Day 1 Lecture 4

Multilayer Perceptron

Elisa Sayrol

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Department of Signal Theory
and Communications
Image Processing Group

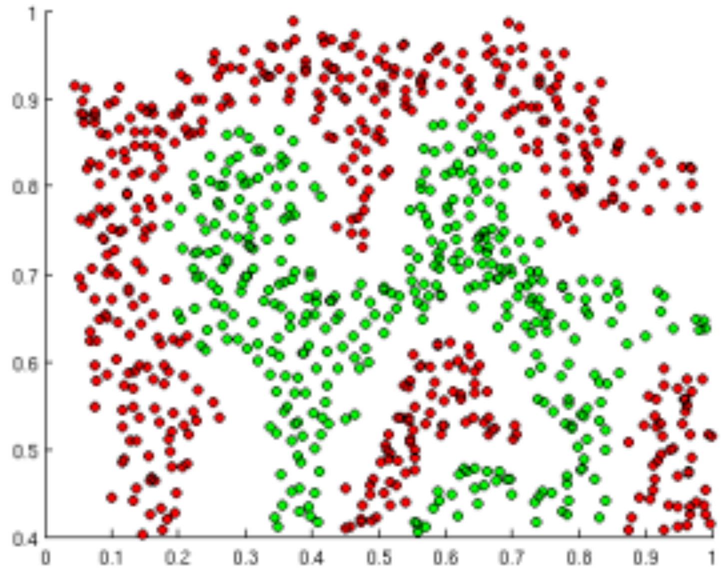
Acknowledgements: To my colleagues of this seminar and previous ones

Non-linear decision boundaries

Linear models can only produce linear decision boundaries

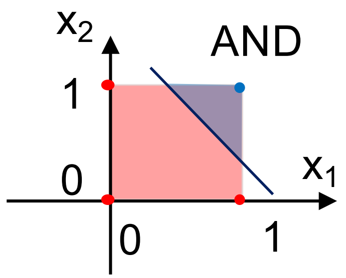
Real world data often needs a non-linear decision boundary

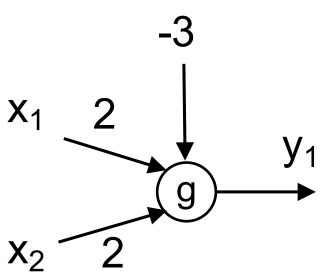
- Images
- Audio
- Text



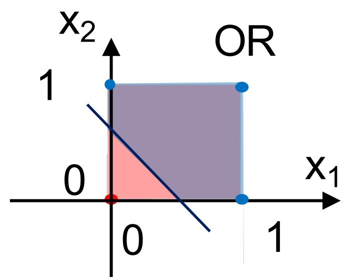
Example: X-OR.

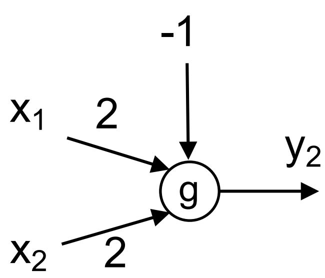
AND and OR can be generated with a single perceptron





Input vector (x1,x2)	Class AND
(0,0)	0
(0,1)	0
(1,0)	0
(1,1)	1



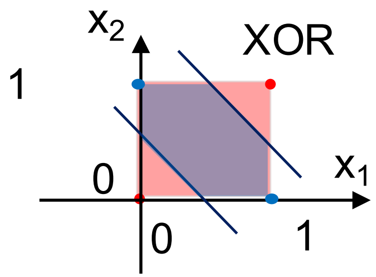


Input vector (x1,x2)	Class OR
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	1

$$y_1 = g\left(\mathbf{w}^T\mathbf{x} + b\right) = u((2 \quad 2) \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 3)$$

