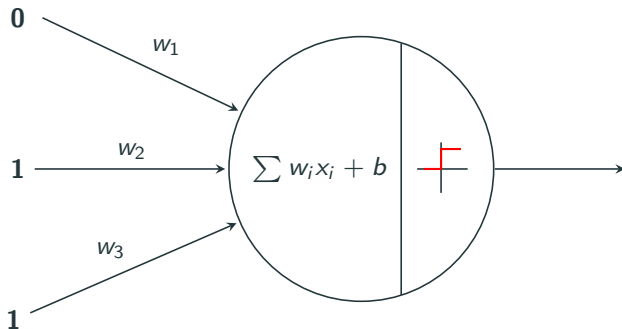
We analyze a perceptron with 3 fixed input signals  $(x_1, x_2, x_3) = (0, 1, 1)$ . We use the **Heavyside function**  $H: \mathbb{R} \rightarrow \{0, 1\}$  as activation function and set the free parameters as  $\theta = (w_1, w_2, w_3, b) = (1, 0, 1, -1)$ .



Thus, we get  $f_{\theta}(\vec{x}) = [1, 0, 1] \cdot [0, 1, 1]^T - 1$

## An example perceptron

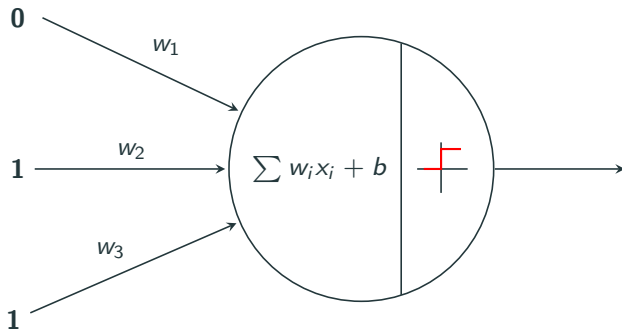
We analyze a perceptron with 3 fixed input signals  $(x_1, x_2, x_3) = (0, 1, 1)$ . We use the **Heavyside function**  $H: \mathbb{R} \rightarrow \{0, 1\}$  as activation function and set the free parameters as  $\theta = (w_1, w_2, w_3, b) = (1, 0, 1, -1)$ .



Thus, we get  $f_{\theta}(\vec{x}) = H([1, 0, 1] \cdot [0, 1, 1]^T - 1)$

# An example perceptron

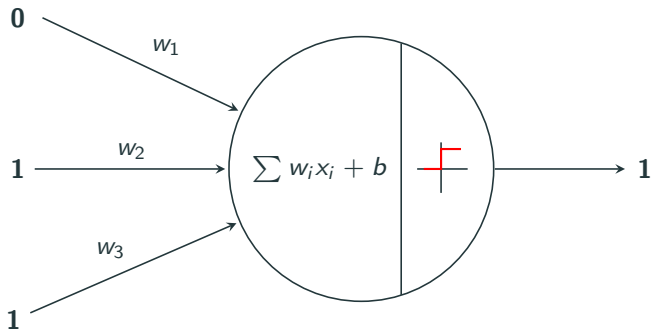
We analyze a perceptron with 3 fixed input signals  $(x_1, x_2, x_3) = (0, 1, 1)$ . We use the **Heavyside function**  $H: \mathbb{R} \rightarrow \{0, 1\}$  as activation function and set the free parameters as  $\theta = (w_1, w_2, w_3, b) = (1, 0, 1, -1)$ .



Thus, we get  $f_{\theta}(\vec{x}) = H([1, 0, 1] \cdot [0, 1, 1]^T - 1) = H(0)$

# An example perceptron

We analyze a perceptron with 3 fixed input signals  $(x_1, x_2, x_3) = (0, 1, 1)$ . We use the **Heavyside function**  $H: \mathbb{R} \rightarrow \{0, 1\}$  as activation function and set the free parameters as  $\theta = (w_1, w_2, w_3, b) = (1, 0, 1, -1)$ .

