Artificial Neural Networks.
Perceptron

Prof. Ioannis Pitas
Aristotle University of Thessaloniki
pitas@csd.auth.gr
www.aiia.csd.auth.gr
Version 4.3.4

Artificial Intelligence &
Information Analysis Lab

Artificial Neural Networks. Perceptron

- Classification/Recognition/Identification
- Biological Neurons
- Artificial Neurons
- Perceptron
- Iterative Perceptron training
- Batch Perceptron training

Classification/Recognition/ Identification

- Given a set of classes $\mathcal{C} = \{\mathcal{C}_i,\ i = 1, \dots, m\}$ and a sample $\mathbf{x} \in \mathbb{R}^n$, the ML model $\hat{\mathbf{y}} = \boldsymbol{f}(\mathbf{x}; \boldsymbol{\theta})$ predicts a class label vector $\hat{\mathbf{y}} \in [0, 1]^m$ for input sample $\mathbf{x}$, where $\boldsymbol{\theta}$ are the learnable model parameters.

- Essentially, a probabilistic distribution $P(\hat{\mathbf{y}}; \mathbf{x})$ is computed.
- Interpretation: likelihood of the given sample $\mathbf{x}$ belonging to each class $\mathcal{C}_i$.

- Single-target classification:
  - Classes $\mathcal{C}_i, i = 1, \dots, m$ are ***mutually exclusive***: $\|\hat{\mathbf{y}}\|_1 = 1$.
- Multi-target classification:
  - Classes $\mathcal{C}_i, i = 1, \dots, m$ are ***not mutually exclusive***: $\|\hat{\mathbf{y}}\|_1 \geq 1$.

Supervised Learning

- A sufficient large training sample set $\mathcal{D}$ is required for Supervised Learning (regression, classification):

$$\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i), i = 1, \ldots, N\}.$$

- $\mathbf{x}_i \in \mathbb{R}^n$: n –dimensional input (feature) vector of the i-th training sample.
- $\mathbf{y}_i$: its target label (output).
- Target form $\mathbf{y}$ can vary:
  - it can be categorical, a real number or a combination of both.

Classification/Recognition/ Identification

- ***Training***: Given N pairs of training samples $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i), i = 1, \dots N\}$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $\mathbf{y}_i \in [0,1]^m$, estimate $\boldsymbol{\theta}$ by minimizing a loss function: $\min_{\boldsymbol{\theta}} J(\mathbf{y}, \hat{\mathbf{y}})$.

- ***Inference/testing***: Given N_t pairs of testing examples $\mathcal{D}_t = \{(\mathbf{x}_i, \mathbf{y}_i), i = 1, \dots, N_t\}$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $\mathbf{y}_i \in [0,1]^m$, compute (***predict***) $\hat{\mathbf{y}}_i$ and calculate a performance metric, e.g., classification accuracy.

Classification/Recognition/ Identification

Optional steps between training and testing:

- ***Validation***: Given N_v pairs of testing examples (different from either training or testing examples) $\mathcal{D}_v = \{(\mathbf{x}_i, \mathbf{y}_i), i = 1, \dots, N_v\}$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $\mathbf{y}_i \in [0,1]^m$, compute (predict) $\hat{\mathbf{y}}_i$ and validate using a performance metric.

or

- ***k-fold cross-validation***: Use only a percentage ($\frac{100}{k}$%, e.g., 80%) of the data for training and the rest for validation ($\frac{100}{k}$%, e.g., 20%). Repeat it k times, until all data used for training and testing).

Classification

- **Classification:**
  - Two class ($m = 2$) and multiple class ($m > 2$) classification.
  - **Example:** ***Face detection*** (*t*wo classes), ***face recognition*** (many classes).

Classification

Multiclass Classification $(m > 2)$**:**

- Multiple $(m > 2)$ hypothesis testing: choose a winner class out of m classes.
- Binary hypothesis testing:
  - One class against all: m binary hypothesis.
    - one must be proven true.
  - Pair-wise class comparisons: $m(m-1)/2$ binary hypothesis.

Face Recognition/identification

Problem statement:

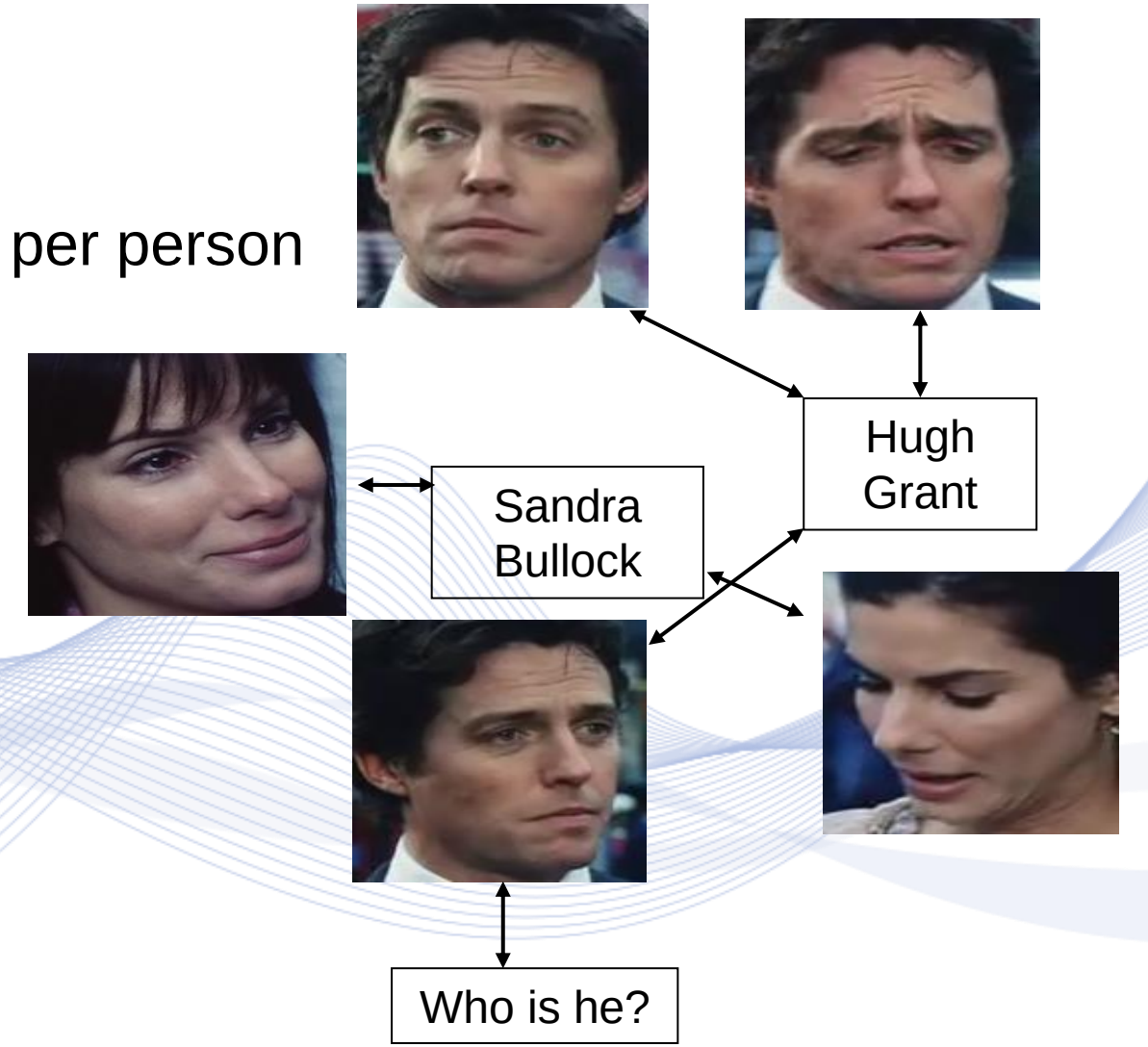
- To identify a face identity
- Input for training: several facial ROIs per person
- Input for inference: a facial ROI
- Inference output: the face id

- Supervised learning
- Applications:

  Biometrics

  Surveillance applications

  Video analytics



Face Recognition/identification

Problem statement:

