



Τεχνητή Νοημοσύνη

12η διάλεξη (2025-26)

Ίων Ανδρουτσόπουλος

<http://www.aueb.gr/users/ion/>

Οι διαφάνειες αυτές βασίζονται σε ύλη των βιβλίων:

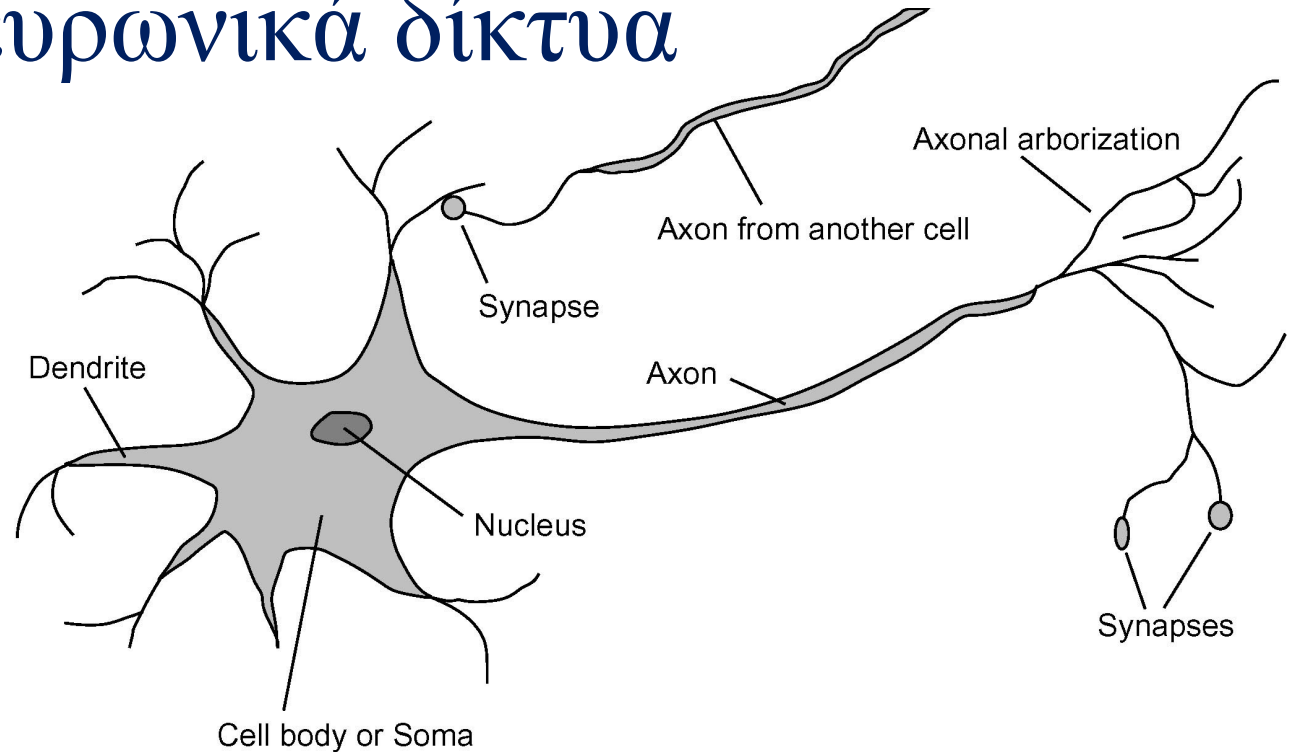
- *Artificial Intelligence – A Modern Approach* των S. Russel και P. Norvig, 2^η και 4^η έκδοση, Prentice Hall, 2003 και 2020,
- *Τεχνητή Νοημοσύνη* των Βλαχάβα κ.ά., 3η έκδοση, Β. Γκιούρδας Εκδοτική, 2006,
- *Machine Learning* του T. Mitchell, McGraw-Hill, 1997.

Τα περισσότερα σχήματα των διαφανειών προέρχονται από τα παραπάνω βιβλία και διαλέξεις.

Τι θα ακούσετε σήμερα

- Νευρωνικά δίκτυα.
 - Φυσικά και τεχνητά νευρωνικά δίκτυα.
 - Perceptron.
 - Πολύ-επίπεδα Perceptron (MLPs).
 - Εισαγωγή στον αλγόριθμο ανάστροφης μετάδοσης (back-propagation).

Φυσικά νευρωνικά δίκτυα

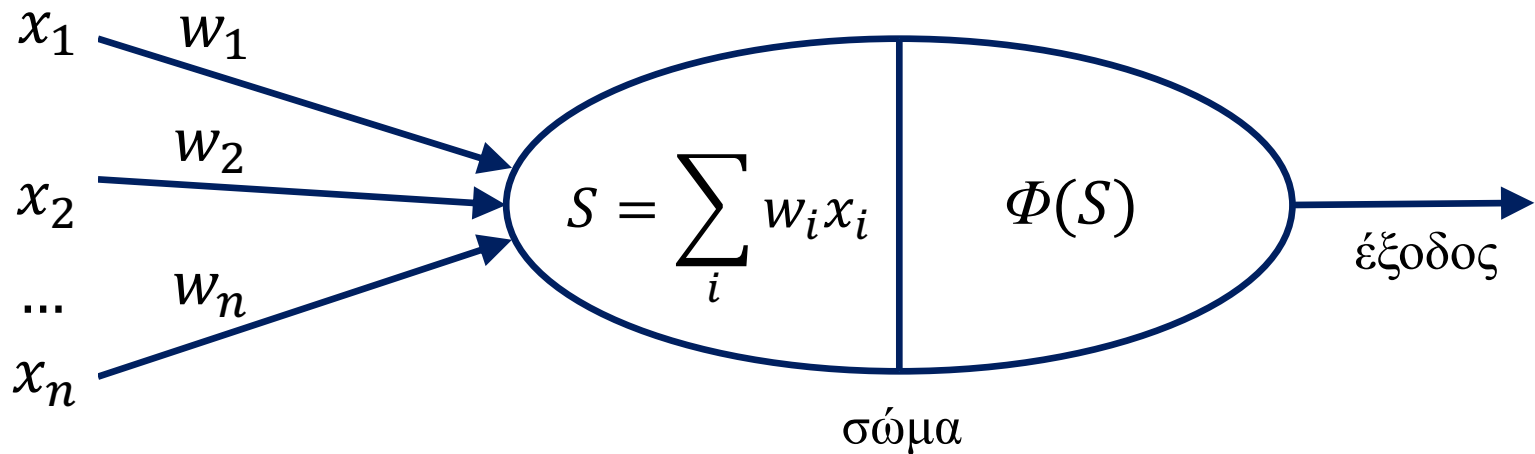


- **Νευρώνας:** κύτταρο του εγκεφάλου.
 - **Σώμα:** το κυρίως μέρος, που περιέχει τον **πυρήνα**.
 - **Δενδρίτες:** λαμβάνουν σήματα από άλλους νευρώνες.
 - **Άξονας:** παρέχει κοινή έξοδο προς πολλούς άλλους νευρώνες.
 - **Σύναψη:** μέσω ηλεκτροχημικών μηχανισμών μεταβάλλεται η αγωγιμότητά της.
- **Νευρωνικό δίκτυο:** δίκτυο συνδεδεμένων νευρώνων.

