# Artificial Neural Networks

Máster Universitario en Ciencia de Datos - Métodos Avanzados en Aprendizaje Automático

Carlos María Alaíz Gudín

Escuela Politécnica Superior
Universidad Autónoma de Madrid

Academic Year 2021/22


Universidad Autónoma
de Madrid

# Contents

# Introduction

# ANN Origins

- **Artificial Neural Networks** (ANNs) are a family of machine learning models that "mimic" the brain.
- They are the state-of-the-art in several machine learning problems.

---

- Hypothesis: the brain does many different things with just a single algorithm.

  Scientific Challenge Figure out how the brain does it.

  Machine Learning Challenge Find an approximation to implement this generic algorithm.

---

- ANNs are good machine learning models, not realistic brain simulators.
- Biology is used as an **inspiration**.

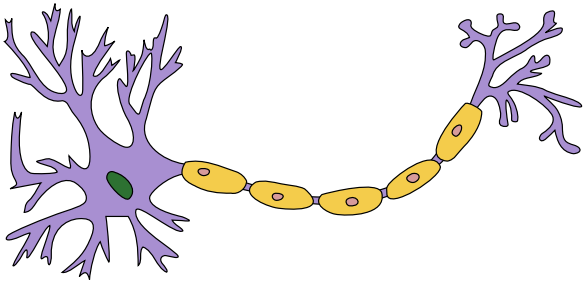
# ANN Historical Overview

- ANNs were widely used in the 80s and 90s.
- They decreased in popularity in late 90s.
  - They are computationally intensive.
  - Kernel methods are a good alternative.

---

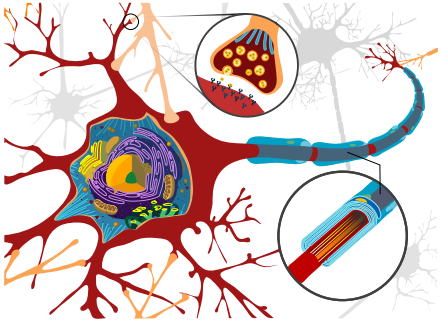
- They resurged recently.
  - Mainly because of **increasing computational power**.
  - Some "heuristics" allow the apparition of practical **Deep Neural Networks**.
- They are the state-of-the-art technique in several machine learning problems.

# Structure of the Neuron



- A neuron collects signals from other neurons through a set of fine structures called **dendrites**.
- These signals release neurotransmitters, which cause changes in **soma** voltages.
- Electric spikes travel though a long, thin swelling (the **axon**), which splits into thousands of branches.

# Communication among Neurons



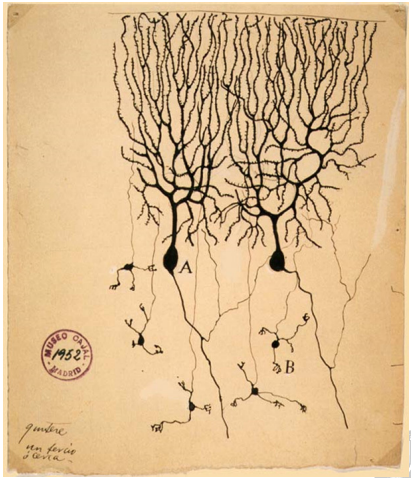
1. The action potential arrives to the axon terminal of the presynaptic neuron, causing the influx of ions.
2. This ion influx initiates the release of neurotransmitters that transport the signal across the synaptic cleft to the postsynaptic neuron.
3. Neurotransmitters can be either inhibitory or excitatory, and either block or stimulate the generation of an action potential in the postsynaptic neuron.
4. The response of the postsynaptic neuron depends on which signals (inhibitory or excitatory) dominate.

# The Brain

- The **brain** is basically a connected network of neurons that communicate by means of electric and chemical signals.
  - $\sim 10^{11}$ neurons.
  - $\sim 1000$ synapses per neuron.

---

- The brain is basically a huge network of relatively simple units.
- **Is it possible to emulate it?**



# Stylized Facts about Biological Neurons

- A neuron receives **impulses** (both **excitatory** and **inhibitory**) of different strength from many neurons.
- The neuron then **integrates (sums) the impulses** received over space and time.
- If the resulting integrated signal is **above a threshold**, the neuron generates an action potential ("**fires**").
- The response of the neuron is of the type "**all-or-nothing**".
- The action potential, generated at the axon hillock, is **transported along the axon** until the axon terminal.
- The signal is then **transmitted** to the following neuron through a **synapse** by means of neurotransmitters.
- The strength of the synapse can change (**learning**).

# Ideas for Machine Learning

- Importance of **connectionism**: units with simple procession that interact in a complex network.
- **Distributed** representation of knowledge.
- **Parallelism**.
- **Thresholding mechanism** for robust classification.
- A **mechanism for learning**: learn by adjusting the synaptic weights.

# The Perceptron

# McCulloch–Pitts Neuron (I)

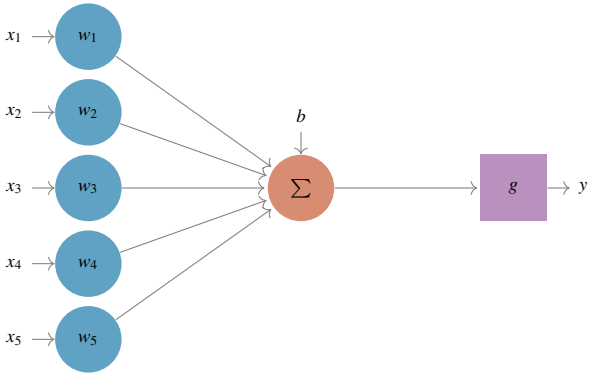
- The first step is to approximate the "simplest" unit: a **single neuron**.
- The first artificial neuron was the Threshold Logic Unit (TLU): the **McCulloch–Pitts neuron**.

