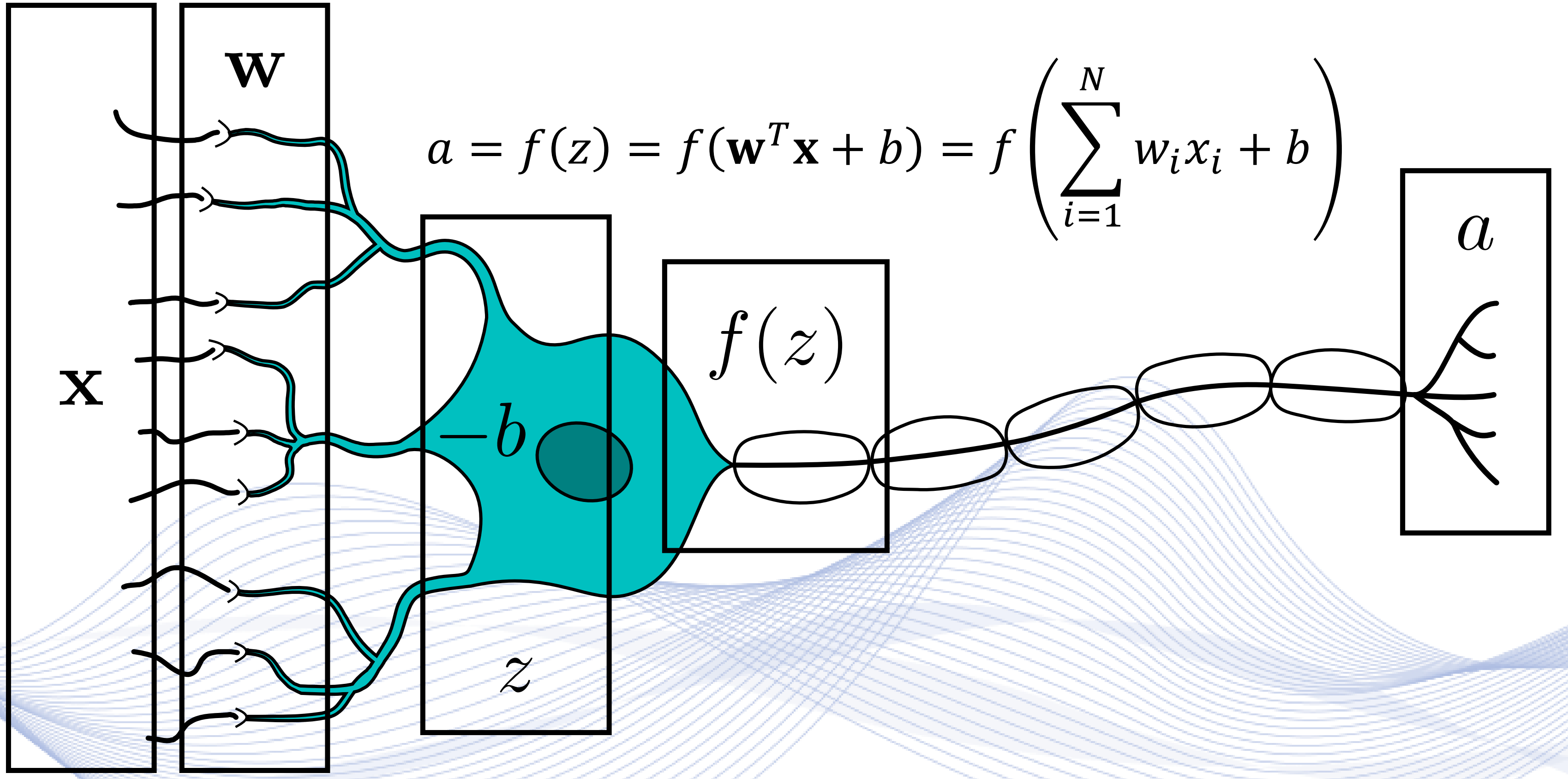
Multilayer Perceptron. Backpropagation summary

Prof. Ioannis Pitas
Aristotle University of Thessaloniki
pitas@csd.auth.gr
www.aiia.csd.auth.gr

Version 4.1

Artificial Intelligence & Information Analysis Lab

Perceptron



$\mathbf{w}$
$a = f(z) = f(\mathbf{w}^T\mathbf{x} + b) = f\left(\sum_{i=1}^{N} w_i x_i + b\right)$
$\mathbf{x}$
$-b$
$f(z)$
a
z

Multi-Layer Perceptrons (MLP)

Universal Approximation Theorem:

Let $f(\cdot)$ be a nonconstant, 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 $\mathrm{G}(\mathbf{x}) \in C(H_n)$, there exist an integer N, real constants $u_i, b_i \in \mathbb{R}$ and real vectors $\mathbf{w}_i \in \mathbb{R}^n$, where $i = 1, \dots, N$, such that we may define:

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

as an approximate realization of the function F:

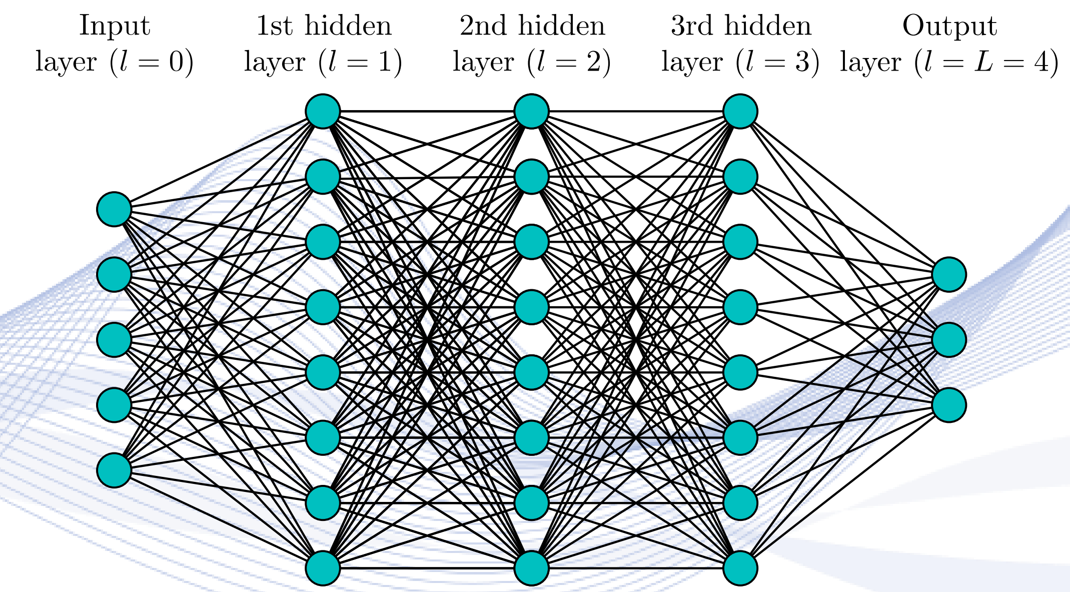
$$|G(\mathbf{x}) - g(\mathbf{x}; \mathbf{w})| < \epsilon, \qquad \forall\, \mathbf{x} \in H_n.$$

Artificial Intelligence & Information Analysis Lab

MLP Architecture

Multilayer perceptrons are ***feed-forward neural networks*** and typically consist of L layers with L_l neurons in each layer: $l = 1, \dots, L$.

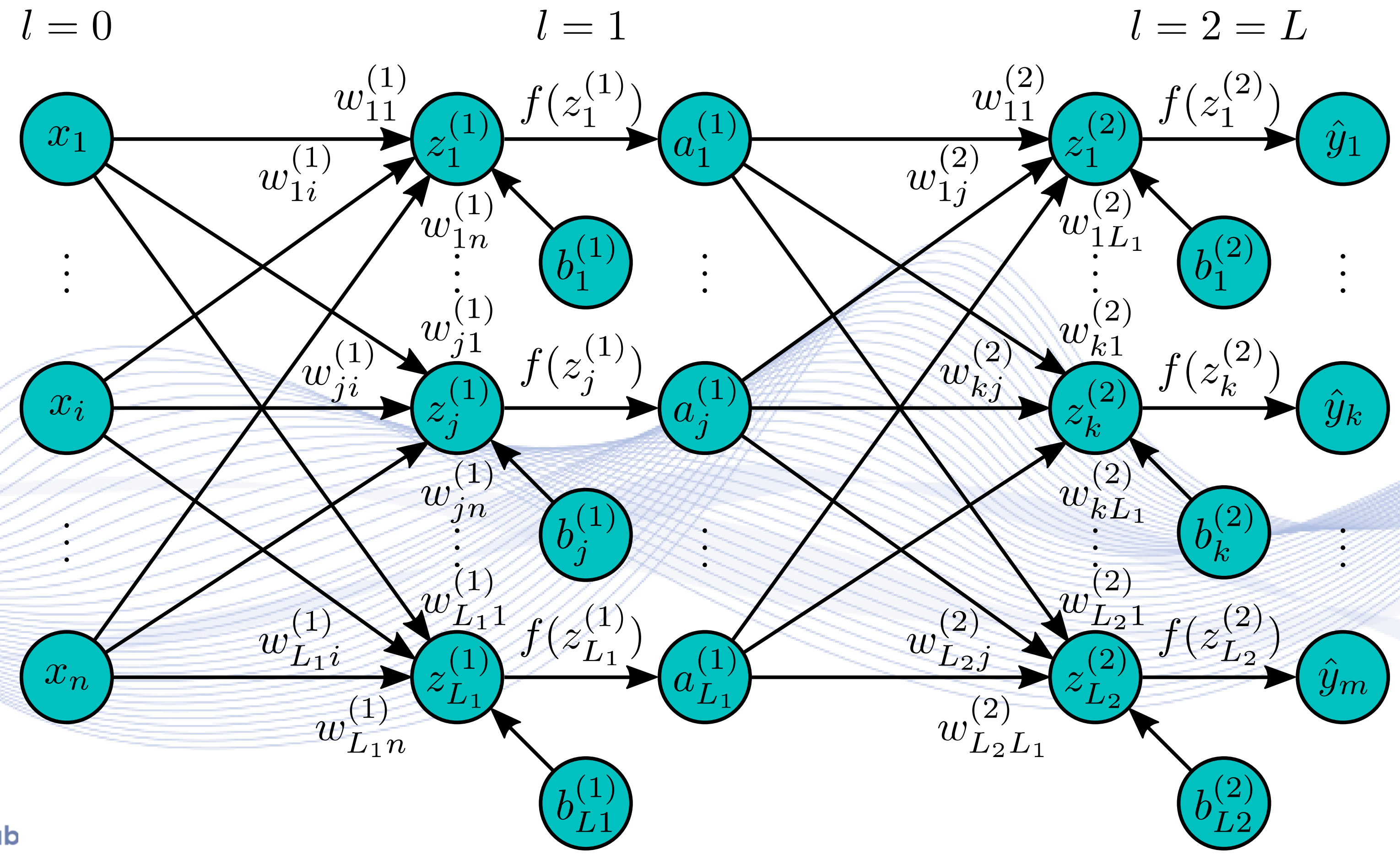
- The first layer (technically layer $l = 0$) contains n inputs, where n is the dimensionality of the input sample vector.
- The $L - 1$ hidden layers $l = 1, \dots, L - 1$ can contain any number of neurons.