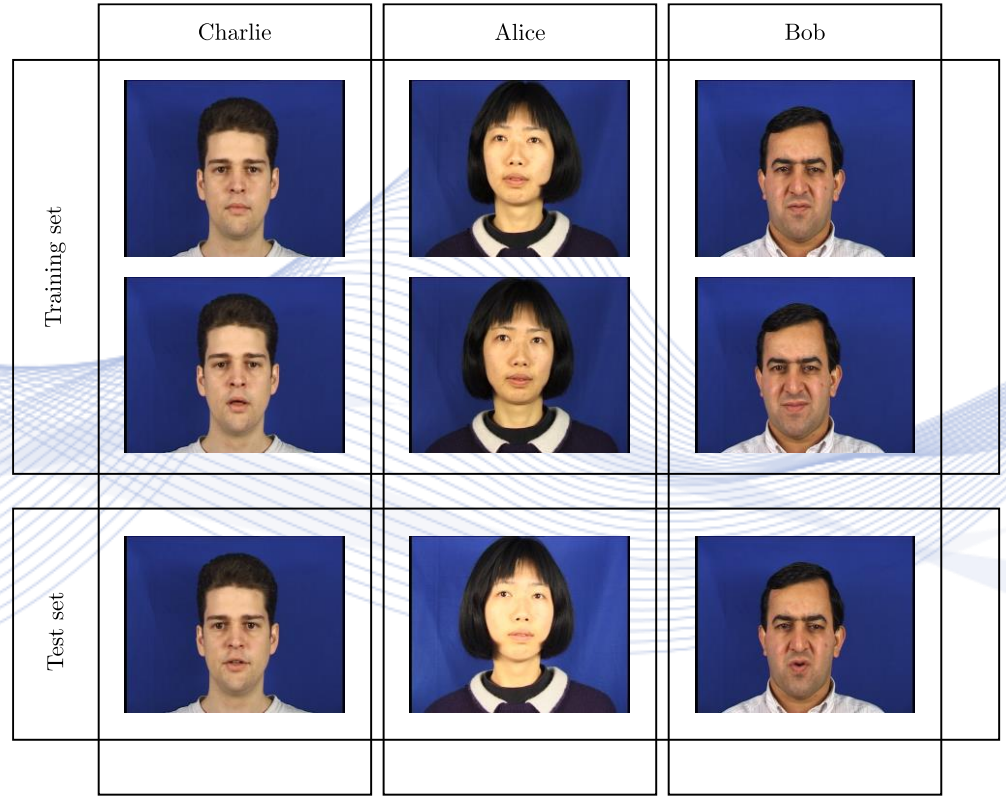
- To identify a face identity
- Input for training: several facial ROIs per person
- Input for inference: a facial ROI
- Inference output: the face id

- Supervised learning
- Applications:

  Biometrics

  Surveillance applications

  Video analytics



Regression

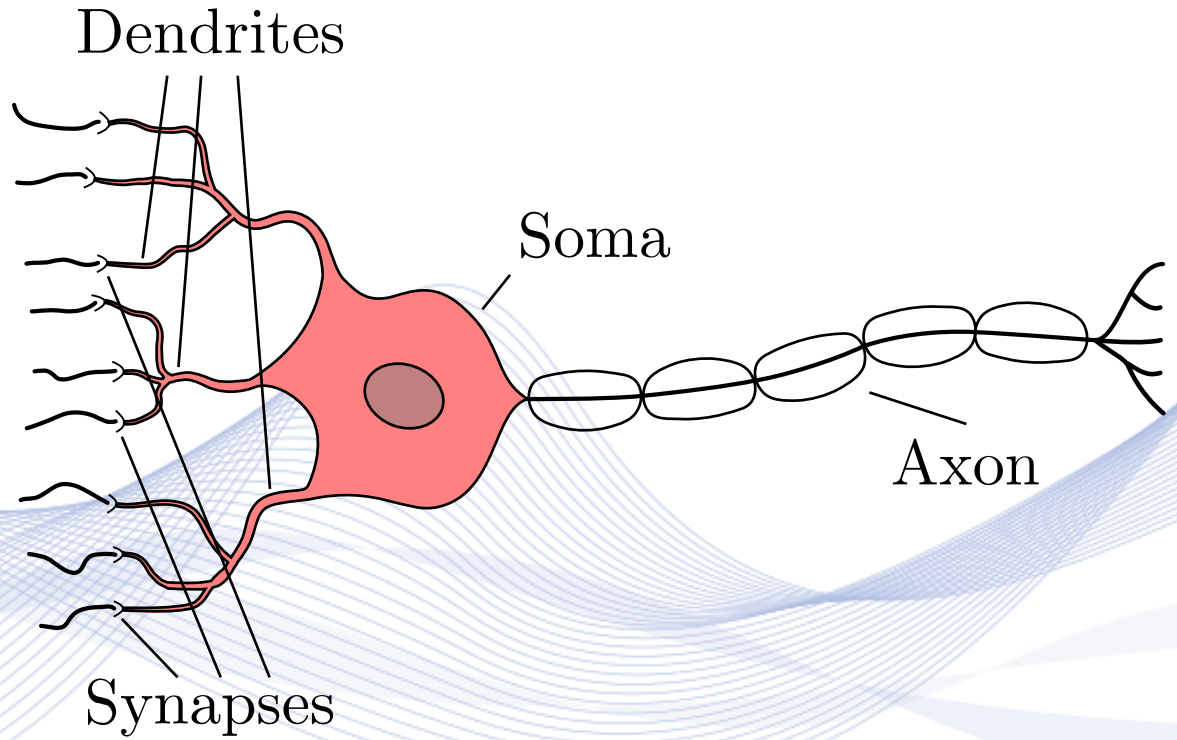
Given a sample $\mathbf{x} \in \mathbb{R}^n$ and a function $\mathbf{y} = \boldsymbol{f}(\mathbf{x})$, the model predicts ***real-valued quantities*** for that sample: $\hat{\mathbf{y}} = \boldsymbol{f}(\mathbf{x}; \boldsymbol{\theta})$, where $\hat{\mathbf{y}} \in \mathbb{R}^m$ and $\boldsymbol{\theta}$ are the learnable parameters of the model.

- **Training**: Given N pairs of training examples $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i), i = 1, \dots N\}$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $\mathbf{y}_i \in \mathbb{R}^m$, estimate $\boldsymbol{\theta}$ by minimizing a loss function: $\min_{\boldsymbol{\theta}} J(\mathbf{y}, \hat{\mathbf{y}})$.

- **Testing**: Given N_t pairs of testing examples $\mathcal{D}_t = \{(\mathbf{x}_i, \mathbf{y}_i), i = 1, \dots, N_t\}$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $\mathbf{y}_i \in \mathbf{y}_i \in \mathbb{R}^m$, compute (predict) $\hat{\mathbf{y}}_i$ and calculate a performance metric, e.g., MSE.

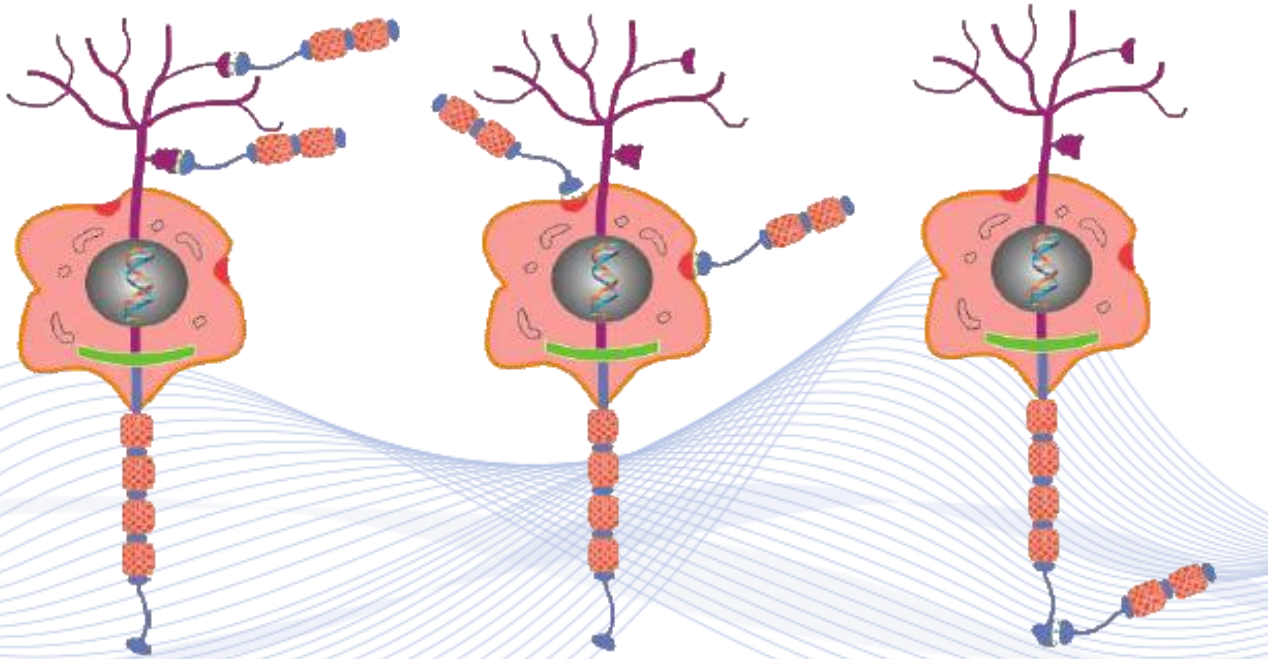
Biological Neuron

- Basic computational unit of the brain.
- Main parts:
  - ***Dendrites***
    - They act as inputs.
  - ***Soma***
    - Main body of neuron.
  - ***Axon***
    - It acts as neuron output.