---

- The elements are approximated as:

| Biological Neuron | Artificial Neuron |
|---|---|
| Dendrite | Input signal |
| Synapse | Weight |
| Soma | Summation |
| Spike | Activation function |
| Axon | Output |

# McCulloch–Pitts Neuron (II)

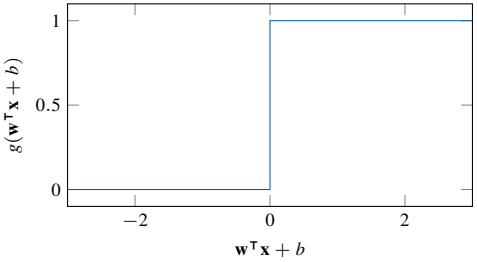


# McCulloch–Pitts Neuron (III)

- The activation function $g$ is the **Heaviside step function**.
- The output is:

$$y = g\left(\sum_{i=1}^{d} w_i x_i + b\right) = \begin{cases} 0 & \text{if } \sum_{i=1}^{d} w_i x_i + b < 0, \\ 1 & \text{if } \sum_{i=1}^{d} w_i x_i + b \geq 0. \end{cases}$$

- The output is activated (spike) when the combination of inputs (summing junction) is above a threshold.
- The resultant model resembles a binary classification linear model.



# McCulloch–Pitts Neuron - Exercise (I)

## Exercise

Given the TLU with Heaviside activation function and weights $\{b = 1, w_1 = 1, w_2 = 2\}$, and given this dataset:

| $x_{i,1}$ | $x_{i,2}$ |
|---|---|
| $-1$ | $-1$ |
| $-2$ | $1$ |
| $1$ | $0$ |

1. Compute the output for each pattern.

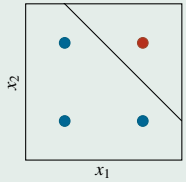
## Solution

1. The outputs are:
   - $g(\mathbf{w}^\intercal \mathbf{x}_1 + b) = g(-2) = 0.$
   - $g(\mathbf{w}^\intercal \mathbf{x}_2 + b) = g(1) = 1.$
   - $g(\mathbf{w}^\intercal \mathbf{x}_3 + b) = g(2) = 1.$

# McCulloch–Pitts Neuron: Examples

## Example (AND/OR)

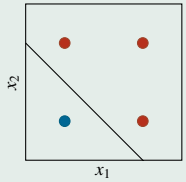
| $x_{i,1}$ | $x_{i,2}$ | $y_i$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |




$$w_1^\star = 1,$$
$$w_2^\star = 1,$$
$$b^\star = -1.5.$$

| $\mathbf{w}^\intercal \mathbf{x}_i + b$ | $g(\mathbf{w}^\intercal \mathbf{x}_i + b)$ |
|---|---|
| $-1.5$ | 0 |
| $-0.5$ | 0 |
| $-0.5$ | 0 |
| 0.5 | 1 |

| $x_{i,1}$ | $x_{i,2}$ | $y_i$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |




$$w_1^\star = 1,$$
$$w_2^\star = 1,$$
$$b^\star = -0.5.$$

| $\mathbf{w}^\intercal \mathbf{x}_i + b$ | $g(\mathbf{w}^\intercal \mathbf{x}_i + b)$ |
|---|---|
| $-0.5$ | 0 |
| 0.5 | 1 |
| 0.5 | 1 |
| 1.5 | 1 |

# McCulloch–Pitts Neuron: Possibilities

- This model can solve linearly separable problems such as AND and OR gates.
- By combining AND and OR gates any logical function can be devised, no matter how complex.
- A network of McCulloch–Pitts neurons should be able to compute anything.

# McCulloch–Pitts Neuron - Exercise (II)

## Exercise

Given the NOT AND (NAND) dataset:

| $x_{i,1}$ | $x_{i,2}$ | $y_i$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

1. Compute the weights $\{b, w_1, w_2\}$ of a TLU that solves this problem.

## Solution

1. The NOT AND is the negation of the AND problem, so it is enough to invert the weights of the AND problem:

$$\{b = 1.5, w_1 = -1, w_2 = -1\}.$$

| $\mathbf{w}^\mathsf{T}\mathbf{x}_i + b$ | $g(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b)$ |
|---|---|
| 1.5 | 1 |
| 0.5 | 1 |
| 0.5 | 1 |
| −0.5 | 0 |

# Rosenblatt Perceptron

- The difference between the circuit gates and the neurons is that **neurons can learn**.
- Rosenblatt invented an algorithm that automatically learns the weights of a McCulloch–Pitts neuron to solve a given binary classification problem. This learning machine is called **Rosenblatt Perceptron**.

## Rosenblatt Perceptron: Algorithm

**Require:** $\mathcal{D} = \{(\mathbf{x}_i, t_i)\}_{i=1}^{N}, \eta > 0$

**Ensure:** $\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$

Set $\mathbf{w} \sim \mathcal{U}[-0.5, 0.5]^d$, $b \sim \mathcal{U}[-0.5, 0.5]$ ▷ Random initial weights and bias.