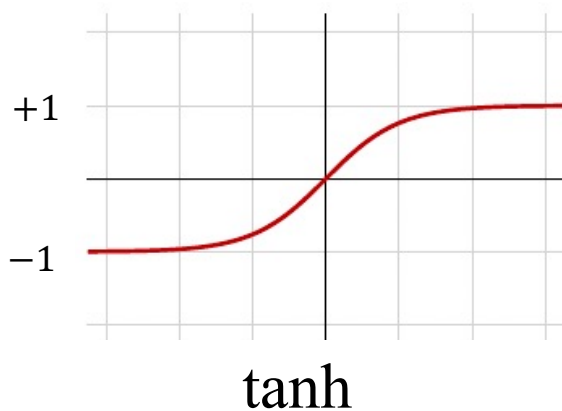
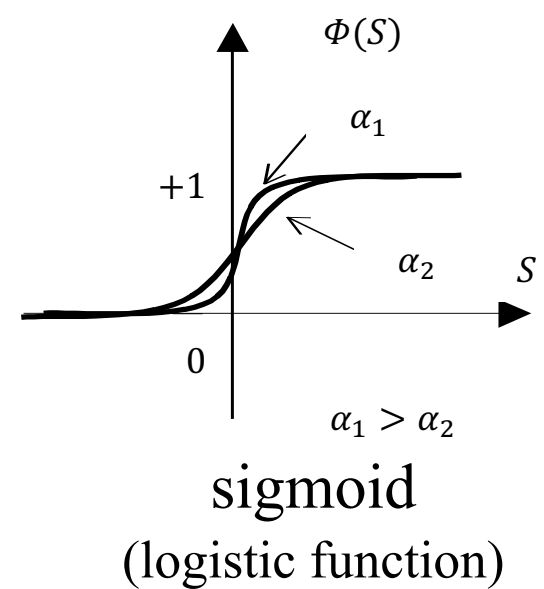
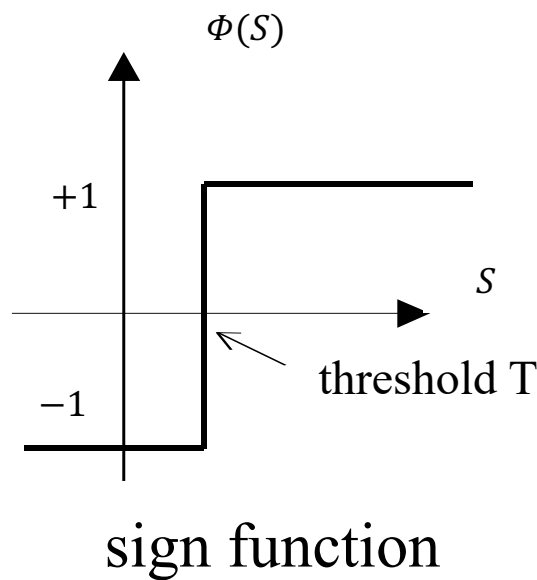
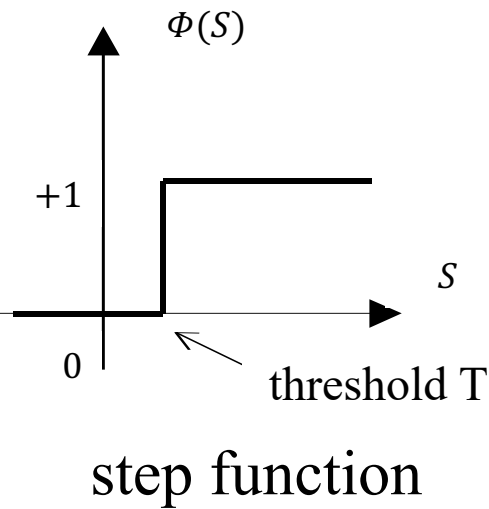
Τεχνητά νευρωνικά δίκτυα

- **Τεχνητός νευρώνας:**

- **Είσοδοι:** πραγματικές τιμές.
- **Βάρη εισόδων:** πραγματικές τιμές (χονδρικά συνάψεις).
- **Σώμα:** υπολογίζει το ζυγισμένο άθροισμα των εισόδων, κατόπιν εφαρμόζει τη **συνάρτηση ενεργοποίησης** στο ζυγισμένο άθροισμα.



Activation functions

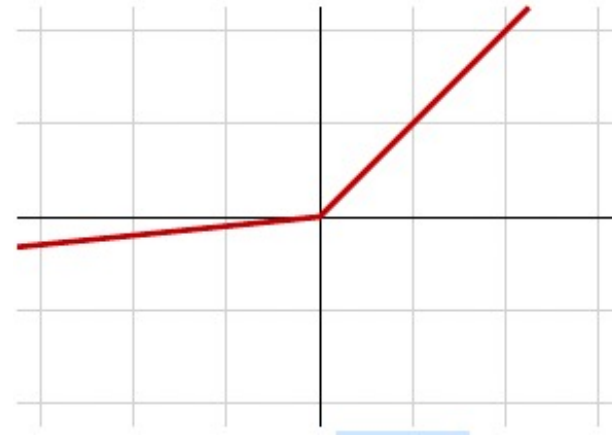
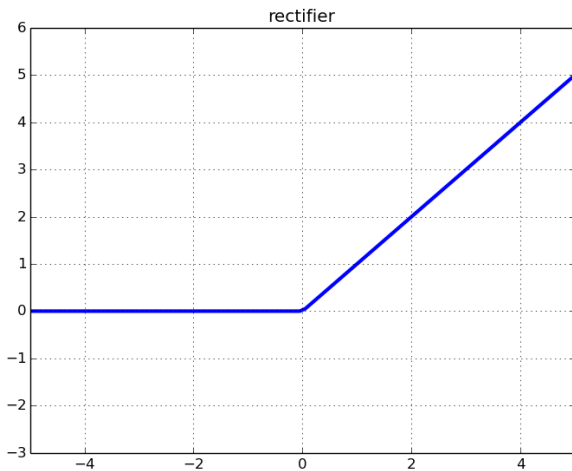


$$\Phi(S) = \frac{1}{1 + e^{-a \cdot S}}$$

The **sigmoid** is **differentiable**.

The hyperbolic tangent (**tanh**) is very similar to the sigmoid, but with values from -1 to $+1$. Usually better, unless we really want values in $(0, 1)$.

Activation functions – continued



Rectified Linear Unit (ReLU)

$$\text{relu}(x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{otherwise} \end{cases}$$

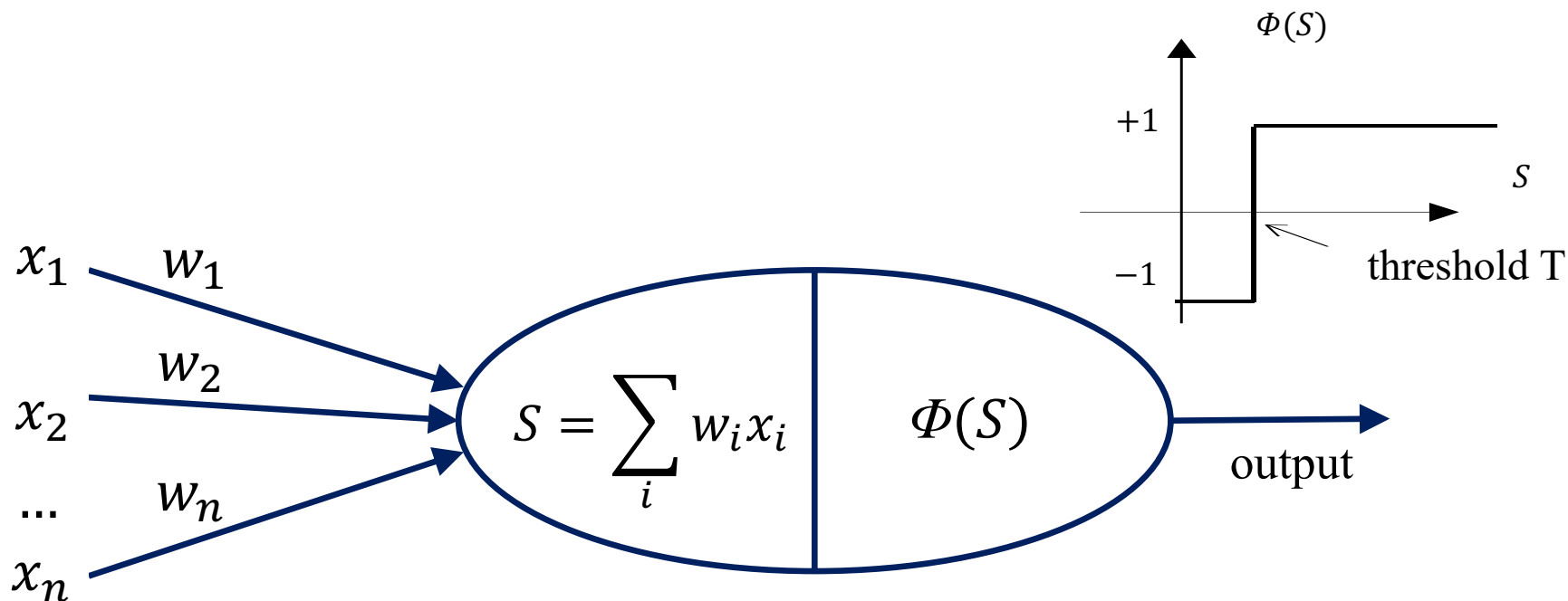
$$\text{leakyrelu}(x) = \begin{cases} \alpha x & \text{if } x < 0, \\ x & \text{otherwise} \end{cases}$$