Biological Neuron Connectivity

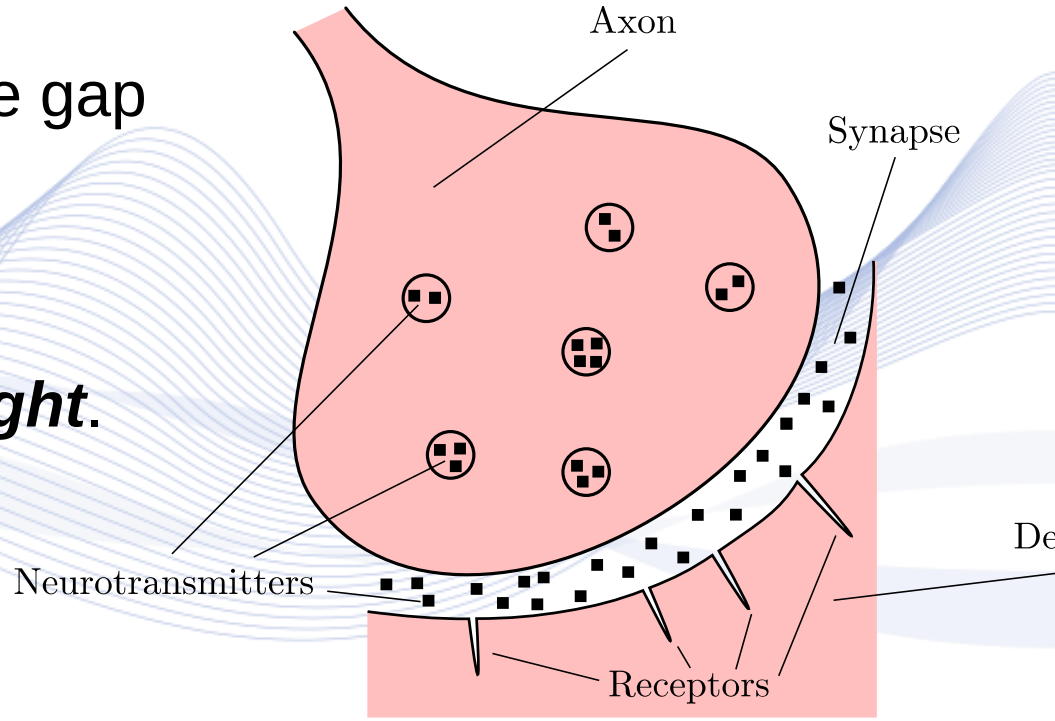
- Neurons connect with other neurons through ***synapses***.



Axodendritic Axosomatic Axoaxonal

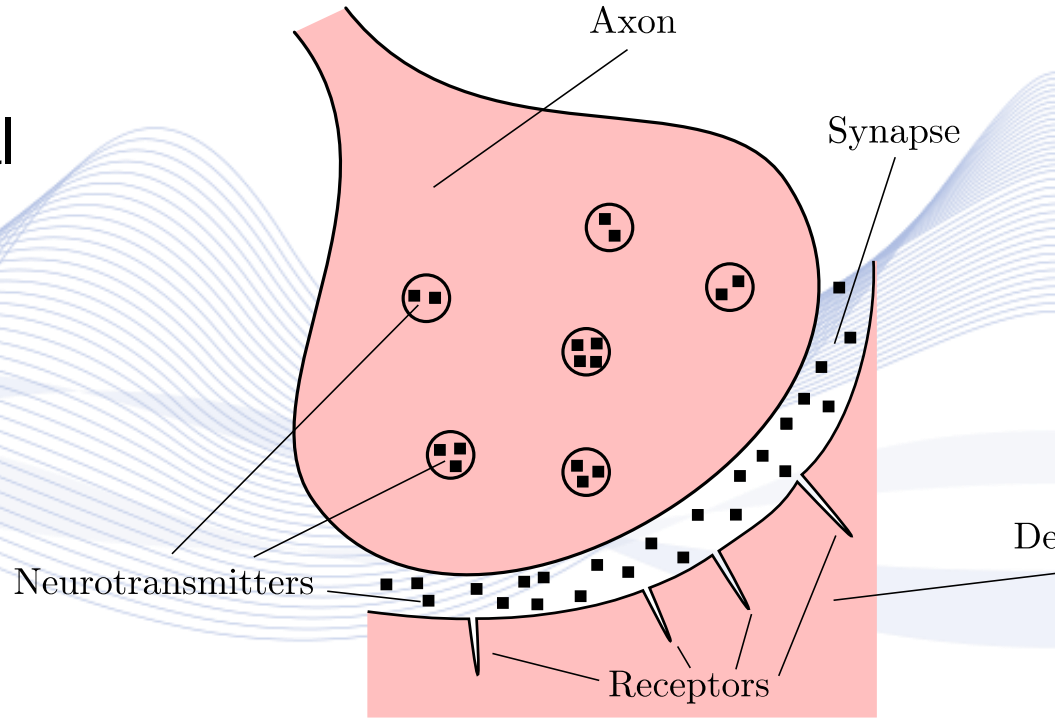
Biological Neuron Connectivity

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



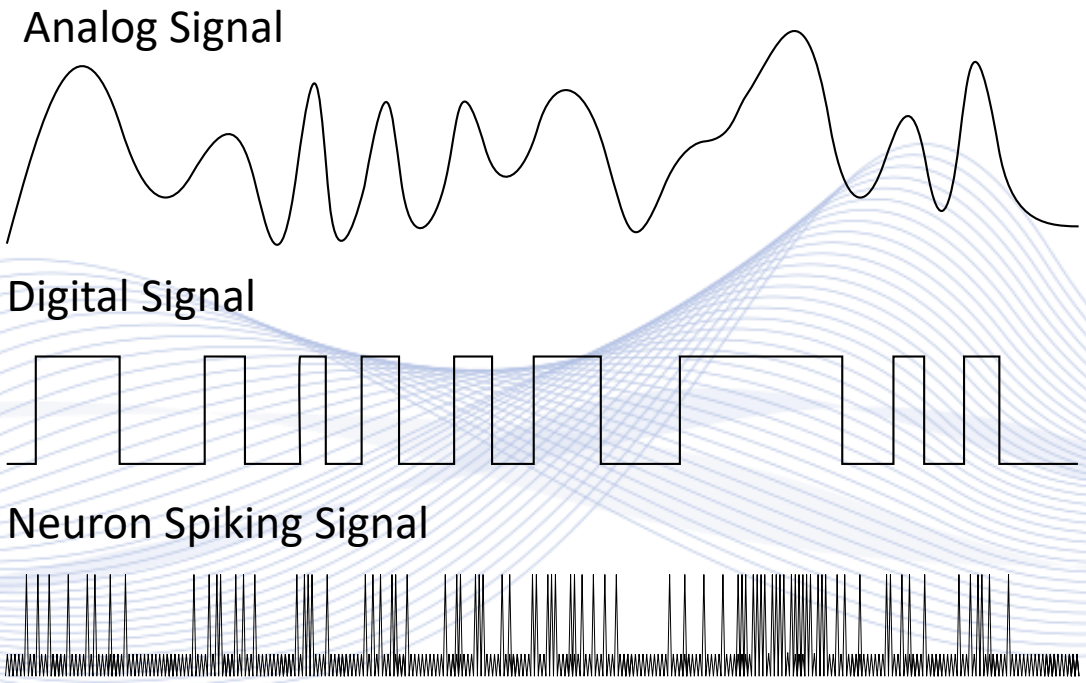
Biological Neuron Connectivity

- Synaptic weights can be:
  - Positive (***excitatory synapses***).
  - Negative (***inhibitory synapses***).
- Transmitted signal is a series of electrical impulses.
- The stronger the transmitted signal, the bigger the impulse frequency.