**while** convergence criteria not met **do** ▷ Loop through epochs.

**for** $i = 1, \cdots, N$ **do** ▷ Loop through instances.

$\delta_i \leftarrow g(\mathbf{w}^\intercal \mathbf{x}_i + b) - t_i$ ▷ Signed error.

$\mathbf{w} \leftarrow \mathbf{w} - \eta \delta_i \mathbf{x}_i$ ▷ Update weights if there is an error.

$b \leftarrow b - \eta \delta_i$ ▷ Update bias if there is an error.

**return** $\mathbf{w}, b$

Notebook

Perceptron: AND Gate
OR Gate

# Perceptron: Limitations

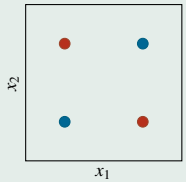
- The algorithm is guaranteed to finish if the problem is linearly separable (Novikoff).

---

- The model has all the limitations of a linear model.
- If the problem is not linearly separable, the algorithm will not converge.

## Example (XOR)

| $x_{i,1}$ | $x_{i,2}$ | $y_i$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




$$w_1^\star = ?,$$
$$w_2^\star = ?,$$
$$b^\star = ?.$$

| $\mathbf{w}^\intercal \mathbf{x}_i + b$ | $g(\mathbf{w}^\intercal \mathbf{x}_i + b)$ |
|---|---|
| ? | 0 |
| ? | 1 |
| ? | 1 |
| ? | 0 |

## Notebook

Perceptron: XOR Gate
Moons

# Neural Networks

# Stacking Perceptrons

- Following their biological inspiration, stacking perceptrons should increase their computational power.

---

- The intermediate perceptrons should build a good representation of the data.
- A mapping of the original data is done, but it is not fixed beforehand.
  - Adaptive basis function.

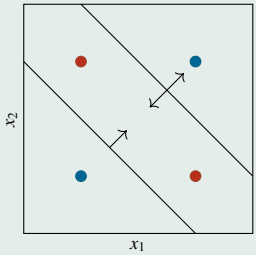
---

- The last perceptron should build a linear model over the new representation.

# Stacking Perceptrons: Example (I)

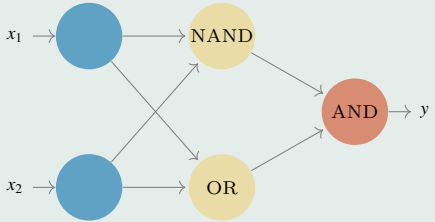
## Example (XOR)

| $x_{i,1}$ | $x_{i,2}$ | $y_i$ | $x_{i,1}$ AND $x_{i,2}$ | $x_{i,1}$ NAND $x_{i,2}$ ($\bar{y}_i$) | $x_{i,1}$ OR $x_{i,2}$ ($\bar{\bar{y}}_i$) | $\bar{y}_i$ AND $\bar{\bar{y}}_i$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 |



# Stacking Perceptrons: Example (II)

## Example (XOR)



Notebook

Multilayer Perceptron: Stacking Perceptrons

# Feedforward Neural Networks (I)

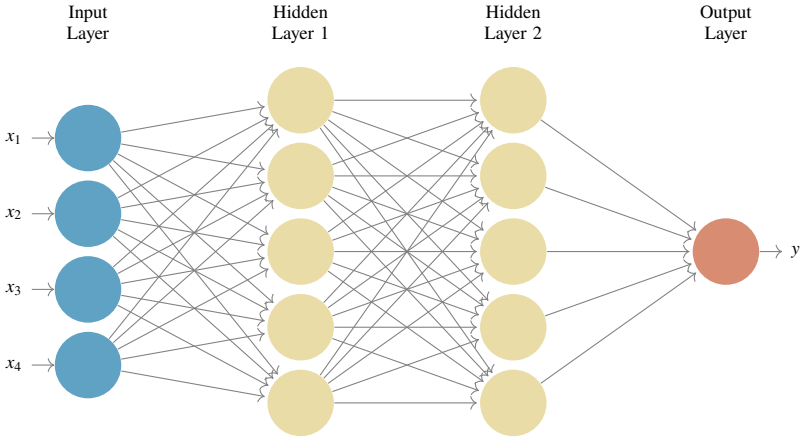
## Definition (Feedforward Neural Network)

- A **Neural Network** (NN) is a structured network of neurons.
- A **Feedforward Neural Network** (FFNN) is a NN with one or more hidden layers and without loops. In practice, **MultiLayer Perceptron** (MLP) denotes a FFNN with usually one hidden layer.
- The network receives a vector $\mathbf{x} \in \mathbb{R}^d$ in the form of an input signal through the input nodes.
- The input is processed by the network, which eventually provides an output.
- The prediction is a readout of the output of a neuron or a group of neurons in the network.

## Definition (Neuron)

- A **neuron** is a set of linear synaptic links, each of which channels an incoming signal (plus a bias).
- The synaptic links weight the corresponding incoming signals.
- The weighted sum of the incoming signals defines the induced **local field**.
- The neuron outputs a non-linear transformation of the local field by means of an **activation function**.
- The output can be used as input of another neuron or as output of the network as a whole.