α is a small positive.

ReLU and variants are popular choices.

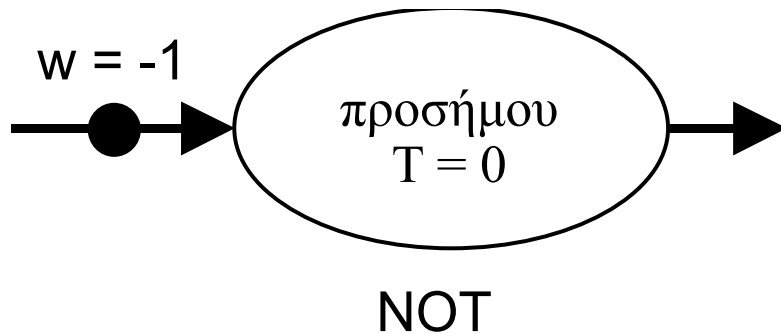
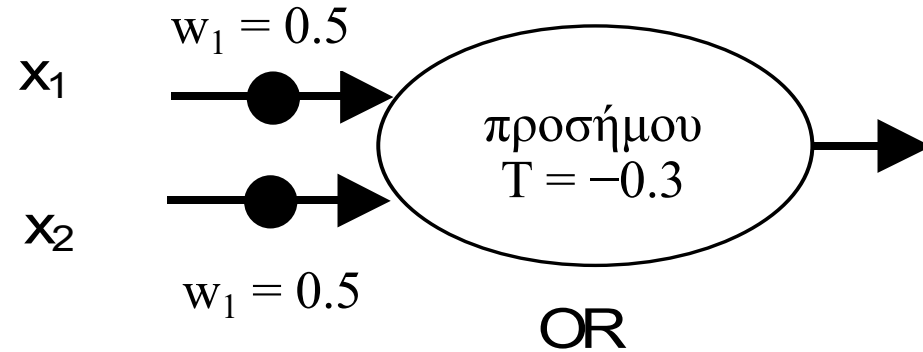
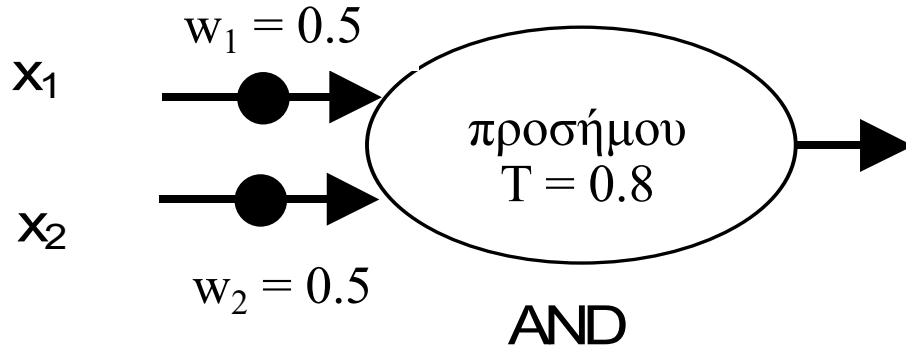
Perceptron

- Μόνο ένας νευρώνας, με συνάρτηση ενεργοποίησης **προσήμου**.
 - **Ισοδύναμα**, με **βηματική** (step) συνάρτηση ενεργοποίησης.
 - Το Perceptron μπορεί να **γενικευθεί** (έγινε αργότερα), ώστε να χρησιμοποιεί **σιγμοειδή ή άλλη** συνάρτηση ενεργοποίησης.
 - Ένα **μεμονωμένο Perceptron** είναι **γραμμικός διαχωριστής**.