MLP with $L = 4$ layers.

Fully connected MLP Example

- Example architecture with $L = 2$ layers, n input features, L_1 neurons at the first layer and m output units.



MLP Training

- ***Mean Square Error (MSE)***:

$$J(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N}\sum_{j=1}^{m}\left(\hat{y}_{ij} - y_{ij}\right)^2.$$

  - It is suitable for regression and classification.

- ***Categorical Cross Entropy Error***:

$$J(\boldsymbol{\theta}) = -\sum_{i=1}^{N}\sum_{\mathrm{j}=1}^{m} y_{ij}\log\left(\hat{y}_{ij}\right).$$

  - It is suitable for classifiers that use softmax output layers.

MLP Training

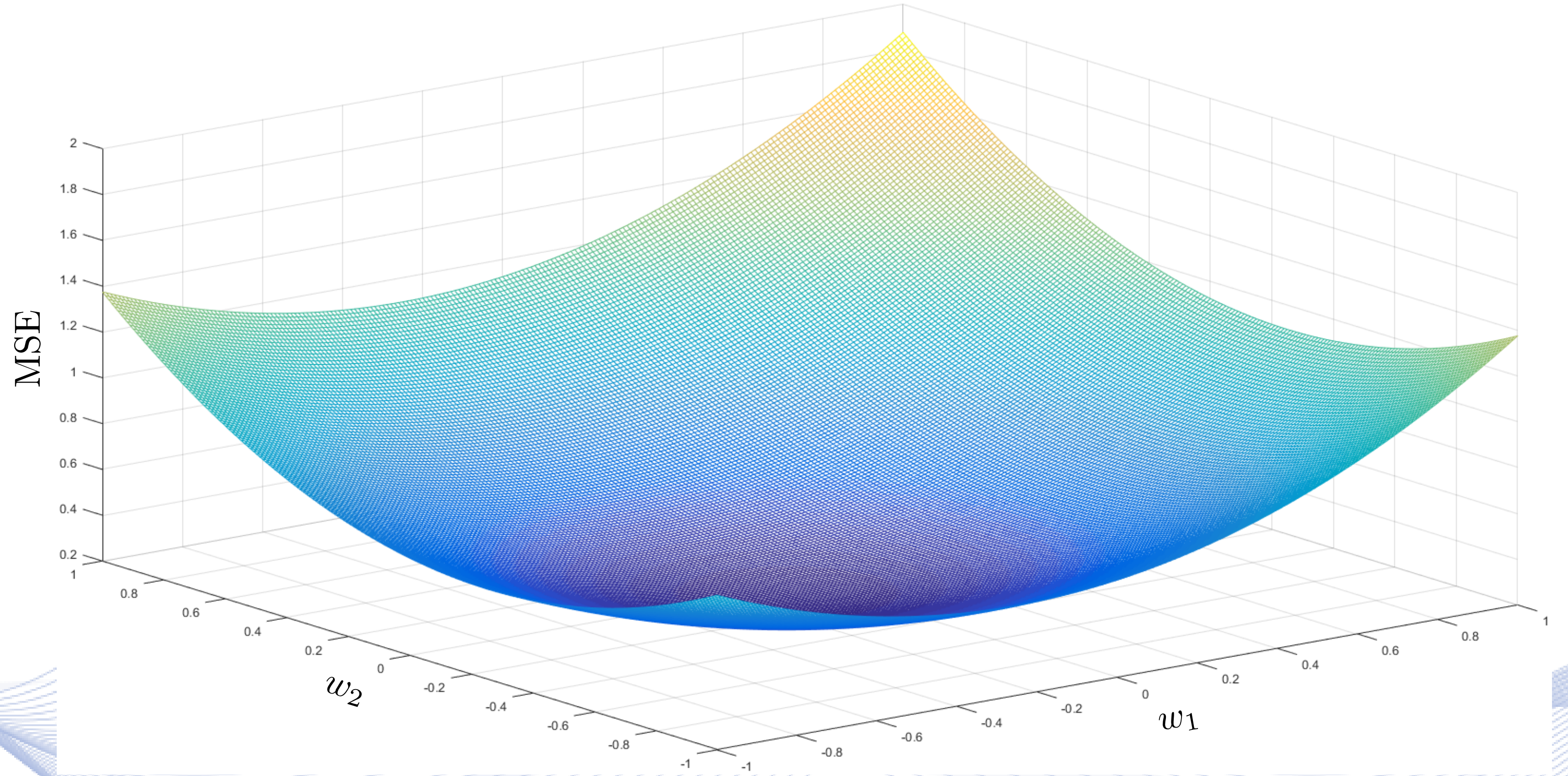
- Differentiation: $\nabla J(\boldsymbol{\theta}) = \mathbf{0}$ can provide the critical points of multivariate function $J(\boldsymbol{\theta})$:

  Minima, ***maxima*** and ***saddle points***.

- Analytical differentiation is usually impossible.