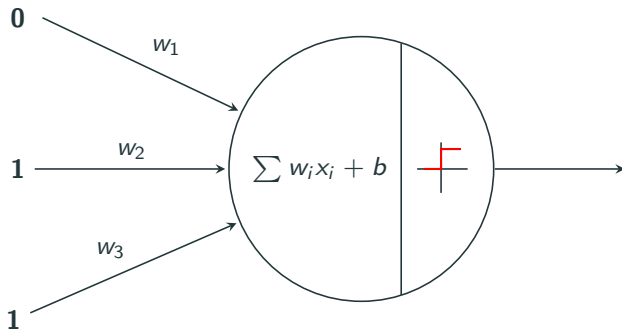


Thus, we get  $f_{\theta}(\vec{x}) = H([1, 0, 1] \cdot [0, 1, 1]^T - 1) = H(0) = 1$



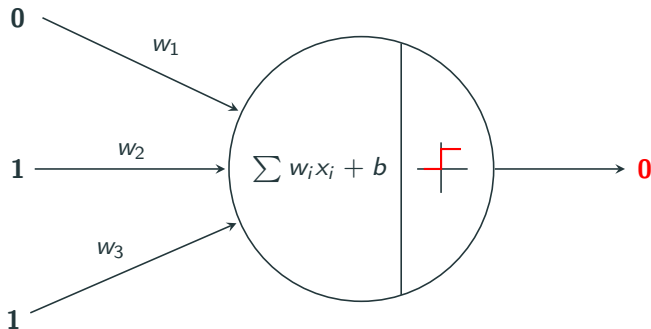
## An example perceptron

We analyze a perceptron with 3 fixed input signals  $(x_1, x_2, x_3) = (0, 1, 1)$ . We use the Heavyside function  $H: \mathbb{R} \rightarrow \{0, 1\}$  as activation function and set the free parameters as  $\theta = (w_1, w_2, w_3, b) = (1, 0.5, -1, 0)$ .



## An example perceptron

We analyze a perceptron with 3 fixed input signals  $(x_1, x_2, x_3) = (0, 1, 1)$ . We use the Heavyside function  $H: \mathbb{R} \rightarrow \{0, 1\}$  as activation function and set the free parameters as  $\theta = (w_1, w_2, w_3, b) = (1, 0.5, -1, 0)$ .

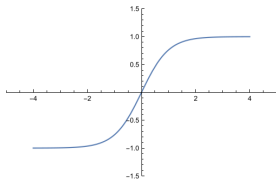


Thus, we get  $f_{\theta}(\vec{x}) = H([1, 0.5, -1] \cdot [0, 1, 1]^T + 0) = H(-0.5) = 0$

# Continuous activation functions

The following **continuous activation functions** are commonly used in artificial neurons due to their **nice analytic properties**:

## Tanh

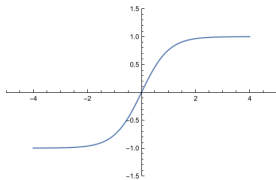


$$\psi(t) := \tanh(t)$$

# Continuous activation functions

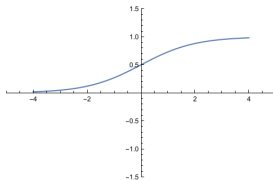
The following **continuous activation functions** are commonly used in artificial neurons due to their **nice analytic properties**:

**Tanh**



$$\psi(t) := \tanh(t)$$

**Logistic**

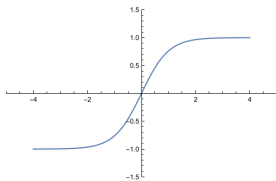


$$\psi(t) := \frac{1}{1 + e^{-t}}$$

# Continuous activation functions

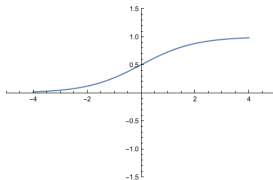
The following **continuous activation functions** are commonly used in artificial neurons due to their **nice analytic properties**:

**Tanh**



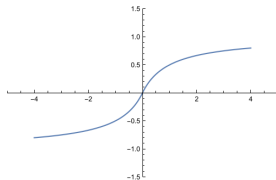
$$\psi(t) := \tanh(t)$$

**Logistic**



$$\psi(t) := \frac{1}{1 + e^{-t}}$$

**Softsign**



$$\psi(t) := \frac{t}{1 + |t|}$$

# Observations on the perceptron

## Observations:

- **weights**  $\vec{w} = (w_1, \dots, w_n)$  determine the **influence of the input**  
→ weight  $w_k = 0$  disregards respective input  $x_k$  completely

# Observations on the perceptron

## Observations:

- **weights**  $\vec{w} = (w_1, \dots, w_n)$  determine the **influence of the input**  
→ weight  $w_k = 0$  disregards respective input  $x_k$  completely
- **bias**  $b$  defines a **base probability for activation** of the artificial neuron  
→ bias  $b \ll 0$  makes an activation very unlikely  
→ bias  $b \gg 0$  makes an activation very likely

# Observations on the perceptron

## Observations:

- **weights**  $\vec{w} = (w_1, \dots, w_n)$  determine the **influence of the input**  
→ weight  $w_k = 0$  disregards respective input  $x_k$  completely
- **bias**  $b$  defines a **base probability for activation** of the artificial neuron  
→ bias  $b \ll 0$  makes an activation very unlikely  
→ bias  $b \gg 0$  makes an activation very likely
- simple perceptrons with Heavyside activation function realize **linear binary classifiers**

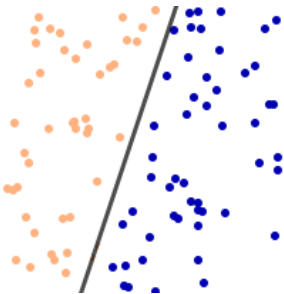


# Observations on the perceptron

## Observations:

- **weights**  $\vec{w} = (w_1, \dots, w_n)$  determine the **influence of the input**  
→ weight  $w_k = 0$  disregards respective input  $x_k$  completely
- **bias**  $b$  defines a **base probability for activation** of the artificial neuron  
→ bias  $b \ll 0$  makes an activation very unlikely  
→ bias  $b \gg 0$  makes an activation very likely
- simple perceptrons with Heavyside activation function realize **linear binary classifiers**  
→ for more complex applications a perceptron is **too restricted**

# Observations on the perceptron



Given a set of input data  $\{\vec{x}^{(1)}, \dots, \vec{x}^{(N)}\}$  with  $\vec{x}^{(i)} \in \mathbb{R}^n$ , the free parameters  $\theta \in \mathbb{R}^{n+1}$  induce a **hyperplane** that **linearly** separates the data in **two classes**.

$$f_{\theta}(\vec{x}^{(i)}) := \begin{cases} 1, & \text{if } \langle \vec{w}, \vec{x}^{(i)} \rangle + b > 0, \\ 0, & \text{otherwise.} \end{cases}$$

# Artificial neural networks

**Idea:** Combine multiple perceptrons to perform more complex tasks.

# Artificial neural networks

**Idea:** Combine **multiple perceptrons** to perform **more complex tasks**.

- align artificial neurons in consecutive layers
  - convention: use designated input layer and output layer
  - all intermediate layers are called **hidden layer**
  - number of layers is called **depth** of the neural network
  - number of nonzero weights is called **connectivity** of the neural network

# Artificial neural networks

**Idea:** Combine **multiple perceptrons** to perform **more complex tasks**.

- align artificial neurons in consecutive layers
  - convention: use designated input layer and output layer
  - all intermediate layers are called **hidden layer**
  - number of layers is called **depth** of the neural network
  - number of nonzero weights is called **connectivity** of the neural network
- artificial neural networks can be represented by directed graphs