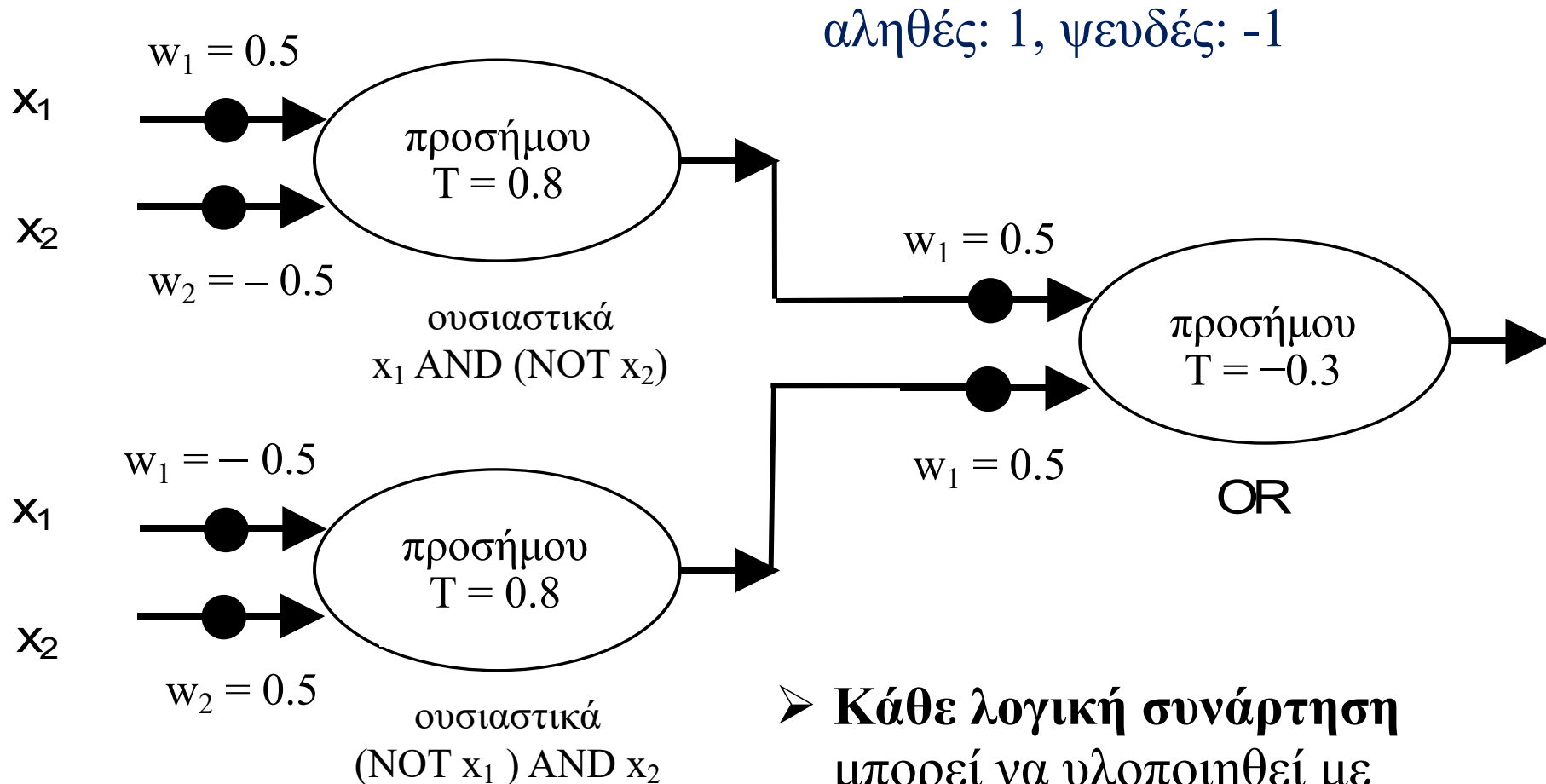
Λογικές συναρτήσεις με Perceptron

αληθές: 1, ψευδές: -1



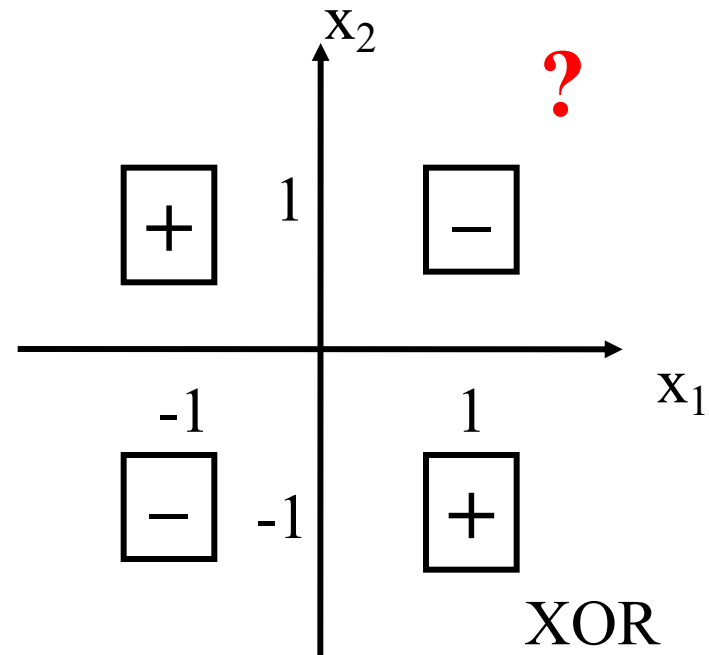
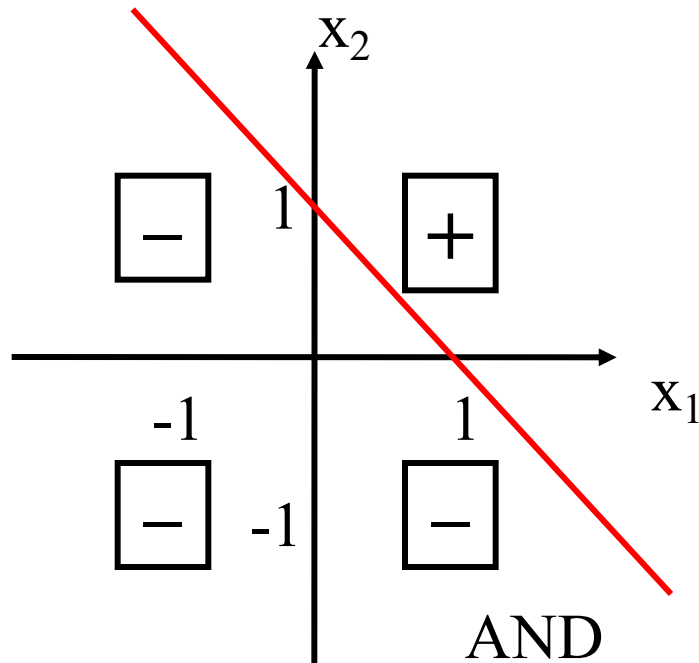
- Δεν μπορούν να υλοποιηθούν όλες οι λογικές συναρτήσεις με μόνο ένα Perceptron.
 - Π.χ. XOR.

Υλοποίηση XOR με δύο επίπεδα



➤ **Κάθε λογική συνάρτηση μπορεί να υλοποιηθεί με δίκτυο Perceptrons δύο επιπέδων.**

Γραμμική διαχωρισιμότητα



- Το **Perceptron** (όπως και ένας ταξινομητής λογιστικής παλινδρόμησης) μπορεί να μάθει **μόνο γραμμικά διαχωρίσιμα** προβλήματα (άσκηση μελέτης).
- Για **μη γραμμικά διαχωρίσιμα** προβλήματα, χρειαζόμαστε **πολυ-επίπεδο Perceptron (MLP, βλ. παρακάτω)**.

Perceptron's original learning algorithm

1. Start with random weights \vec{w} .
 2. Set $i \leftarrow 1$ and $s \leftarrow 0$.
 3. Let $t^{(i)}$ be the correct output for the i -th training instance and $o^{(i)}$ the current output for that instance.
 4. Set $s \leftarrow s + E_i(\vec{w})$, with: $E_i(\vec{w}) = 1/2 \cdot [t^{(i)} - o^{(i)}]^2$
 5. Update the weights: $w_l \leftarrow w_l + \eta \cdot (t^{(i)} - o^{(i)}) \cdot x_l^{(i)}$
 6. If there is a next training instance, set $i \leftarrow i + 1$ and go to step 3.
 7. If s has not converged and max number of scans (epochs) of training data not exceeded go to step 2.
- In the simplest case, η is a small positive constant.

Perceptron with sigmoid or other Φ

- More generally, when using a Perceptron with **activation function Φ instead of step/sign**:

$$w_l \leftarrow w_l + \eta \cdot \Phi' \left(\sum_l w_l x_l^{(i)} \right) \cdot (t^{(i)} - \Phi \left(\sum_l w_l x_l^{(i)} \right)) \cdot x_l^{(i)}$$

- For sigmoid $\Phi(S) = \frac{1}{1 + e^{-S}}$, $\Phi'(S) = \Phi(S) \cdot (1 - \Phi(S))$.

