MultiDrone

Artificial Neural Networks

Contributors: Sotiris Papadopoulos, Christos Chadoulos, Pantelis I. Kaplanoglou, Ioannis Pitas

Presenter: Prof. Ioannis Pitas

Aristotle University of Thessaloniki

pitas@aiia.csd.auth.gr

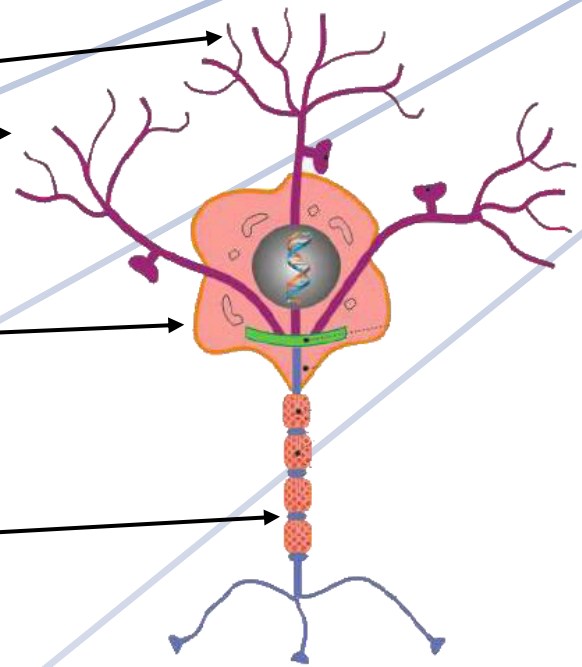
www.multidrone.eu

Presentation version 1.3

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Biological Neuron

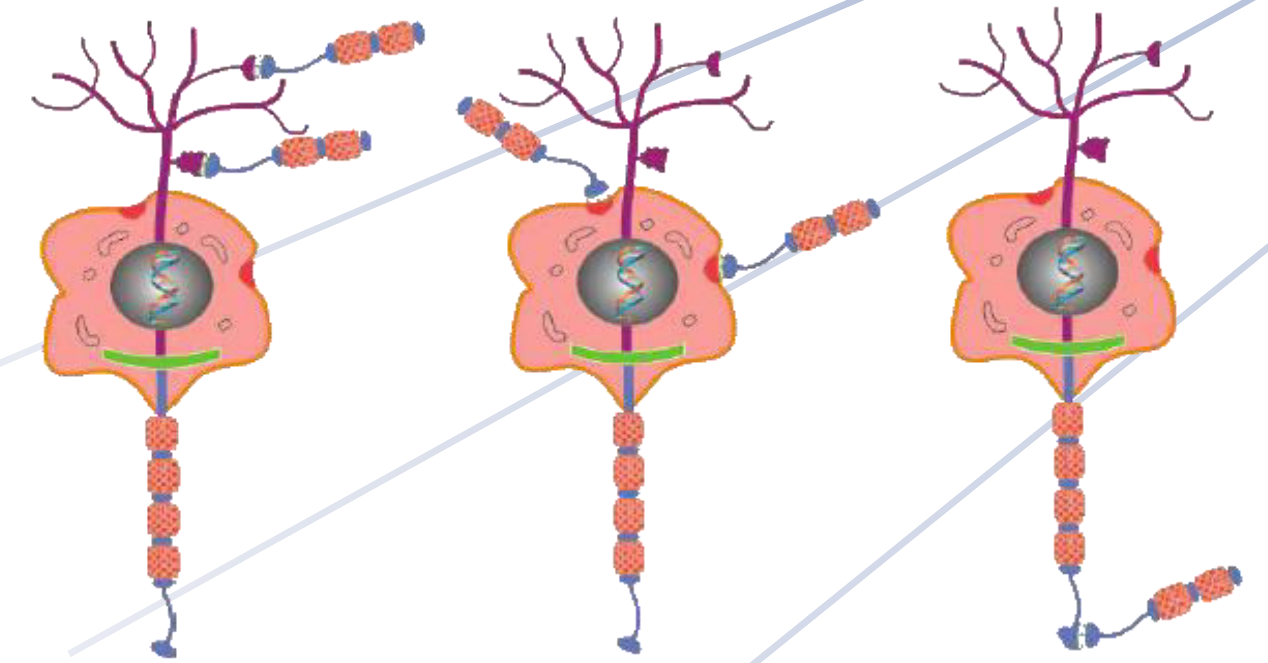
- Basic computational unit of the brain.
- Main parts:
  - Dendrites
    - ➔ Act as inputs.
  - Soma
    - ➔ Main body of neuron.
  - Axon
    - ➔ Acts as output.
- Neurons connect with other neurons via synapses.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Biological Neuron Connectivity

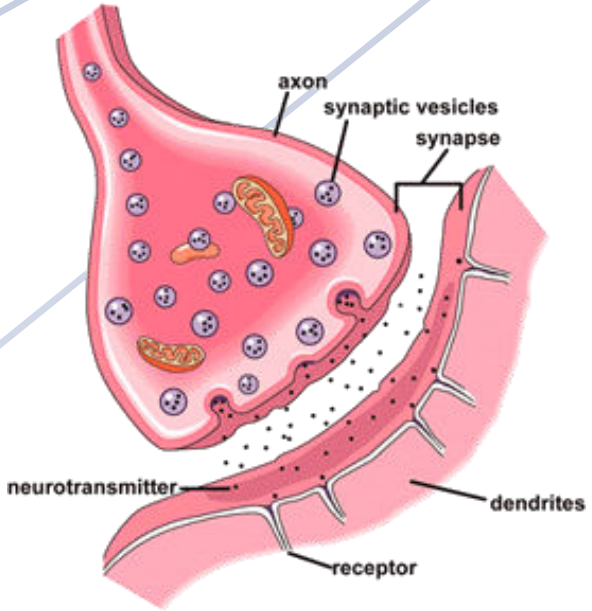
- Neurons connect with other neurons through *synapses.*



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Biological Neuron Conncetivity

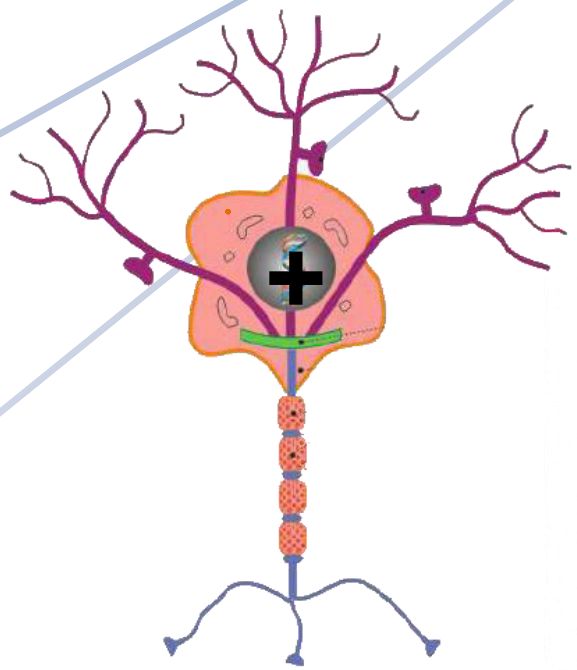
- An electric action potential is propagated through the axon.
- Signal is transmitted through the synapse gap by neurotransmitter molecules.
- Transmitted signal is an electrical impulse.
- Each synapse has its own synaptic weight.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Synaptic Integration

- Electric potential received by all dendrites of a neuron is accumulated inside its soma
- When the electric potential at the membrane reaches a certain threshold, the neuron fires an electrical impulse
- The signal is propagated through the axon and information is “fed” forward to all connected neurons.



This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MultiDrone

Motivation for Neural Network Research

- Goal: Build computational models $y = f(\boldsymbol{x})$ that can learn from data.
- Data: $D = \{\mathbf{x}_i, y_i\}_{i=1}^{N}, \quad \mathbf{x}_i \in R^n, \quad y_i \in Y \subseteq R$ (supervised learning)
  - → $\mathbf{x}_i$ are n-dimensional real-valued vectors, called features.
  - → y_i are real valued variables, called targets.
- Assumption: There is an underlying function F for which $\mathrm{y} = F(\mathbf{x}) + \epsilon$.
- *Statistical modeling approach*: search through a class of functions C for a function f that best approximates the true function F.
- *Machine learning approach*: search the function space C for an appropriate function $f(\boldsymbol{x})$.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Learning (Supervised)

- In most cases, function f can have one of the following forms:
  - $y = f(\mathbf{x}; \boldsymbol{w}) = \mathbf{w}^T\mathbf{x} + b$: linear function (or mapping from $\mathbf{x}$ to y).
  - $y = f(\mathbf{x}; \boldsymbol{w}) = \mathbf{w}^T\phi(\mathbf{x}) + b$: nonlinear function
  - $\phi: R^n \rightarrow H$ is a nonlinear mapping of $\mathbf{x}$ to a (possibly) high-dimensional space, $\mathrm{m} = \dim(H) > n, \boldsymbol{w} \in R^m$.

- Learning consists of finding the optimal parameters $\mathbf{w}$ so that $\hat{\mathrm{y}} = f(\mathbf{x}; \mathbf{w})$ is as close as possible to the target y.
- Optimization of parameters $\mathbf{w}$ is accomplished by minimizing a cost function $J(\mathbf{w})$ where cost is defined as a measure of discrepancy between y and $\hat{\mathrm{y}}$.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Learning (Supervised)

- Depending on the form of the target y, the estimated function f can perform different tasks:

  1. Classification: $\boldsymbol{y}$ is categorical, $y_i \in \{k_1, k_2, \ldots, k_c\}$, c: number of classes. The goal is to learn a function $\boldsymbol{f}(\mathbf{x}; \mathbf{w})$ that correctly classifies each sample $\mathbf{x}_i$ to its corresponding class. Typically, the NN has vectorial output $\hat{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{w})$ where: $\hat{\boldsymbol{y}} \in R^c$ and $\hat{y}_i = \begin{cases} 1, & \text{if } y_i = i \\ 0, & \text{elsewhere} \end{cases}$, $i = 1, \ldots, c$.

  2. Regression: $\boldsymbol{y}$ is real: $y_i \in R$. The goal is to learn a function $\boldsymbol{f}(\mathbf{x}; \mathbf{w})$ that produces an estimate $\hat{\boldsymbol{y}}$, as close as possible to $\boldsymbol{y}$.

- Success depends on: a) the functional form of f (linear, nonlinear, etc.), b) the parameters $\mathbf{w}$, c) the cost function $J(\boldsymbol{w})$, d) the optimization procedure.

- A possible approach: mimic the functionality of biological neurons.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Learning (Unsupervised)

- In addition to supervised learning, where training samples come with known labels (or real valued targets), there are cases where unsupervised learning is needed.
- Unsupervised learning is the process of training a model without any targets.
- Some types of unsupervised learning:

1. **Clustering:** each sample is assigned to a cluster, in which all samples are similar to the rest of the samples of this cluster and dissimilar to the samples outside the cluster.
2. **Dimensionality reduction:** learning a transformation matrix $\boldsymbol{W_{n\times d}}$ where $n > d$, that when applied to a sample $\boldsymbol{x} \in \mathbb{R}^n$ it is transformed to $\hat{\boldsymbol{x}} \in \mathbb{R}^d$, while keeping as much information as possible.
3. **Autoencoders:** ANNs that are taught to reconstruct their inputs after having them compressed to a more compact representation. Used in feature extraction, pretraining, sample generation, etc.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Artificial Neurons

- Artificial neurons are mathematical models loosely inspired by their biological counterparts.
- Incoming signals through the axons serve as the input vector:

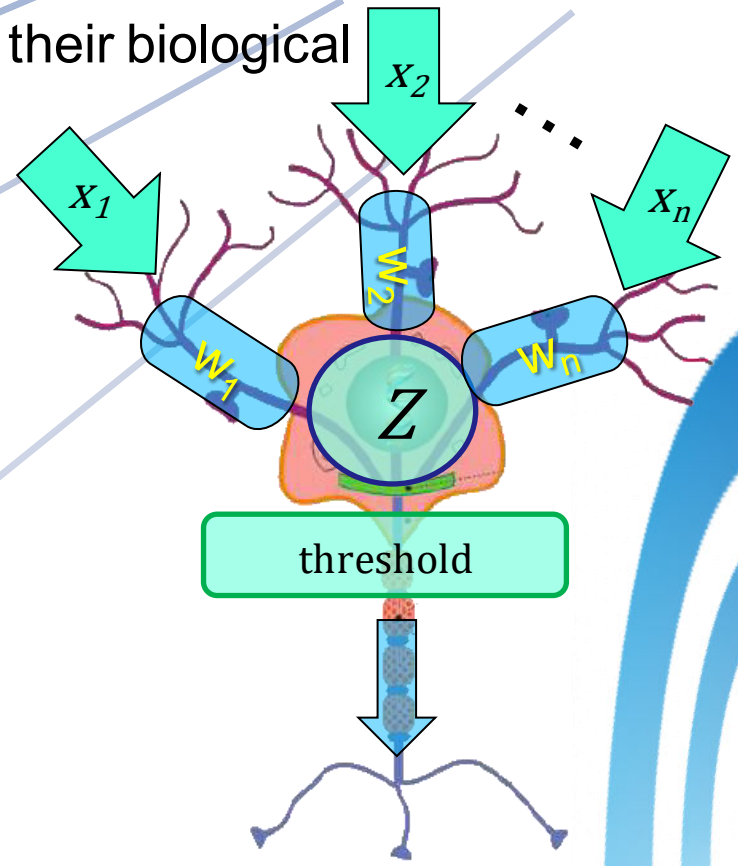
$$\mathbf{x} = [x_1, x_2, \dots, x_n]^T, \qquad x_i \in R.$$

- The synaptic weights are grouped in a weight vector:

$$\mathbf{w} = [w_1, w_2, \dots, w_n]^T, \quad w_i \in R.$$

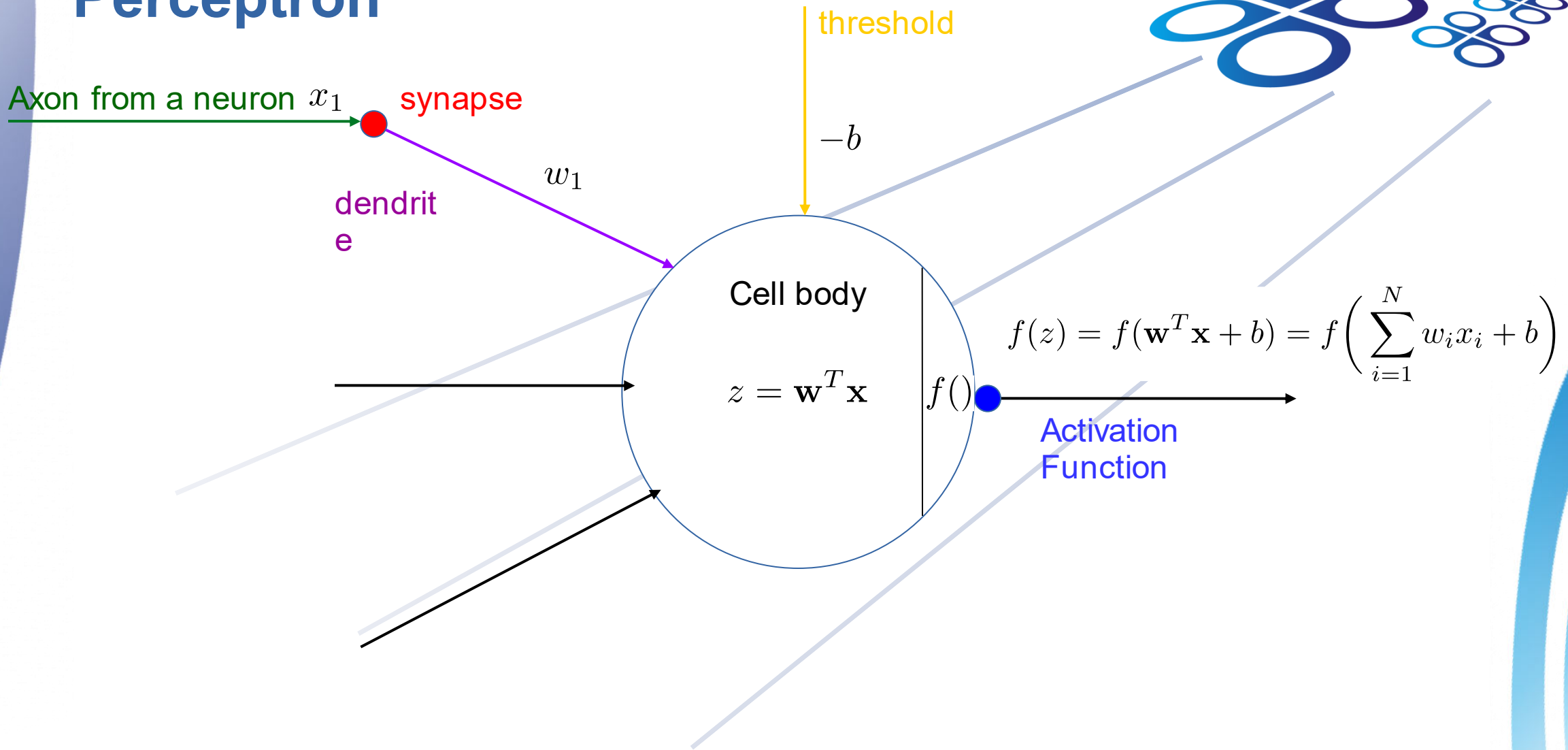
- Synaptic integration is modeled as the inner product:

$$z = \sum_{i=1}^{N} w_i x_i = \mathbf{w}^T \mathbf{x}.$$



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Perceptron

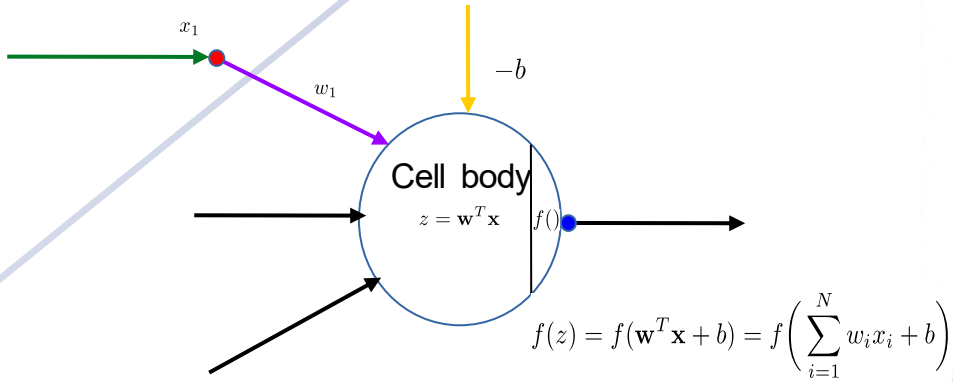


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Perceptron

- Simplest mathematical model of a neuron (Rosenblatt – McCulloch & Pitts)
- Inputs are binary, $x_i \in \{0,1\}$.
- Produces single, binary outputs, $y_i \in \{0,1\}$, through an activation function $f(\cdot)$.
- Output y_i signifies whether the neuron will fire or not.
- Firing threshold:

$$\mathbf{w}^T\mathbf{x} \geq -b \Rightarrow \mathbf{w}^T\mathbf{x} + b \geq 0.$$



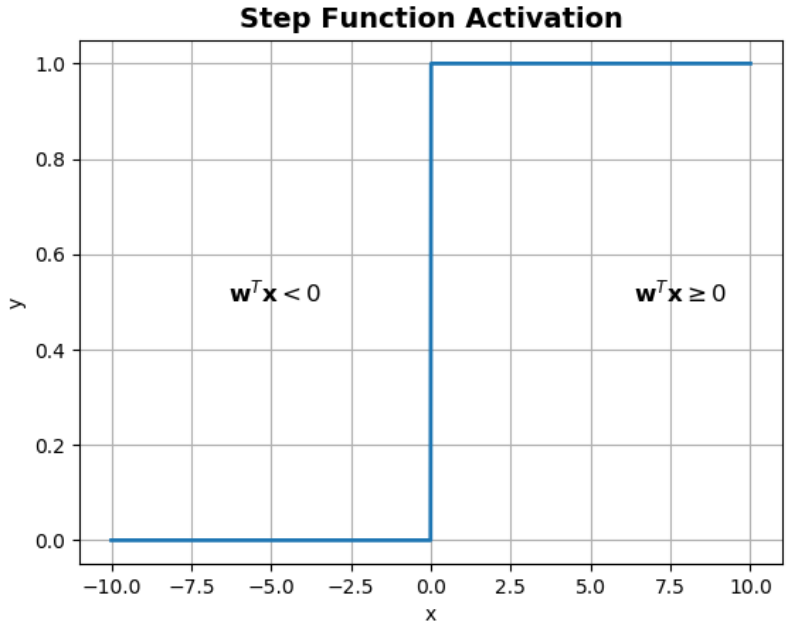
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Perceptron – Activation

- Threshold can be incorporated in the weight vector.
- Augmented input and weight vectors: $\mathbf{x}' = [1, x_1, \dots, x_n]^T, \quad \mathbf{x}' \in R^{n+1}$

$$\mathbf{w}' = [1, w_1, \dots, w_n]^T, \quad \mathbf{w}' \in R^{n+1}$$

- Step function to model the activation of the perceptron :



$$y = f(z) = f(\mathbf{w}^T\mathbf{x}) = \begin{cases} 0, & \mathbf{w}'^T\mathbf{x}' < 0 \\ 1, & \mathbf{w}'^T\mathbf{x}' \geq 0 \end{cases}$$

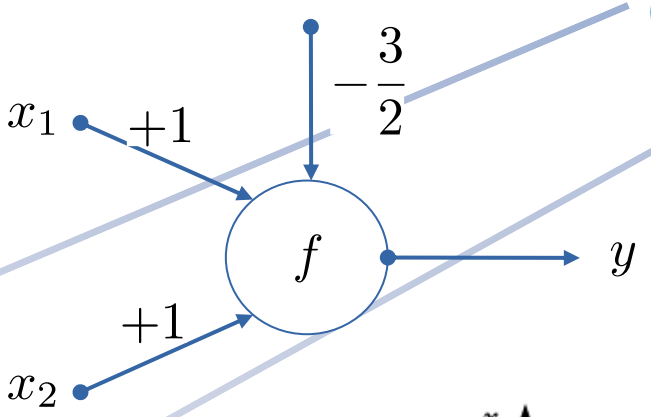
Suitable for 2-class problems:

- If $f(z) \geq 0$, assign $\mathbf{x}$ to class $+1$
- If $f(z) < 0$, assign $\mathbf{x}$ to class -1.

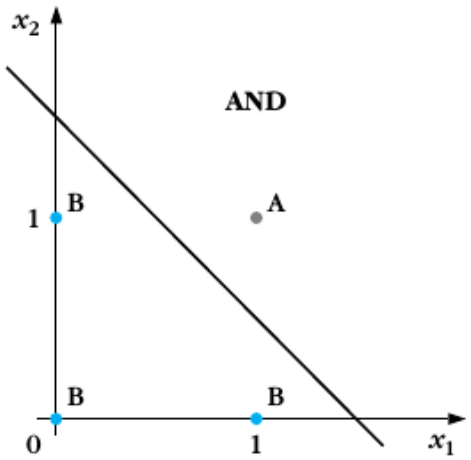
ı Union's Horizon 2020 research
ement No 731667 (MULTIDRONE)

Perceptron – AND function model

- Input vector: $\mathbf{w} = \left[-\frac{3}{2}, 1, 1\right]^T$
- Weight vector: $\mathbf{x} = [1, x_1, x_2]^T$



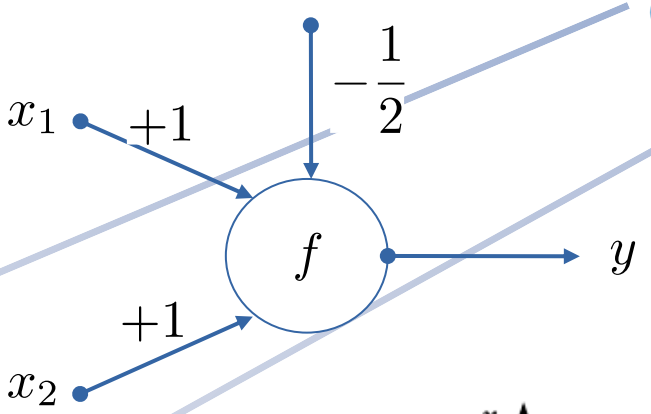
x_1	x_2	$y = \mathbf{w}^T\mathbf{x}$
0	0	$\mathbf{w}^T\mathbf{x} < 0 \Rightarrow y = 0$
0	1	$\mathbf{w}^T\mathbf{x} < 0 \Rightarrow y = 0$
1	0	$\mathbf{w}^T\mathbf{x} < 0 \Rightarrow y = 0$
1	1	$\mathbf{w}^T\mathbf{x} \geq 0 \Rightarrow y = 1$