- We must resort to numerical optimization methods.
- Iteratively search the parameter space for the optimal values.
- In gradient descent, weights are update in the opposite direction of the gradient, factored by the *learning rate* η:

$$\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) + \eta \nabla J(\boldsymbol{\theta}).$$

MLP Training



Steepest descent on a function surface.

Backpropagation

- Consider the MSE objective function for a single input vector:

$$J = \frac{1}{2}\sum_{i=1}^{m}(y_i - a_i^{(L)})^2 .$$

- We introduce the notation for the *delta rule* as:

$$\delta_i^{(l)} = \frac{\partial J}{\partial z_i^{(l)}} = \frac{\partial J}{\partial a_i^{(l)}}\frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} .$$

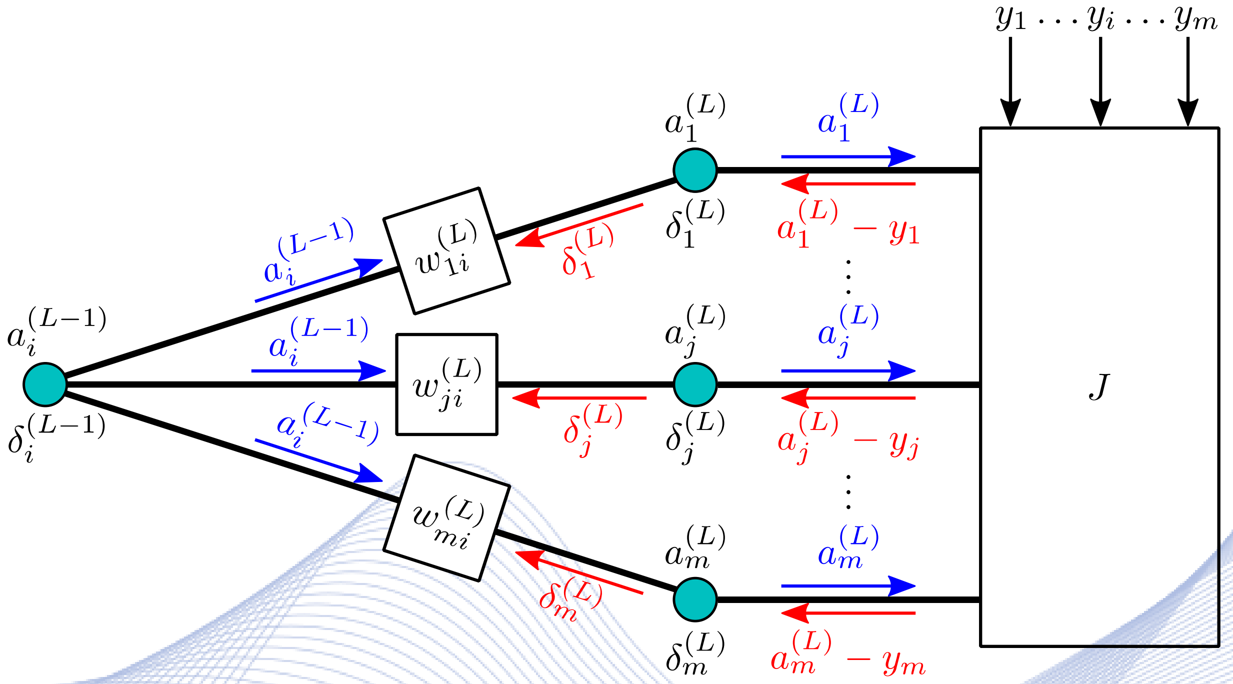
- The sample is first fed forward throughout the network and the neuron outputs are stored. The computation of the error starts at the output and propagates backwards.