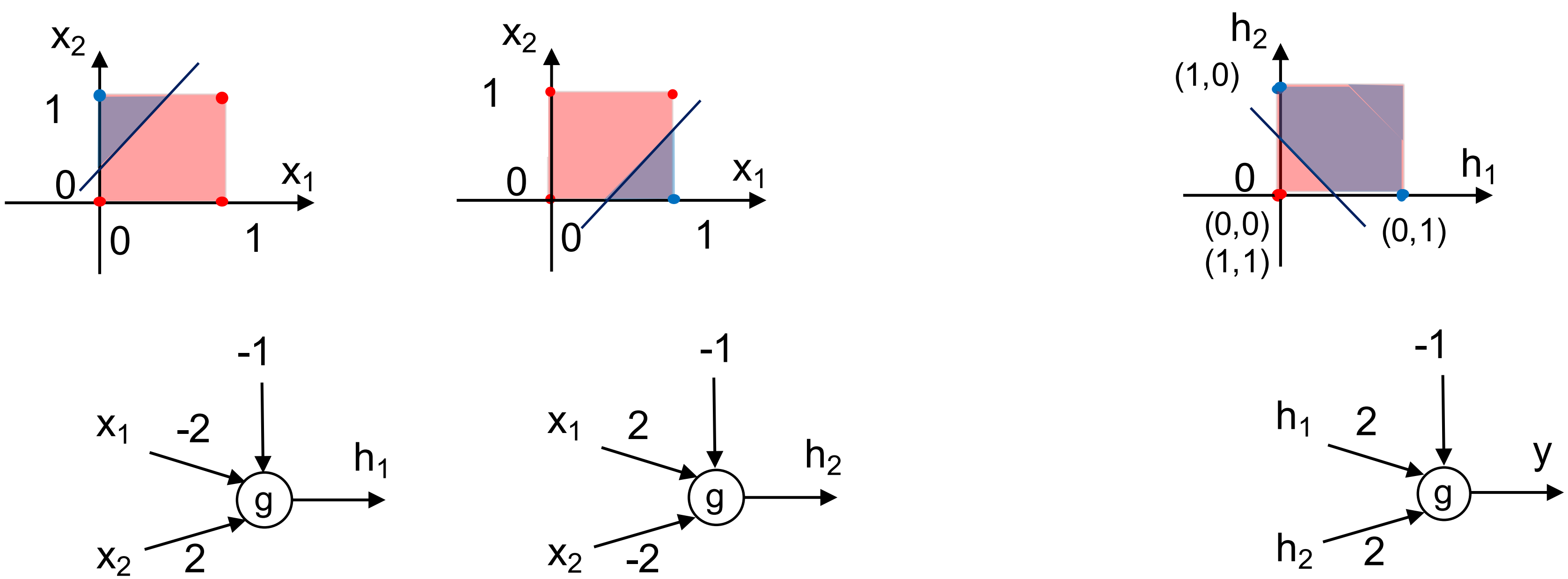
$$y_2 = g\left(\mathbf{w}^T\mathbf{x} + b\right) = u((2 \quad 2) \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 1)$$

Example: X-OR
X-OR a Non-linear separable problem can not be generated with a single perceptron



Input vector (x1,x2)	Class XOR
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0

Example: X-OR. However…..

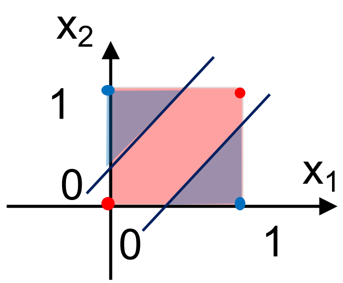


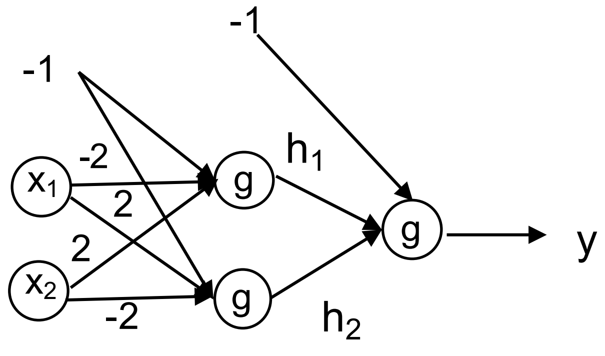
$$h_1 = g\left(\mathbf{w}_{11}^T\ \mathbf{x} + b_{11}\right) = u((-2 \quad 2) \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 1)$$

$$h_2 = g\left(\mathbf{w}_{12}^T\ \ \mathbf{x} + b_{12}\right) = u((2 \quad -2) \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 1)$$

$$y = g\left(\mathbf{w}_2^T\ \ \mathbf{h} + b_2\right) = u((2 \quad 2) \cdot \begin{pmatrix} h_1 \\ h_2 \end{pmatrix} - 1)$$