Biological Neuron Connectivity

- Transmitted signal is a series of electrical impulses.
- The stronger the transmitted signal, the bigger the impulse frequency.



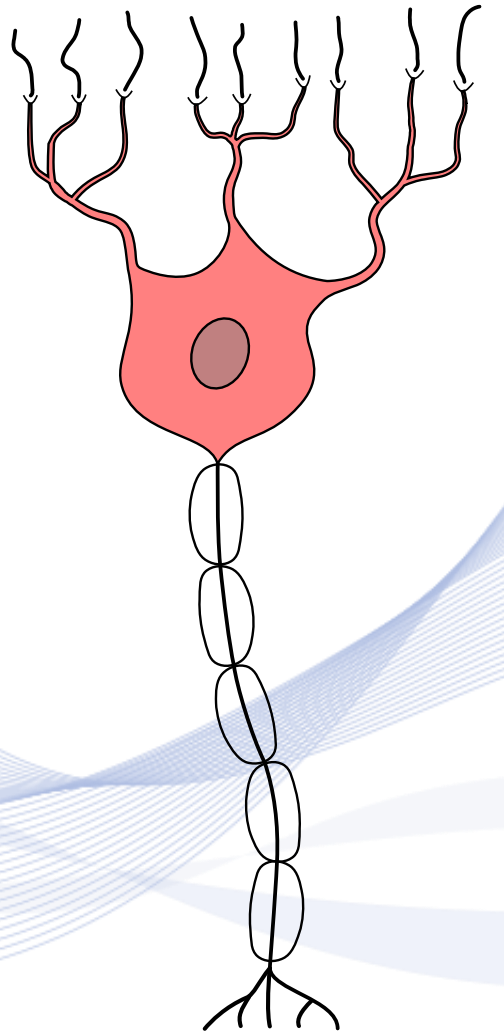
Neural spiky signal [EXT].

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.



Artificial Neurons

Artificial neurons are mathematical models loosely inspired by their biological counterparts.

- Previous dendrites fetch the input vector:

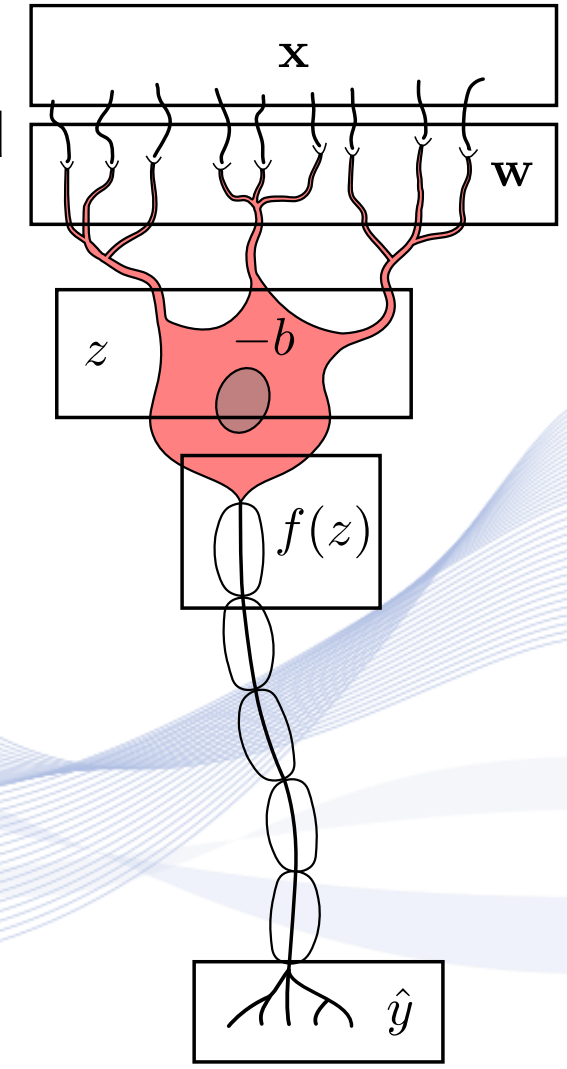
$$\mathbf{x} = [x_1, x_2, \dots, x_n]^T, \qquad x_i \in \mathbb{R}.$$

- The synaptic weights are grouped in a weight vector:

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

- ***Synaptic integration*** is modeled as the inner product:

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



Artificial Neurons

ANN synaptic summation models:

- linear function (or mapping from $\mathbf{x}$ to y):

$$\hat{y} = \mathbf{w}^T\mathbf{x} + b.$$

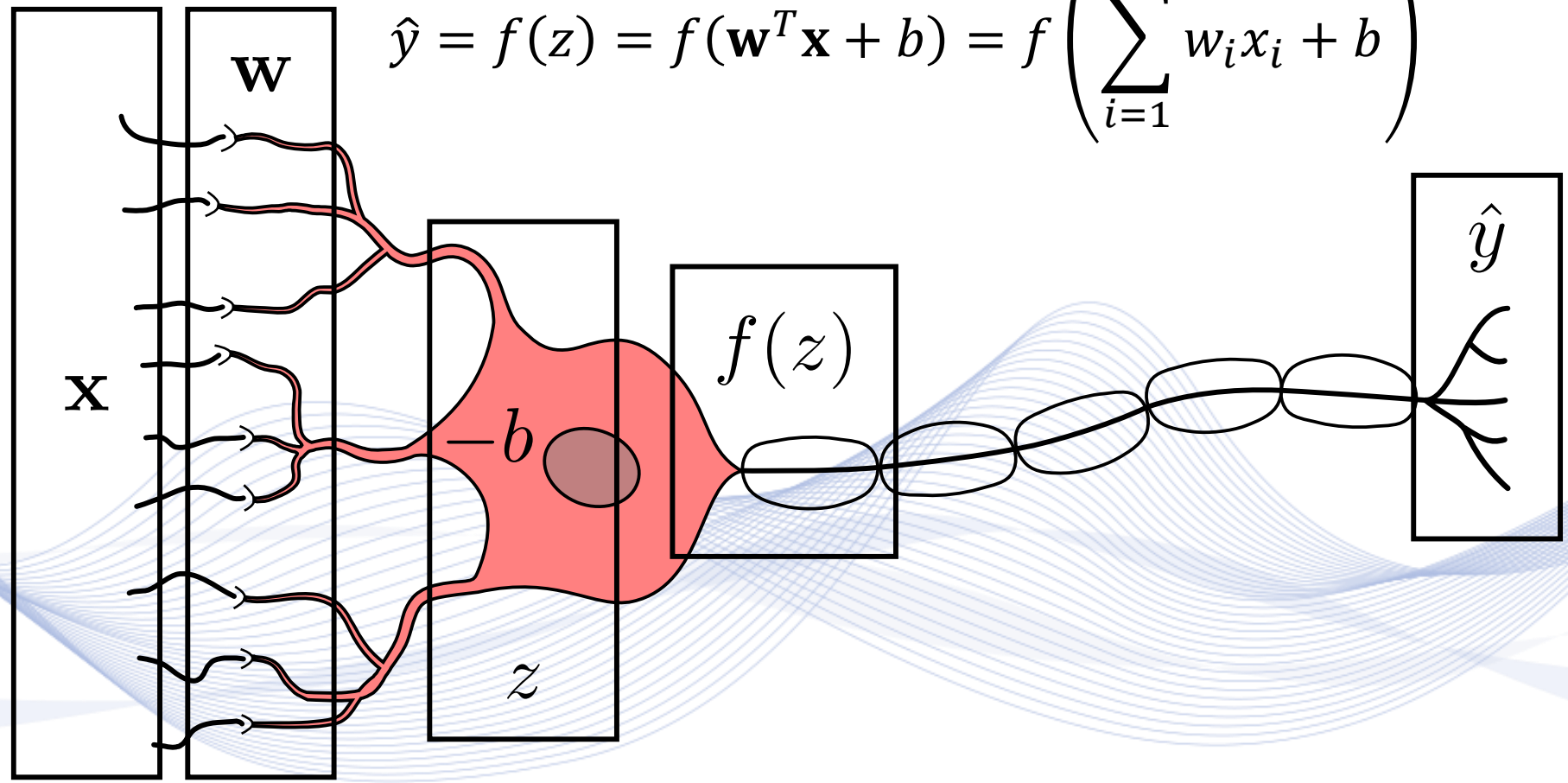
- nonlinear model:

$$\hat{y} = \mathbf{w}^T\phi(\mathbf{x}) + b.$$

- $\phi: \mathbb{R}^n \rightarrow H$: Nonlinear mapping of $\mathbf{x}$ to a (possibly) high-dimensional space, $L = \dim(H) > n, \mathbf{w} \in \mathbb{R}^L$.
- Learning consists of finding the optimal parameters $\mathbf{w}$, so that $\hat{y} = f(\mathbf{x}; \mathbf{w})$ is as close as possible to the target y, by minimizing a cost function $J(\mathbf{w})$ (a measure of discrepancy between y and $\hat{y}$).