Backpropagation

- Overall:

$$\frac{\partial J}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)},$$

$$\delta_i^{(l)} = \frac{\partial a_i^{(l)}}{\partial z_i^{(l)}} \begin{cases} \left(a_i^{(L)} - y_i\right), l = L, \\ \sum_{j=1}^{L_{l+1}} \delta_j^{(l+1)} w_{ji}^{(l+1)}, l < L. \end{cases}$$



Gradient Descent Overview

When multiple training samples are available, parameter updates in the Backpropagation algorithm can be applied in three different ways:

- **Batch Gradient Descent**, where the gradients are computed for each sample of the training set $\{(\mathbf{x}_i, \mathbf{y}_i), i = 1, \ldots N\}$, and then the update rule becomes:

$$\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) - \eta \nabla J(\boldsymbol{\theta}(t))$$

$$\nabla J(\boldsymbol{\theta}(t)) = \frac{\sum_{i=1}^{N} \nabla J_{\mathbf{x}_i}(\boldsymbol{\theta}(t))}{N},$$

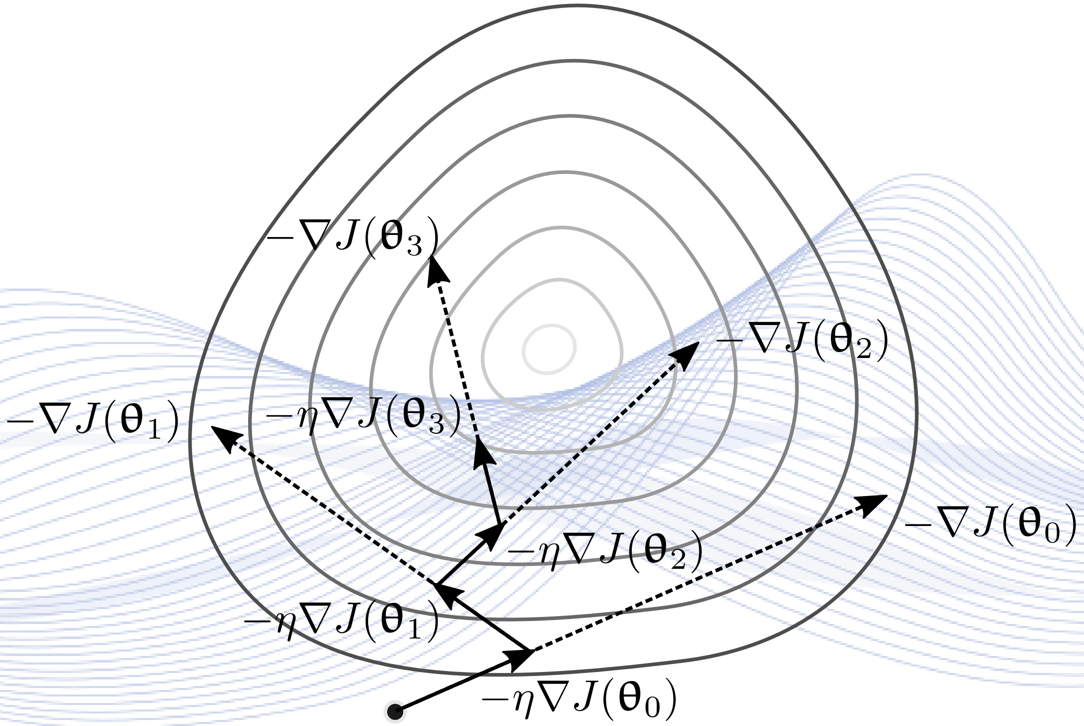
  - $J_{\mathbf{x}_i}$ is the value of cost function J when given input sample $\mathbf{x}_i$.

