ΕΘΝΙΚΟ ΑΣΤΕΡΟΣΚΟΠΕΙΟ ΑΘΗΝΩΝ
Institute for Astronomy, Astrophysics, Space Applications and Remote Sensing (IAASARS)

ΕΑΡΙΝΟ ΣΧΟΛΕΙΟ ΒΑΡΥΤΗΤΑΣ ΚΑΙ ΚΟΣΜΟΛΟΓΙΑΣ

Εισαγωγή στη Μηχανική Μάθηση και Ορθές Πρακτικές Εφαρμογής

Μαρία Κασελίμη

Ερευνήτρια Γ’ Βαθμίδας, ΙΑΑΔΕΤ

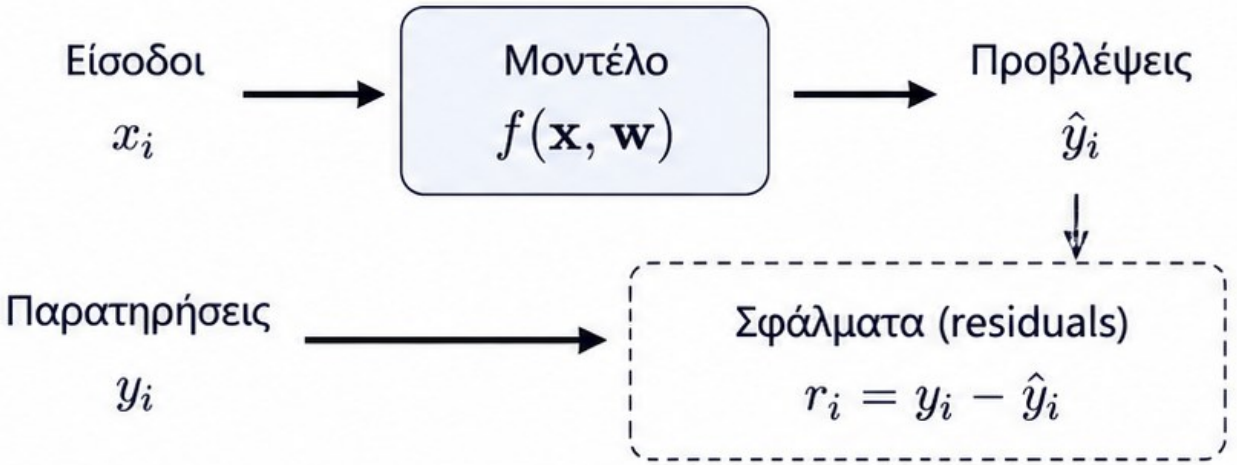
19 Μαϊου 2026

Εισαγωγή: Least Squares & Νευρωνικά Δίκτυα

Ιδέα

Η μέθοδος ελαχίστων τετραγώνων (LS)
και τα Νευρωνικά Δίκτυα (NN)
μοιράζονται θεμελιώδεις ιδέες.
Θα δούμε πώς έννοιες που ήδη γνωρίζουμε
από την προσαρμογή, αποτελούν
τη βάση των μεθόδων Μηχανικής Μάθησης.

Πρόβλημα (αντίστροφο πρόβλημα)



Κοινός στόχος

Να βρούμε τις παραμέτρους **w** που ελαχιστοποιούν
τη διαφορά μεταξύ παρατηρήσεων y_i και
προβλέψεων $\hat{y}_i = f(\mathbf{x}_i, \mathbf{w})$.

Στόχος (LS):

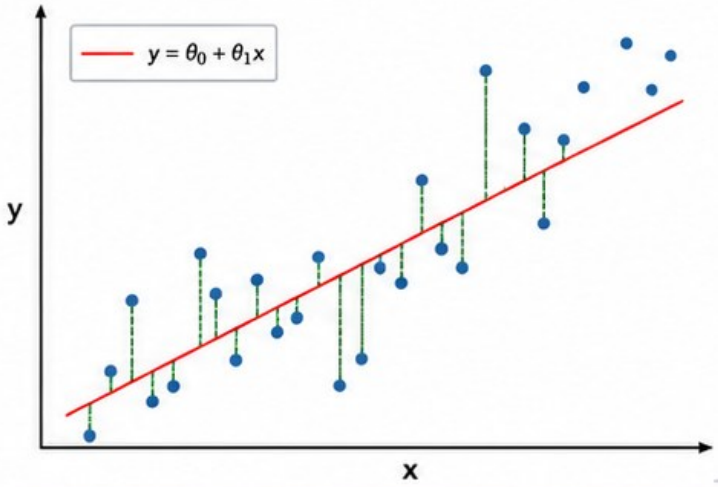
Ελαχιστοποίηση των τετραγώνων των residuals

$$E(\mathbf{w}) = \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

ΓΡΑΜΜΙΚΗ ΠΑΛΙΝΔΡΟΜΗΣΗ & ΜΕΓΙΣΤΗ ΠΙΘΑΝΟΦΑΝΕΙΑ

Από τα δεδομένα στη βέλτιστη εκτίμηση παραμέτρων

1. ΔΕΔΟΜΕΝΑ & ΓΡΑΜΜΙΚΟ ΜΟΝΤΕΛΟ



Έχουμε δεδομένα:

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$$

Γραμμικό μοντέλο:

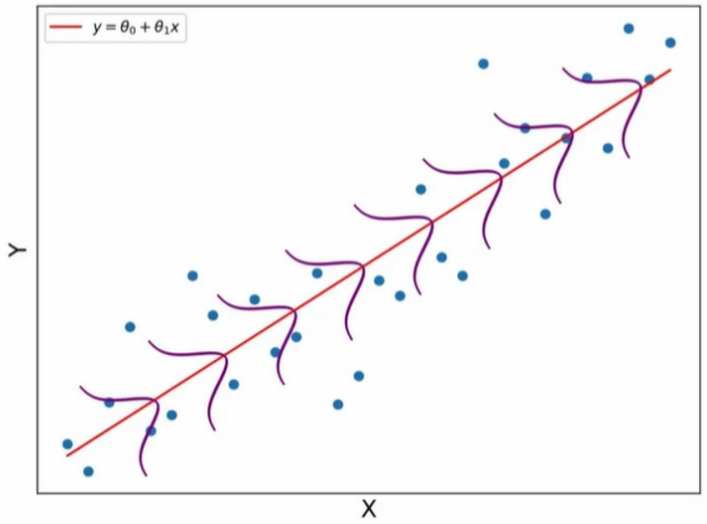
$$\hat{y}_i = \theta_0 + \theta_1 x_i$$

όπου:

θ_0 : σταθερός όρος (intercept)

θ_1 : κλίση (slope)

2. ΘΟΡΥΒΟΣ & ΠΙΘΑΝΟΤΗΤΑ (GAUSSIAN ΥΠΟΘΕΣΗ)



Υποθέτουμε ότι κάθε παρατήρηση:

$$y_i = \hat{y}_i + \epsilon_i = (\theta_0 + \theta_1 x_i) + \epsilon_i$$

Υπόθεση θορύβου:

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2)$$

(Gaussian με μέση τιμή 0 και διακύμανση σ^2)