Perceptron

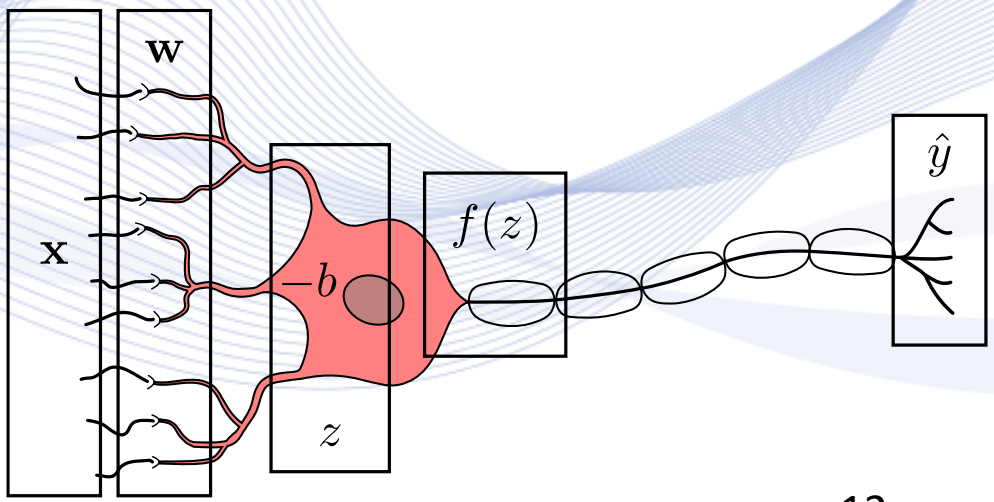
$$\hat{y} = f(z) = f(\mathbf{w}^T\mathbf{x} + b) = f\left(\sum_{i=1}^{N} w_i x_i + b\right)$$



Perceptron

- McCulloch & Pitts model is the simplest mathematical model of a neuron.
- It has real inputs in the range $x_i \in [0,1]$.
- It produces a single output $\hat{y} \in [0,1]$, through ***activation function*** $f(\cdot)$.
- Output y signifies whether the neuron will fire or not.
- Firing threshold:

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



Perceptron

- Threshold can be incorporated into the augmented vectors:

$$\hat{y} = f(\mathbf{x}; \mathbf{w}) = f(\mathbf{w}'^T \mathbf{x}').$$

- Augmented input vector: $\mathbf{x}' = [1, x_1, \ldots, x_n]^T$
- Augmented weight vector $\mathbf{w}' = [b, w_1, \ldots, w_n]^T$.

- From now on, for notation simplicity, we discard augmentation and work with $\mathbf{x} \in \mathbb{R}^{n+1}$ and $\mathbf{w} \in \mathbb{R}^{n+1}$.
- In this case, the parameter vector to be optimized is $\boldsymbol{\theta} = \mathbf{w}$.

Perceptron – Activation

Neural ***activation function*** that is suitable for 2-class problems:

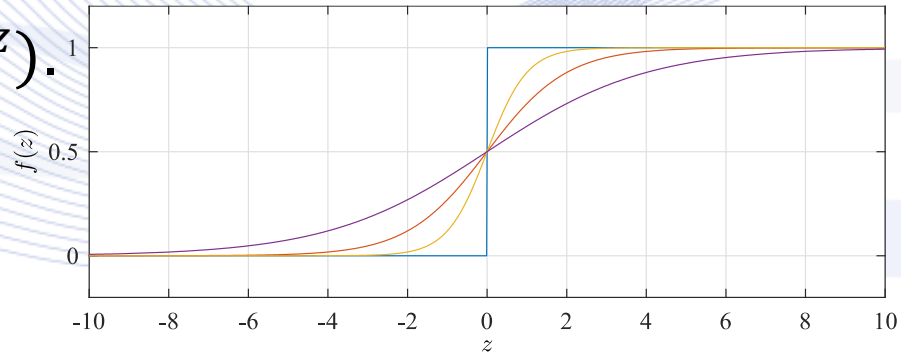
- If $f(z) \geq 0$, assign $\mathbf{x}$ to class $\mathcal{C}_1$;
- If $f(z) < 0$, assign $\mathbf{x}$ to class $\mathcal{C}_2$.

- ***Step activation function***:

$$\hat{y} = f(z) = f(\mathbf{w}^T\mathbf{x}) = \begin{cases} 0, & \mathbf{w}^T\mathbf{x} < 0 \\ 1, & \mathbf{w}^T\mathbf{x} \geq 0. \end{cases}$$

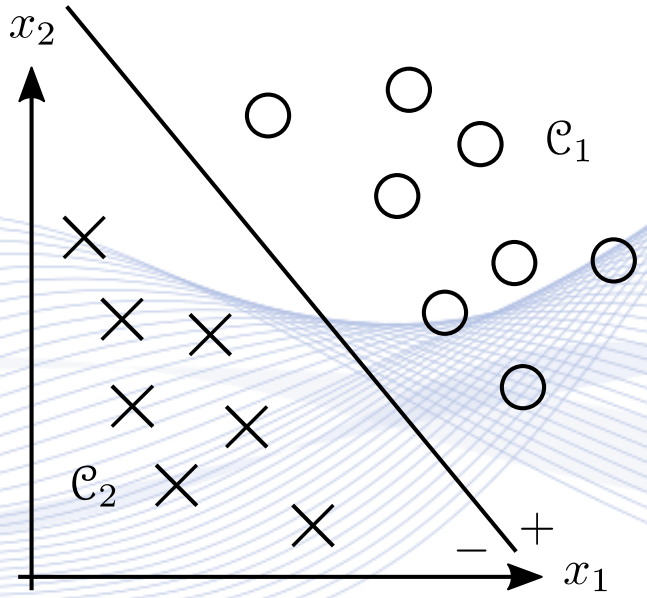
- ***Sigmoid activation function***:

$$\hat{y} = f(z) = 1/(1 + e^{-cz}).$$



Perceptron-Decision Hyperplanes

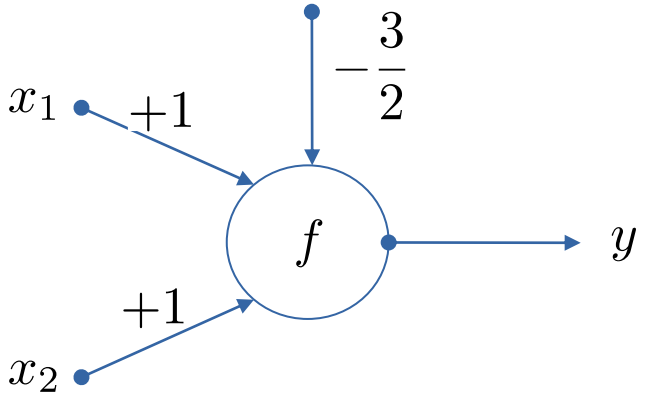
Perceptron decision surface with step activation function:
a) Line in $\mathbb{R}^2$; b) plane in $\mathbb{R}^3$; c) hyperplane in $\mathbb{R}^n$.



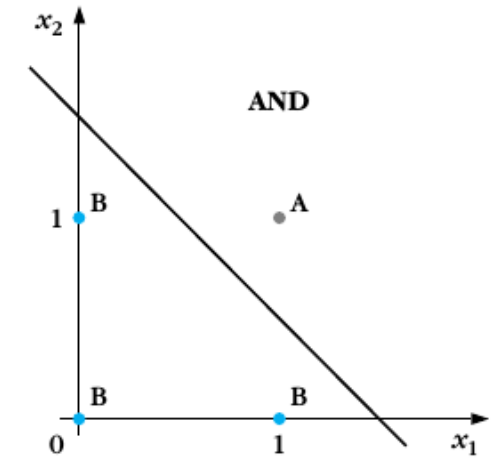
Linear decision line.