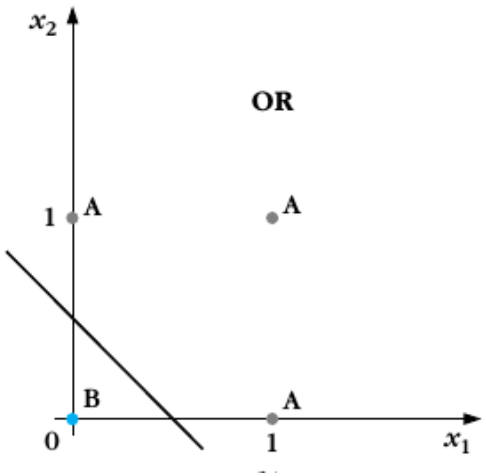
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Perceptron – OR function model

- Input vector: $\mathbf{w} = \left[-\frac{1}{2}, 1, 1\right]^T$
- Weight vector: $\mathbf{x} = [1, x_1, x_2]^T$



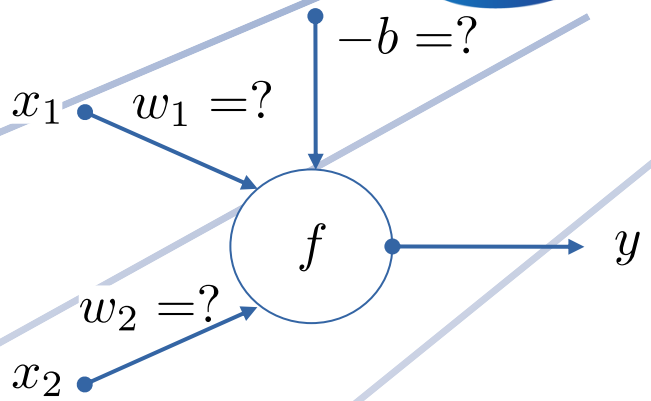
x_1	x_2	$y = \mathbf{w}^T\mathbf{x}$
0	0	$\mathbf{w}^T\mathbf{x} < 0 \Rightarrow y = 0$
0	1	$\mathbf{w}^T\mathbf{x} \geq 0 \Rightarrow y = 1$
1	0	$\mathbf{w}^T\mathbf{x} \geq 0 \Rightarrow y = 1$
1	1	$\mathbf{w}^T\mathbf{x} \geq 0 \Rightarrow y = 1$

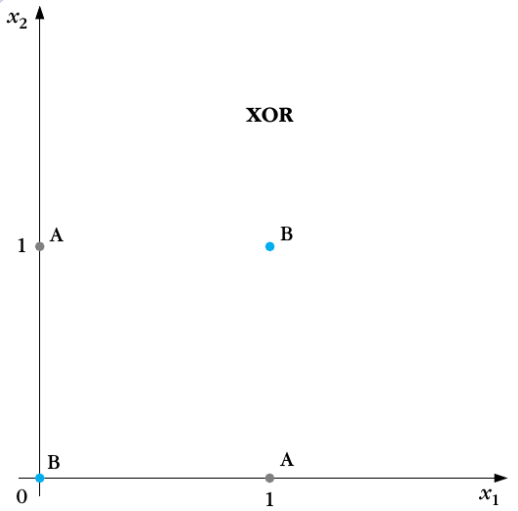


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

XOR function modeling

- There is no linear separating line in R^2 for XOR function.
- No set of parameters $\mathbf{w}, b$ can produce a line that can correctly model XOR.
- Solution: Add an extra layer of neurons before the perceptron output y.
- Extra layer consists of two perceptrons, computing the AND and the OR function respectively.
- The new functional form will be $f(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$, where $\phi(\mathbf{x})$ is the output of the extra layer, given as input to the output layer.





This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Two-layer Perceptron – XOR function model

- Notation:

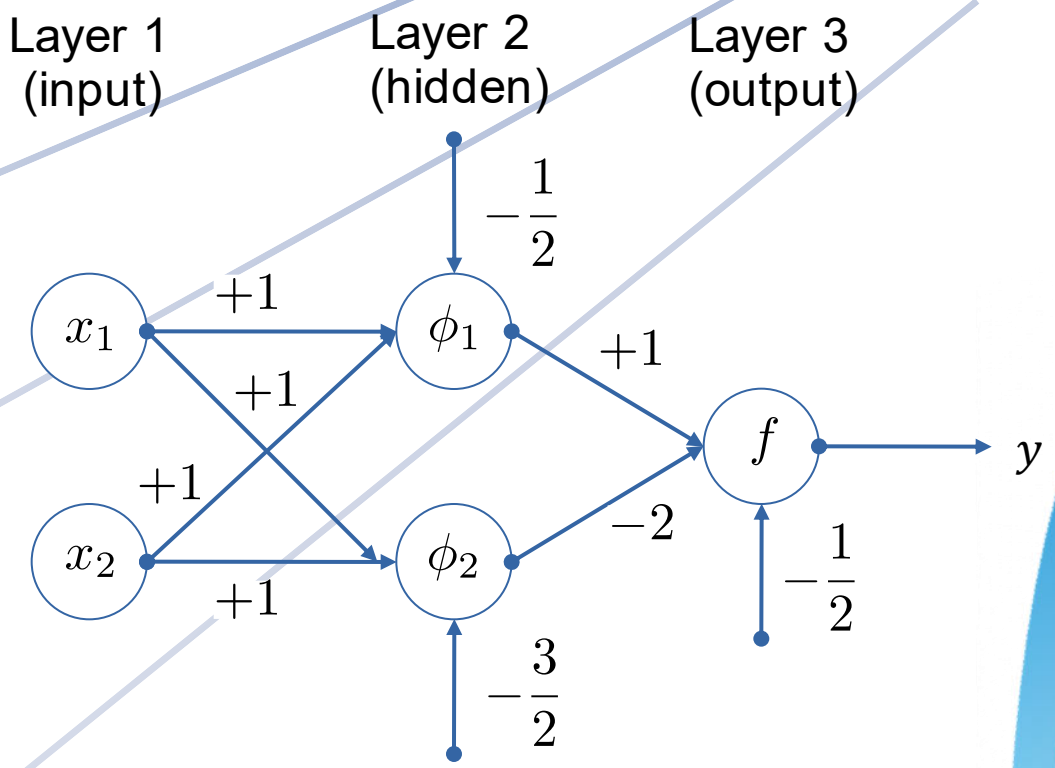
$$\mathbf{x} = [1, x_1, x_2]^T$$

$$\mathbf{w}_1 = \left[-\tfrac{1}{2}, 1, 1\right]^T \qquad \phi_1(\mathbf{x}) = \mathbf{w}_1^T\mathbf{x}$$

$$\mathbf{w}_2 = \left[-\tfrac{3}{2}, 1, 1\right]^T \qquad \phi_2(\mathbf{x}) = \mathbf{w}_2^T\mathbf{x}$$

$$\mathbf{w} = \left[\tfrac{3}{2}, 1, -2\right]^T \qquad y = \mathbf{w}^T\boldsymbol{\Phi}$$

$$\boldsymbol{\Phi} = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x})]^T$$

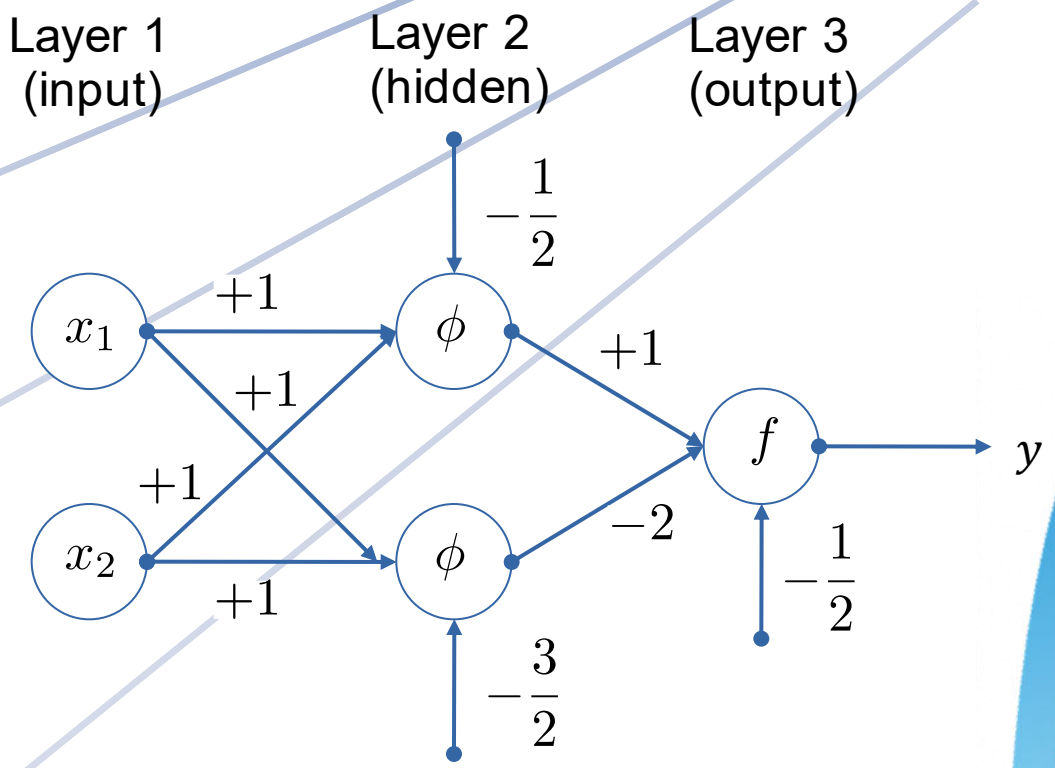


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Two-layer Perceptron – XOR function model

x_1	x_2	$\phi_1(\mathbf{x})$	$\phi_2(\mathbf{x})$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

$\phi_1(\mathbf{x})$	$\phi_2(\mathbf{x})$	$y = \mathbf{w}^T\mathbf{\Phi}$
0	0	$\mathbf{w}^T\mathbf{\Phi} < 0 \Rightarrow y = 0$
0	1	$\mathbf{w}^T\mathbf{\Phi} \geq 0 \Rightarrow y = 1$
1	0	$\mathbf{w}^T\mathbf{\Phi} \geq 0 \Rightarrow y = 1$
1	1	$\mathbf{w}^T\mathbf{\Phi} < 0 \Rightarrow y = 0$



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Single Perceptron training

- The target is to find the optimal model parameters $\mathbf{w}$, so that the model produces the minimal number of false predictions.
- This problem is approached as an optimization problem, so there is a need to:
  a) construct an appropriate cost function.
  b) define an optimization algorithm to minimize this function.
- Perceptron cost function: $J(\boldsymbol{w}) = \sum_{\boldsymbol{x}\in Y} d_x(\boldsymbol{w}^T\boldsymbol{x})$,

Y the subset of training samples that have been misclassified, $d_x \triangleq -1$ if $\boldsymbol{x}$ is a sample of the first class, and $d_x \triangleq 1$ if $\boldsymbol{x}$ is a sample of the second class.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Single Perceptron training

$$J(\boldsymbol{w}) = \sum_{\boldsymbol{x} \in Y} d_x(\boldsymbol{w}^T \boldsymbol{x})$$

- It is apparent that when all samples have been correctly classified, $Y = \emptyset \Rightarrow J(\boldsymbol{w}) = 0$.
- If $\boldsymbol{x} \in class\ 1$ and has been misclassified, then $\boldsymbol{w}^T \boldsymbol{x} < 0$ and $d_x < 0$, and thus, the product of the two is positive.
- Same applies for the samples of $class\ 2$ that have been misclassified.
- $J(\boldsymbol{w})$ is continuous and linear in parts. It is also differentiable only in parts.
- An appropriate iterative minimization algorithm for this function is the *Gradient Descent* algorithm.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MultiDrone

Single Perceptron training – Gradient Descent

- One of the most popular optimization algorithms.
- Iteratively searches the parameter space by following the direction of the steepest descent. Given any multivariate function $f(\boldsymbol{\theta})$, the gradient vector points to the direction of the steepest ascent.

$$\nabla f(\boldsymbol{\theta}) = \frac{\partial f}{\partial \boldsymbol{\theta}} = \left[\frac{\partial f}{\partial \boldsymbol{\theta}_1}, \cdots, \frac{\partial f}{\partial \boldsymbol{\theta}_m}\right]^T$$

- Correspondingly, $-\nabla f$ points to the direction where f decreases more rapidly.
- A solution to the choice of the correction term: it must be proportional to the direction of steepest descent:

$$\Delta\boldsymbol{\theta} = -\mu\frac{\partial f}{\partial \boldsymbol{\theta}} \rightarrow \boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) - \mu\nabla\big(\boldsymbol{\theta}(t)\big)$$

- μ: a parameter controlling model parameters updates called **learning rate**.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Single Perceptron training – Gradient Descent

- Applying the previous parameter update rule to the Perceptron model:

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \mu \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}}.$$

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = \frac{\partial}{\partial \boldsymbol{w}} \sum_{\boldsymbol{x} \in Y} d_x (\boldsymbol{w}^T \boldsymbol{x}) = \sum_{\boldsymbol{x} \in Y} d_x \boldsymbol{x}.$$

- The complete form of the update rule, widely known as the Perceptron algorithm becomes as follows:

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \mu \sum_{\boldsymbol{x} \in Y} d_x \boldsymbol{x}.$$

- It is proven that this algorithm converges.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MultiDrone

Towards Multi-Layer Perceptrons (MLP)

- Single neuron models (Perceptron) are only suitable for learning linear hyperplanes.
- Introducing more layers between the input and the output allows us to model more complex functions.
- The following theorem, imposing only mild assumptions on the activation function, states that a neural network, with a single hidden layer containing a finite number of neurons, can represent a wide variety of functions.