Example: X-OR. Finally





Input layer

Hidden layer

Output Layer

Three layer Network:

-Input Layer
-Hidden Layer
-Output Layer

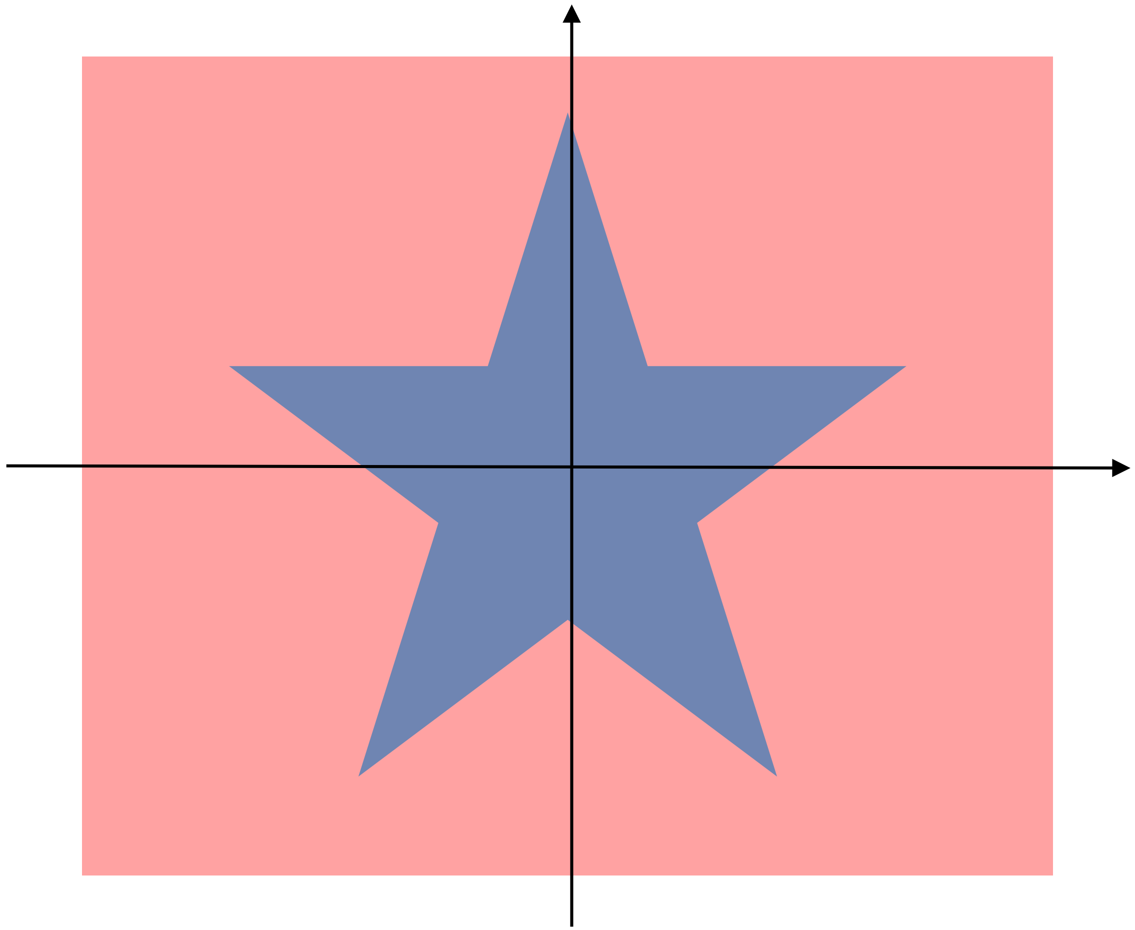
2-2-1 **Fully connected** topology
(all neurons in a layer connected
Connected to all neurons in the
following layer)

$$h_1 = g\left(\mathbf{w}_{11}^T\ \mathbf{x} + b_{11}\right) = u\left(\begin{pmatrix}-2 & 2\end{pmatrix} \cdot \begin{pmatrix}x_1 \\ x_2\end{pmatrix} - 1\right)$$

$$h_2 = g\left(\mathbf{w}_{12}^T\ \mathbf{x} + b_{12}\right) = u\left(\begin{pmatrix}2 & -2\end{pmatrix} \cdot \begin{pmatrix}x_1 \\ x_2\end{pmatrix} - 1\right)$$

$$y = g\left(\mathbf{w}_2^T\ \mathbf{h} + b_2\right) = u\left(\begin{pmatrix}2 & 2\end{pmatrix} \cdot \begin{pmatrix}h_1 \\ h_2\end{pmatrix} - 1\right)$$