AND function model

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

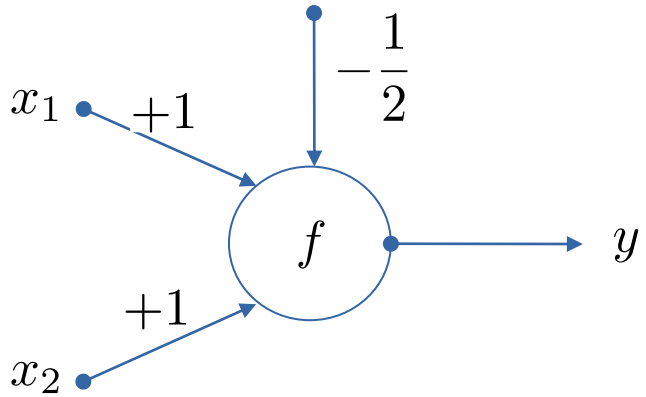


x_1	x_2	$y = \mathbf{w}^T\mathbf{x}$
0	0	$\mathbf{w}^T\mathbf{x} < 0 \Rightarrow y = 0$
0	1	$\mathbf{w}^T\mathbf{x} < 0 \Rightarrow y = 0$
1	0	$\mathbf{w}^T\mathbf{x} < 0 \Rightarrow y = 0$
1	1	$\mathbf{w}^T\mathbf{x} \geq 0 \Rightarrow y = 1$

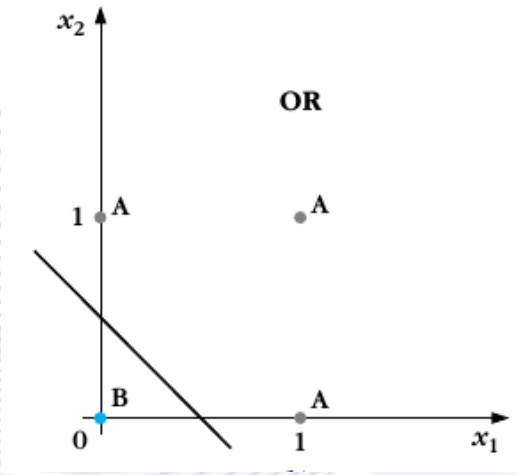


OR function model

- Perceptron weight vector: $\mathbf{w} = \left[-\frac{1}{2}, 1, 1\right]^T$
- Input 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	$\mathbf{w}^T\mathbf{x} \geq 0 \Rightarrow y = 1$

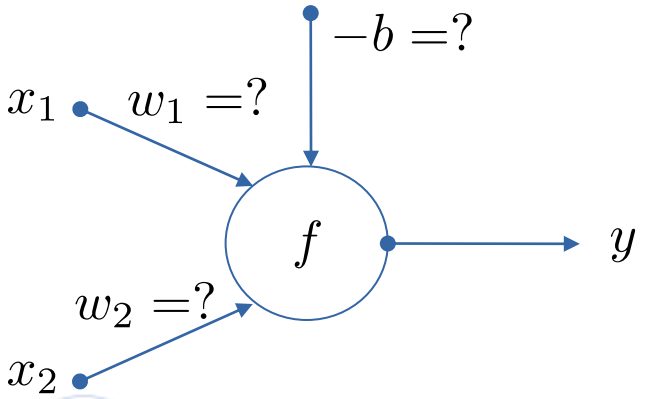


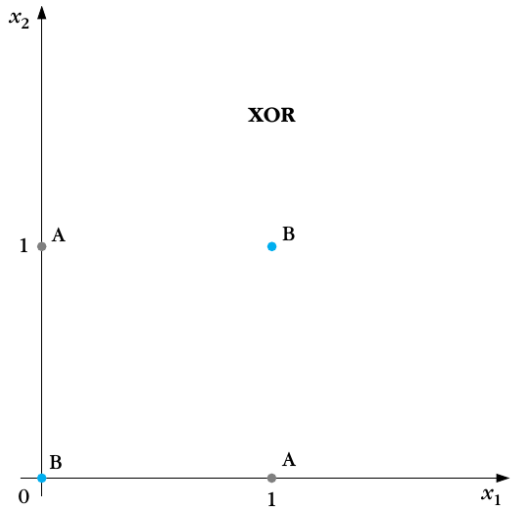
XOR function mod

- There is no linear separating line in $\mathbb{R}^2$ for the XOR function.
- 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:

$$f(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi},$$

where $\boldsymbol{\phi}$ is the output of the extra layer, given as input to the output layer.





Two-layer Perceptron – XOR function model

- Notation:

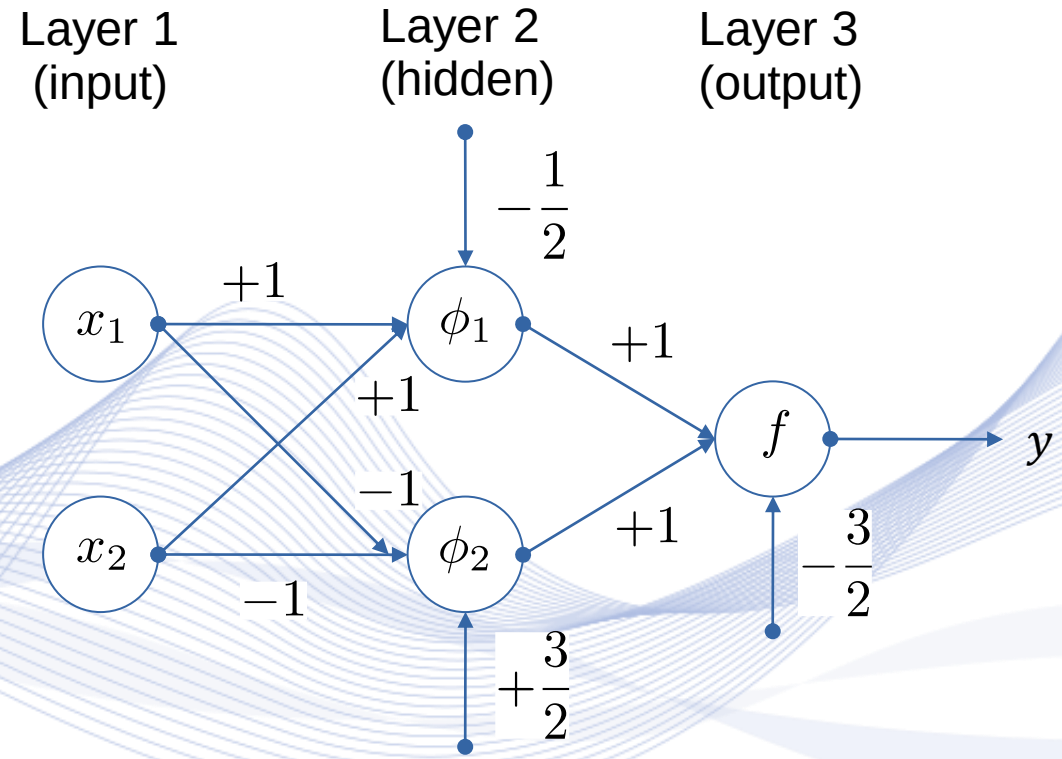
$\mathbf{x} = [1, x_1, x_2]^T$ $\quad f(\cdot)$: step function

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

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

$\mathbf{w} = \left[-\frac{3}{2}, 1, 1\right]^T$ $\quad y = \mathbf{w}^T\boldsymbol{\phi},$