Universal Approximation Theorem:
Let $\phi(\,)$ be a noncostant, bounded and continuous function. Let H_n denote the n-dimensional unit Hypercube $[0,1]^n$. The space of continuous functions on H_n is denoted as $C(H_n)$. Then, given any $\epsilon > 0$ and any function $F(\boldsymbol{x}) \in C(H_n)$, there exist an integer N, real constants $u_i, b_i \in R$ and real vectors $\mathbf{w}_i \in R^n$ where $i = 1, \ldots, N$ such that we may define:

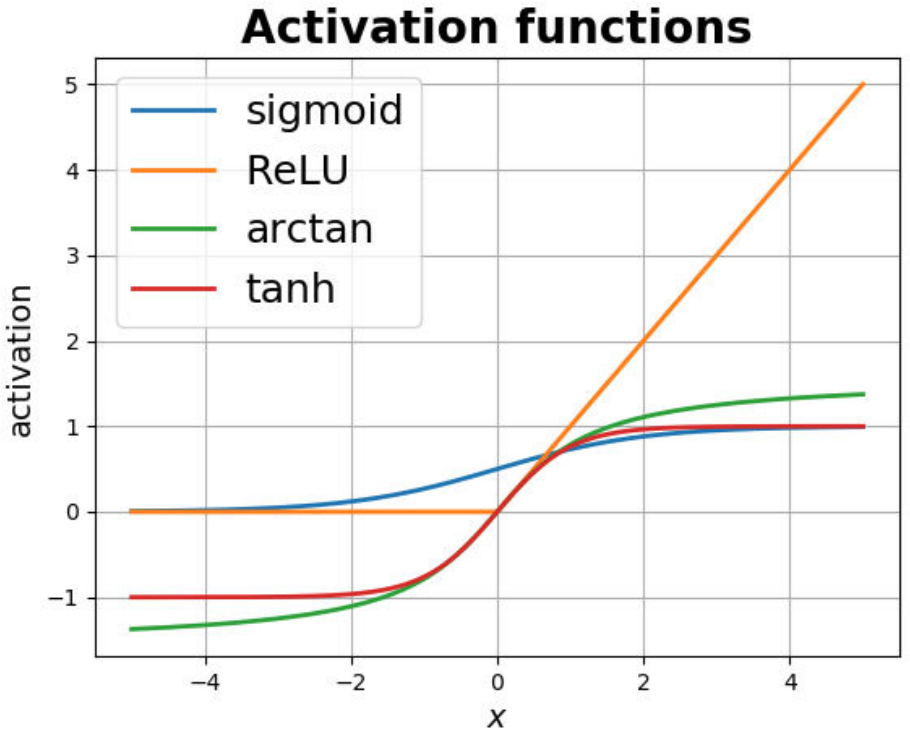
$$f(\mathbf{x}) = \sum_{i=1}^{N} u_i \phi\left(\mathbf{w}^T \mathbf{x} + b_i\right)$$

as an approximate realization of the function F, that is for all $\mathbf{x} \in H_n, \left|F(\mathbf{x}) - f(\mathbf{x})\right| < \epsilon$.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Activation Functions

- Previous theorem verifies MLPs as universal approximators, under the condition that the activation functions $\phi(\cdot)$ for each neuron are continuous.
- Continuous (and differentiable) activation functions:
  - $\sigma(x) = \frac{1}{1+e^{-x}}, \quad R^m \to [0,1]$
  - $ReLU(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}, \quad R^m \to R_+$
  - $f(x) = \tan^{-1}(x), \quad R^m \to [-\pi/2, \pi/2]$
  - $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad R^m \to [-1, +1]$



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Architecture

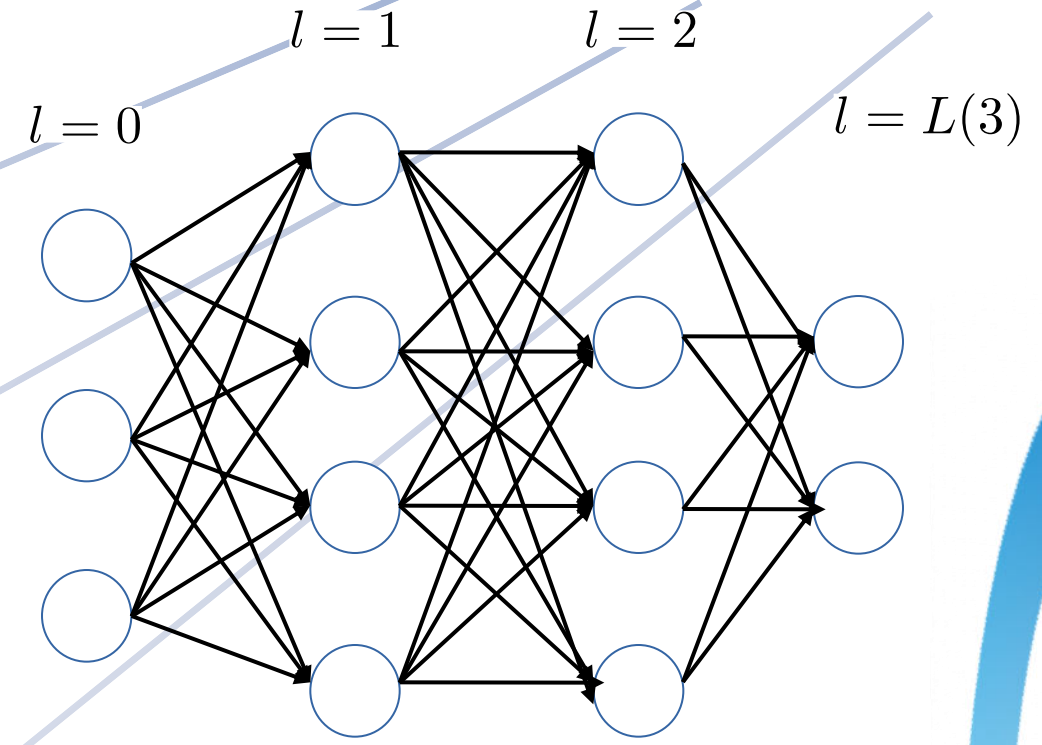
- Multilayer perceptrons (feed-forward neural networks) typically consist of $L+1$ layers with k_l neurons in each layer: $l = 0, \dots, L$.
- The first layer (input layer $l = 0$) contains n inputs, where n is the dimensionality of the input sample vector.
- The $L-1$ hidden layers $l = 1, \dots, L-1$ can contain any number of neurons.
- The output layer $l = L$ typically contains either one neuron (especially when dealing with regression problems) or c neurons, where c is the number of classes.
- Thus, neural networks can be described by the following characteristic numbers:

  1. Depth: Number of hidden layers L.
  2. Width: Number of neurons per layer $k_l, l = 1, \dots, L$.
  3. $k_L = 1$, or $k_L = c$.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Architecture

- Suppose we have a network with L layers and k_l neurons per layer.
- Input vector in layer 0: $\mathbf{x} = [x_1, \ldots, x_n]^T$
- For each layer:
  - b_j^l: the bias of the j^{th} neuron in the l^{th} layer.
  - w_{jk}^l: the weight for k^{th} neuron in the $(l-1)^{th}$ layer to j^{th} neuron in the l^{th} layer.
  - z_j^l: the argument of the activation function of the j^{th} neuron in the l^{th} layer.
  - a_j^l:the output of the activation function of the j^{th} neuron in the l^{th} layer.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Forward Propagation

- The activation for each neuron of the first layer $l = 1$ is defined by the following equations:

$$z_j^{(1)} = \sum_{k=1}^{n} w_{jk}^{(1)T} x_k + b_j^{(1)} = \boldsymbol{w}_j^{(1)T} \boldsymbol{x} + b_j^{(1)}$$

$$a_j^{(1)} = f(\boldsymbol{w}_j^{(1)T} \boldsymbol{x} + b_j^{(1)}).$$

- Grouping the weights and biases, we can rewrite the above equations in a more compact matrix/vector form:

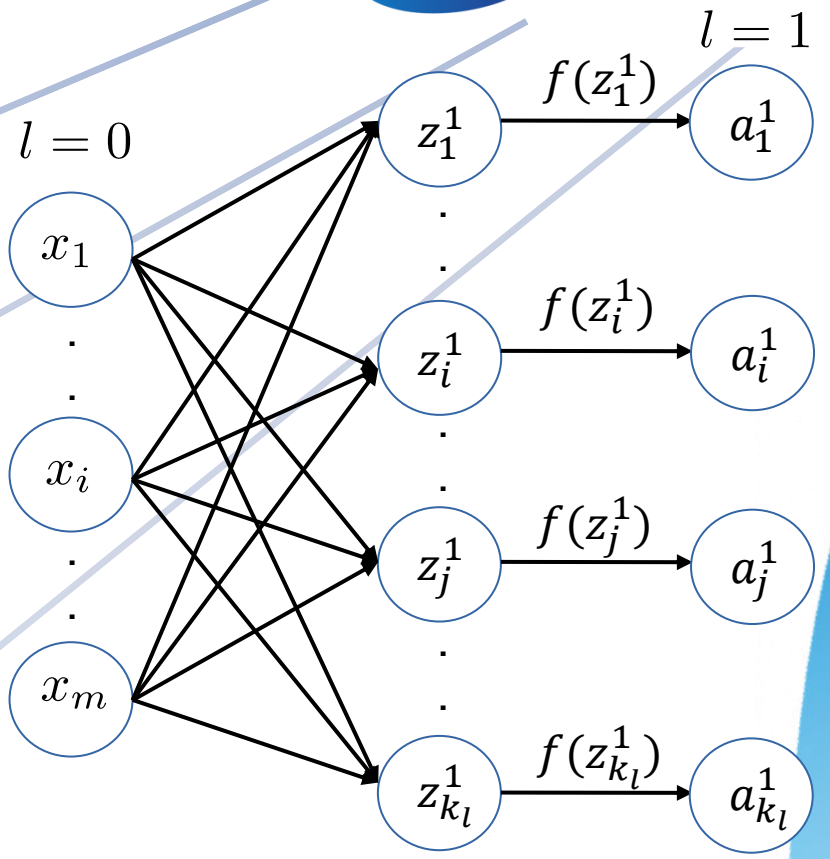
$$\mathbf{W}^{(1)} = [\mathbf{w}_j^{(1)}]_{j=1}^{k_l}$$

$$\mathbf{b}^{(1)} = [b_j^{(1)}]_{j=1}^{k_l}$$

$$\Longrightarrow$$

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)T}\mathbf{x} + \mathbf{b}^{(1)}$$

$$\boldsymbol{\alpha}^{(1)} = f(\mathbf{z}^{(1)}).$$



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Forward Propagation

- Following the same procedure for the l^{th} hidden layer, we get:

$$z_j^{(l)} = \mathbf{w}_j^{(l)T} \boldsymbol{\alpha}^{(l-1)} + b_j^{(l)}$$

$$a_j^{(1)} = f(\mathbf{w}_j^{(l)T} \boldsymbol{\alpha}^{(l-1)} + b_j^{(l)}).$$

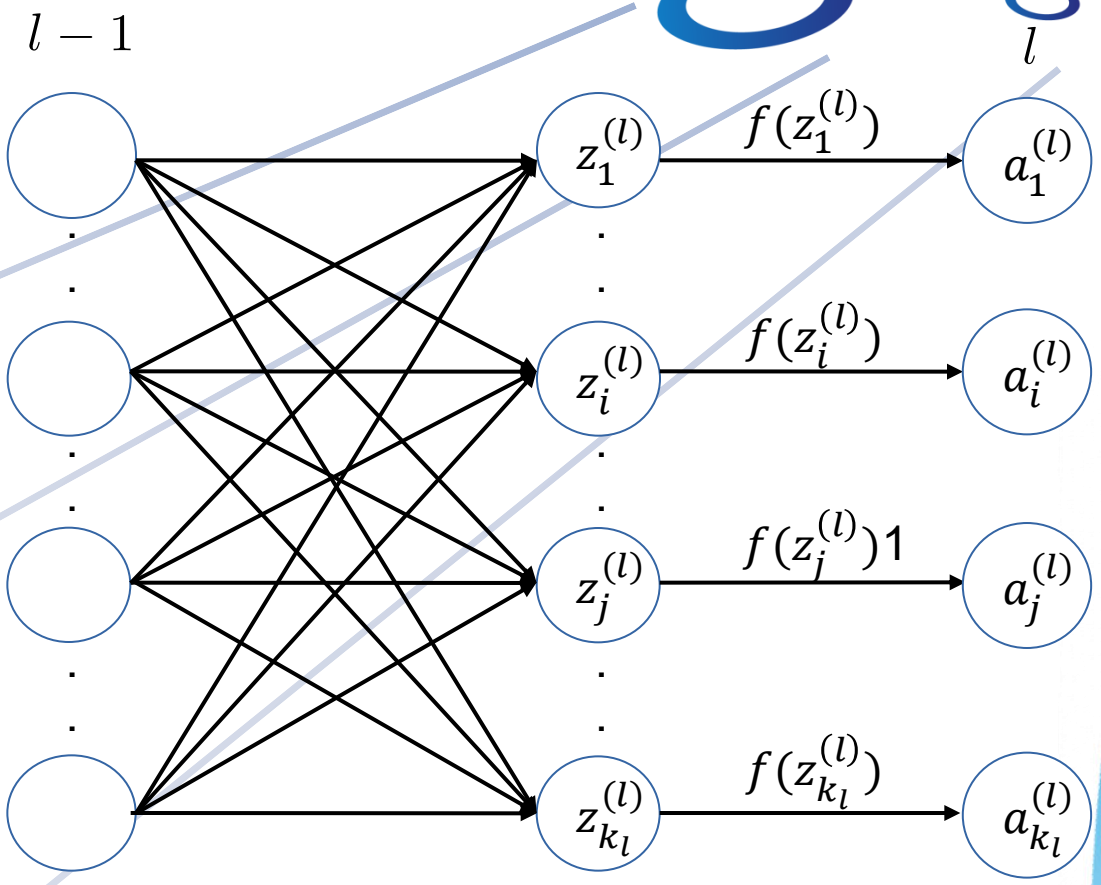
- In compact form:

$$\mathbf{W}^{(l)} = [\mathbf{w}_j^{(l)}]_{j=1}^{k_l}$$

$$\mathbf{b}^{(l)} = [b_j^{(l)}]_{j=1}^{k_l}$$

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)T} \boldsymbol{\alpha}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\boldsymbol{\alpha}^{(l)} = f(\mathbf{z}^{(l)}).$$



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

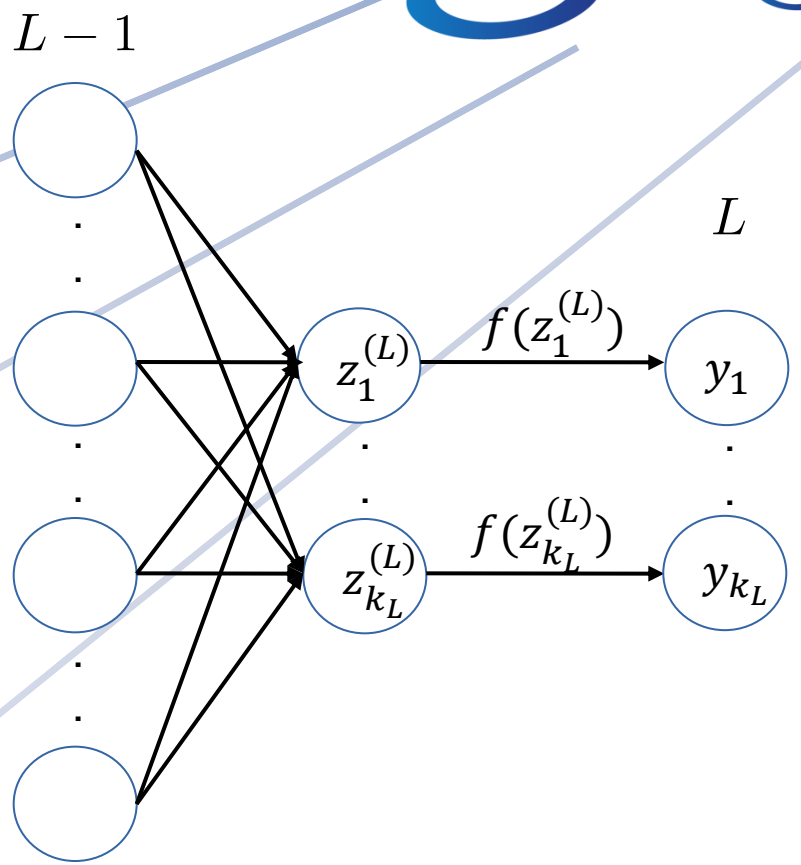
MLP Forward Propagation

- Finally, in the last (output) layer $l = L$, with $k_l = 1$ (regression, two class classifiaction) or c (multi-class classification):

$$z_j^{(L)} = \mathbf{w}_j^{(L)T} \boldsymbol{\alpha}^{(L-1)} + b_j^{(L)}$$

$$y_j = f(\mathbf{w}_j^{(L)T} \boldsymbol{\alpha}^{(L-1)} + b_j^{(L)})$$

- MLP in total computes the following function: $\hat{\mathbf{y}} = F_{NN}(\mathbf{x}; \boldsymbol{\theta})$, where we have grouped all the parameters $\{\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(L)}, \boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(L)}\}$ in a vector $\boldsymbol{\theta} \in R^m$ to be learnt.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Softmax Layer

- It is the last layer in a neural network classifier.
- The response of neuron i in the softmax layer L is calculated with regard to the value of its activation function $a_i = f(z_i)$.

$$\hat{y}_i = g(a_i) = \frac{e^{a_i}}{\sum_{k=1}^{k_L} e^{a_k}} : \mathbb{R} \to [0,1], \quad i = 1,..,k_L$$

- The responses of softmax neurons sum up to one: $\sum_{i=1}^{k_L} \hat{y}_i = 1.$
- Better representation for mutually exclusive class labels.

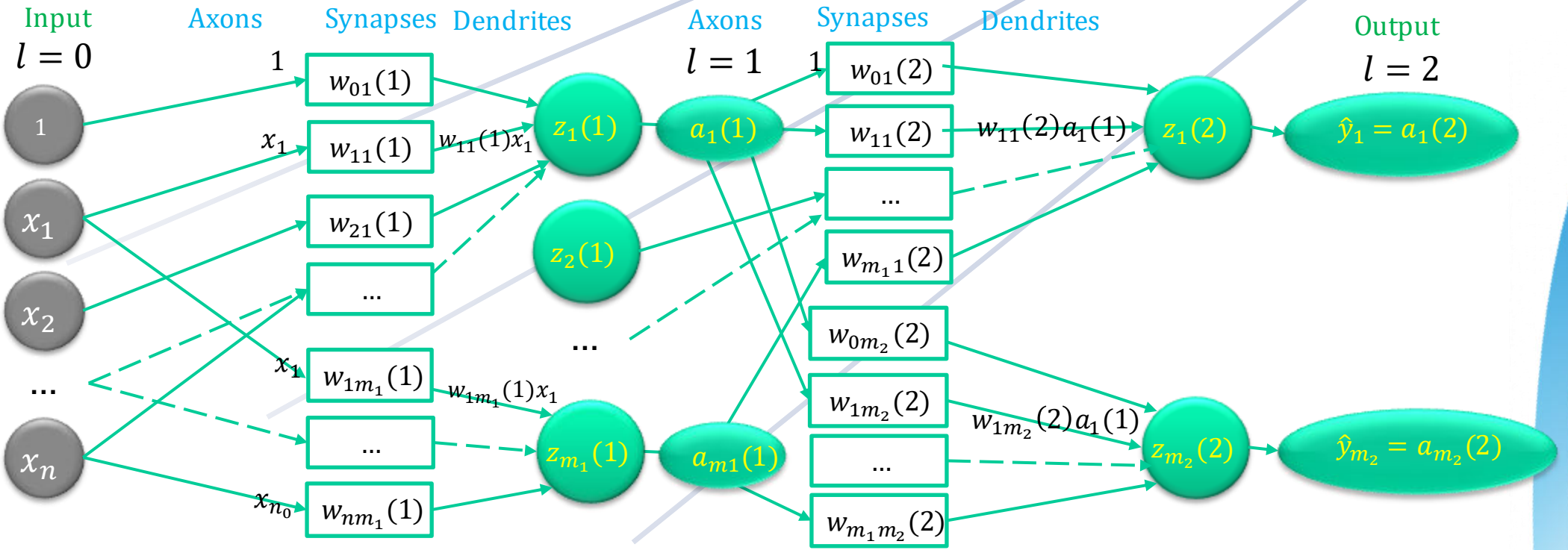
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Example

Fully connected

- Example architecture with $L = 2$ layer and $n = m_0$ input features, m_1 neurons at the first layer and m_2 output units.

$$a_j(l) = f_l\left(\sum_{i=1}^{m_{l-1}} w_{ij}(l)\, a_i(l-1) + w_{0j}(l)\right)$$



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Training

The process of computing the optimal parameters for a model is called *training*. There are three ways of training a network:

- ***Supervised Learning***: Train a NN model by presenting sample set $X = \{\boldsymbol{x}_i, i = 1, \dots, N\}$, where each feature vector $\boldsymbol{x}_i$ is paired with a target vector $\boldsymbol{y}_i = [y_{i1}, \dots, y_{ik_L}]^T$, $y_{ij} = \begin{cases} 1, & \text{if } \boldsymbol{x}_i \text{ belongs to class } j = 1, \dots, k = c \\ 0, & \text{elsewhere} \end{cases}$ for classification, and $\boldsymbol{y}_i \in R^{k_L}$ for regression.

- ***Unsupervised Learning***: Train a model with samples X without any targets. This is used by the process of *pre-training* that aims to create an initial *neural network state* that is better than random.

- ***Fine-tuning***: Reuse a neural network model that was created through training on samples X_a, to run some training steps on different samples X_b. This is method implements *transfer learning*.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Training

The process of computing the optimal parameters for a model is called *training*. There are three ways of training a network:

- ***Supervised Learning***: Train a NN model by presenting sample set $X = \{\boldsymbol{x}_i, i = 1, \dots, N\}$, where each feature vector $\boldsymbol{x}_i$ is paired with a target vector $\boldsymbol{y}_i = [y_{i1}, \dots, y_{ik_L}]^T$, $y_{ij} = \begin{cases} 1, & \text{if } \boldsymbol{x}_i \text{ belongs to class } j = 1, \dots, k = c \\ 0, & \text{elsewhere} \end{cases}$ for classification, and $\boldsymbol{y}_i \in R^{k_L}$ for regression.

- ***Unsupervised Learning***: Train a model with samples X without any targets. This is used by the process of *pre-training* that aims to create an initial *neural network state* that is better than random.

- ***Fine-tuning***: Reuse a neural network model that was created through training on samples X_a, to run some training steps on different samples X_b. This is method implements *transfer learning*.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Training

The process of computing the optimal parameters for a model is called *training*. There are three ways of training a network:

- ***Supervised Learning***: Train a NN model by presenting sample set $X = \{\boldsymbol{x}_i, i = 1, \dots, N\}$, where each feature vector $\boldsymbol{x}_i$ is paired with a target vector $\boldsymbol{y}_i = [y_{i1}, \dots, y_{ik_L}]^T$, $y_{ij} = \begin{cases} 1, & \text{if } \boldsymbol{x}_i \text{ belongs to class } j = 1, \dots, k = c \\ 0, & \text{elsewhere} \end{cases}$ for classification, and $\boldsymbol{y}_i \in R^{k_L}$ for regression.

- ***Unsupervised Learning***: Train a model with samples X without any targets. This is used by the process of *pre-training* that aims to create an initial *neural network state* that is better than random.

- ***Fine-tuning***: Reuse a neural network model that was created through training on samples X_a, to run some training steps on different samples X_b. This is method implements *transfer learning*.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Training

- Supervised training worths elaborating the most. It consists of the following steps:

  1. Acquire a training dataset: $D_{train} = \{\mathbf{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}, \quad \mathbf{x}_i \in R^n, \boldsymbol{y}_i \in R^{k_L}$

  2. Define the functional form $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})$ and the model parameters vector $\boldsymbol{\theta}$. This step usually involves randomly initializing the parameters $\boldsymbol{\theta}$.