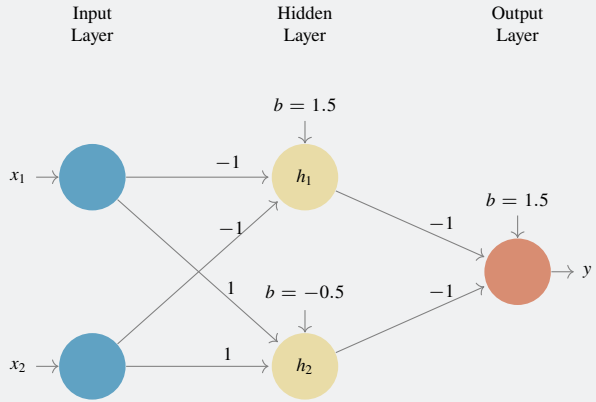
# Feedforward Neural Networks (II)



# Feedforward Neural Networks - Exercise (I)

## Exercise

Given this FFNN, where the activation functions (for both the hidden and the output layer) are Heaviside.



# Feedforward Neural Networks - Exercise (II)

## Exercise

1. Complete this table with the outputs of the hidden units $h_1$ and $h_2$, and with the output of the NN $y$, for the four samples.

| $x_{i,1}$ | $x_{i,2}$ | $h_{i,1}$ | $h_{i,2}$ | $y_i$ |
|---|---|---|---|---|
| 0 | 0 | | | |
| 0 | 1 | | | |
| 1 | 0 | | | |
| 1 | 1 | | | |

## Solution

1. The resultant outputs are:

| $h_{i,1}$ | $h_{i,2}$ | $y_i$ |
|---|---|---|
| $g(+1.5) = 1$ | $g(-0.5) = 0$ | $g(+0.5) = 1$ |
| $g(+0.5) = 1$ | $g(+0.5) = 1$ | $g(-0.5) = 0$ |
| $g(+0.5) = 1$ | $g(+0.5) = 1$ | $g(-0.5) = 0$ |
| $g(-0.5) = 0$ | $g(+1.5) = 1$ | $g(+0.5) = 1$ |

# Feedforward Neural Networks: Formalisation (I)

- $H$ hidden layers, $d_\ell$ units on the layer $\ell$.

---

- Local field of unit $i$ at layer $\ell$:

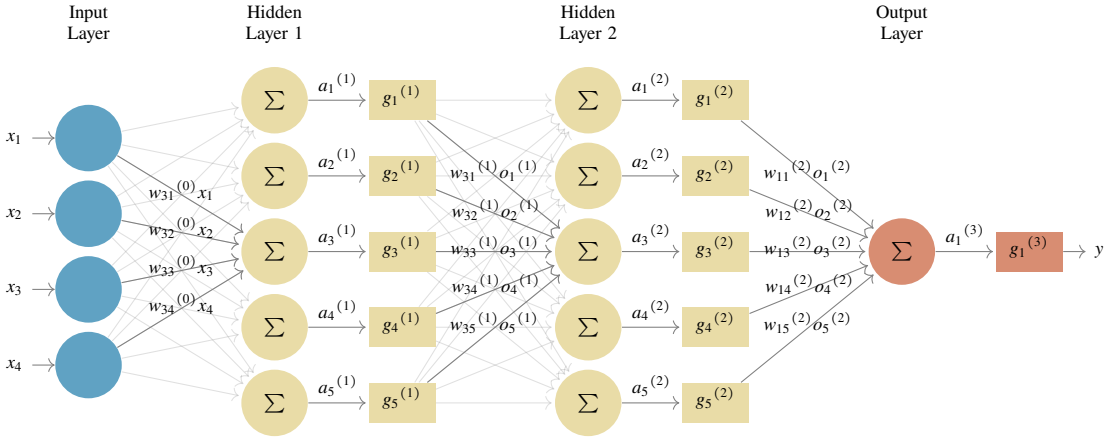
$$a_i^{(\ell)} = \sum_{j=1}^{d_{\ell-1}} w_{ij}^{(\ell-1)} o_j^{(\ell-1)} + b_i^{(\ell-1)}.$$

- Activation function of unit $i$ at layer $\ell$: $g_i^{(\ell)}$.
- Activation (output) of unit $i$ at layer $\ell$: $o_i^{(\ell)} = g_i^{(\ell)}(a_i^{(\ell)})$.

---

- Input layer, $d_0 = d$, $o_i^{(0)} = a_i^{(0)} = x_i$.

# Feedforward Neural Networks: Formalisation (II)



# Universal Approximation Property

**Theorem (Universal Approximation Property)**

*A feed-forward network with a single hidden layer containing a sufficiently large number of hidden neurons can uniformly approximate any continuous function on compact subsets of $\mathbb{R}^d$, under mild conditions on the activation function.*

- A neural network can be used to make **optimal predictions**.
- Finding such a network is not easy.

# Feedforward Neural Networks: Defining the Network

- The network is defined by:

  **Architecture** Number of hidden layers $H$ and number of units at each layer, $d_\ell$.

  **Activation Functions** ${g_i}^{(\ell)}$, $1 \leq i \leq d_\ell$, $0 \leq \ell \leq H+1$.

  **Set of parameters** $\boldsymbol{\theta} = \{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(H)}; \mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \ldots, \mathbf{b}^{(H+1)}\}$, with $\mathbf{W}^{(\ell)} \in \mathbb{R}^{d_{\ell+1} \times d_\ell}$ and $\mathbf{b}^{(\ell)} \in \mathbb{R}^{d_\ell}$.

---

- How are these parameters selected?

  **Architecture** $H$ and $d_\ell$ are hyper-parameters of the model.
  - Validation.
  - Cross-validation.
  - Expert knowledge.