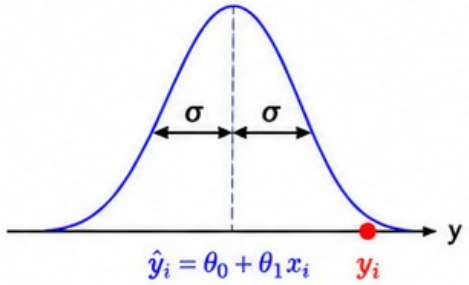
Άρα:

$$y_i \sim \mathcal{N}(\hat{y}_i, \sigma^2)$$

3. ΠΙΘΑΝΟΦΑΝΕΙΑ (LIKELIHOOD)

Η πιθανότητα να παρατηρήσουμε το y_i, δεδομένου x_i και παραμέτρων (θ_0, θ_1):

$$p(y_i \mid x_i, \theta_0, \theta_1) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \hat{y}_i)^2}{2\sigma^2}\right)$$



Για όλα τα δεδομένα $(i = 1, \ldots, N)$:

Πιθανοφάνεια:

$$L(\theta_0, \theta_1) = \prod_{i=1}^{N} p(y_i \mid x_i, \theta_0, \theta_1)$$

$$= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \hat{y}_i)^2}{2\sigma^2}\right)$$

Λογαριθμική πιθανοφάνεια:

$$\ell(\theta_0, \theta_1) = \log L(\theta_0, \theta_1)$$

$$= -\frac{N}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Συμπέρασμα: Επιλέγουμε τις παραμέτρους (θ_0, θ_1) που κάνουν τα παρατηρηθέντα δεδομένα να είναι όσο το δυνατόν πιο πιθανά σύμφωνα με το μοντέλο.

ΠΙΘΑΝΟΦΑΝΕΙΑ $L(\mathbf{w})$ ΚΑΙ ΜΕΓΙΣΤΗ ΠΙΘΑΝΟΦΑΝΕΙΑ

1 Αν έχουμε δεδομένα

Δεδομένα: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$

Μοντέλο: $\hat{y}_i = f(\mathbf{x}_i, \mathbf{w})$

Για κάθε παρατήρηση ορίζουμε την πιθανότητα:

$$p(y_i \mid \mathbf{x}_i, \mathbf{w})$$

- Εκφράζει πόσο πιθανό είναι να παρατηρήσουμε το y_i όταν το μοντέλο με παραμέτρους $\mathbf{w}$ δέχεται ως είσοδο το x_i.

2 Η πιθανοφάνεια

Για όλα τα δεδομένα μαζί, η πιθανοφάνεια είναι:

$$L(\mathbf{w}) = \prod_{i=1}^{N} p(y_i \mid \mathbf{x}_i, \mathbf{w})$$

Το $L(\mathbf{w})$ είναι ένα **score** για τις παραμέτρους $\mathbf{w}$: όσο μεγαλύτερο είναι, τόσο καλύτερα το μοντέλο εξηγεί τα δεδομένα.

παράμετροι $\mathbf{w}$ → μοντέλο $f(\mathbf{x}, \mathbf{w})$ → πιθανότητα δεδομένων $L(\mathbf{w})$

3 Μέγιστη πιθανοφάνεια

Αναζητούμε τις παραμέτρους που μεγιστοποιούν την πιθανοφάνεια:

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} L(\mathbf{w})$$

Συνήθως χρησιμοποιούμε και τη λογαριθμική πιθανοφάνεια:

$$\ell(\mathbf{w}) = \log L(\mathbf{w})$$

Για Gaussian θόρυβο, η μεγιστοποίηση της πιθανοφάνειας είναι ισοδύναμη με την ελαχιστοποίηση του αθροίσματος τετραγωνικών σφαλμάτων.

$$\sum_{i=1}^{N} \left(y_i - f(\mathbf{x}_i, \mathbf{w})\right)^2$$

Με απλά λόγια: η μέγιστη πιθανοφάνεια βρίσκει τις τιμές των παραμέτρων $\mathbf{w}$ που κάνουν τα παρατηρημένα δεδομένα όσο το δυνατόν πιο πιθανά σύμφωνα με το μοντέλο.

ΜΗ ΓΡΑΜΜΙΚΑ ΣΥΣΤΗΜΑΤΑ & ΜΕΓΙΣΤΗ ΠΙΘΑΝΟΦΑΝΕΙΑ

1 Από τη Γραμμική στη Μη Γραμμική Μοντελοποίηση

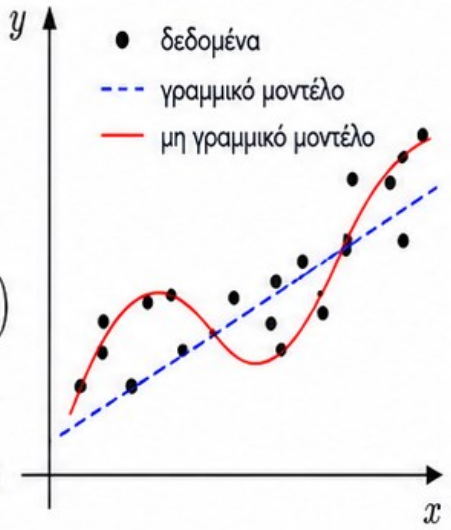
Γραμμικό μοντέλο

$$\hat{y} = w_0 + w_1 x_1 + \cdots + w_D x_D$$

Μη γραμμικό μοντέλο (γενίκευση)

$$\hat{y} = f(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \, \phi_j(\mathbf{x})$$

- $\phi_j(\mathbf{x})$: μη γραμμικές συναρτήσεις βάσης
- Παραδείγματα $\phi_j(\mathbf{x})$:
  - Πολυωνυμικές συναρτήσεις (π.χ. $x_1^2,\ x_1 x_2,\ \sin x_1$)
  - Radial Basis Functions (RBF): $\phi_j(\mathbf{x}) = \exp\left(-\gamma \|\mathbf{x} - \mathbf{c}_j\|^2\right)$
  - Activation Functions (Neural Nets): Sigmoid, ReLU, tanh, ...
- M: συνολικός αριθμός παραμέτρων (βάρη w_j και bias w_0)



2 Υπόθεση Gaussian Θορύβου

Θεωρούμε ότι κάθε παρατήρηση y_i ακολουθεί Gaussian κατανομή γύρω από την πρόβλεψη $f(\mathbf{x}_i, \mathbf{w})$:

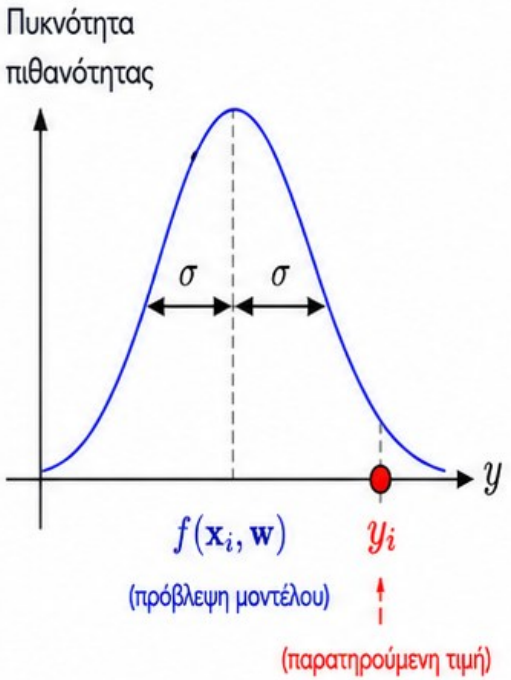
$$P(y_i \,|\, \mathbf{x}_i, \mathbf{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - f(\mathbf{x}_i, \mathbf{w}))^2}{2\sigma^2}\right)$$

Ισοδύναμα:

$$y_i = f(\mathbf{x}_i, \mathbf{w}) + \epsilon_i$$

με $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$

- σ^2: γνωστή και σταθερή διακύμανση
- ϵ_i: ανεξάρτητος Gaussian θόρυβος



Άρα τα δεδομένα θεωρούνται: $y_i \sim \mathcal{N}\left(f(\mathbf{x}_i, \mathbf{w}), \sigma^2\right)$ δηλαδή Gaussian κατανομή με μέση τιμή το μοντέλο και διακύμανση σ^2.