  3. Define a suitable cost (objective) function, that produces a measure of the error between an estimated output $\widehat{\boldsymbol{y}_i} = \boldsymbol{f}(\mathbf{x}_i; \boldsymbol{\theta})$ and the real output $\boldsymbol{y}_i$.

  4. Using an optimization algorithm, search the space of parameters $(\mathbf{W}, \mathbf{b})$ for a set of values that minimizes the cost function.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Training – Objective functions

- **Mean Square Error (MSE)** (suitable for regression and classification):

$$J(\boldsymbol{\theta}) = J(\mathbf{W}, \mathbf{b}) = \frac{1}{N}\sum_{i=1}^{N} \|\widehat{\boldsymbol{y}}_i - \boldsymbol{y}_i\|^2$$

- **Categorical Cross Entropy Error** (Suitable for classifiers that use softmax output layers):

$$J_{CCE} = -\sum_{i=1}^{N}\sum_{\mathrm{j}=1}^{\mathrm{C}} \boldsymbol{y}_{ij} \log\left(\widehat{\boldsymbol{y}}_{ij}\right)$$

- **Exponential Loss:**

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{N} e^{-\beta y_i \widehat{y_i}}$$

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Training – Gradient Based Learning

- Having a cost function $J(\mathbf{W}, \mathbf{b})$, the equations $\frac{\partial J}{\partial \mathbf{W}} = \frac{\partial J}{\partial \mathbf{b}} = 0$ can provide the critical points (minima, maxima, saddle points).
- Analytical computations of this kind are usually impossible.
- Need to resort to numerical optimization methods.
- Iteratively search the parameter space for the optimal values. In each iteration, apply a small correction to the previous value.

$$\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) + \Delta\boldsymbol{\theta}$$

- In the neural network case, jointly optimize $(\mathbf{W}, \mathbf{b})$

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \Delta\mathbf{W}$$

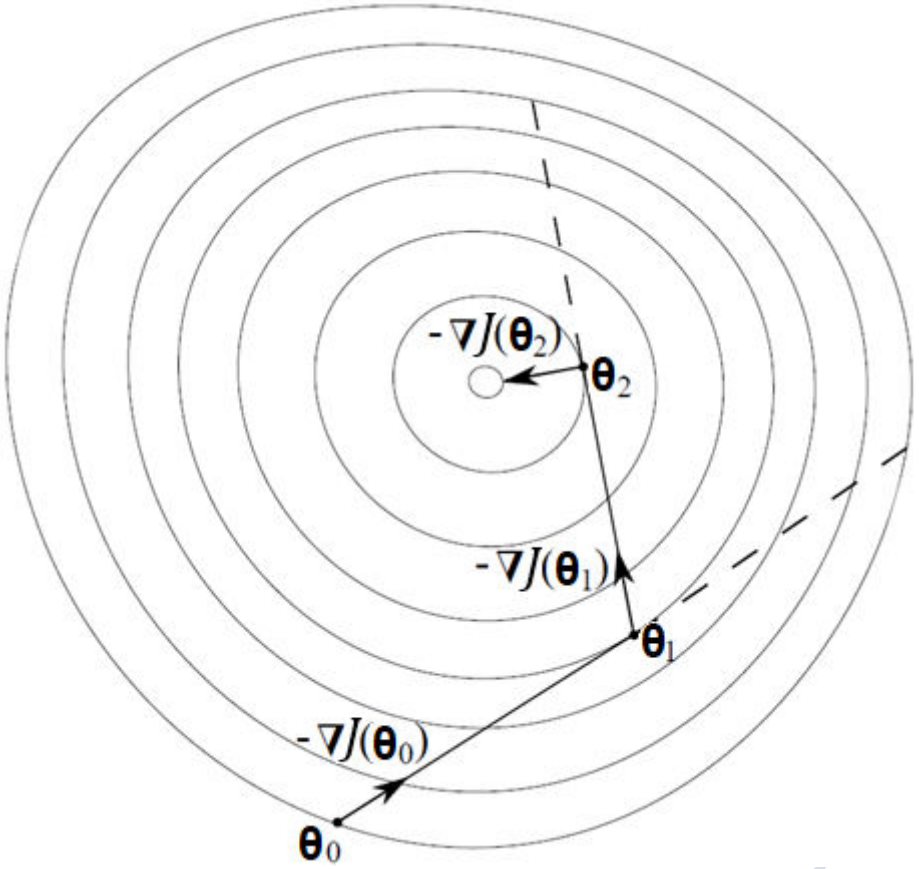
$$\mathbf{b}(t+1) = \mathbf{b}(t) + \Delta\mathbf{b}$$

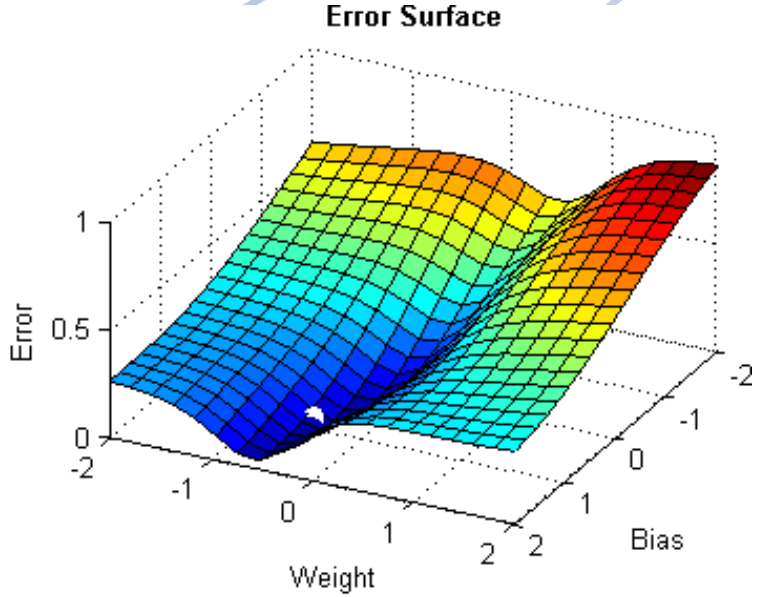
- What is an appropriate choice for the correction term?

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Training – Gradient Descent (GD)

Steepest descent on a function surface





This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Training – Gradient Descent (GD)

- In the Neural Network case: $\mathbf{W}(t+1) = \mathbf{W}(t) - \mu \nabla_{\mathbf{W}} J(\mathbf{W}, \mathbf{b})$

$$\mathbf{b}(t+1) = \mathbf{b}(t) - \mu \nabla_{\mathbf{b}} J(\mathbf{W}, \mathbf{b})$$

- Less compactly, the equations can be rewritten for each neuron, in each layer of the neural network separately. For the j^{th} neuron in the l^{th} layer:

$$\mathbf{w}_j^{(l)}(t+1) = \mathbf{w}_j^{(l)}(t) - \mu \frac{\partial J}{\partial \mathbf{w}_j^{(l)}} = \left[w_{j1}^{(l)}(t) - \mu \frac{\partial J}{\partial w_{j1}^{(l)}}, \cdots, w_{jk_{l-1}}^{(l)}(t) - \mu \frac{\partial J}{\partial w_{jk_{l-1}}^{(l)}} \right]^T$$

$$b_j^{(l)}(t+1) = b_j^{(l)}(t) - \mu \frac{\partial J}{\partial b_j^{(l)}}$$

- Essentially, we need to compute the amount of change ΔJ caused to the cost function by a small change $\Delta w, \Delta b$ in the model parameters, and move towards the direction this change is negative.

- A huge number of partial derivatives to be computed, even for moderate networks.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Training – Backpropagation (BP)

- It is an elegant way of computing the partial derivatives of the cost function w.r.t. the network parameters $(\mathbf{W}, \mathbf{b})$.
- Define the error of the j^{th} neuron in the l^{th} layer as: $\delta_j^{(l)} \triangleq \frac{\partial J}{\partial z_j^{(l)}}$
- The ultimate goal is to compute this partial derivative for every neuron in every layer and associate it with the partial derivatives $\frac{\partial J}{\partial b_j^{(l)}}, \frac{\partial J}{\partial w_{jk}^{(l)}}$.
- To achieve this we need the following expressions:
  1. An equation for the error in the output layer $\boldsymbol{\delta}^{(L)}$.
  2. An equation for the error $\boldsymbol{\delta}^{(l)}$ in terms of the error in the next layer $\boldsymbol{\delta}^{(l+1)}$.
  3. An equation for the rate of change of the cost w.r.t. any bias in the network.
  4. An equation for the rate of change of the cost w.r.t. any weight in the network.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MultiDrone

MLP Training – Backpropagation (BP) (cont.)

- Starting from the last layer, we have:

$$\delta_j^{(L)} \triangleq \frac{\partial J}{\partial z_j^{(L)}} = \frac{\partial J}{\partial a_j^{(L)}} \frac{df\left(z_j^{(L)}\right)}{dz_j^{(L)}}, \qquad a_j^{(L)} = f(\mathbf{w}_j^{(L)T} \boldsymbol{\alpha}^{(L-1)} + b_j^{(L)}) = f\left(z_j^{(L)}\right)$$

- Measures how rapidly the cost is changing as a function of the j^{th} output. Second term measures how fast activation function is changing w.r.t. its argument.

- For the quadratic (MSE) cost function $J = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - a_i^{(L)}\right)^2$, the partial derivative is trivial to compute:

$$\frac{\partial J}{\partial a_j^{(L)}} = y_j - a_j^{(L)}$$

- In matrix-based notation, the above equation can be expressed as: $\boldsymbol{\delta}^{(L)} = \nabla_\alpha \circ \frac{df\left(z_j^{(L)}\right)}{dz_j^{(L)}}$ (BP1)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Training – Backpropagation (BP) (cont.)

- Next, we extract the formula for that associates the errors in successive layers. Using the **chain rule** of calculus:

$$\delta_j^{(l)} = \frac{\partial J}{\partial z_j^{(l)}} = \sum_k \frac{\partial J}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \sum_k \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} \delta_k^{(l+1)}$$

since $z_k^{(l+1)} = \sum_j w_{kj}^{(l+1)} a_j^{(l)} + b_k^{(l+1)} = \sum_j w_{kj}^{(l+1)} f(z_j^{(l)}) + b_k^{(l+1)}$

it is easy to differentiate and obtain $\frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = w_{kj}^{(l+1)} f(z_j^{(l)})$

- Finally, we retrieve the following formula $\delta_j^{(l)} = \sum_k w_{kj}^{(l+1)} \delta_k^{(l+1)} \frac{df(z_j^{(l)})}{dz_j^{(l)}}$.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Training – Backpropagation (BP) (cont.)

- Written in compact, matrix-based notation, the last equation is expressed as:

$$\boldsymbol{\delta}^{(l)} = \left[\mathbf{w}^{(l+1)}\boldsymbol{\delta}^{(l+1)}\right] \circ \frac{df(\mathbf{z}^{(l)})}{d\mathbf{z}^{(l)}} \qquad \text{(BP2)}$$

- (BP2) essentially performs the backwards propagation of the error from the $(l+1)^{th}$ to the l^{th} layer. Combining equations (BP1) and (BP2) we can compute the error in every neuron in every layer.

- To complete the algorithm, we still need the expressions of the partial derivatives of the cost w.r.t. the network parameters. Applying the chain rule:

$$\frac{\partial J}{\partial b_j^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}}\frac{\partial z_k^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \qquad \text{(BP3)}$$

$$\frac{\partial J}{\partial w_{jk}^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}}\frac{\partial z_k^{(l)}}{\partial w_{jk}^{(l)}} = \delta_j^{(l)}\alpha_k^{(l-1)} \qquad \text{(BP4)}$$

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Training – Backpropagation (BP) (cont.)