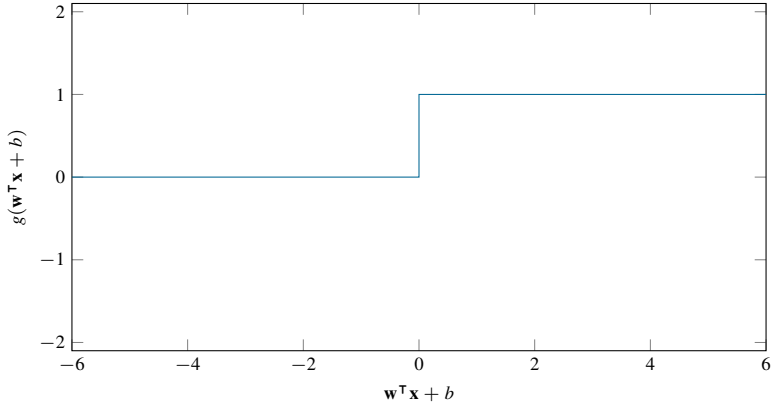
  **Activation Functions** The activation functions have to be fixed, according to:
  - Nature of the problem.
  - Efficiency (e.g. sparsity).
  - Convenience.

  **Set of parameters** They have to be learnt for each problem.

# Activation Functions (I)

## Heaviside

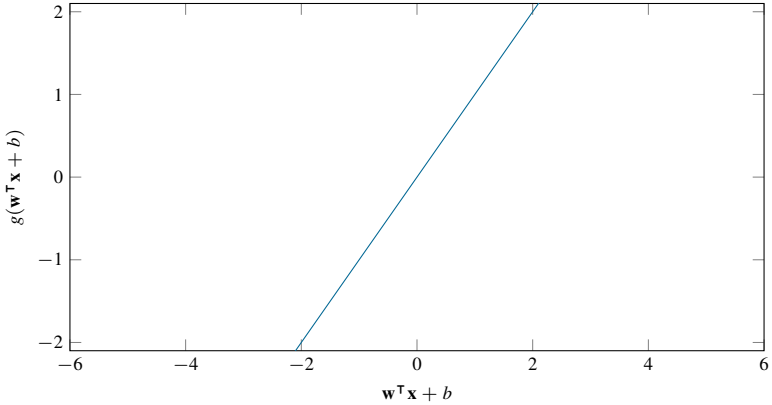
- Used in the perpceptron.
- The discontinuity affects the optimization.



# Activation Functions (II)

## Identity

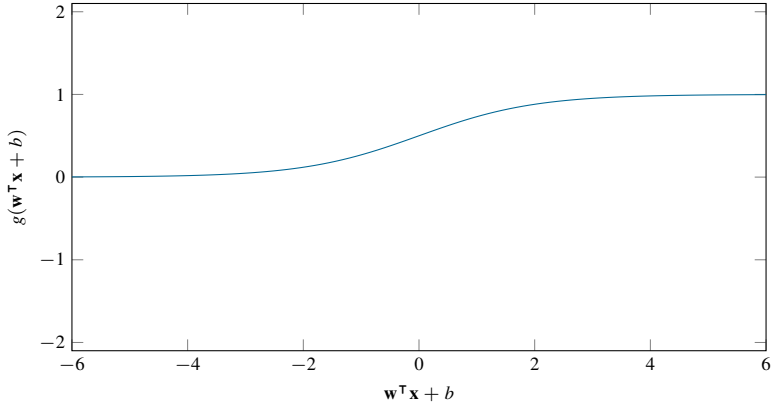
- Used in input neurons.
- In output neurons allows regression.



# Activation Functions (III)

## Logistic (sigmoid)

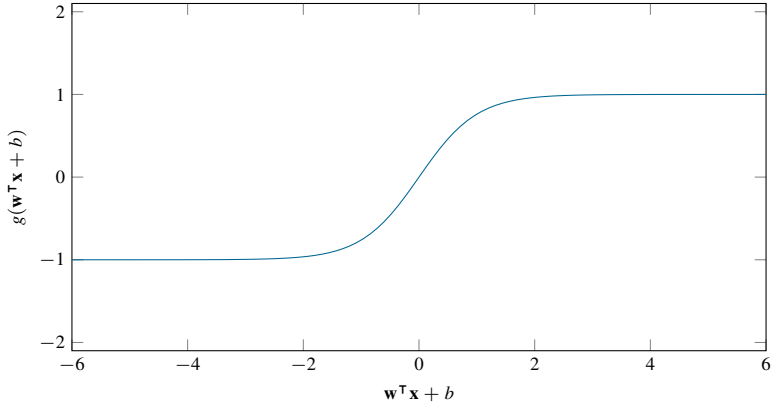
- Commonly used in Multiplayer Perceptrons.
- Also used in logistic regression.



# Activation Functions (IV)

## Hyperbolic Tangent (sigmoid)

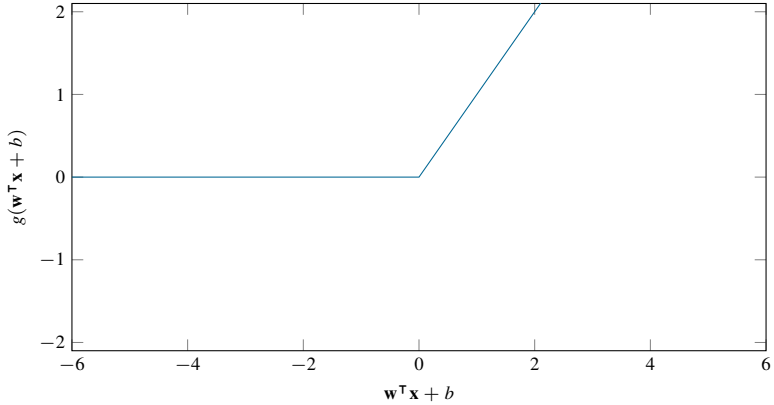
- Alternative to logistic sigmoid.
- Derivative can be expressed in terms of the function itself.