$\boldsymbol{\phi} = [1, \phi_1(\mathbf{x}), \phi_2(\mathbf{x})]^T.$

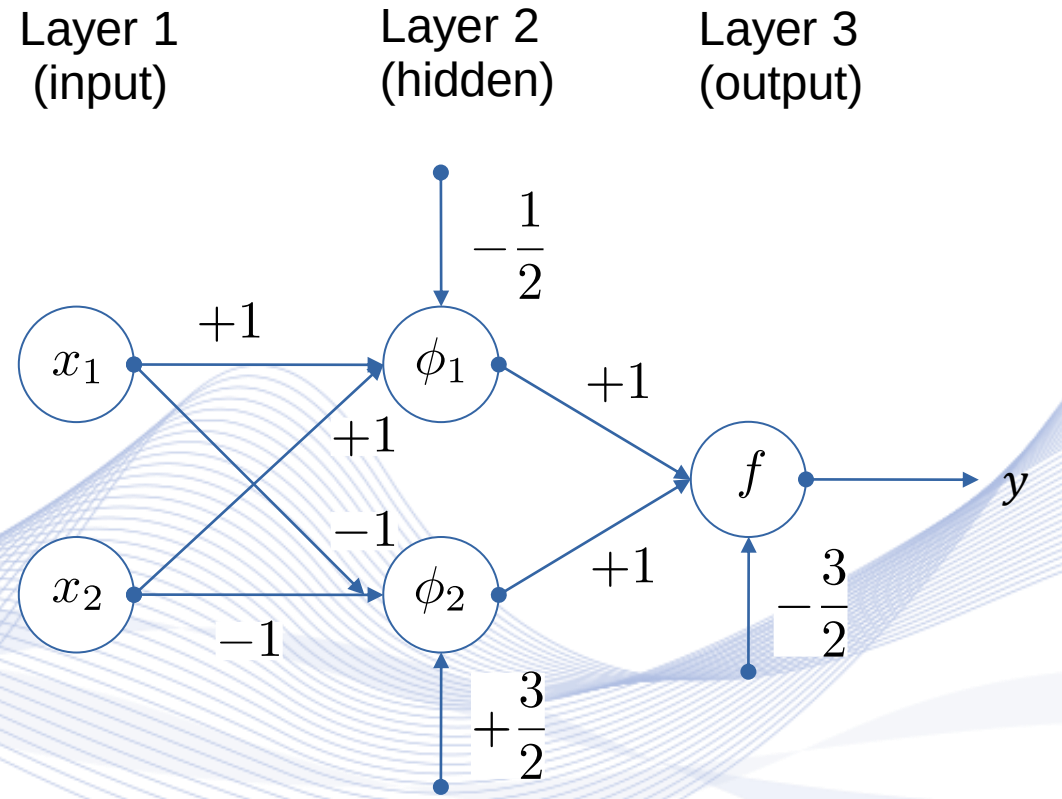


Two-layer Perceptron – XOR function model

- (A OR B) AND (NOT (A AND B))

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

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



Iterative Perceptron training

Perceptron training as an optimization problem:

- Perceptron has to be optimized to minimize error function $J(\mathbf{w})$.
- Differentiation:

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \mathbf{0}$$

can provide the ***critical points*** of multivariate function $J(\mathbf{w})$:

- ***Minima***, ***maxima*** and ***saddle points***.
- Analytical differentiation is usually impossible.
- We must resort to numerical optimization methods.

Steepest Gradient Descent

Steepest Gradient Descent is one of the most popular optimization algorithms.

- The parameter space $\mathbf{w} \in \mathbb{R}^n$ is iteratively searched, along the direction of the error function gradient $-\nabla J(\mathbf{w})$.
- The gradient vector points to the direction of the ***steepest ascent***.

$$\nabla J(\mathbf{w}) = \frac{\partial J}{\partial \mathbf{w}} = \left[\frac{\partial J}{\partial w_1}, \ldots, \frac{\partial J}{\partial w_n}\right]^T.$$

- Correspondingly, $-\nabla J(\mathbf{w})$ points to the direction of the ***steepest descent*** to a mimimum, along which $J(\mathbf{w})$ decreases more rapidly.

Steepest Gradient Descent

- Steepest Gradient Descent is an iterative method, fining solutions

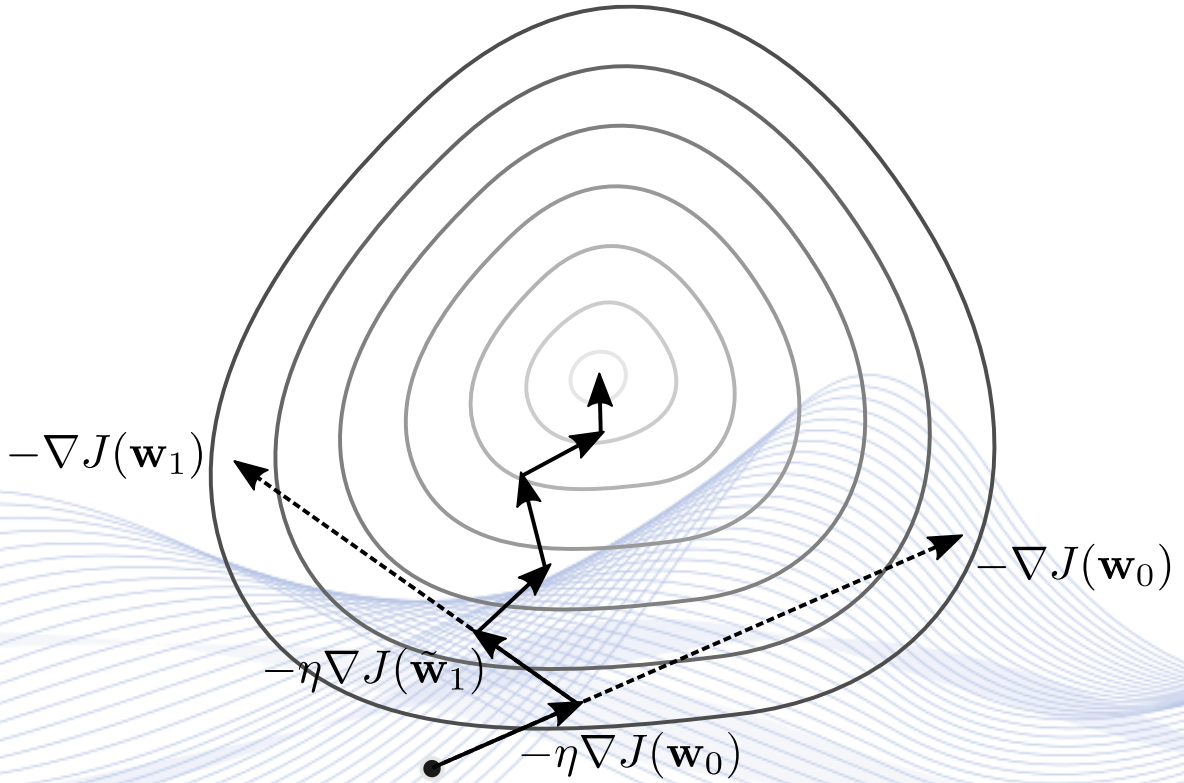
$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}\ (t),$$

- t is the iteration number.
- The correction term can be proportional to the direction of steepest descent $\nabla_{\mathbf{w}} J(\mathbf{w})(t)$:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla_{\mathbf{w}} J(\mathbf{w})(t).$$

- η: ***learning rate*** is a parameter controlling model parameters updates***.***

Steepest Gradient Descent



Steepest descent for a 2D error function $J(w_1, w_2)$.

Steepest Gradient Descent

- There is no guarantee of convergence to a global minimum.
- The solution depends on the initial staring point $\mathbf{w}(0)$.
- Numerical estimates of the gradient $\nabla J(\mathbf{w}) = \left[\frac{\partial J}{\partial w_1}, \ldots, \frac{\partial J}{\partial w_n}\right]^T$ can be noisy.
- Convergence can be slow.
- Convergence speed depends on learning rate η.

Iterative Perceptron training

2-class Classification:

If $\hat{y} = f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x} \geq 0$, assign $\mathbf{x}$ to class $\mathcal{C}_1$,

If $\hat{y} = f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x} < 0$, assign $\mathbf{x}$ to class $\mathcal{C}_2$.

- Goal: find the optimal model parameters $\mathbf{w}$, so that the model produces the minimal number of false class assignments (decisions/predictions),

Iterative Perceptron training

Classification is transformed to an optimization problem:

- construct a cost function, choose an optimization algorithm.

Perceptron cost function:

$$\min_{\mathbf{w}} J(\mathbf{w}) = \sum_{\mathbf{x}\in\mathcal{D}'} d_x \mathbf{w}^T \mathbf{x}.$$

- $\mathcal{D}'$ is the subset of training samples that have been misclassified.
- $d_x \triangleq -1$, if $\mathbf{x} \in \mathcal{C}_1$ and has been misclassified to $\mathcal{C}_2$,
- $d_x \triangleq 1$, if $\mathbf{x} \in \mathcal{C}_2$ and has been misclassified to $\mathcal{C}_1$.

Iterative Perceptron training

- When all samples have been correctly classified: $\mathcal{D}' = \emptyset \Rightarrow J(\mathbf{w}) = 0$.
- If $\mathbf{x} \in \mathcal{C}_1$ and has been misclassified, then $\mathbf{w}^T\mathbf{x} < 0$ and $d_x < 0$. Thus, they produce positive error $J(\mathbf{w})$.
- The same applies for misclassified samples of class $\mathcal{C}_2$.
- $J(\mathbf{w})$ is continuous and piece-wise linear and differentiable.