ΜΗ ΓΡΑΜΜΙΚΑ ΣΥΣΤΗΜΑΤΑ & ΜΕΓΙΣΤΗ ΠΙΘΑΝΟΦΑΝΕΙΑ

3 Μέγιστη Πιθανοφάνεια (Maximum Likelihood)

Συνολική Πιθανοφάνεια

Η πιθανότητα όλων των παρατηρήσεων:

$$L(\mathbf{w}) = \prod_{i=1}^{n} P(y_i \,|\, \mathbf{x}_i, \mathbf{w})$$

Αρνητικός Λογάριθμος Πιθανοφάνειας

Αντί να μεγιστοποιούμε το $L(\mathbf{w})$, ελαχιστοποιούμε το:

$$E(\mathbf{w}) = -\log L(\mathbf{w})$$

Κλειστή Μορφή

Μετά από αλγεβρικούς χειρισμούς:

$$E(\mathbf{w}) = \frac{n}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y_i - f(\mathbf{x}_i, \mathbf{w})\right)^2$$

4 Βασικό Συμπέρασμα

Επειδή ο πρώτος όρος $\frac{n}{2}\log(2\pi\sigma^2)$ δεν εξαρτάται από τα $\mathbf{w}$ η ελαχιστοποίηση του $E(\mathbf{w})$ είναι ισοδύναμη με την ελαχιστοποίηση του:

$$\sum_{i=1}^{n}\left(y_i - f(\mathbf{x}_i, \mathbf{w})\right)^2$$

δηλαδή του γνωστού **Mean Squared Error (MSE)**.

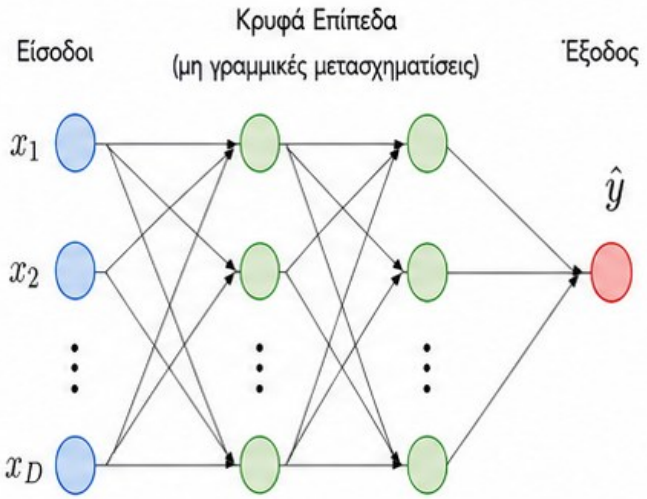
Συμπέρασμα: Για μη γραμμικό μοντέλο με Gaussian θόρυβο, η ελαχιστοποίηση του Negative Log-Likelihood ισοδυναμεί με την ελαχιστοποίηση του MSE.

5 Σύνδεση με Neural Networks

Τα Neural Networks:

- χρησιμοποιούν μη γραμμικές συναρτήσεις (activations) ως $\phi(\mathbf{x})$
- εκπαιδεύονται συνήθως με MSE (ή άλλα loss functions)
- βασίζονται στην ίδια φιλοσοφία optimization:

$$\min_{\mathbf{w}} E(\mathbf{w}) = \min_{\mathbf{w}} \sum_{i=1}^{n}\left(y_i - f(\mathbf{x}_i, \mathbf{w})\right)^2$$



Η έξοδος $\hat{y}$ είναι μια μη γραμμική συνάρτηση των εισόδων και των παραμέτρων $\mathbf{w}$.

Άρα: Οι αρχές των Ελαχίστων Τετραγώνων αποτελούν τη **θεωρητική βάση** της σύγχρονης Μηχανικής Μάθησης!

Ομοιότητες LSE και DNN

Πότε το DNN ταυτίζεται με LSE

- ✓ **1 layer** (χωρίς hidden layers) και **γραμμική** ενεργοποίηση ⇒ γραμμικό μοντέλο
- ✓ Με Loss = MSE:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

ίδιο αντικείμενο με το LS.

- ✓ **Κυρτότητα:** Για γραμμικά μοντέλα, το σφάλμα είναι κυρτό ⇒ μοναδικό παγκόσμιο ελάχιστο. Gradient descent συγκλίνει στο παγκόσμιο ελάχιστο (όπως στο LS).

Πού το DNN υπερέχει

- **Πολλαπλά** hidden layers και **μη γραμμικές ενεργοποιήσεις** ⇒ πολύ μεγαλύτερη εκφραστική ικανότητα (μοντελοποίηση πολύπλοκων σχέσεων).
- **Η συνάρτηση σφάλματος δεν είναι κυρτή (πολλά τοπικά ελάχιστα)** ⇒ το gradient descent μπορεί να καταλήξει σε τοπικό ελάχιστο.
- Με σύγχρονες παραλλαγές (SGD, Adam, καλή αρχικοποίηση, regularization) μπορούμε να πλοηγηθούμε αποτελεσματικά και να επιτύχουμε πολύ καλή απόδοση.

Συμπέρασμα: Το LS είναι το γραμμικό, κλειστής λύσης ειδικής περίπτωσης.
Τα DNN είναι η μη γραμμική γενίκευση με τεράστια εκφραστική δύναμη.

Θεμελιώδη Στοιχεία Νευρωνικών Δικτύων

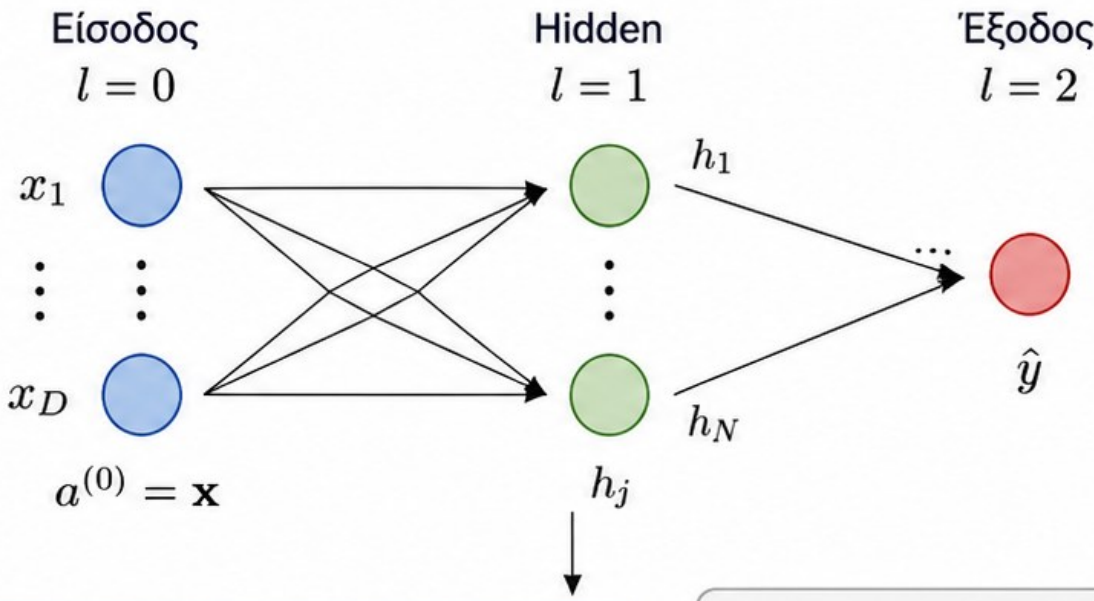
Νευρώνας

Για το στρώμα l:

$$a_j^{(l)} = \sigma\left(\sum_i w_{ji}^{(l)} \, a_i^{(l-1)} + b_j^{(l)} \right)$$

- $a_i^{(l-1)}$: ενεργοποιήσεις προηγούμενου στρώματος
- $w_{ji}^{(l)}$: βάρη από το στρώμα $l-1$ στο l
- $b_j^{(l)}$: bias
- $\sigma(\cdot)$: μη γραμμική συνάρτηση ενεργοποίησης (π.χ. sigmoid, ReLU)

Shallow Νευρωνικό Δίκτυο (1 hidden layer)



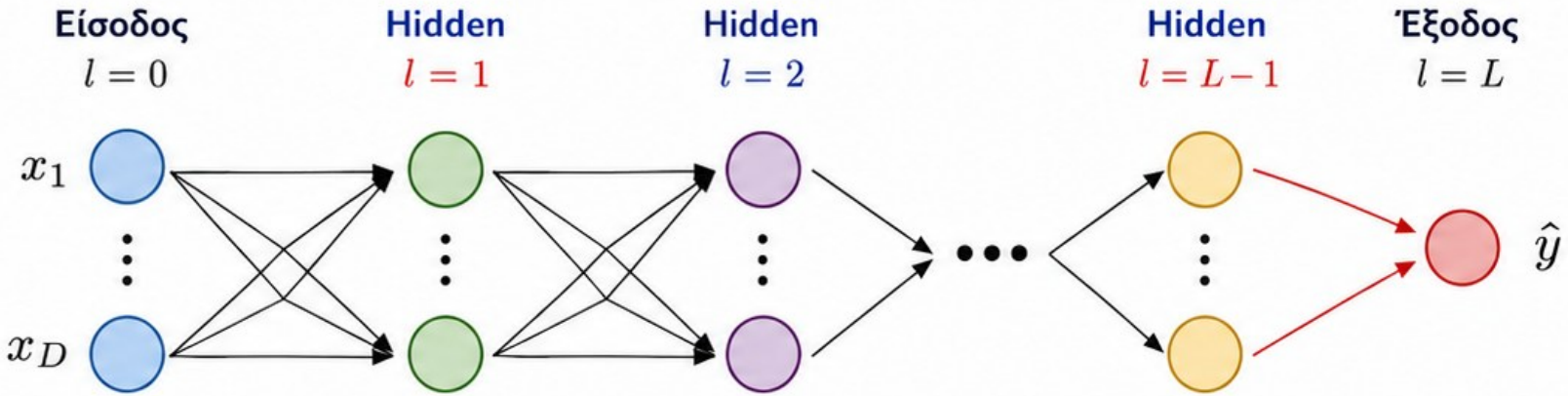
$$a_j^{(1)} = \sigma\left(\sum_i w_{ji}^{(1)} \, x_i + b_j^{(1)} \right)$$

$$\hat{y} = z^{(2)} = \sum_j w_{ji}^{(2)} \, a_j^{(1)} + b_j^{(2)}$$

(για παλινδρόμηση: γραμμική έξοδος)

Κάθε νευρώνας υπολογίζει τον σταθμισμένο άθροισμα των εισόδων του, προσθέτει bias και εφαρμόζει μια μη γραμμική συνάρτηση ενεργοποίησης.

Βαθιά Νευρωνικά Δίκτυα (Deep Neural Networks)



Έξοδος (layer L)

$$\hat{y} = z^{(L)} = \sum_j w_{ji}^{(L)}\, a_i^{(L-1)} + b_j^{(L)}$$

(γραμμική έξοδος για παλινδρόμηση)

Ενεργοποίηση νευρώνα στη στοιβάδα l $(1 \le l \le L-1)$

$$a_j^{(l)} = \sigma\left(\sum_i w_{ji}^{(l)}\, a_i^{(l-1)} + b_j^{(l)} \right), \quad 1 \le l \le L-1$$

- $a_i^{(l-1)}$: ενεργοποιήσεις της προηγούμενης στοιβάδας (στρώματος)
- $w_{ji}^{(l)}$: βάρος από τον νευρώνα i της στοιβάδας $l-1$ στον νευρώνα j της στοιβάδας l
- $b_j^{(l)}$: bias του νευρώνα j της στοιβάδας l
- $\sigma(\cdot)$: μη γραμμική συνάρτηση ενεργοποίησης (π.χ. ReLU, sigmoid, tanh)

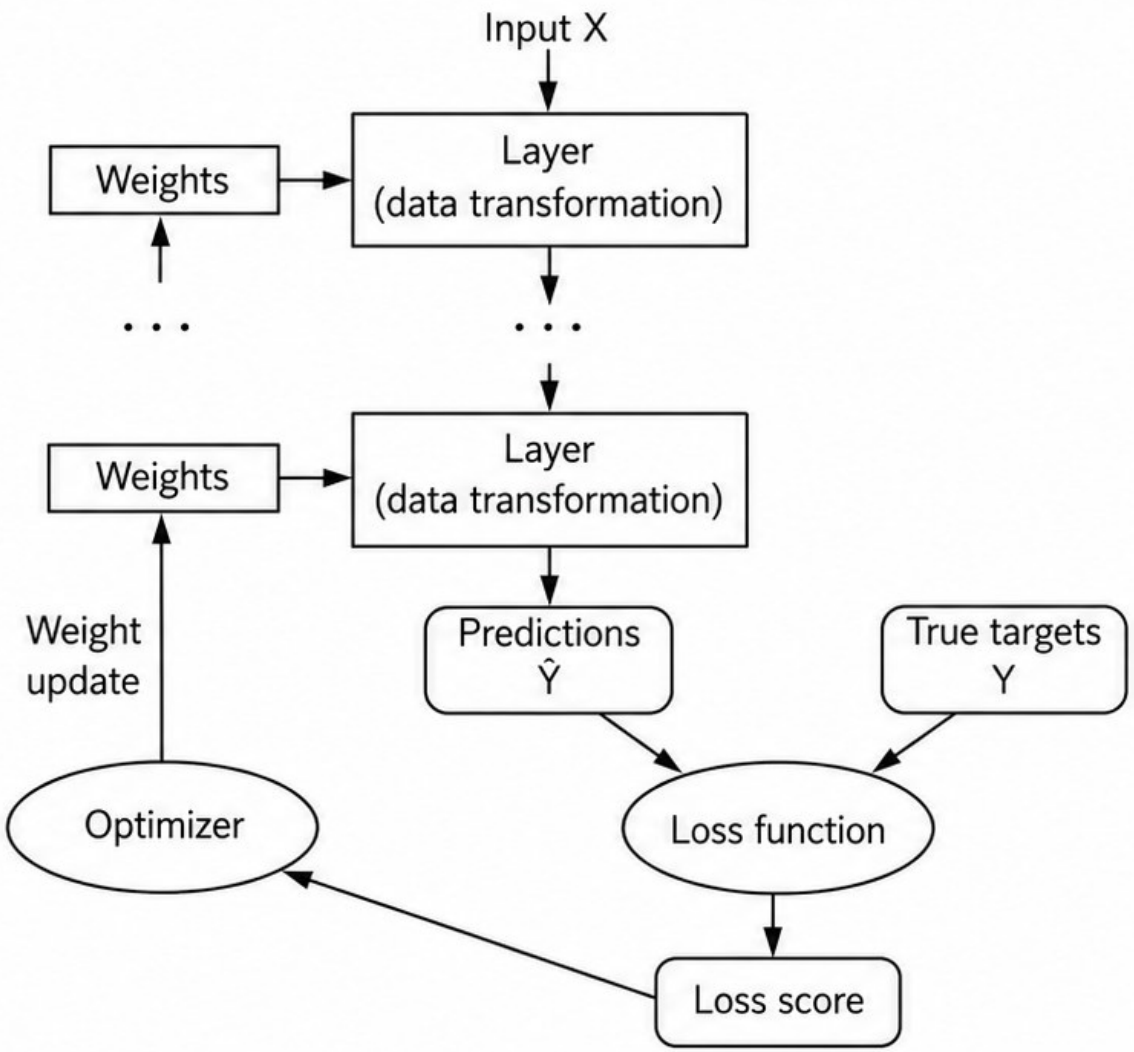
Στόχος εκπαίδευσης (τυπικά MSE)