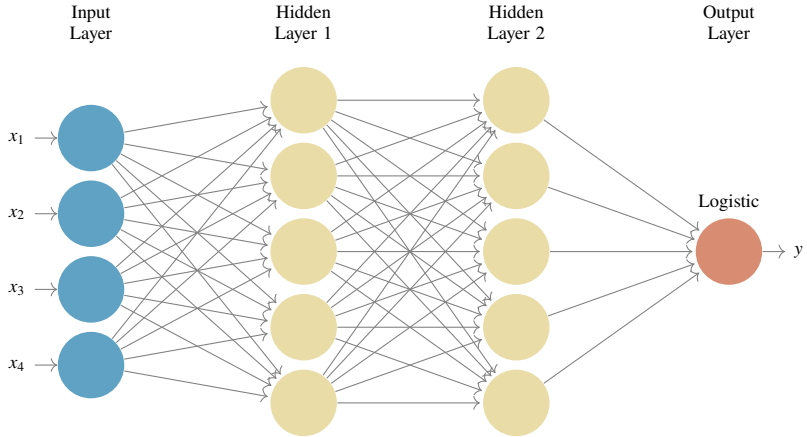
# Activation Functions (V)

## Rectified Linear (ReLU)

- Most popular for Deep Learning.
- Continuous but non-differentiable.

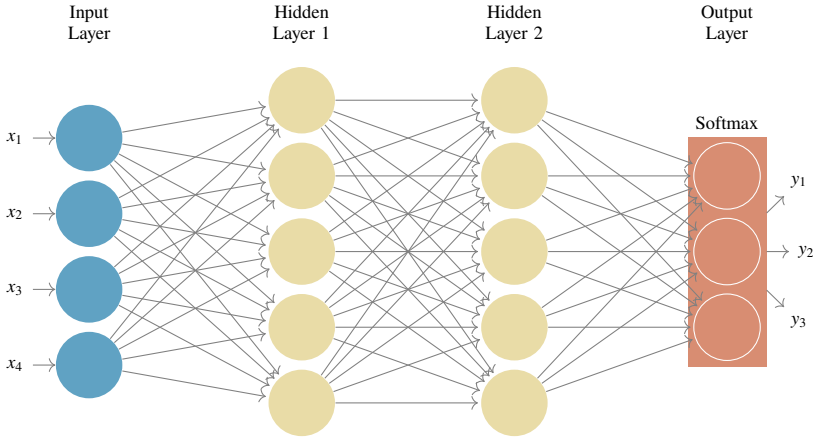


# Architecture for Binary Classification



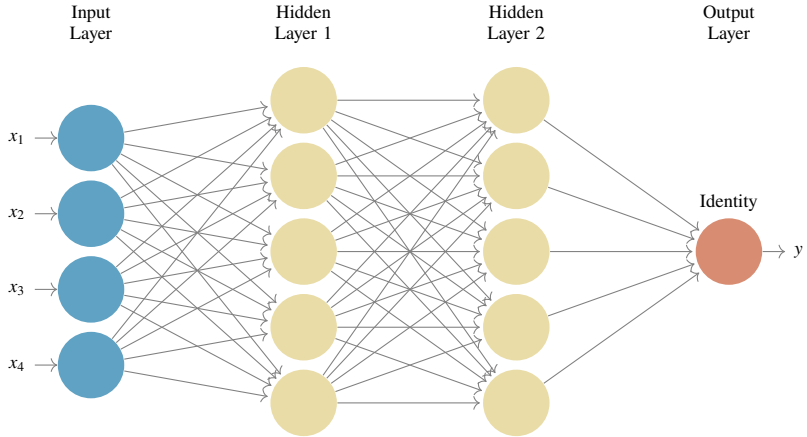

- The network performs a generalized logistic regression.

# Architecture for Multi-Class Classification




- The network performs a generalized multi-class logistic regression.

# Architecture for Regression




- The network performs a generalized linear regression.

Notebook

Multilayer Perceptron: XOR Gate

# Optimization

- Optimization problems:

$$\textbf{Classification} \quad \min_{\boldsymbol{\theta}} \{\mathrm{CE}(\boldsymbol{\theta}) + \mathcal{R}(\boldsymbol{\theta})\},$$

$$\textbf{Regression} \quad \min_{\boldsymbol{\theta}} \{\mathrm{MSE}(\boldsymbol{\theta}) + \mathcal{R}(\boldsymbol{\theta})\}.$$

- Both problems are non-convex (there can be local minima).
- There is no closed-form solution.

---

- The NN can be trained by (stochastic) gradient descent.
- The gradient with respect to the set of weights $\boldsymbol{\theta}$, $\nabla_{\boldsymbol{\theta}} \mathcal{E}$, has to be computed.
- This can be done using **backpropagation**, based on two phases:
  1. **Forward pass** to compute the current estimates, starting at the input layer.
  2. **Backward pass** to compute the gradient, starting at the output layer.
- Some activation functions allow to use the same operations for both phases.

# Backpropagation (I)

- Since usually the error function $\mathcal{E}$ separates across the patterns, the gradient of the error over each individual sample $\mathbf{x}$, $\nabla_{\boldsymbol{\theta}}\mathcal{E}_{\mathbf{x}}$, can be computed separately and then averaged to compose the whole $\nabla_{\boldsymbol{\theta}}\mathcal{E}$.
- A single output NN is considered for simplicity (the extension to several outputs is straightforward).