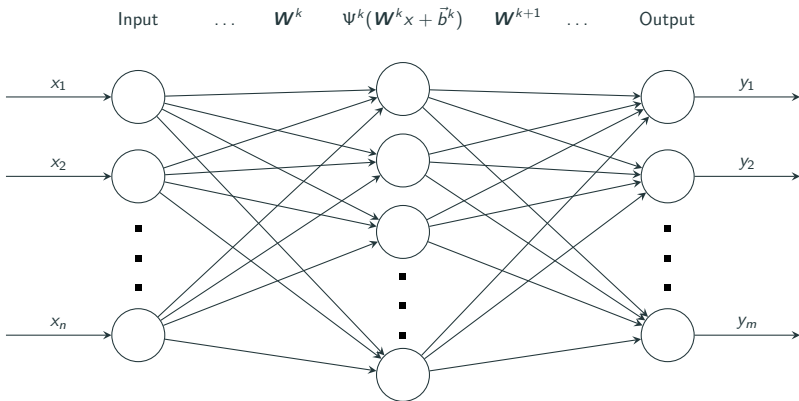
# Artificial neural networks

**Idea:** Combine **multiple perceptrons** to perform **more complex tasks**.

- align artificial neurons in consecutive layers
  - convention: use designated input layer and output layer
  - all intermediate layers are called **hidden layer**
  - number of layers is called **depth** of the neural network
  - number of nonzero weights is called **connectivity** of the neural network
- artificial neural networks can be represented by directed graphs
- connections between neurons can be (almost) **arbitrary**
  - often there are no connections within same layer (except in recurrent neural networks)
  - certain network structures have proved to be successful for different applications, e.g., convolutional neural networks

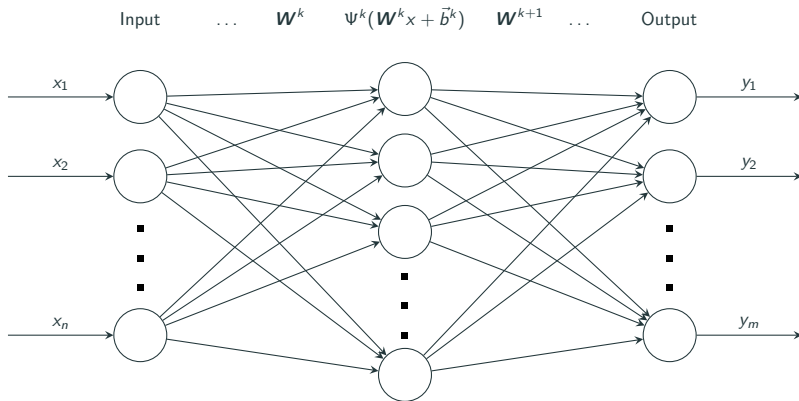
# Fully-connected feedforward neural network

**Classical representation:** Mappings from  $k$ th to  $(k + 1)$ st layer:



# Fully-connected feedforward neural network

**Classical representation:** Mappings from  $k$ th to  $(k + 1)$ st layer:



**Compact representation:**





# Fully-connected feedforward neural network

- a fully-connected feedforward neural network can be written as a parametrized map  $f_{\Theta}: \mathbb{R}^n \rightarrow \mathbb{R}^m$  that is realized by a concatenation of  $d \in \mathbb{N}$  perceptron layers via

$$f_{\Theta} := f_{\Theta_d}^d \circ \dots \circ f_{\Theta_1}^1$$

# Fully-connected feedforward neural network

- a fully-connected feedforward neural network can be written as a parametrized map  $f_{\Theta} : \mathbb{R}^n \rightarrow \mathbb{R}^m$  that is realized by a concatenation of  $d \in \mathbb{N}$  perceptron layers via

$$f_{\Theta} := f_{\Theta_d}^d \circ \dots \circ f_{\Theta_1}^1$$

- each layer is a map  $f_{\Theta_k}^k : \mathbb{R}^{n_{k-1}} \rightarrow \mathbb{R}^{n_k}$  with  $f_{\Theta_k}^k(x) = \Psi^k(\mathbf{W}^k x + \vec{b}^k)$