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- y_i : πραγματική τιμή στόχος
- $\hat{y}_i$: πρόβλεψη του δικτύου
- n : αριθμός παρατηρήσεων

Τα βαθιά νευρωνικά δίκτυα αποτελούνται από πολλές κρυφές στοιβάδες (layers) ποι.ίον επιτρέπουν την εκμάθηση πολύπλοκων, μη γραμμικών σχέσεων στα δεδομένα.

Εκπαιδεύοντας ένα Βαθύ Νευρωνικό Δίκτυο



Βασικά Βήματα

1. **Forward pass:** το input X περνά από τα layers και παράγεται πρόβλεψη Ŷ
2. **Υπολογισμός σφάλματος:** η loss function συγκρίνει Ŷ με τον πραγματικό στόχο Y
3. **Backpropagation:** υπολογίζονται οι κλίσεις των βαρών
4. **Optimizer update:** τα weights ενημερώνονται ώστε να μειώνεται το loss

Κύριες Έννοιες

- **Weights:** οι παράμετροι που μαθαίνει το δίκτυο
- **Layers:** διαδοχικοί μετασχηματισμοί δεδομένων
- **Loss function:** μετρά πόσο καλή είναι η πρόβλεψη
- **Optimizer:** αλγόριθμος ενημέρωσης των βαρών

Στόχος της εκπαίδευσης: ελαχιστοποίηση του loss ώστε οι προβλέψεις να πλησιάζουν όσο γίνεται τους πραγματικούς στόχους.

Εκπαίδευση

1 Forward Pass

Η είσοδος $\mathbf{x}$ περνάει από το δίκτυο μέχρι την έξοδο και υπολογίζεται το σφάλμα $E(\mathbf{w})$.

2 Backward Pass

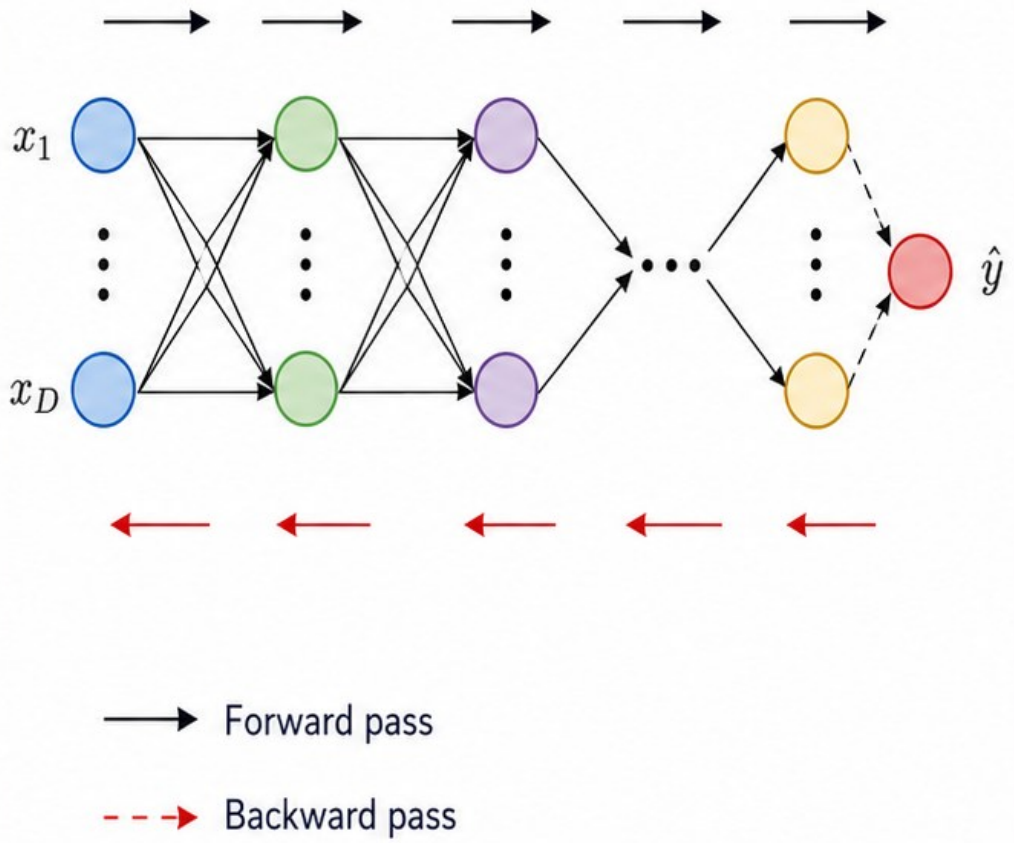
Υπολογίζουμε τα gradients $\nabla_{\mathbf{w}} E(\mathbf{w})$ με τον κανόνα αλυσίδας, από την έξοδο προς την είσοδο.