## Forward Pass

**Require:** $\boldsymbol{\theta}, \mathbf{x} \in \mathbb{R}^d$

**for** $i = 1, \cdots, d$ **do** ▷ Loop over the input units.

  $o_i^{(0)} \leftarrow x_i$ ▷ The input units take the value of the input.

**for** $\ell = 1, \cdots, H$ **do** ▷ Loop through layers.

  **for** $i = 1, \cdots, d_\ell$ **do** ▷ Loop over the hidden units of layer $\ell$.

    $a_i^{(\ell)} \leftarrow \sum_{j=1}^{d_{\ell-1}} w_{ij}^{(\ell-1)} o_j^{(\ell-1)} + b_i^{(\ell-1)}$ ▷ Local field of unit $i$ of layer $\ell$.

    $o_i^{(\ell)} \leftarrow g_i^{(\ell)}(a_i^{(\ell)})$ ▷ Output of unit $i$ of layer $\ell$.

$a_1^{(H+1)} \leftarrow \sum_{j=1}^{d_H} w_{1j}^{(H)} o_j^{(H)} + b_1^{(H)}$ ▷ Local field of output unit.

$f_{\boldsymbol{\theta}}(\mathbf{x}) \leftarrow g_1^{(H+1)}(a_1^{(H+1)})$ ▷ Output of the NN.

# Backpropagation (II)

- The objective now is to compute $\nabla_{\boldsymbol{\theta}} \mathcal{E}_{\mathbf{x}}$, composed by the following partial derivatives:

$$\frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial w_{ij}^{(\ell-1)}}; \quad \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial b_i^{(\ell-1)}}.$$

- The procedure is based on the chain rule, and simplified defining the following **error deltas**:

$$\delta_i^{(\ell)} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial a_i^{(\ell)}}.$$

- Once these deltas are computed, the derivatives are simply:

$$\frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial w_{ij}^{(\ell-1)}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial a_i^{(\ell)}} \frac{\partial a_i^{(\ell)}}{\partial w_{ij}^{(\ell-1)}} = \delta_i^{(\ell)} o_j^{(\ell-1)},$$
$$\frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial b_i^{(\ell-1)}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial a_i^{(\ell)}} \frac{\partial a_i^{(\ell)}}{\partial b_i^{(\ell-1)}} = \delta_i^{(\ell)}.$$

- How is $\delta_i^{(\ell)}$ computed?

# Backpropagation (III)

- Since $a_i^{(\ell)}$ influences all $a_j^{(\ell+1)}$, the chain rule can be used again:

$$\begin{aligned}\delta_i^{(\ell)} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial a_i^{(\ell)}} &= \sum_{j=1}^{d_{\ell+1}} \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial a_j^{(\ell+1)}} \frac{\partial a_j^{(\ell+1)}}{\partial a_i^{(\ell)}} \\ &= \sum_{j=1}^{d_{\ell+1}} \delta_j^{(\ell+1)} \frac{\partial a_j^{(\ell+1)}}{\partial o_i^{(\ell)}} \frac{\partial o_i^{(\ell)}}{\partial a_i^{(\ell)}} \\ &= \sum_{j=1}^{d_{\ell+1}} \delta_j^{(\ell+1)} w_{ij}^{(\ell)} g_i'^{(\ell)}(a_i^{(\ell)}).\end{aligned}$$

- There is a recursion to compute $\delta_i^{(\ell)}$, but a starting point is needed: $\delta_1^{(H+1)}$, the delta in the output unit:

$$\delta_1^{(H+1)} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial a_1^{(H+1)}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial f_{\boldsymbol{\theta}}(\mathbf{x})} \frac{\partial f_{\boldsymbol{\theta}}(\mathbf{x})}{\partial a_1^{(H+1)}} = \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial f_{\boldsymbol{\theta}}(\mathbf{x})} g_1'^{(H+1)}(a_1^{(H+1)}).$$

- The first term depends on the error function. For example, in the case of the MSE, $\frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial f_{\boldsymbol{\theta}}(\mathbf{x})} = f_{\boldsymbol{\theta}}(\mathbf{x}) - y$.

# Backpropagation (IV)

## Backward Pass

**Require:** $\boldsymbol{\theta}, y \in \mathbb{R}, a_i^{(\ell)}, o_i^{(\ell)}, f_{\boldsymbol{\theta}}(\mathbf{x})$

$\delta_1^{(H+1)} \leftarrow \frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial f_{\boldsymbol{\theta}}(\mathbf{x})} g_1'^{(H+1)}(a_1^{(H+1)})$ ▷ Error delta of output unit.

**for** $\ell = H, \cdots, 1$ **do** ▷ Loop through layers (backwards).

  **for** $i = 1, \cdots, d_\ell$ **do** ▷ Loop over the hidden units of layer $\ell$.

    $\delta_i^{(\ell)} \leftarrow g_i'^{(\ell)}(a_i^{(\ell)}) \sum_{j=1}^{d_{\ell+1}} \delta_j^{(\ell+1)} w_{ij}^{(\ell)}$ ▷ Error delta of unit $i$ of layer $\ell$.

**for** $\ell = 0, \cdots, H$ **do** ▷ Loop through layers.

  **for** $i = 1, \cdots, d_\ell$ **do** ▷ Loop over the hidden units of layer $\ell$.

    $\frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial b_i^{(\ell)}} \leftarrow \delta_i^{(\ell+1)}$ ▷ Partial derivative with respect to the bias.