# Fully-connected feedforward neural network

- a fully-connected feedforward neural network can be written as a parametrized map  $f_{\Theta}: \mathbb{R}^n \rightarrow \mathbb{R}^m$  that is realized by a concatenation of  $d \in \mathbb{N}$  perceptron layers via

$$f_{\Theta} := f_{\Theta_d}^d \circ \dots \circ f_{\Theta_1}^1$$

- each layer is a map  $f_{\Theta_k}^k: \mathbb{R}^{n_{k-1}} \rightarrow \mathbb{R}^{n_k}$  with  $f_{\Theta_k}^k(x) = \psi^k(\mathbf{W}^k x + \vec{b}^k)$
- the free parameters can be written as matrix  $\Theta_k = (\mathbf{W}^k, \vec{b}^k)$  with weights  $\mathbf{W}^k \in \mathbb{R}^{n_k \times n_{k-1}}$  and biases  $\vec{b}^k \in \mathbb{R}^{n_k}$

# Fully-connected feedforward neural network

- a fully-connected feedforward neural network can be written as a parametrized map  $f_{\Theta} : \mathbb{R}^n \rightarrow \mathbb{R}^m$  that is realized by a concatenation of  $d \in \mathbb{N}$  perceptron layers via

$$f_{\Theta} := f_{\Theta_d}^d \circ \dots \circ f_{\Theta_1}^1$$

- each layer is a map  $f_{\Theta_k}^k : \mathbb{R}^{n_{k-1}} \rightarrow \mathbb{R}^{n_k}$  with  $f_{\Theta_k}^k(x) = \Psi^k(\mathbf{W}^k x + \vec{b}^k)$
- the free parameters can be written as matrix  $\Theta_k = (\mathbf{W}^k, \vec{b}^k)$  with weights  $\mathbf{W}^k \in \mathbb{R}^{n_k \times n_{k-1}}$  and biases  $\vec{b}^k \in \mathbb{R}^{n_k}$
- the activation function  $\Psi^k$  acts pointwise on the resulting vector of the affine linear map, i.e.,  $\Psi^k(x_1, \dots, x_{n_k}) := (\psi^k(x_1), \dots, \psi^k(x_{n_k}))$  where  $\psi^k : \mathbb{R} \rightarrow \mathbb{R}$  is the chosen activation function for this layer

# Fully-connected feedforward neural network

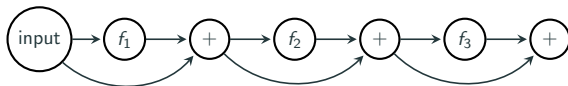
- a fully-connected feedforward neural network can be written as a parametrized map  $f_{\Theta} : \mathbb{R}^n \rightarrow \mathbb{R}^m$  that is realized by a concatenation of  $d \in \mathbb{N}$  perceptron layers via

$$f_{\Theta} := f_{\Theta_d}^d \circ \dots \circ f_{\Theta_1}^1$$

- each layer is a map  $f_{\Theta_k}^k : \mathbb{R}^{n_{k-1}} \rightarrow \mathbb{R}^{n_k}$  with  $f_{\Theta_k}^k(x) = \Psi^k(\mathbf{W}^k x + \vec{b}^k)$
- the free parameters can be written as matrix  $\Theta_k = (\mathbf{W}^k, \vec{b}^k)$  with weights  $\mathbf{W}^k \in \mathbb{R}^{n_k \times n_{k-1}}$  and biases  $\vec{b}^k \in \mathbb{R}^{n_k}$
- the activation function  $\Psi^k$  acts pointwise on the resulting vector of the affine linear map, i.e.,  $\Psi^k(x_1, \dots, x_{n_k}) := (\psi^k(x_1), \dots, \psi^k(x_{n_k}))$  where  $\psi^k : \mathbb{R} \rightarrow \mathbb{R}$  is the chosen activation function for this layer
- the network is **fully-connected** if each weight matrix  $\mathbf{W}^k$  is dense

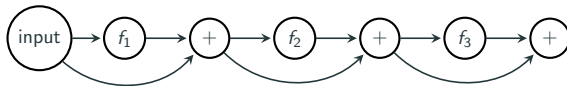
# Non-sequential artificial neural networks

- **Residual network:** Popular architecture involving *residual connections*. Can be interpreted as **forward Euler method**.