- **Stochastic Gradient Descent**, where the parameters are updated for every training sample:

$$\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) - \eta \nabla J_{\mathbf{x}_i}(\boldsymbol{\theta}(t)).$$

Gradient Descent Overview

- A training ***epoch*** is a complete cycle of training, in which all training samples have been processed once.
- Usually, multiple epochs are used when training a network, for better convergence.

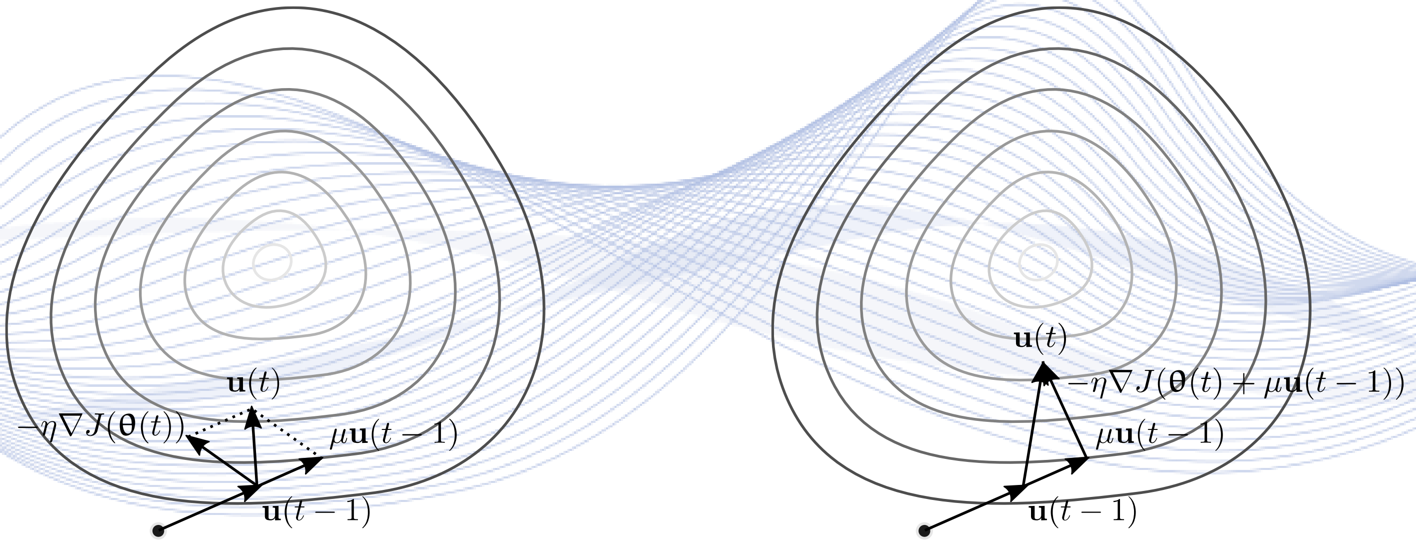


Gradient Descent Overview

- A more refined and popular momentum approach is the Nesterov momentum method, in which the momentum is applied first and then the gradient is computed:

$$\boldsymbol{u}(t) = \mu\boldsymbol{u}(t-1) - \eta\nabla J\big(\boldsymbol{\theta}(t) + \mu\boldsymbol{u}(t-1)\big),$$

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



Artificial Intelligence & Information Analysis Lab

Gradient Descent Overview

***AdaGrad algorithm*:**

- Maintains the sum of squares of all previous gradients.

$$\eta(t) = \frac{\eta(0)}{\sqrt{\sum_{i=1}^{t-1}\left(\frac{\partial J}{\partial \theta}(i)\right)^2 + \epsilon}}$$

- The learning rate decreases faster for more frequently updated parameters.

- The problem is that eventually the learning rate vanishes and the training stops.

Gradient Descent Overview

ADAM algorithm:

$$u(t) = \frac{\beta_1 u(t) + (1 - \beta_1)\frac{\partial J}{\partial \theta}(t)}{1 - \beta_1^t},$$

$$r(t) = \frac{\beta_2 r(t) + (1 - \beta_2)\left(\frac{\partial J}{\partial \theta}(t)\right)^2}{1 - \beta_2^t},$$