Another Example: Star Region (Univ. Texas)



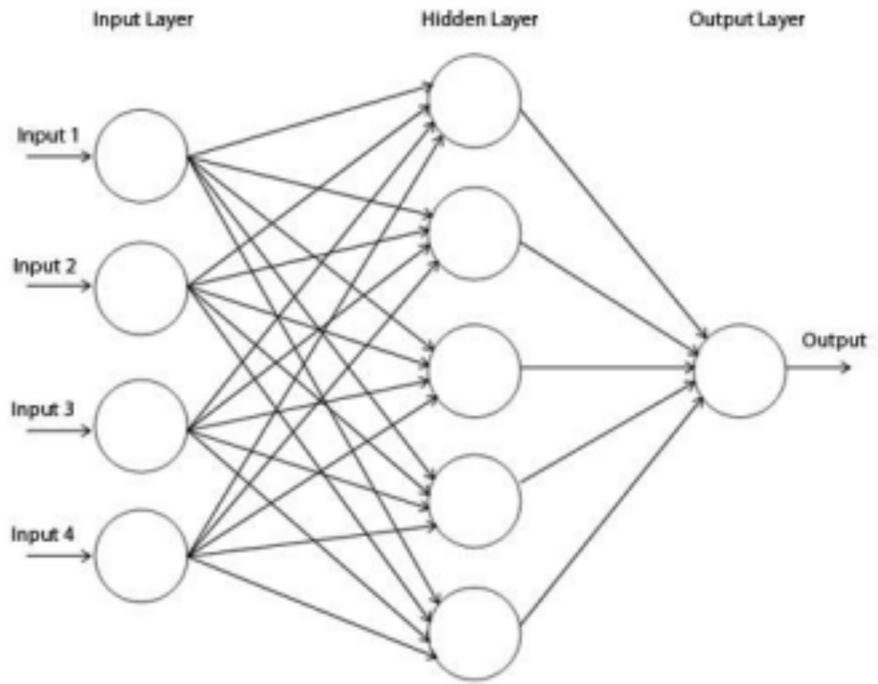
Neural networks

A neural network is simply a **composition** of simple neurons into several layers

Each neuron simply computes a **linear combination** of its inputs, adds a bias, and passes the result through an **activation function**

The network can contain one or more **hidden layers**. The outputs of these hidden layers can be thought of as a new **representation** of the data (new features).

The final output is the **target** variable ($y = f_\theta(x)$)



Multilayer perceptrons

When each node in each layer is a linear combination of **all inputs from the previous layer** then the network is called a multilayer perceptron (MLP)

Weights can be organized into matrices.