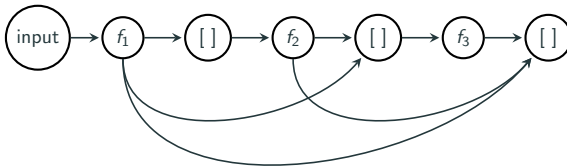


# Non-sequential artificial neural networks

- **Residual network:** Popular architecture involving *residual connections*. Can be interpreted as **forward Euler method**.

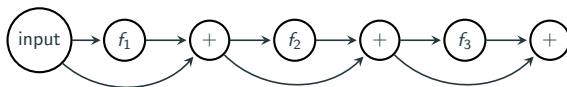


- **Concatenation:** Result from all previous layers are concatenated ([ ] indicates concatenation) to form the input to the next layer

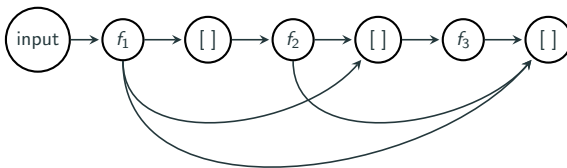


# Non-sequential artificial neural networks

- **Residual network:** Popular architecture involving *residual connections*. Can be interpreted as **forward Euler method**.



- **Concatenation:** Result from all previous layers are concatenated ([ ] indicates concatenation) to form the input to the next layer



- **Sparse network:** Network architectures where most weights are zero, i.e., the connectivity is small relative to the number of connections possible.



# Convolutional neural networks (CNNs)

**Idea:** Encode the **geometry** of data (e.g., proximity, directions) in network structure.

# Convolutional neural networks (CNNs)

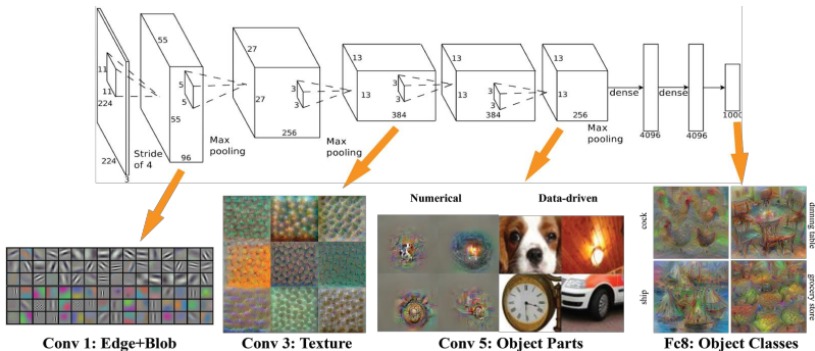
**Idea:** Encode the **geometry** of data (e.g., proximity, directions) in network structure.

- especially suitable for **images, volumes, graphs**
- easily parallelizable

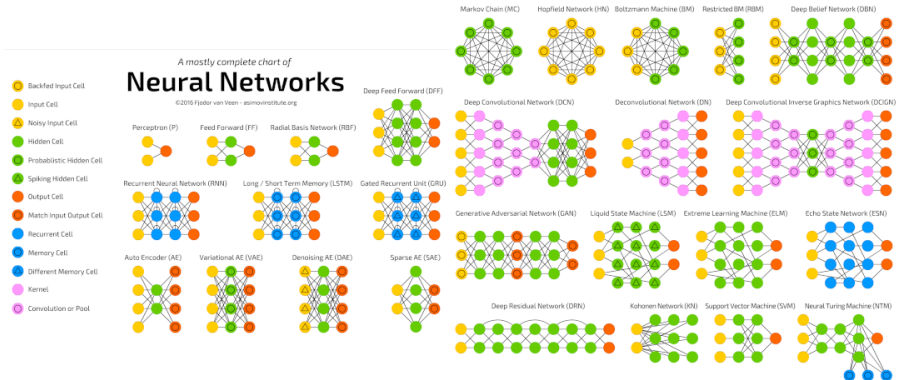
# Convolutional neural networks (CNNs)