Iterative Perceptron training

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

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla J(\mathbf{w}),$$

$$\nabla J(\mathbf{w}) = \frac{\partial}{\partial \mathbf{w}} \sum_{\mathbf{x} \in \mathcal{D}'} d_x \mathbf{w}^T \mathbf{x} = \sum_{\mathbf{x} \in \mathcal{D}'} d_x \mathbf{x}.$$

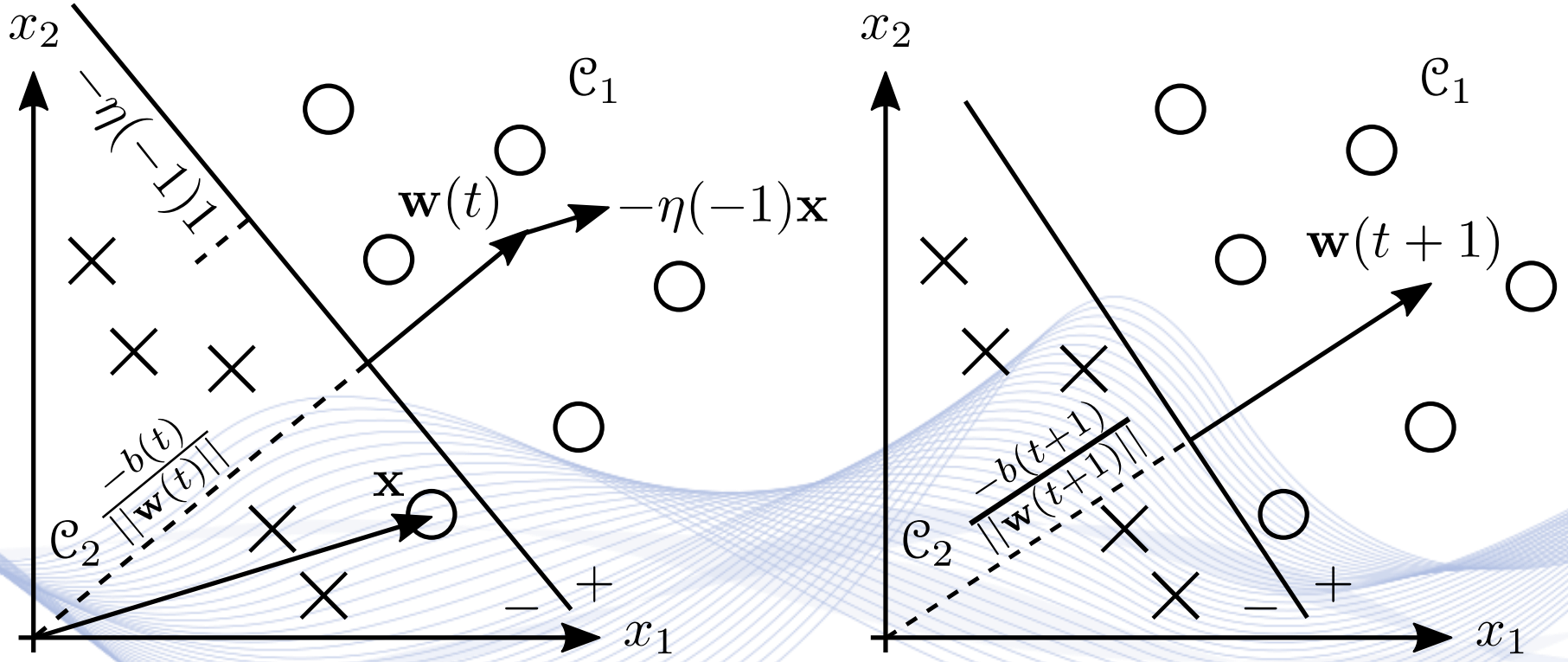
Iterative Perceptron training

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

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \sum_{\mathbf{x} \in \mathcal{D}'} d_x \mathbf{x}.$$

- It is proven that this algorithm converges.
- It is an on-line algorithm (training data can be used as they come).

Iterative Perceptron training



Perceptron training.

Batch Perceptron training

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i), i = 1, \dots, N\}$,an analytical solution to the Mean Squared Error function (MSE) minimization problem is possible:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 .$$

- $J(\mathbf{w})$ gradient is given by:

$$\nabla J(\mathbf{w}) = \mathbf{X}^T\mathbf{X}\mathbf{w}^T - \mathbf{X}^T\mathbf{y}.$$

- $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]^T$: $N \times n$ data matrix.
- $\mathbf{y} = [y_1, \dots, y_N]^T$: target vector.

Batch Perceptron training

- Setting $\nabla J(\mathbf{w}) = \mathbf{0}$ gives us :

$$\mathbf{X}^T\mathbf{X}\mathbf{w}^T = \mathbf{X}^T\mathbf{y}.$$

- Therefore, MSE minimization solution is given by:

$$\mathbf{w}^T = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

- Matrix $\mathbf{X}^\dagger = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is known as the ***pseudoinverse*** of matrix $\mathbf{X}$.
- It has the property:

$$\mathbf{X}^\dagger\mathbf{X} = \mathbf{I}.$$

Batch Perceptron training

- If $(\mathbf{X}^T\mathbf{X})^{-1}$ is singular, then we can use:

$$\mathbf{X}^{\dagger} \triangleq \lim_{\epsilon \to 0} (\mathbf{X}^T\mathbf{X} - \epsilon\mathbf{I})^{-1}\mathbf{X}^T.$$

- The solution is given by:

$$\mathbf{w} = (\mathbf{X}^{\dagger}\mathbf{y})^T.$$

Bibliography

[GOO2016] Goodfellow I, Bengio Y, Courville A, Bengio Y., *Deep learning* MIT press, 2016.

[HAY2009] S. Haykin, *Neural networks and learning machines*, Prentice Hall, 2009.

[BIS2006] C.M. Bishop, Pattern recognition and machine learning, Springer, 2006.

[GOO2016] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, MIT press, 2016

[THEO2011] S. Theodoridis, K. Koutroumbas, Pattern Recognition, Elsevier, 2011.

[ZUR1992] J.M. Zurada, *Introduction to artificial neural systems*. Vol. 8. St. Paul: West publishing company, 1992.

[ROS1958] Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review* 65.6 (1958): 386.

[YEG2009] Yegnanarayana, Bayya. *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[DAN2013] Daniel, Graupe. *Principles of artificial neural networks*. Vol. 7. World Scientific, 2013.

[HOP1988] Hopfield, John J. "Artificial neural networks." *IEEE Circuits and Devices Magazine* 4.5 (1988): 3-10.

[SZE2013] C. Szegedy, A. Toshev, D. Erhan. "Deep neural networks for object detection." *Advances in neural information processing systems*. 2013.

Bibliography

[SAL2016] T. Salimans, D. P. Kingma "Weight normalization: A simple reparameterization to accelerate training of deep neural networks." *Advances in Neural Information Processing Systems*. 2016.

[MII2019] Miikkulainen, Risto, et al. "Evolving deep neural networks." *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Academic Press, 2019. 293-312.

[EXT] https://www.extremetech.com/extreme/144248-neuristors-the-future-of-brain-like-computer-chips

Bibliography

[1] I. Pitas, “Artificial Intelligence Science and Society Part A: Introduction to AI Science and Information Technology“, Amazon/Kindle Direct Publishing, 2022,
https://www.amazon.com/dp/9609156460?ref_=pe_3052080_397514860

[2] I. Pitas, “Artificial Intelligence Science and Society Part B: AI Science, Mind and Humans“, Amazon/Kindle Direct Publishing, 2022,
https://www.amazon.com/dp/9609156479?ref_=pe_3052080_397514860

[3] I. Pitas, “Artificial Intelligence Science and Society Part C: AI Science and Society”, Amazon/Kindle Direct Publishing, 2022,
https://www.amazon.com/dp/9609156487?ref_=pe_3052080_397514860

[4] I. Pitas, “Artificial Intelligence Science and Society Part D: AI Science and the Environment“, Amazon/Kindle Direct Publishing, 2022,
https://www.amazon.com/dp/9609156495?ref_=pe_3052080_397514860

Q & A

Thank you very much for your attention!

More material in
http://icarus.csd.auth.gr/cvml-web-lecture-series/

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