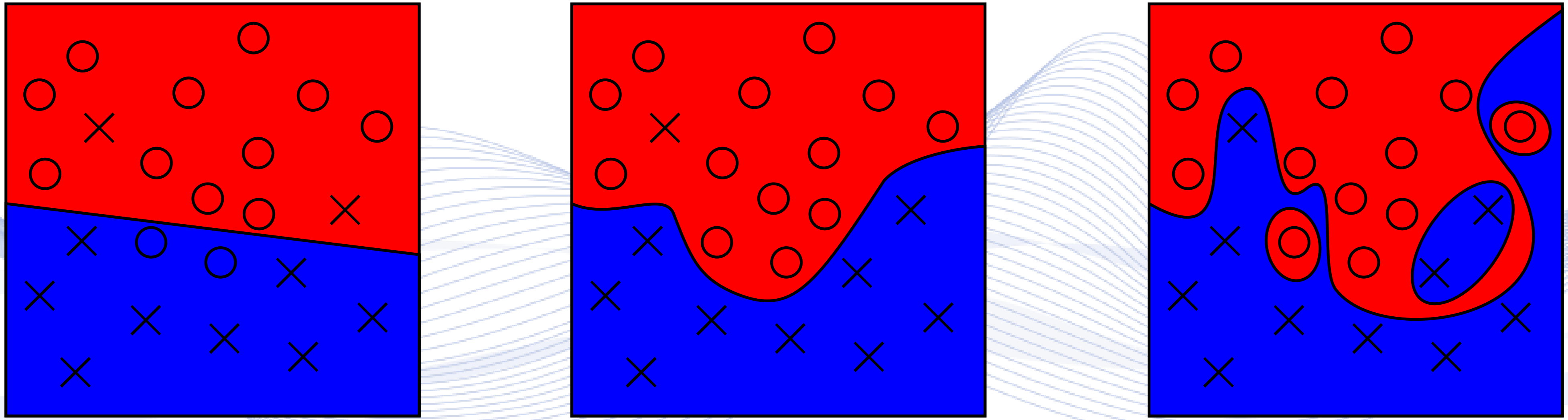
$$\theta(t) = \theta(t-1) - \frac{\eta}{\sqrt{r(t) + \epsilon}} u(t).$$

- β_1, β_2: exponential decay rates for the moment estimates.
- u, r: first and second moment values respectively.

Generalization

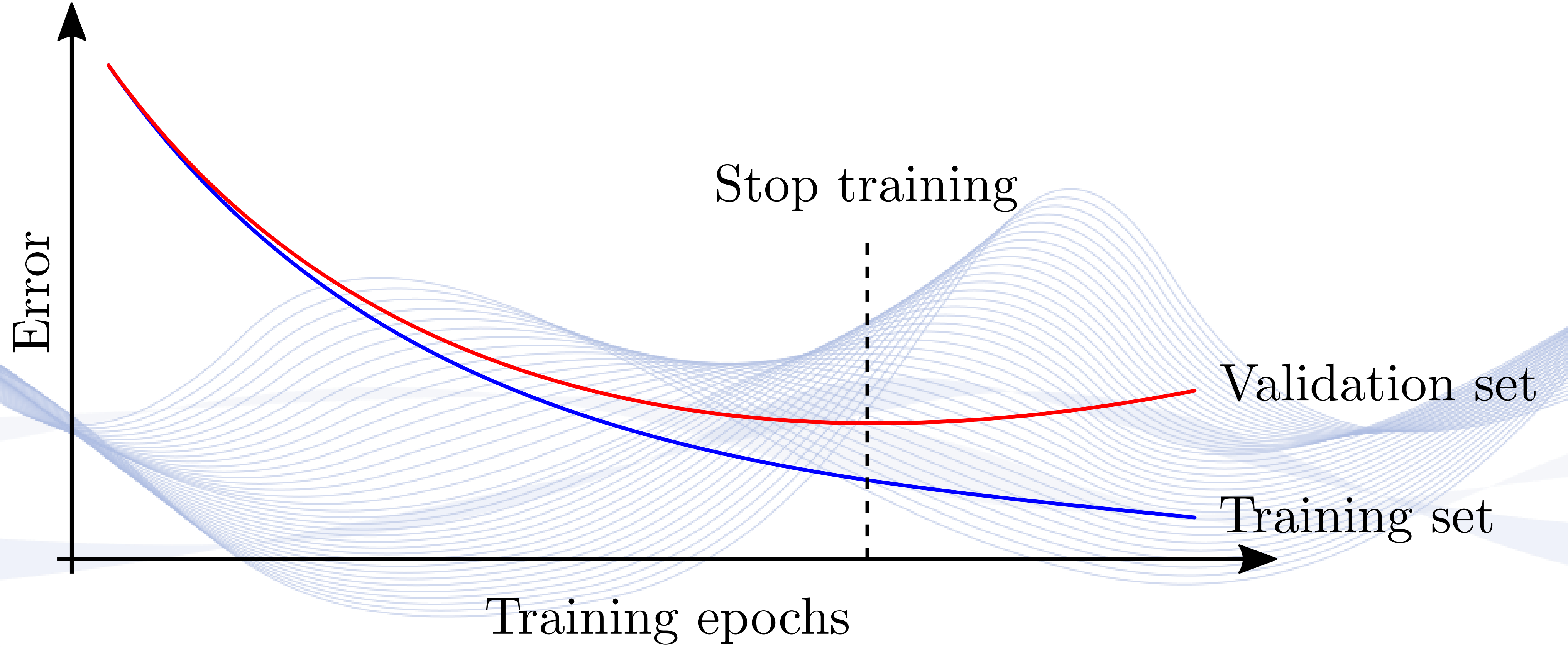
Underfitting occurs when a model cannot accurately capture the underlying data structure.