3 Ενημέρωση Παραμέτρων

Ενημερώνουμε τα βάρη και biases:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} E(\mathbf{w})$$

(η: learning rate)



Gradient (chain rule)

$$\frac{\partial E}{\partial w_{ji}^{(l)}} = \frac{\partial E}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{ji}^{(l)}}$$

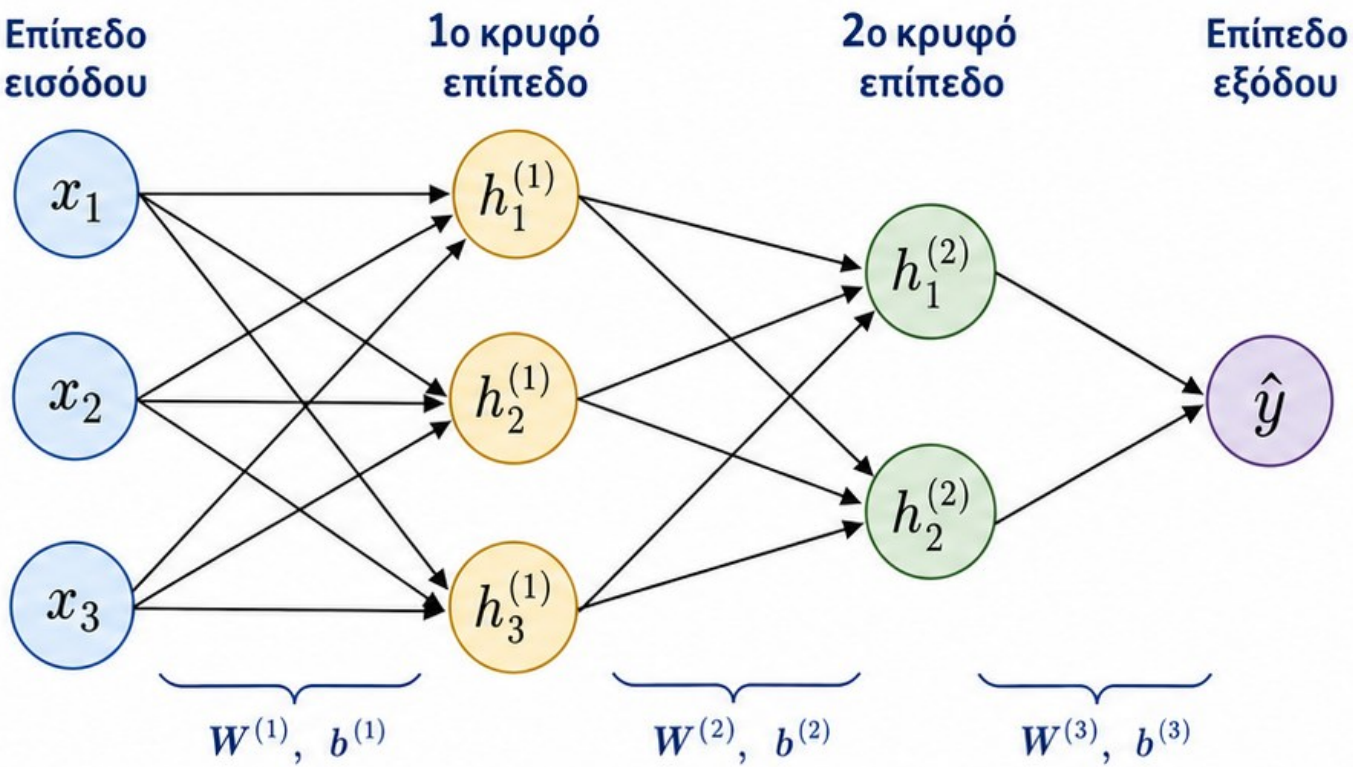
όπου $z_j^{(l)} = \sum_i w_{ji}^{(l)} a_i^{(l-1)} + b_j^{(l)}$

Ο αλγόριθμος backpropagation επιτρέπει αποδοτικό υπολογισμό gradients ακόμα και σε δίκτυα με εκατομμύρια παραμέτρους.

Bishop, 2008. Pattern Recognition and Machine Learning https://www.cs.uoi.gr/~arly/courses/ml/tmp/Bishop_book.pdf

Καλό βιβλίο με τη θεωρητική βάση γυρω απο το ML

Εκπαίδευση Νευρωνικού Δικτύου 2 Επιπέδων



Κάθε επίπεδο εφαρμόζει γραμμικό μετασχηματισμό και μη γραμμική ενεργοποίηση.

1 Forward pass

$$
\begin{aligned}
z^{(1)} &= W^{(1)} x + b^{(1)} \\
a^{(1)} &= \phi\left(z^{(1)}\right) \\
z^{(2)} &= W^{(2)} a^{(1)} + b^{(2)} \\
a^{(2)} &= \phi\left(z^{(2)}\right) \\
\hat{y} &= W^{(3)} a^{(2)} + b^{(3)}
\end{aligned}
$$

2 Συνάρτηση κόστους

$$L(\mathbf{w}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Για παλινδρόμηση χρησιμοποιούμε συχνά MSE.

3 Backpropagation & ενημέρωση βαρών

$$
\begin{aligned}
\mathbf{w} &\leftarrow \mathbf{w} - \eta \, \nabla_{\mathbf{w}} L(\mathbf{w}) \\
b &\leftarrow b - \eta \, \nabla_{b} L(\mathbf{w})
\end{aligned}
$$

Ο optimizer ενημερώνει βάρη και bias με ρυθμό μάθησης η.

Στόχος: να βρούμε παραμέτρους που ελαχιστοποιούν το σφάλμα ώστε οι προβλέψεις $\hat{y}$ να προσεγγίζουν τις πραγματικές τιμές y.

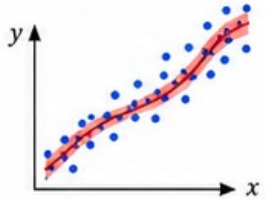
Τύποι Μάθησης στη Μηχανική Μάθηση

(Types of Learning in Machine Learning)

1 Επιβλεπόμενη Μάθηση (Supervised Learning)

Ερώτηση:
«Μπορούμε να προβλέψουμε το y από το x;»

Δεδομένα:
- είσοδοι x
- γνωστές ετικέτες y



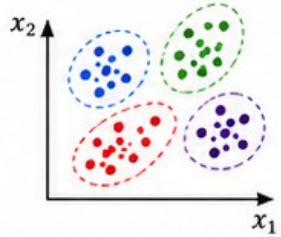
Παραδείγματα:
- πρόβλεψη θερμοκρασίας
- ταξινόμηση γαλαξιών
- ανίχνευση σεισμικών γεγονότων

Τυπικές εργασίες:
- Regression (παλινδρόμηση)
- Classification (ταξινόμηση)

2 Μη Επιβλεπόμενη Μάθηση (Unsupervised Learning)

Ερώτηση:
«Υπάρχει κρυφή δομή στα δεδομένα;»

Δεδομένα:
- μόνο είσοδοι x
- χωρίς labels



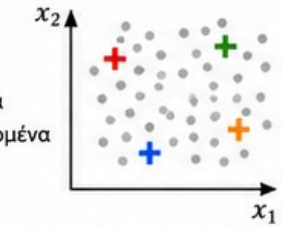
Παραδείγματα:
- Clustering δορυφορικών εικόνων
- grouping σεισμικών σημάτων
- ανάλυση κοινωνικών δικτύων

Τυπικές εργασίες:
- Clustering (ομαδοποίηση)
- Dimensionality Reduction (μείωση διαστατικότητας)
- Anomaly Discovery (ανίχνευση ανωμαλιών)

3 Ημι-Επιβλεπόμενη Μάθηση (Semi-Supervised Learning)

Ερώτηση:
«Μπορούμε να μάθουμε με λίγα labels;»

Δεδομένα:
- λίγα labeled δεδομένα
- πολλά unlabeled δεδομένα



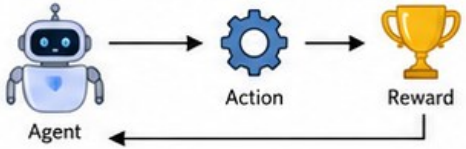
Παραδείγματα:
- ιατρικές εικόνες
- remote sensing
- geospatial mapping

Πλεονέκτημα:
Μειώνει το κόστος annotation και αξιοποιεί μεγάλο όγκο μη επισημασμένων δεδομένων.