- And since $o^{(i)} = \Phi \left(\sum_l w_l x_l^{(i)} \right)$, the weights update rule becomes:

$$w_l \leftarrow w_l + \eta \cdot o^{(i)} \cdot (1 - o^{(i)}) \cdot (t^{(i)} - o^{(i)}) \cdot x_l^{(i)}$$

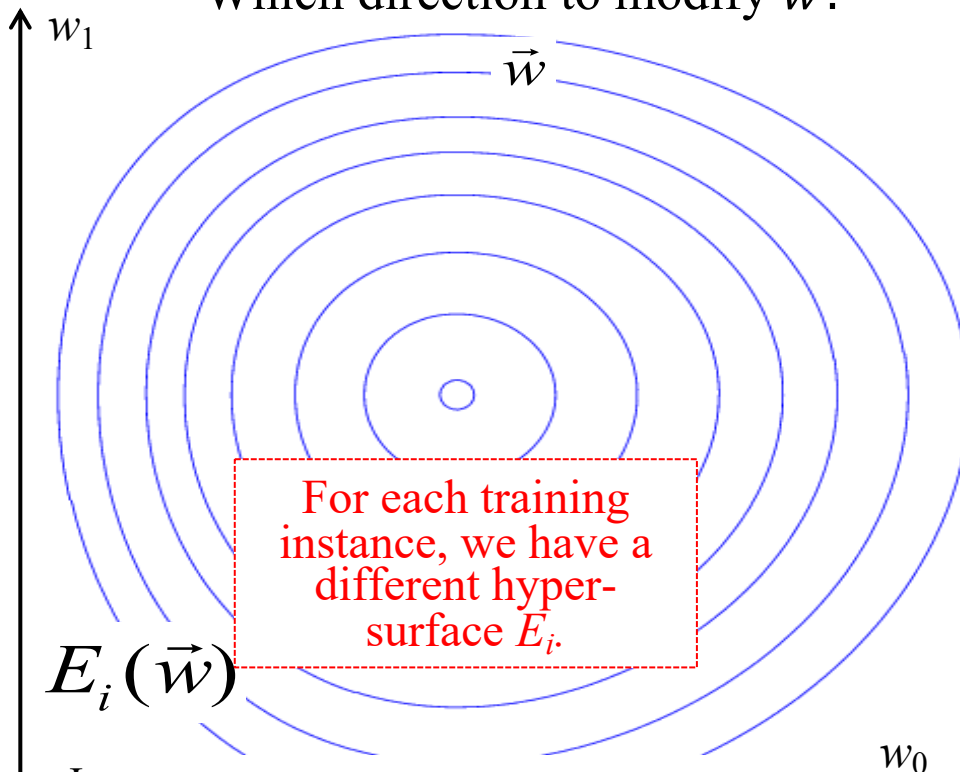
Derivation of the update rule

- **Squared error loss on the current training instance:**

$$E_i(\vec{w}) = \frac{1}{2} (t^{(i)} - o^{(i)})^2 = \frac{1}{2} (t^{(i)} - \Phi \left(\sum_l w_l x_l^{(i)} \right))^2$$

Random initial weights. Loss on current training instance is $E_i(\vec{w})$.

Which direction to modify \vec{w} ?



For each training instance, we have a different hyper-surface E_i .

The **gradient** $\nabla E_i(\vec{w})$ is a vector showing the **direction** we need to **modify** \vec{w} to obtain the **steepest increase** of $E_i(\vec{w})$.

At each iteration, take a **step** to the direction $-\nabla E_i(\vec{w})$:

$$\vec{w} \leftarrow \vec{w} - \eta \cdot \nabla E_i(\vec{w})$$

Simplest case: η is a **small positive constant**. **Stochastic gradient descent** to minimize the **total squared error loss**.

Image source:

http://en.wikipedia.org/wiki/Gradient_descent →

Derivation of the update rule

$$E_i(\vec{w}) = \frac{1}{2} (t^{(i)} - o^{(i)})^2 = \frac{1}{2} (t^{(i)} - \Phi \left(\sum_{l=1}^n w_l x_l^{(i)} \right))^2$$

$$\nabla E_i(\vec{w}) = \left\langle \frac{\partial E_i(\vec{w})}{\partial w_0}, \dots, \frac{\partial E_i(\vec{w})}{\partial w_l}, \dots, \frac{\partial E_i(\vec{w})}{\partial w_n} \right\rangle$$

$$\frac{\partial E_i}{\partial w_l} = (t^{(i)} - \Phi \left(\sum_l w_l x_l^{(i)} \right)) \cdot \frac{\partial (t^{(i)} - \Phi \left(\sum_l w_l x_l^{(i)} \right))}{\partial w_l} = -(t^{(i)} - o^{(i)}) \cdot \Phi' \left(\sum_l w_l x_l^{(i)} \right) \cdot x_l^{(i)}$$

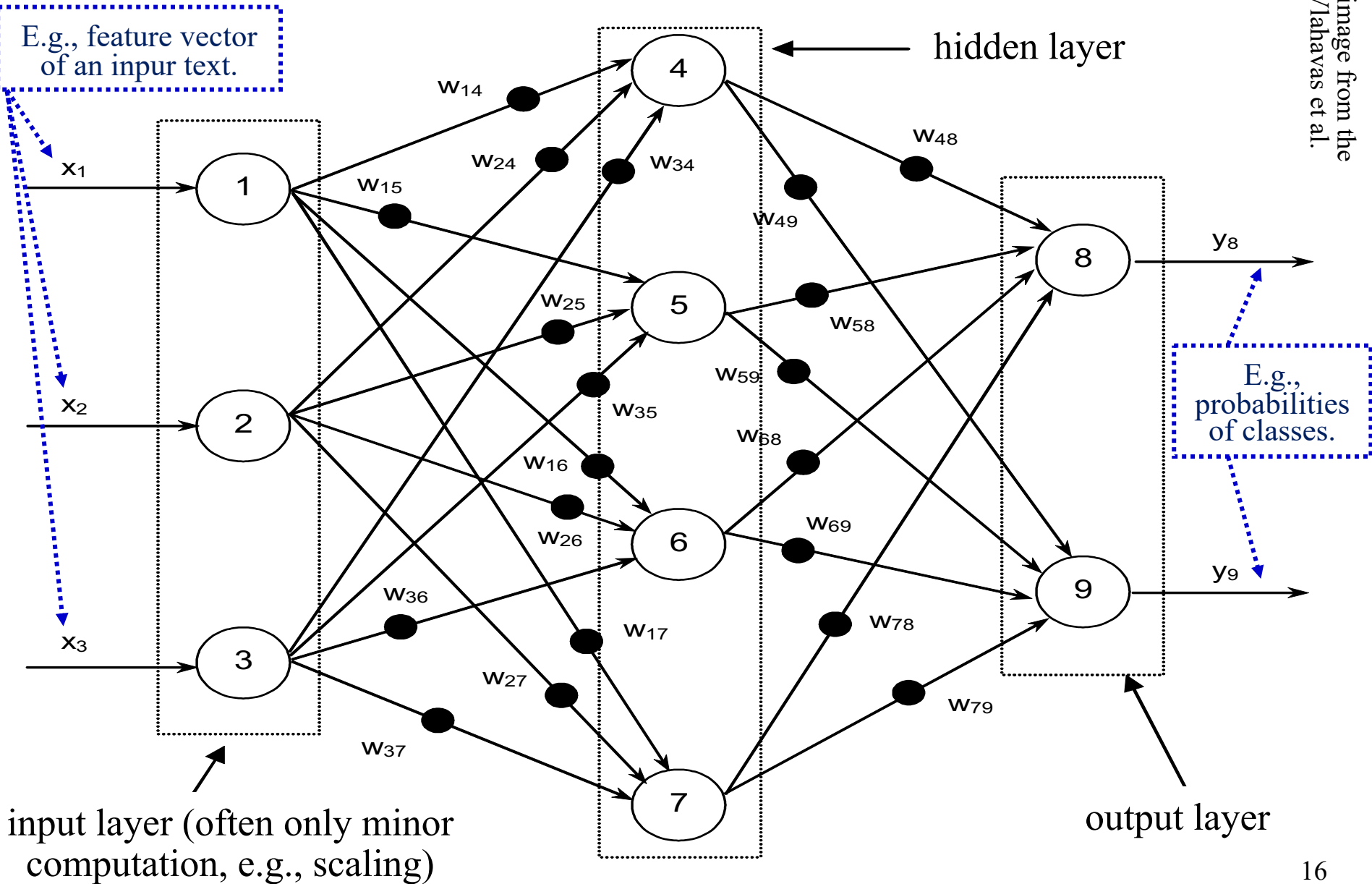
$$\begin{aligned} \text{Hence: } \nabla E_i(\vec{w}) &= -(t^{(i)} - o^{(i)}) \cdot \Phi' \left(\sum_l w_l x_l^{(i)} \right) \cdot \langle x_1^{(i)}, \dots, x_n^{(i)} \rangle \\ &= -(t^{(i)} - o^{(i)}) \cdot \Phi'(\vec{w} \cdot \vec{x}^{(i)}) \cdot \vec{x}^{(i)} \end{aligned}$$