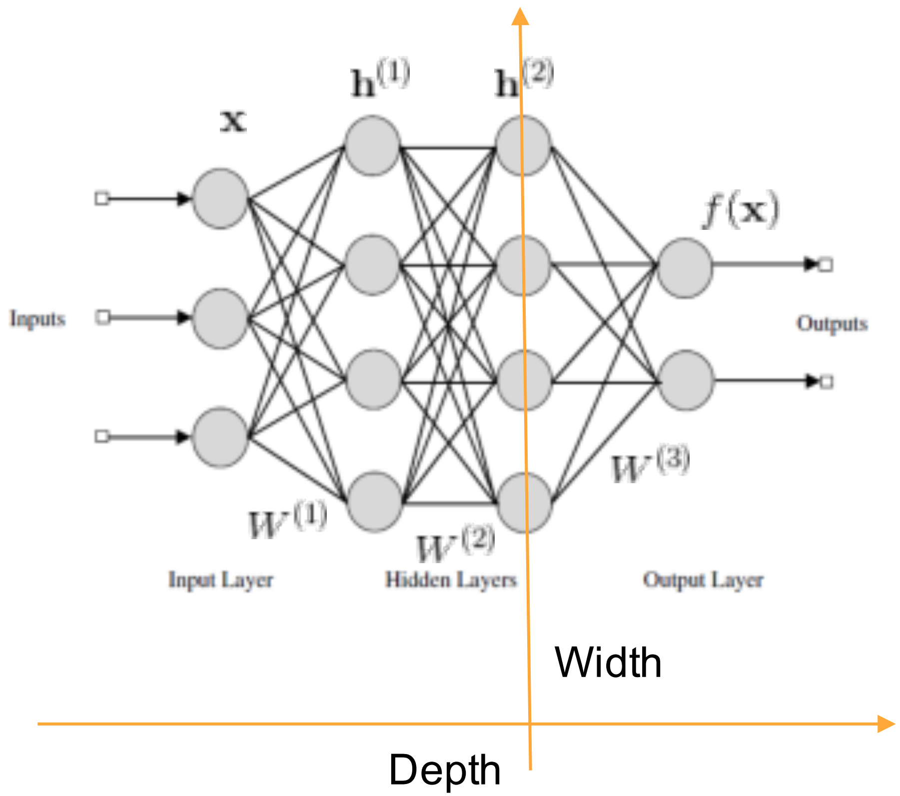
Forward pass computes

$$\mathbf{h}_0 = \mathbf{x}$$

$$\mathbf{h}^{(t)} = g(W^{(t)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(t)})$$

$$f(\mathbf{x}) = \mathbf{h}^{(L)}$$

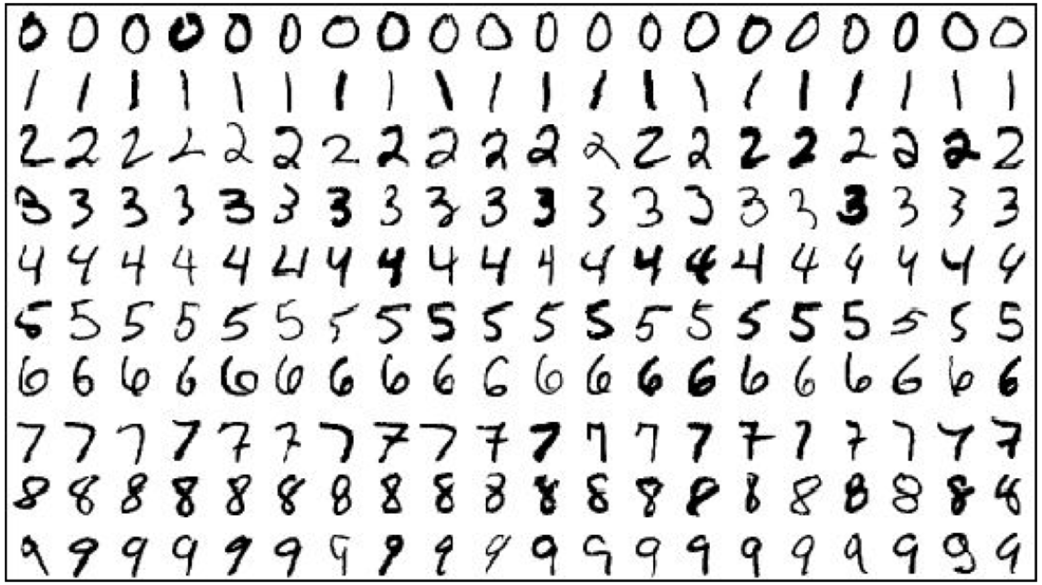
$$\mathbf{h}^{(2)} = g(W^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$



MNIST Example

Handwritten digits

- 60.000 training examples
- 10.000 test examples
- 10 classes (digits 0-9)
- 28x28 grayscale images(784 pixels)
- http://yann.lecun.com/exdb/mnist/