- We can now state the four essential equations for the implementation of the backpropagation algorithm:

$$\boldsymbol{\delta}^{(L)} = \nabla_{\alpha} J \circ \frac{df(z_j^{(L)})}{dz_j^{(L)}} \qquad \text{(BP1)}$$

$$\boldsymbol{\delta}^{(l)} = [\mathbf{w}^{(l+1)} \boldsymbol{\delta}^{(l+1)}] \circ \frac{df(\mathbf{z}^{(l)})}{d\mathbf{z}^{(l)}} \qquad \text{(BP2)}$$

$$\frac{\partial J}{\partial b_j^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} \frac{\partial z_k^{(l)}}{\partial b_j^{(l)}} = \delta_j^{(l)} \qquad \text{(BP3)}$$

$$\frac{\partial J}{\partial w_{jk}^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} \frac{\partial z_k^{(l)}}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \alpha_k^{(l-1)} \qquad \text{(BP4)}$$

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

MLP Training – Complete Algorithm

Algorithm 1: Neural Network Training with Gradient Descent and Back-propagation

Input: learning rate μ, dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$

Random initialization of network parameters $(\mathbf{W}, \mathbf{b})$

Define cost function $J = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$

while *optimization criteria not met* **do**

- $J = 0$
- **Forward pass of whole batch** $\mathbf{y} = f_{NN}(\mathbf{x}; \mathbf{W}, \mathbf{b})$
- $J = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
- $\boldsymbol{\delta}^{(l)} = [\mathbf{w}^{(l+1)}\boldsymbol{\delta}^{(l+1)}] \circ \frac{df(\mathbf{z}^{(l)})}{d\mathbf{z}^{(l)}} \quad \forall l = 1, \ldots, L$
- $\frac{\partial J}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} \alpha_k^{(l-1)} \quad \forall l = 1, \ldots, L$
- $\frac{\partial J}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad \forall l = 1, \ldots, L$
- $\mathbf{W} \leftarrow \mathbf{W} - \mu \nabla_{\mathbf{W}} J(\mathbf{W}, \mathbf{b})$
- $\mathbf{b} \leftarrow \mathbf{b} - \mu \nabla_{\mathbf{b}} J(\mathbf{W}, \mathbf{b})$

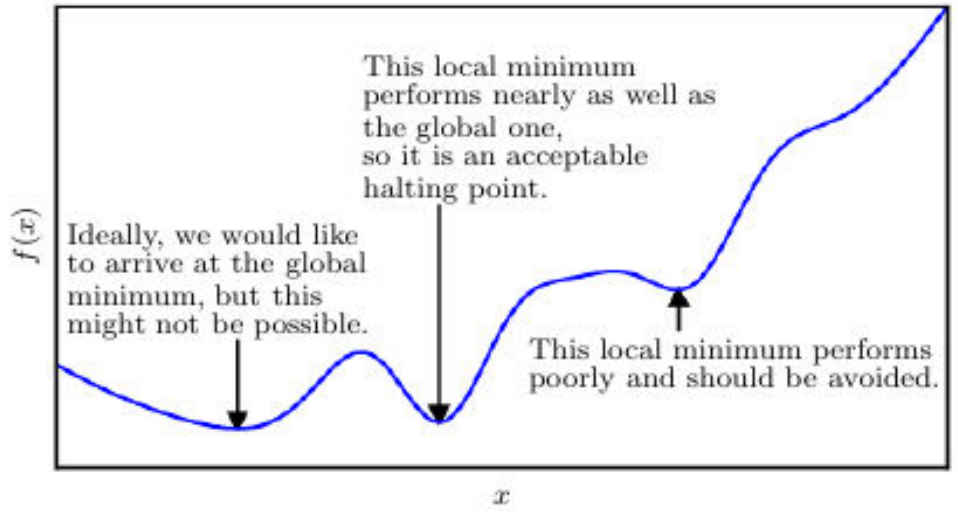
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

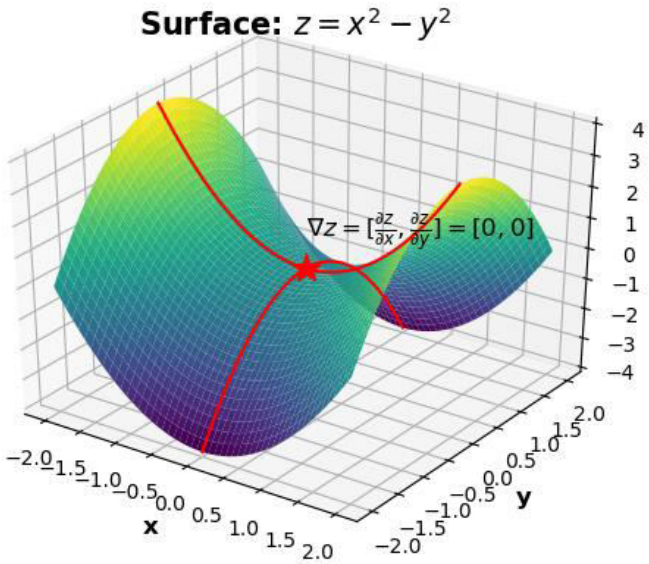
Optimization Review – Gradient Descent

- **Batch Gradient Descent**:

1. In order to compute the gradients, the whole dataset must be fed to the network at once. A computationally inefficient practice, especially for datasets numbering millions/billions of samples.

2. Can be trapped in saddle points or local minima.





This project has received funding from the European Union's Horizon 2020 and innovation programme under grant agreement No 731667 (MU

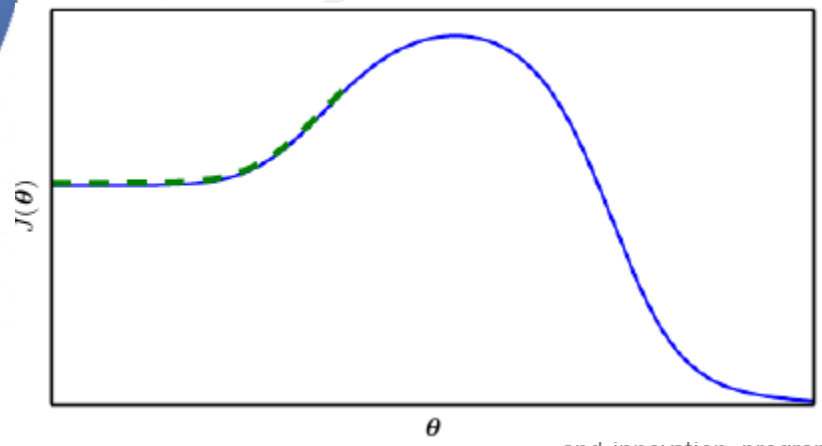
Optimization Review – Gradient Descent (cont)

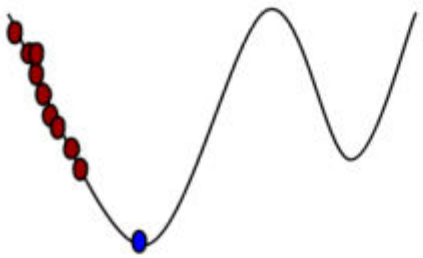
- **Batch Gradient Descent**:

3) Ill conditioning: Takes into account only first – order information (1^{st} derivative). No information about curvature of cost function may lead to zig-zagging motions during optimization, resulting in very slow convergence.

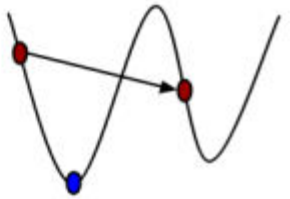
4) May depend too strongly on the initialization of the parameters.

5) Learning rate must be chosen very carefully.

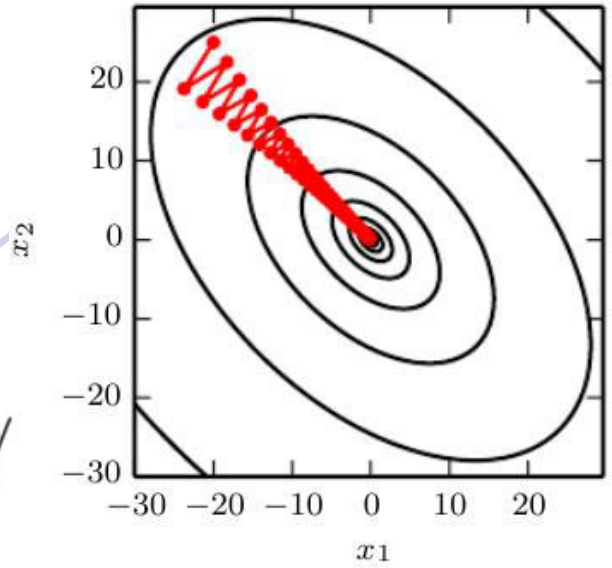




very small learning rate needs lots of steps



too big learning rate: missed the minimum



from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Optimization Review – Alternatives

- **Stochastic Gradient Descent (SGD)**:

1. Passing the whole dataset through the network just for a small update in the parameters is inefficient.

2. Most cost functions used in Machine Learning and Deep Learning applications can be decomposed into a sum over training samples. Instead of computing the gradient through the whole training set, use a small sample (mini-**batch**) to estimate it.

$$\frac{\sum_{s=1}^{S} \nabla J_{\mathbf{x}_s}}{S} \approx \frac{\sum_{x} \nabla J_{x}}{N} = \nabla J \Rightarrow \nabla J \approx \frac{1}{S} \sum_{s=1}^{S} \nabla J_{\mathbf{x}_s}$$

3. Therefore, we can restate the parameter update rules as follows:

$$\mathbf{w}_j^{(l)}(t+1) = \mathbf{w}_j^{(l)}(t) - \frac{\mu}{S} \sum_{s} \frac{\partial J_{\mathbf{x}_s}}{\partial \mathbf{w}_j^{(l)}} \qquad b_j^{(l)}(t+1) = b_j^{(l)}(t) - \frac{\mu}{S} \sum_{s} \frac{\partial J_{\mathbf{x}_s}}{\partial b_j^{(l)}}$$

4. Using all the batches to update the parameters once is called a training **epoch**. Usually, multiple epochs are used when training a network.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Optimization Review – Alternatives

- **Stochastic Gradient Descent with momentum (SGD-m)**

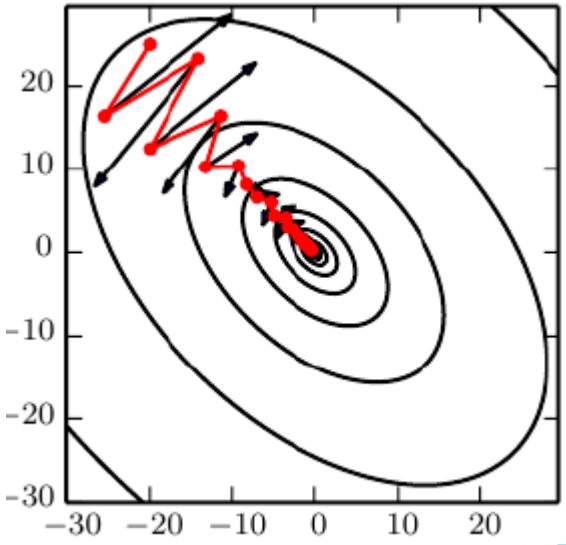
1. To overcome the various difficulties associated with the learning rate μ, a momentum term is utilized, especially to speed up learning and to combat the noise introduced by estimating the gradient by a small number of samples.

2. SGD-m accumulates an exponentially decaying moving average of past gradients in order to compute the descend direction.

$$\boldsymbol{u} \leftarrow a\boldsymbol{u} - \mu\nabla_{\theta}\left(\frac{1}{m}\sum_{i=1}^{m} J\left(f\left(\boldsymbol{x}^{(i)};\boldsymbol{\theta}^{(i)}\right),\boldsymbol{y}_i\right)\right)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + u$$

1. SGD-m performs demonstrably better than non-momentum gradient descent in ill-conditioned problems.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Optimization Review – Alternatives

- **Adaptive Learning Rate Algorithms**:
  - Up to this point, the learning rate was given as an input to the algorithm and was kept stable throughout the whole optimization procedure.
  - To overcome the difficulties of using a set rate, adaptive rate algorithms assign a different learning rate for each separate model parameter.
    1. <u>AdaGrad algorithm:</u>
       i. Gradient accumulation variable $\mathbf{r} = 0$
       ii. Compute and accumulate gradient $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i J\left(f\left(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(i)}\right), \boldsymbol{y}_i\right), \mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \circ \mathbf{g}$
       iii. Compute update $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \circ \mathbf{g}$ and apply update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
       iv. Parameters with largest partial derivative of the loss have a more rapid decrease in their learning rate.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Optimization Review – Alternatives

- **Adaptive Learning Rate Algorithms**:.

2. RMSProp algorithm:

i. RMSProp builds on AdaGrad and introduces exponentially weighted moving average.

ii. Accumulation variable is updated according to $\mathbf{r} \leftarrow \rho\mathbf{r} + (1-\rho)\mathbf{g} \circ \mathbf{g}$

iii. The rest of the steps remain the same.

iv. Outperforms AdaGrad in non-convex settings, by identifying locally convex bowls (possible local minima locations).

v. One of the go-to optimization algorithms for modern Deep Neural Networks.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Optimization Review – Alternatives

- **Second Order Methods**: Second order methods incorporate information from second-order derivatives in the optimization process. This allows primarily accounting for the loss function convexity.