4 Ενισχυτική Μάθηση (Reinforcement Learning)

Ερώτηση:
«Ποια ενέργεια μεγιστοποιεί την ανταμοιβή;»

Βασική ιδέα:
Agent → Action → Reward



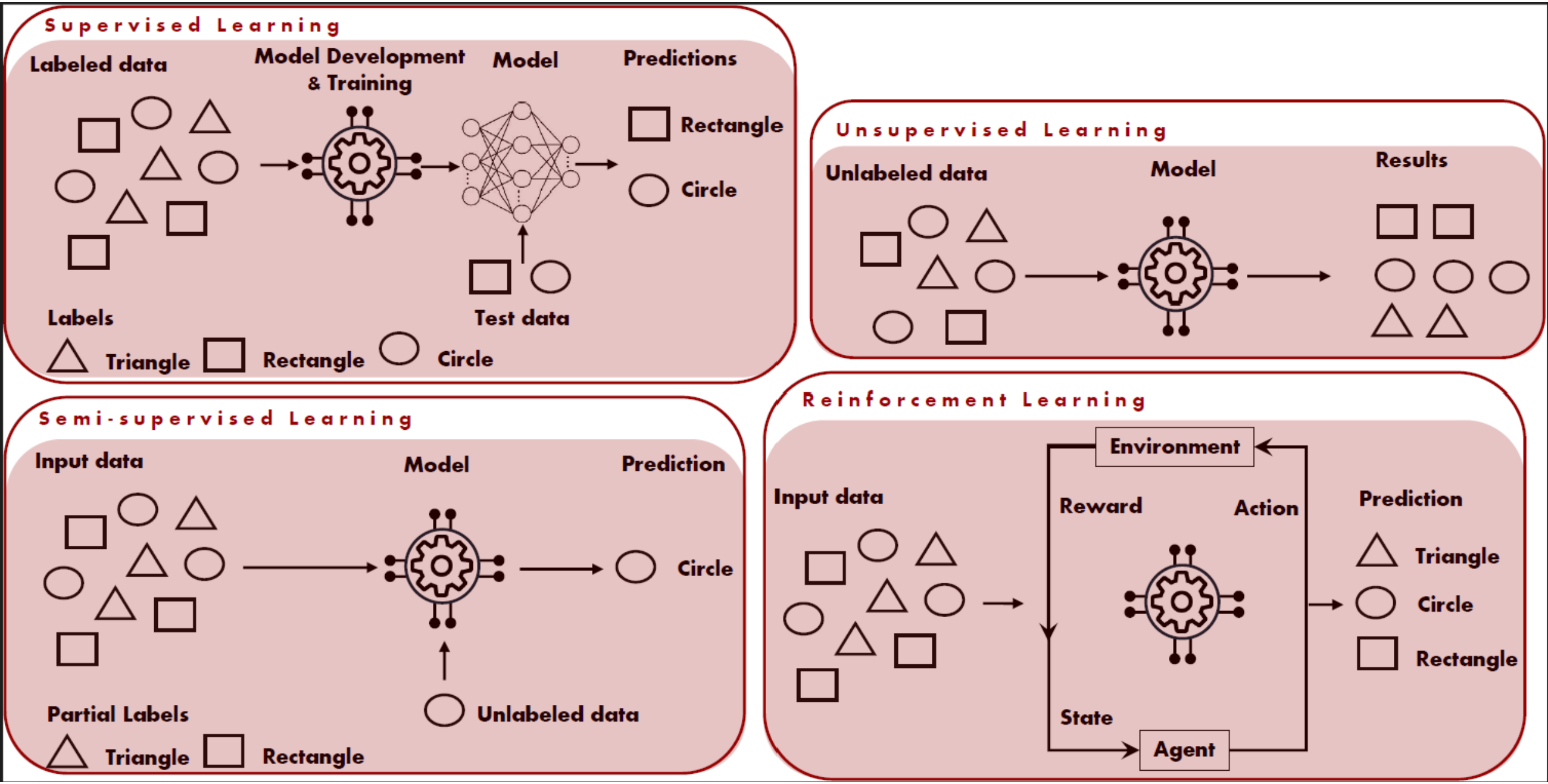
Παραδείγματα:
- autonomous vehicles
- robotics
- adaptive observation systems

Έννοιες-κλειδιά:
- policy (στρατηγική)
- reward (ανταμοιβή)
- environment (περιβάλλον)

Κεντρική ιδέα: Το ίδιο μοντέλο μπορεί να αλλάξει "ρόλο" ανάλογα με:

- τα διαθέσιμα δεδομένα (labels / χωρίς labels)
- τον ορισμό της εξόδου (τι προβλέπουμε / ανακαλύπτουμε)
- το learning objective / loss function (ποιο είναι το κριτήριο βελτίωσης)

Τύποι Μάθησης στη Μηχανική Μάθηση



Σχεδιάζοντας ένα Σύστημα Μηχανικής Μάθησης

(Designing a Machine Learning System)

1 Ορίζουμε το πρόβλημα (δεδομένα)

- Συλλέγουμε δεδομένα από τον πραγματικό κόσμο.

2 Ορίζουμε τι θέλουμε να μάθουμε

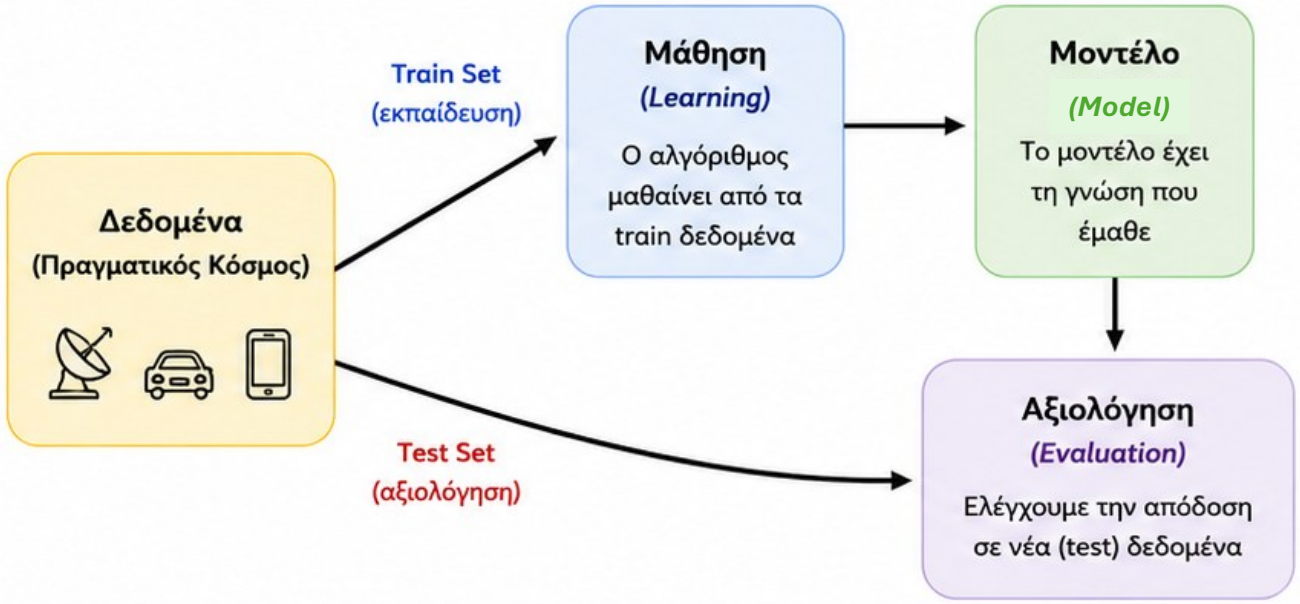
- Δηλαδή τη **συνάρτηση στόχο (target function)** που θέλουμε να προβλέπει το μοντέλο.