- Underfitting can be detected by a very low performance in the training set.



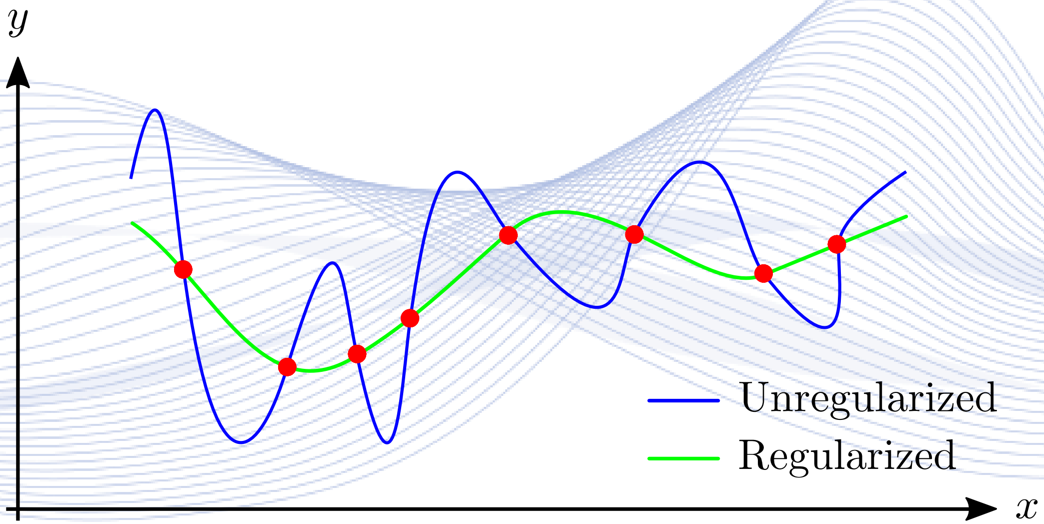
Generalization

- The whole process uses the validation error as a proxy for the generalization performance.



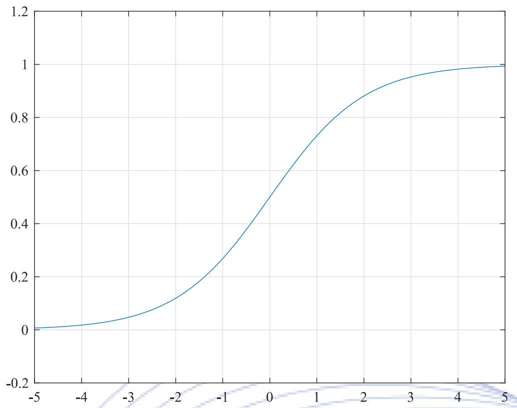
Generalization

- Depending on the functional form of $\Omega(\cdot)$, the effect on the model parameters is different:
  - L_2 regularization: $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|^2 = \sum_i \theta_i^2$.
  - L_1 regularization: $\Omega(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\| = \sum_i |\theta_i|$.

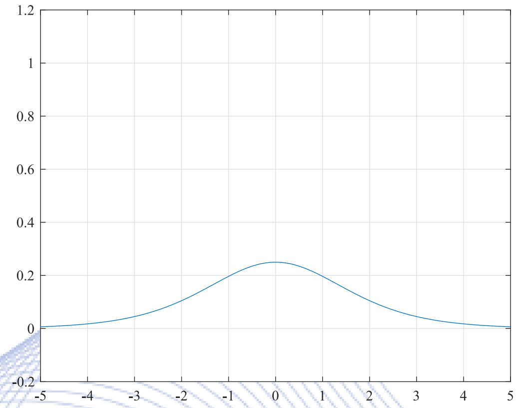


Revisiting Activation Functions

- Sigmoid function, until recently, was the default choice for activation function.
- Sigmoid functions saturate, which can prevent some neurons from updating.



Sigmoid



Derivative of sigmoid

- Sigmoid functions lead to the ***vanishing gradients problem***, as the delta signal is repeatedly multiplied by a value smaller than 1. Near zero gradients effectively mean that earlier layers stop learning.

Artificial Intelligence & Information Analysis Lab

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.

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.)

Q & A

Thank you very much for your attention!

Contact: Prof. I. Pitas
pitas@csd.auth.gr