1. Newton's Method:

i. Newton's method is the most well-known second order method. Taking the second-order Taylor expansion of the cost function, discarding higher order terms and solving for the optimum, we arrive at the following update rule:

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \mathbf{H} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

$\Delta\boldsymbol{\theta} = -\mathbf{H}^{-1}\mathbf{g}$, where $\mathbf{H}$ is the Hessian matrix.

ii. Computing the inverse Hessian matrix for every update can be computationally infeasible for large numbers of parameters.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Generalization

- Good performance on the training set alone is not a satisfying indicator of a good model.
- We are interested in building models that can perform well on data they have not been used the training process. Bad generalization capabilities mainly occur through 2 forms:

1. Overfitting:

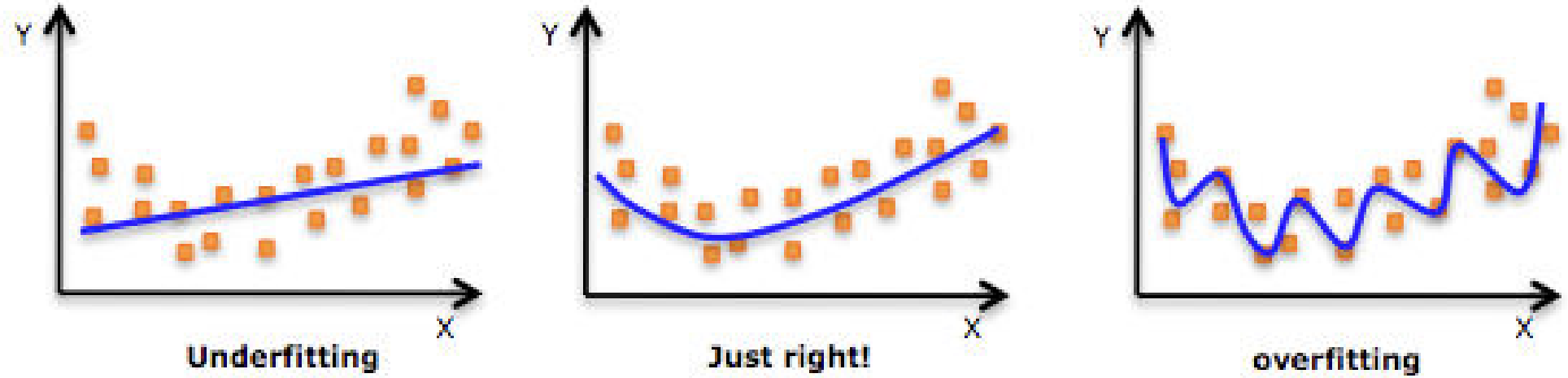
   - Overfitting occurs when the complexity of our model is so high that it memorizes the entire training dataset. This results in the model corresponding to that particular set and failing to produce accurate results in any other set of data drawn from the same distribution.
   - Overfitting can be detected when there is great discrepancy between the performance of the model in the training set, and that in any other set.
   - An example of overfitting would be trying to model a quadratic equation with a 10-th degree polynomial.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Generalization

2. Underfitting:

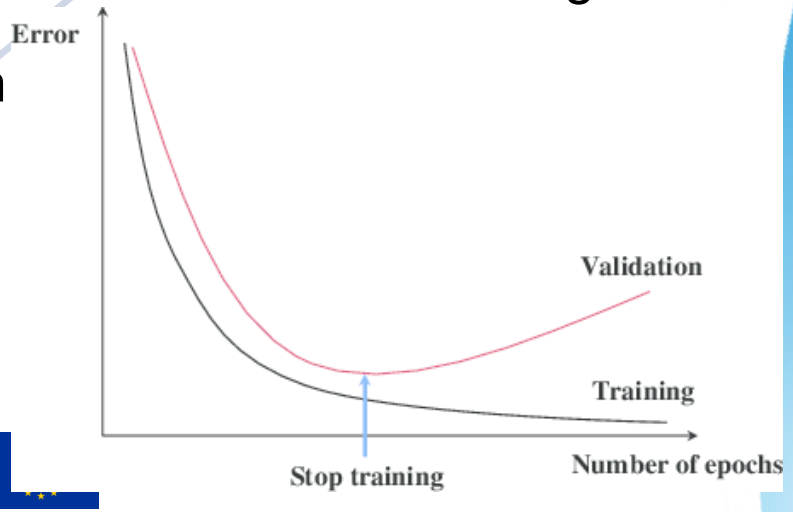
- On the other hand, underfitting occurs when a model cannot accurately capture the underlying structrure of data.
- Underfitting can be detected by a very low performance in the training set.
- Example: trying to model a non-linear equation using linear ones.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Generalization – Early Stopping

- One of the most common ways to improve generalization performance in neural networks is early stopping.
- Given an initial dataset $D = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$, split it into three dijoint subsets, called training, validation and testing.
- While training the neural network using the training subset, use the validation subset to measure its performance.
- Stop the training procedure when the error on the validation set starts increasing.
- Test the NN at the testing set to estimate generalization performance.
- The whole process uses the validation error as a proxy for the generalization performance.



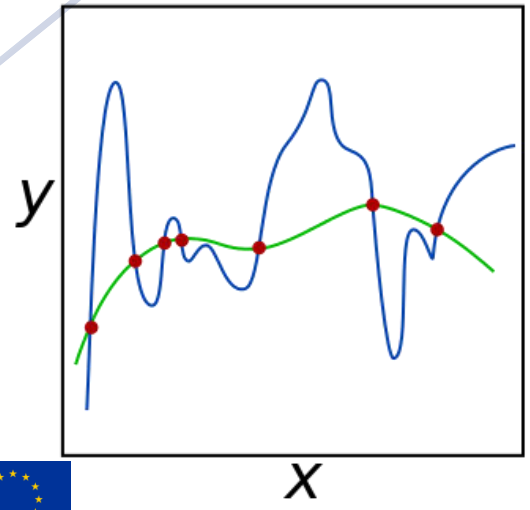
This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Generalization – Regularization

- Another approach to enhancing the generalization performance is regularization.
- One way to regularize a neural network (and any learning machine) is to add an extra term $\Omega(\boldsymbol{\theta})$ in the cost function, penalizing large values on the parameters. The new cost function is: $J_{reg} = J + \frac{\lambda}{N}\Omega(\boldsymbol{\theta})$.
- λ is a regularization parameter controlling the trade – off between minimizing the cost function and penalizing large parameters. Depending on the functional form of $\Omega(\cdot)$, the effect on the model parameters is different.

1. L_2 - Regularization: $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|^2 = \sum_\theta \theta^2$.

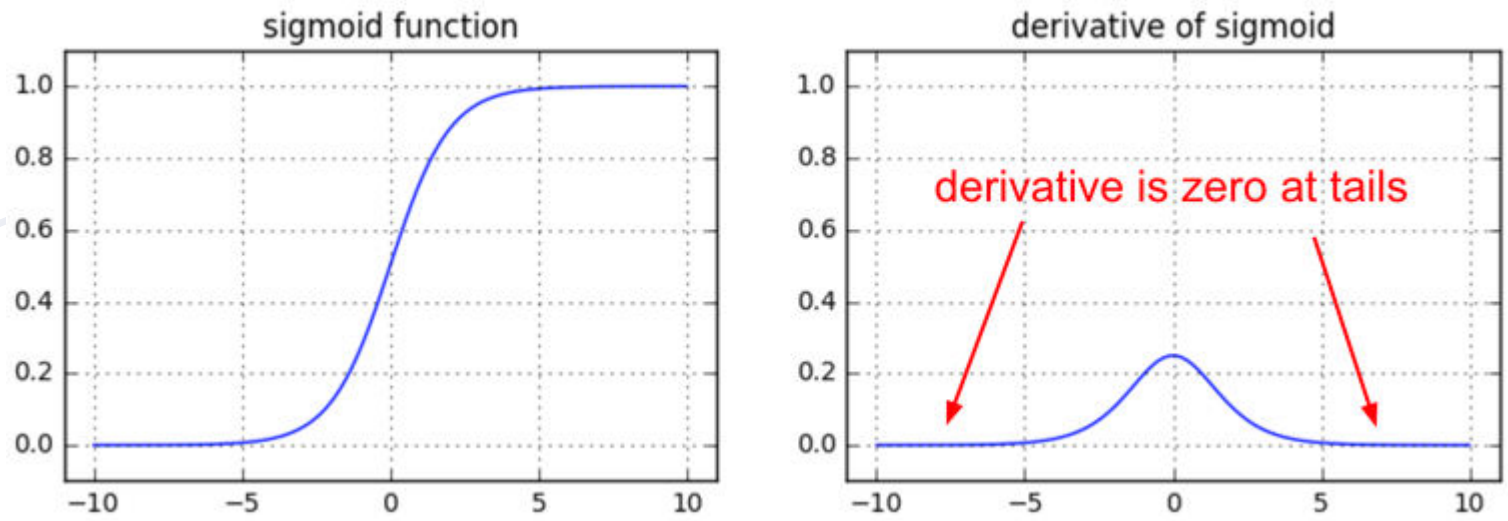
1. L_1 - Regularization: $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\| = \sum_\theta |\theta|$.



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Revisiting Activation Functions

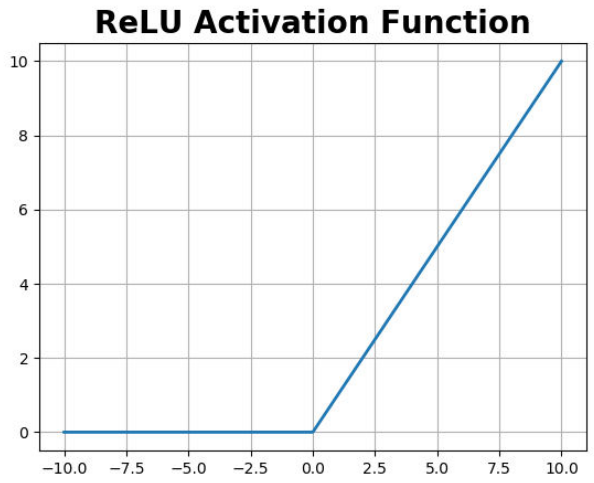
- Sigmoid function -until recently, was the default choice for activation function in neuron
- Sigmoid functions saturate and can lead to a problem known as *vanishing gradients.*
- Zero gradients effectively mean that the network stops learning.

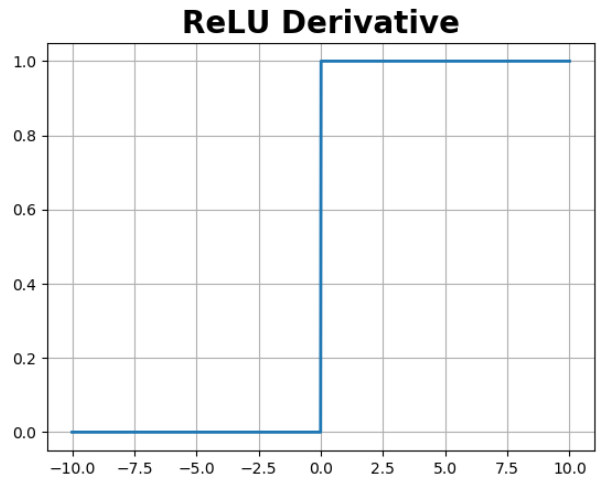


This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Revisiting Activation Functions

- To overcome the shortcomings of the sigmoid activation function, the Rectified Linear Unit (ReLU) was introduced.
- Very easy to compute.
- Due to its quasi-linear form leads to faster convergence (does not saturate at high values)
- Units with zero derivative will not activate at all – may lead to regularization effect





Training on Large Scale Datasets

- Large number of training samples in the magnitude of hundreds of thousands.
  - Problem: Datasets do not fit in memory.
  - Solution: Using mini-batch SGD method.

- Many classes, in the magnitude of hundreds up to one thousand.
  - Problem: Difficult to converge using MSE error.
  - Solution: Using Categorical Cross Entropy (CCE) loss on Softmax output.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Towards Deep Learning

- Increasing the network depth (layer number) L can result in negligible weight updates in the first layers, because the corresponding deltas become very small or vanish completely
  - Problem: Vanishing gradients.
  - Solution: Replacing sigmoid with an activation function without an upper bound, like a rectifier (a.k.a. ramp function, ReLU).
- Full connectivity has high demands for memory and computations
- Very deep fully connected DNNs are difficult to implement.
- New architectures come into play (Convolutional Neural Networks, Deep Autoencoders etc.)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)

Q & A

MultiDrone

Thank you very much for your attention!

Contact: Prof. I. Pitas
pitas@csd.auth.gr
www.multidrone.eu

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731667 (MULTIDRONE)