Weights update rule:

$$\vec{w} \leftarrow \vec{w} - \eta \cdot \nabla E_i(\vec{w}) = \vec{w} + \eta \cdot (t^{(i)} - o^{(i)}) \cdot \Phi'(\vec{w} \cdot \vec{x}^{(i)}) \cdot \vec{x}^{(i)}$$

$$\text{For each weight: } w_l \leftarrow w_l + \eta \cdot (t^{(i)} - o^{(i)}) \cdot \Phi'(\vec{w} \cdot \vec{x}^{(i)}) \cdot x_l^{(i)}$$

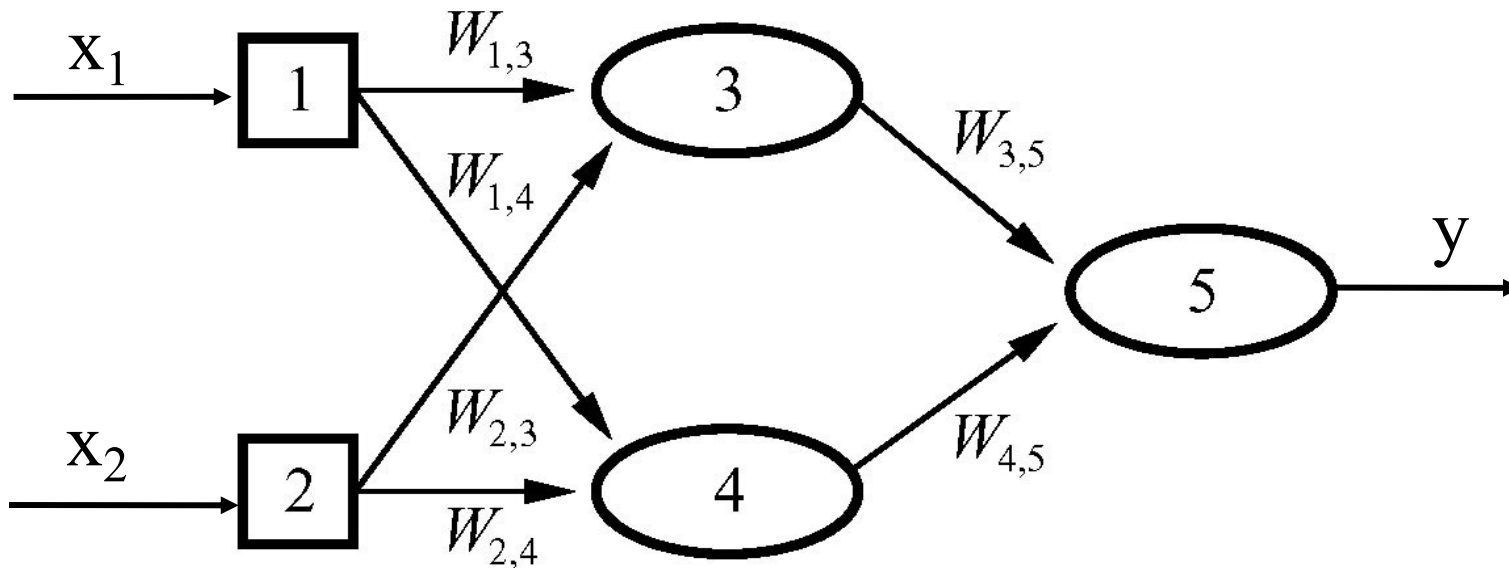
Multi-layer Perceptron (MLP)



Πολυ-επίπεδο Perceptron (MLP)

- Νευρώνες **εισόδου**: συχνά χωρίς ουσιαστικούς υπολογισμούς.
 - Π.χ. μόνο μία είσοδος στον καθένα, μοναδιαίο βάρος, $\Phi(S) = S$.
 - Χρήσιμοι αν θέλουμε να προσθέσουμε μια σταθερά στην κάθε είσοδο ή να πολλαπλασιάσουμε την είσοδο με κάποιο βάρος (ή πίνακα).
- Επίπεδο **εξόδου** και **ενδιάμεσα** («κρυφά») επίπεδα.
- ΤΝΔ με **απλή τροφοδότηση** (feed-forward networks):
 - Οι είσοδοι ενός νευρώνα **δεν μπορούν** να προέρχονται από νευρώνες του **ίδιου** ή **επόμενου** επιπέδου.
- ΤΝΔ με **ανατροφοδότηση** (recurrent networks, RNNs):
 - Είσοδοι και από νευρώνες του **ίδιου** ή **επόμενου** επιπέδου.
 - Η έξοδος εξαρτάται και από την **προηγούμενη κατάσταση** του δικτύου.
 - Χρησιμοποιούνται σε ΤΝΔ που επεξεργάζονται **ακολουθίες** (π.χ. ακολουθίες λέξεων κειμένων).

Μάθηση με ΤΝΔ



- $y = \Phi(w_{3,5} \cdot x_{3,5} + w_{4,5} \cdot x_{4,5}) =$
 $\Phi(w_{3,5} \cdot \Phi(w_{1,3} \cdot x_1 + w_{2,3} \cdot x_2) + w_{4,5} \cdot \Phi(w_{1,4} \cdot x_1 + w_{2,4} \cdot x_2))$
- Μεταβάλλοντας τα **βάρη** μαθαίνουμε **διαφορετική συνάρτηση**.
- **Ποιες συναρτήσεις** μπορούμε να μάθουμε; Εξαρτάται από τη Φ , το **πλήθος των επιπέδων**, το **πλήθος νευρώνων ανά επίπεδο** κ.λπ.