    **for** $j = 1, \cdots, d_{\ell+1}$ **do** ▷ Loop over the hidden units of layer $\ell + 1$.

      $\frac{\partial \mathcal{E}_{\mathbf{x}}}{\partial w_{ij}^{(\ell)}} \leftarrow \delta_i^{(\ell+1)} o_j^{(\ell)}$ ▷ Partial derivative with respect to the weights.

# Some Practical Issues

| | |
|---|---|
| Data Preprocessing | It is crucial to preprocess data for ANNs. |
| Early Stopping | The longer the training, the smaller the training error. To avoid over-fitting, after each epoch the validation error can be controlled. |
| Weight Initialization | Random initialization of the weights, or taking the weights of a network trained for a similar task (pre-training). |
| Stochastic Gradient Descent | Randomly choosing a small subset of patterns (mini-batches) to estimate the gradient. |
| Learning Rate | It influences the convergence of the gradient descent. |
| Momentum | An additional term in weight updates that remembers the direction of previous updates. |
| Regularizers | $\ell_2$ regularizer (weight decay), $\ell_1$ regularizer, dropout... |

Notebook

Multilayer Perceptron: Moons

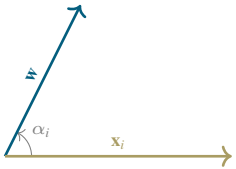
# Artificial Neural Networks

Carlos María Alaíz Gudín

# Additional Material - Rosenblatt Perceptron

# Rosenblatt Perceptron: Geometric Interpretation (I)

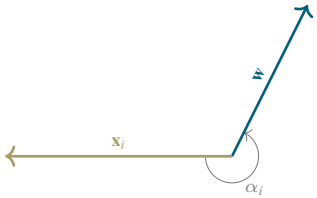
- If $-\frac{\pi}{2} \leq \alpha_i \leq \frac{\pi}{2}$, $g(\mathbf{w}^\intercal \mathbf{x}_i) = 1$ and $\mathbf{x}_i$ is classified as $\mathcal{C}_1$.




For simplicity, $\mathbf{x}_i$ is assumed to contain a constant term, and $\mathbf{w}$ the corresponding weight (the intercept).

# Rosenblatt Perceptron: Geometric Interpretation (II)

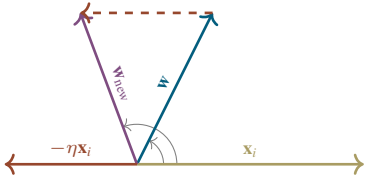
- If $\frac{\pi}{2} \leq \alpha_i \leq \frac{3\pi}{2}$, $g(\mathbf{w}^\intercal \mathbf{x}_i) = 0$ and $\mathbf{x}_i$ is classified as $\mathcal{C}_0$.




For simplicity, $\mathbf{x}_i$ is assumed to contain a constant term, and $\mathbf{w}$ the corresponding weight (the intercept).

# Rosenblatt Perceptron: Geometric Interpretation (III)

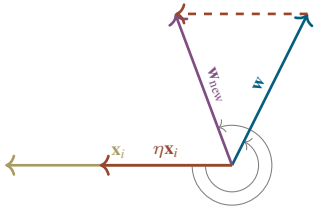
- If $g(\mathbf{w}^\intercal \mathbf{x}_i) = 1$ but $t_i = 0$, then $\delta_i = +1$. The new prediction satisfies $\mathbf{w}_{\text{new}}^\intercal \mathbf{x}_i = \mathbf{w}^\intercal \mathbf{x}_i - \eta \|\mathbf{x}_i\|_2^2 \leq \mathbf{w}^\intercal \mathbf{x}_i$.




For simplicity, $\mathbf{x}_i$ is assumed to contain a constant term, and $\mathbf{w}$ the corresponding weight (the intercept).

# Rosenblatt Perceptron: Geometric Interpretation (IV)

- If $g(\mathbf{w}^\intercal \mathbf{x}_i) = 0$ but $t_i = 1$, then $\delta_i = -1$. The new prediction satisfies $\mathbf{w}_{\text{new}}^\intercal \mathbf{x}_i = \mathbf{w}^\intercal \mathbf{x}_i + \eta \|\mathbf{x}_i\|_2^2 \geq \mathbf{w}^\intercal \mathbf{x}_i$.




For simplicity, $\mathbf{x}_i$ is assumed to contain a constant term, and $\mathbf{w}$ the corresponding weight (the intercept).

# Rosenblatt Perceptron: Optimization Interpretation

- The error function for Rosenblatt perceptron can be defined as:

$$\mathcal{E}(\mathbf{w}, b) = \sum_{i=1}^{N} \delta_i (\mathbf{w}^\intercal \mathbf{x}_i + b) = \sum_{i=1}^{N} (g(\mathbf{w}^\intercal \mathbf{x}_i + b) - t_i)(\mathbf{w}^\intercal \mathbf{x}_i + b).$$

- From its definition, $\nabla_{\mathbf{w}} \delta_i = 0$ except at $\mathbf{w}^\intercal \mathbf{x}_i + b = 0$.
- Therefore,

$$\nabla_{\mathbf{w}} \mathcal{E}(\mathbf{w}, b) = \sum_{i=1}^{N} \delta_i \mathbf{x}_i,$$

$$\frac{\partial}{\partial b} \mathcal{E}(\mathbf{w}, b) = \sum_{i=1}^{N} \delta_i.$$

- Online gradient descent becomes Rosenblatt algorithm.