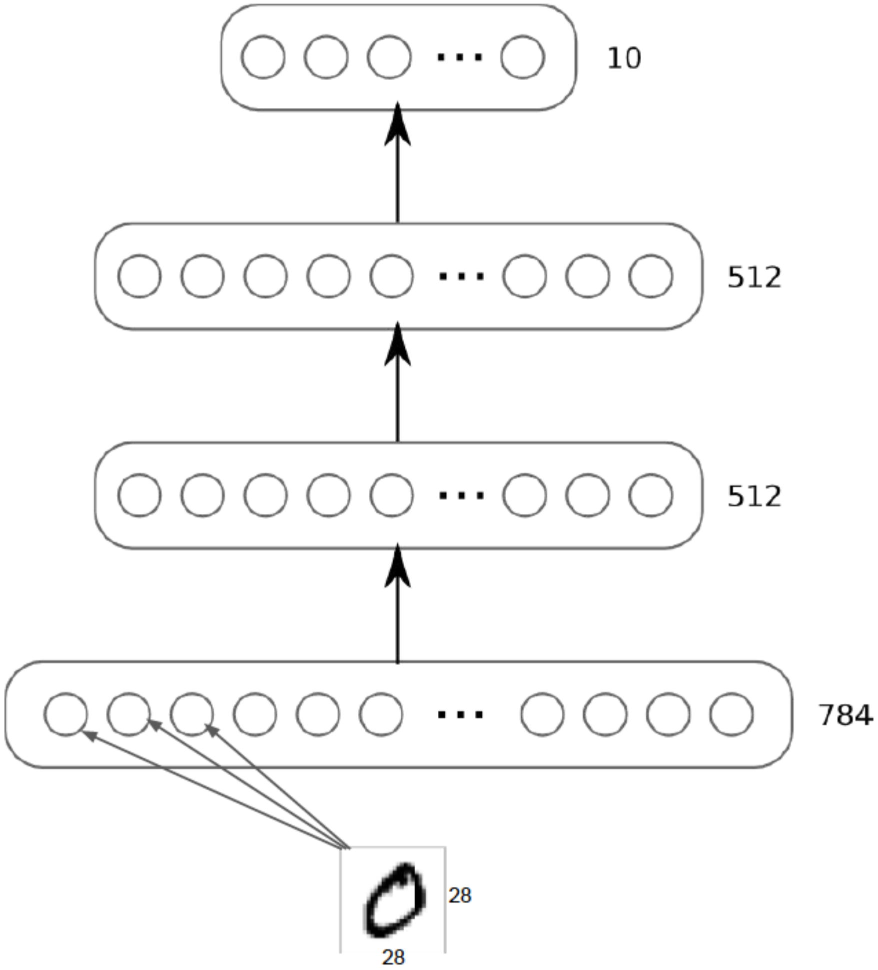
The objective is to learn a function that predicts the digit from the image

MNIST Example

Model

- 3 layer neural-network (2 hidden layers)
- Tanh units (activation function)
- 512-512-10
- Softmax on top layer
- Cross entropy Loss

Layer	#Weights	#Biases	Total
1	784 x 512	512	401,920
2	512 x 512	512	262,656
3	512 x 10	10	5,130
			669,706



MNIST Example

Training

- 40 epochs using min-batch SGD
- Batch Size: 128
- Leaning Rate: 0.1 (fixed)
- Takes 5 minutes to train on GPU

Accuracy Results

- 98.12% (188 errors in 10.000 test examples)

there are ways to improve accuracy…

Metrics

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

there are other metrics….

Training

- Estimate parameters $\theta(W^{(k)}, b^{(k)})$ from training examples given a Loss Function

  $$W^* = argmin_\theta \mathcal{L}(f_\theta(x), y)$$

- Iteratively adapt each parameter

  Basic idea: **gradient descent.**

- Dependencies are very complex.

  Global minimum: challenging. Local minima: can be good enough.

- Initialization influences in the solutions.

Training

- Gradient Descent: Move the parameter θ_j in small steps in the direction opposite sign of the derivative of the loss with respect j.

$$\theta^{(n)} = \theta^{(n-1)} - \alpha^{(n-1)} \cdot \nabla_\theta \mathcal{L}(y, f_\theta(x))$$

- Stochastic gradient descent (SGD): estimate the gradient with one sample, or better, with a **minibatch** of examples.

- **Momentum:** the movement direction of parameters averages the gradient estimation with previous ones.

- Several strategies have been proposed to update the weights: Adam, RMSProp, Adamax, etc. known as: **optimizers**

Training MLPs

With **Multiple Layer Perceptrons** we need to find the gradient of the loss function with respect to all the parameters of the model ($W^{(k)}$, $b^{(k)}$)

These can be found using the **chain rule** of differentiation.

The calculations reveal that the gradient wrt the parameters in layer k only depends on the error from the above layer and the output from the layer below.

This means that the gradients for each layer can be computed iteratively, starting at the last layer and propagating the error back through the network. This is known as the **backpropagation** algorithm.