Μάθηση με ΤΝΔ

- **Επιβλεπόμενη μάθηση:**

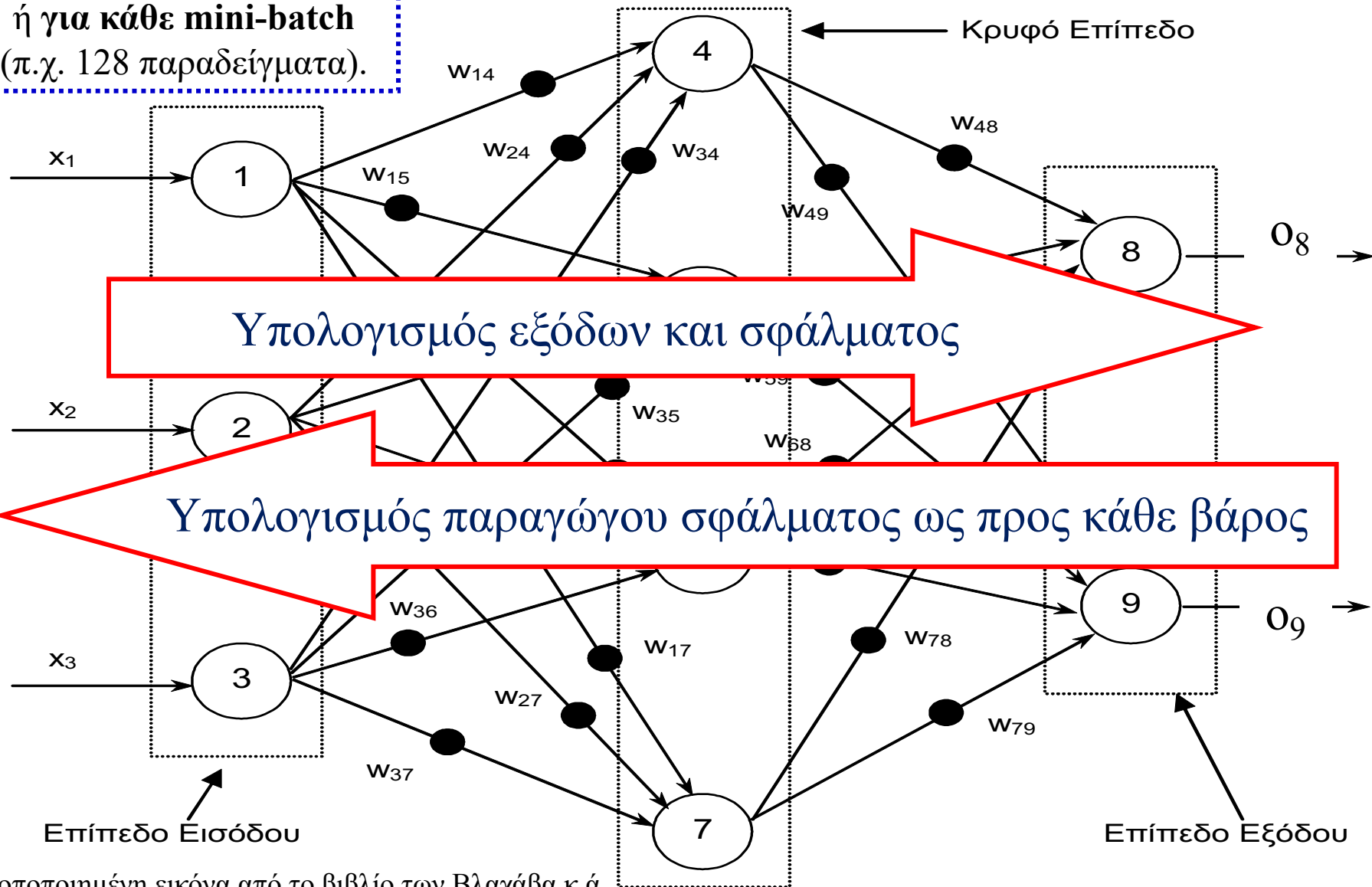
- **Εκπαίδευση:** παρέχονται είσοδοι και οι επιθυμητές αποκρίσεις. Τα **βάρη προσαρμόζονται**, ώστε να παράγονται οι επιθυμητές αποκρίσεις.
- **Χρήση:** παρέχονται είσοδοι. Χρησιμοποιούμε τα **βάρη που προέκυψαν από την εκπαίδευση** και εμπιστευόμαστε τις αποκρίσεις.
- Κατάταξη (κατηγοριοποίηση) σε **δύο αμοιβαία αποκλειόμενες κατηγορίες:** συνήθως **ένας νευρώνας εξόδου**. Αν η έξοδός του ξεπερνά ένα **κατώφλι**, κατατάσσουμε στη μία κατηγορία, διαφορετικά στην άλλη.
- Με περισσότερες **κατηγορίες:** συνήθως **ένας νευρώνας εξόδου για κάθε κατηγορία**.
- Μάθηση συνεχούς συνάρτησης (προβλήματα **παλινδρόμησης**): συνήθως **ένας ή περισσότεροι νευρώνες εξόδου** (π.χ. στροφή, επιτάχυνση αυτοκινήτου).

- **Μη επιβλεπόμενη μάθηση:**

- Π.χ. ΤΝΔ μπορούν να χρησιμοποιηθούν και για **ομαδοποίηση** (clustering).

Αλγόριθμος ανάστροφης μετάδοσης

Για κάθε ένα παράδειγμα
ή για κάθε mini-batch
(π.χ. 128 παραδείγματα).



Αλγόριθμος ανάστροφης μετάδοσης (back-propagation)

- Χρησιμοποιείται για την εκπαίδευση ΤΝΔ **απλής τροφοδότησης με πολλαπλά επίπεδα**.
 - Μπορεί να χρησιμοποιηθεί (με κάποιες προσαρμογές) και για την εκπαίδευση ΤΝΔ με ανατροφοδότηση (RNNs).
 - Η προηγούμενη διαφάνεια δείχνει μόνο ένα κρυφό επίπεδο αλλά ο αλγόριθμος δουλεύει και για **πολλαπλά κρυφά επίπεδα**.
 - Σε προβλήματα **παλινδρόμησης**, το **τοπικό σφάλμα** (σφάλμα στο τρέχον παράδειγμα) είναι συνήθως: $E = \frac{1}{n} \sum_{i=1}^n (o_i - y_i)^2$
- Συμβολισμός:
 - **Απόκριση** του ΤΝΔ σε ένα παράδειγμα: $\langle o_1, \dots, o_n \rangle$
 - Σωστή απόκριση: $\langle y_1, \dots, y_n \rangle$
 - **Βάρος** σύνδεσης από νευρώνα i σε νευρώνα j : w_{ij}
 - **Είσοδος** από το νευρώνα i στο νευρώνα j : x_{ij}

Αλγόριθμος ανάστροφης μετάδοσης

- Αρχικοποίησε όλα τα βάρη σε μικρές τυχαίες τιμές.
- Όσο δεν ικανοποιείται η συνθήκη τερματισμού:

- Νέα εποχή: Για κάθε παράδειγμα εκπαίδευσης:

- Υπολόγισε (από αριστερά προς τα δεξιά) τις αποκρίσεις στο επίπεδο εξόδου.

- Υπολόγισε το σφάλμα E στο επίπεδο εξόδου.

- Για κάθε βάρος w_{ij} , υπολόγισε (από δεξιά προς τα

αριστερά) την παράγωγο $\frac{\partial E}{\partial w_{ij}}$ (δηλ. πώς επηρεάζει το E).

- Ενημέρωσε κάθε βάρος w_{ij} :