**Idea:** Encode the **geometry** of data (e.g., proximity, directions) in network structure.

- especially suitable for **images, volumes, graphs**
- easily parallelizable



# Zoo of architectures



# Open question

## Machine learning task:

Given pairs of input/output data  $(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})$ . How can we build an artificial neural network  $f_{\Theta}$  such that

$$f_{\Theta}(x^{(k)}) \approx y^{(k)}, \quad k = 1, \dots, N.$$

# Open question

## Machine learning task:

Given pairs of input/output data  $(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})$ . How can we build an artificial neural network  $f_{\Theta}$  such that

$$f_{\Theta}(x^{(k)}) \approx y^{(k)}, \quad k = 1, \dots, N.$$

## Example:

Imagine an artificial neural network with an input layer (10 neurons), 5 hidden layers (10 neurons each), and a single output neuron. This leads to  $10 * 10 + 4 * (10 * 10) + 10 = 510$  free parameters for the *weights* and 51 free parameters for the *biases*.

# Open question

## Machine learning task:

Given pairs of input/output data  $(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})$ . How can we build an artificial neural network  $f_{\Theta}$  such that

$$f_{\Theta}(x^{(k)}) \approx y^{(k)}, \quad k = 1, \dots, N.$$

## Example:

Imagine an artificial neural network with an input layer (10 neurons), 5 hidden layers (10 neurons each), and a single output neuron. This leads to  $10 * 10 + 4 * (10 * 10) + 10 = 510$  free parameters for the *weights* and 51 free parameters for the *biases*.

→ Setting the free parameters  $\Theta$  of a network manually is **not feasible!**

# Open question

## Machine learning task:

Given pairs of input/output data  $(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})$ . How can we build an artificial neural network  $f_{\Theta}$  such that

$$f_{\Theta}(x^{(k)}) \approx y^{(k)}, \quad k = 1, \dots, N.$$

## Example:

Imagine an artificial neural network with an input layer (10 neurons), 5 hidden layers (10 neurons each), and a single output neuron. This leads to  $10 * 10 + 4 * (10 * 10) + 10 = 510$  free parameters for the *weights* and 51 free parameters for the *biases*.

→ Setting the free parameters  $\Theta$  of a network manually is **not feasible!**

**Solution:** Obtain good parameters by **training** the neural network!



# Conclusions

- Artificial neurons were designed to mimic biological processes

# Conclusions

- Artificial neurons were designed to mimic biological processes
- Perceptrons realize **linear classifiers**

# Conclusions

- Artificial neurons were designed to mimic biological processes
- Perceptrons realize linear classifiers
- combining multiple layers of artificial neurons allows to solve more complex tasks

# Conclusions

- Artificial neurons were designed to mimic biological processes
- Perceptrons realize linear classifiers
- combining multiple layers of artificial neurons allows to solve more complex tasks
- some network architectures are well-suited for certain applications

# Conclusions

- Artificial neurons were designed to mimic biological processes
- Perceptrons realize linear classifiers
- combining multiple layers of artificial neurons allows to solve more complex tasks
- some network architectures are well-suited for certain applications
- manually choosing the free parameters of a network is infeasible

# Conclusions

- Artificial neurons were designed to mimic biological processes
- Perceptrons realize linear classifiers
- combining multiple layers of artificial neurons allows to solve more complex tasks
- some network architectures are well-suited for certain applications
- manually choosing the free parameters of a network is infeasible

**Thank you for your attention!**