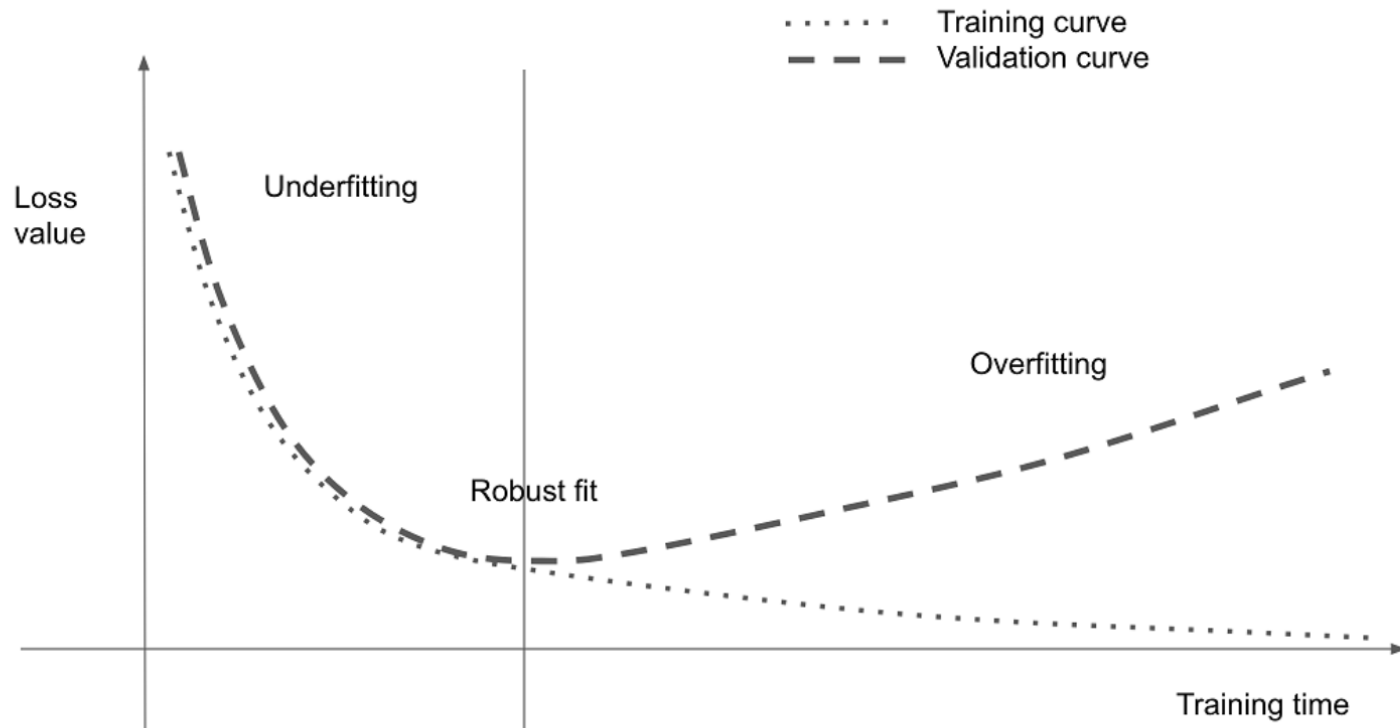
$$w_{ij} \leftarrow w_{ij} - \eta \cdot \frac{\partial E}{\partial w_{ij}}$$

Για όλα τα βάρη μαζί:
 $W \leftarrow W - \eta \cdot \nabla_W E$

Θα δούμε πώς
στην επόμενη
διάλεξη.

Σφάλμα συναρτήσεως των εποχών

Figure 5.1. Canonical overfitting behavior



Σχήμα από το προτεινόμενο βιβλίο «**Deep Learning with Python**» του F. Chollet, Manning Publications, 2η έκδοση. Η 1^η έκδοση παρέχεται δωρεάν.

<https://www.manning.com/books/deep-learning-with-python>

<https://www.manning.com/books/deep-learning-with-python-second-edition>

Ανάστροφη μετάδοση: περισσότερα

- Συνθήκη **τερματισμού**:
 - υπερβήκαμε έναν **μέγιστο αριθμό εποχών** ή
 - **early stopping**: το **συνολικό σφάλμα εποχής** υπολογισμένο σε (διαφορετικά) παραδείγματα **επικύρωσης** (ή «ανάπτυξης») έχει αρχίσει πλέον να χειροτερεύει λόγω υπερ-εφαρμογής.
 - Συνήθως **περιμένουμε και μερικές ακόμα εποχές (patience)**, μήπως το σφάλμα επικύρωσης αρχίσει **πάλι να πέφτει**.
- **Δεν εγγυάται** ότι θα βρει τη **βέλτιστη λύση**:
 - Ουσιαστικά **στοχαστική κατάβαση κλίσης (SGD)**.
 - Βλ. <https://aclanthology.org/2024.eacl-long.157/> για παραλλαγές του SGD (π.χ. **Adam optimizer**).
 - Κίνδυνος εγκλωβισμού σε **τοπικό μέγιστο**. Αντιμετώπιση: **εκπαιδεύουμε πολλές φορές με διαφορετικά αρχικά βάρη**. Επιλέγουμε την επανάληψη με τα **καλύτερα τελικά βάρη**, αξιολογώντας σε **δεδομένα ανάπτυξης/επικύρωσης**.

Το σκεφτόμαστε πια συνήθως ως αλγόριθμο, αλλά ήταν μηχανήμα!

The perceptron is a machine



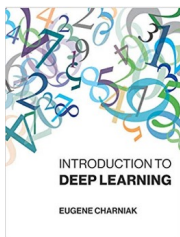
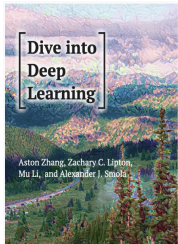
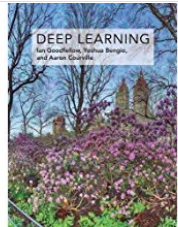
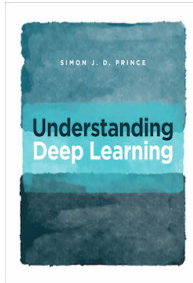
Από την παρουσίαση «Multilayer Neural Networks» του L. Bottou στο Deep Learning Summer School 2015 (βλ. http://videlectures.net/deeplearning2015_montreal/).

Βιβλιογραφία

- Russel & Norvig (4^η έκδοση, ελληνική μετάφραση): κεφάλαιο 21 ως και ενότητα 21.2.3.
 - Θα καλύψουμε αρκετές άλλες ενότητες αυτού του κεφαλαίου σε επόμενες διαλέξεις.
- Βλαχάβας κ.ά: ενότητες 18.10.1, 19.1–19.5.
 - Όσοι ενδιαφέρονται μπορούν να διαβάσουν προαιρετικά και τις υπόλοιπες ενότητες του κεφαλαίου 19.
- Αν σας λείπει μαθηματικό υπόβαθρο, ιδιαίτερα στις ασκήσεις μελέτης, συμβουλευτείτε το:
 - T. Parr και J. Howard, «The Matrix Calculus You Need for Deep Learning». <https://explained.ai/matrix-calculus/>

Πρόσθετη προτεινόμενη βιβλιογραφία

- Τα τελευταία χρόνια η βαθιά μάθηση (deep learning) έχει επιδείξει εξαιρετικά αποτελέσματα σε πολλές εφαρμογές.
 - Το βιβλίο «Understanding Deep Learning» του S.J.D. Prince, MIT Press, διατίθεται δωρεάν: <https://udlbook.github.io/udlbook/>
 - Το βιβλίο «Deep Learning» των I. Goodfellow κ.ά., MIT Press διατίθεται δωρεάν: <http://www.deeplearningbook.org/>
 - Το διαδραστικό βιβλίο «Dive into Deep Learning» των Zhang κ.ά. διατίθεται δωρεάν: <https://d2l.ai/>
 - Το βιβλίο «Introduction to Deep Learning» του E. Charniak, MIT Press υπάρχει στη βιβλιοθήκη του ΟΠΑ.



Πρόσθετη προτεινόμενη βιβλιογραφία

- Ως πιο πρακτική εισαγωγή, προτείνεται το βιβλίο του F. Chollet «Deep Learning in Python», Manning Publications.
 - Καλύπτει σε μεγάλο βαθμό την ύλη αυτών των διαφανειών (βλ. κεφ. 1–4). Αποτελεί και εισαγωγή στο Keras (πλέον μέρος της βιβλιοθήκης Tensorflow).
 - Η 1^η έκδοση (2017) παρέχεται δωρεάν: <https://www.manning.com/books/deep-learning-with-python>
 - Η 2^η έκδοση (2022) απαιτεί πληρωμή αλλά την αξίζει!
 - Αλλά στα φροντιστήρια χρησιμοποιούμε πλέον PyTorch.
- Η PyTorch είναι πλέον η πιο δημοφιλής βιβλιοθήκη ανάπτυξης νευρωνικών δικτύων.
 - Εισαγωγικά μαθήματα PyTorch: <https://pytorch.org/tutorials/>
 - Θα χρησιμοποιήσουμε PyTorch στα φροντιστήρια του μαθήματος και (πιθανότατα) στη 2^η εργασία.