3 Επιλέγουμε το μοντέλο (αναπαράσταση)

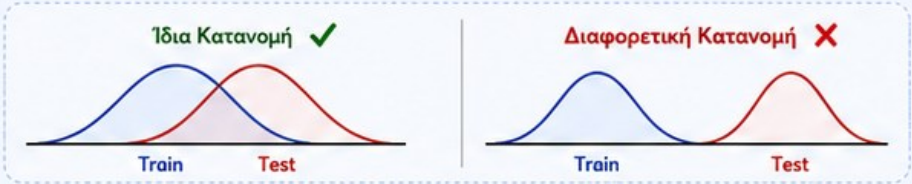
- Πώς θα αναπαραστήσουμε τη συνάρτηση που θέλουμε να μάθουμε.

4 Επιλέγουμε τον αλγόριθμο μάθησης

- Ο αλγόριθμος μαθαίνει τις παραμέτρους του μοντέλου από τα δεδομένα εκπαίδευσης (train set).



Κρίσιμο: Τα δεδομένα εκπαίδευσης (train set) και δοκιμής (test set) πρέπει να προέρχονται από την **ίδια κατανομή (same distribution)** με τα δεδομένα που θα συναντήσει το μοντέλο στην πράξη!



Αν οι κατανομές ειναι διαφορετικές υπάρχουν τεχνικές λύσεις όπως το transfer learning

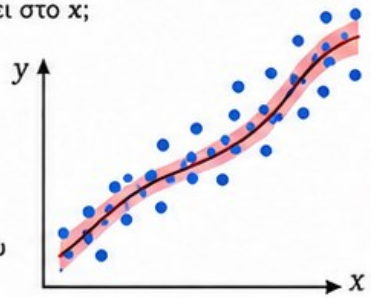
Τι ερωτήσεις μπορεί να απαντήσει ένα μοντέλο;

1 Πρόβλεψη (Regression)

Ερώτηση: Ποια τιμή y ταιριάζει στο x;

Διατύπωση:

$$\hat{y} = f(x, w)$$



Παραδείγματα:

- πρόβλεψη φωτεινότητας άστρου
- πρόβλεψη θερμοκρασίας
- πρόβλεψη τιμής χρονοσειράς

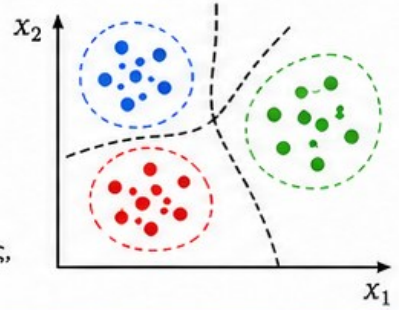
x: χαρακτηριστικά (π.χ. χρώμα, λαμπρότητα, χρόνος)
y: συνεχής τιμή (π.χ. φωτεινότητα, θερμοκρασία)

2 Ταξινόμηση (Classification)

Ερώτηση: Σε ποια κατηγορία ανήκει το x;

Διατύπωση:

$$y \in \{1, \ldots, K\}$$



Παραδείγματα:

- γαλαξίας / άστρο / κβάζαρ
- τύπος σήματος (π.χ. παλωός, συνεχής, θόρυβος)

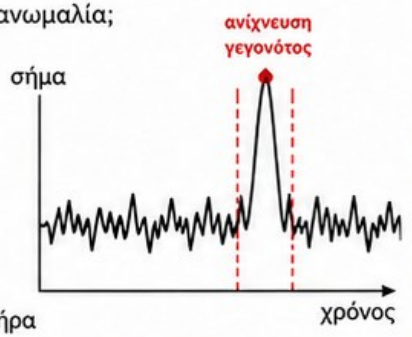
x: χαρακτηριστικά (π.χ. μορφολογία, φάσμα, σήμα)
y: διακριτή κατηγορία $(1, \ldots, K)$

3 Ανίχνευση (Detection)

Ερώτηση: Υπάρχει γεγονός ή ανωμαλία;

Διατύπωση:

$$P(event \mid x)$$



Παραδείγματα:

- ανίχνευση βαρυτικού κύματος
- ανίχνευση έκλαμψης (flare)
- ανίχνευση ανωμαλίας σε αισθητήρα

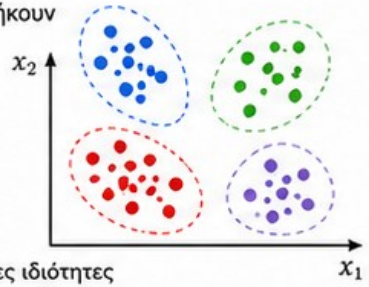
x: χρονική σειρά / δεδομένα αισθητήρα / παρατήρηση
y: πιθανότητα γεγονότος (0 έως 1)

4 Συσταδοποίηση (Clustering)

Ερώτηση: Ποιες παρατηρήσεις ανήκουν στην ίδια ομάδα;

Διατύπωση:

$$y \in \{1, \ldots, K\}$$
(άγνωστα, χωρίς ετικέτες)



Παραδείγματα:

- ομαδοποίηση γαλαξιών με παρόμοιες ιδιότητες
- ομαδοποίηση αστέρων σε αστρικούς πληθυσμούς
- ομαδοποίηση σημάτων χωρίς γνωστές κατηγορίες

x: χαρακτηριστικά (π.χ. χρώμα, φωτεινότητα, μορφολογία)
y: ομάδα / συστάδα $(1, \ldots, K)$ – δεν είναι γνωστές εκ των προτέρων

5 Κατάταξη (Ranking)

Ερώτηση: Ποια επιλογή είναι πιο συμβατή με το x;

Διατύπωση:

$$score(x, y_1) > score(x, y_2)$$

Παραδείγματα:

- κατάταξη υποψήφιων πηγών (π.χ. εξωπλανήτες)
- επιλογή καλύτερου μοντέλου

x: δεδομένα / ερώτημα
y: υποψήφιες επιλογές (λίστα)

6 Λήψη Απόφασης

Ερώτηση: Ποια ενέργεια πρέπει να γίνει;

Διατύπωση:

$$a^* = \arg\max_a U(a, x)$$

Παραδείγματα:

- επιλογή επόμενης παρατήρησης
- ρύθμιση τηλεσκοπίου
- προτεραιοποίηση δεδομένων

x: τρέχουσα κατάσταση / δεδομένα
a: διαθέσιμες ενέργειες, **y**: αποτέλεσμα / ωφέλεια

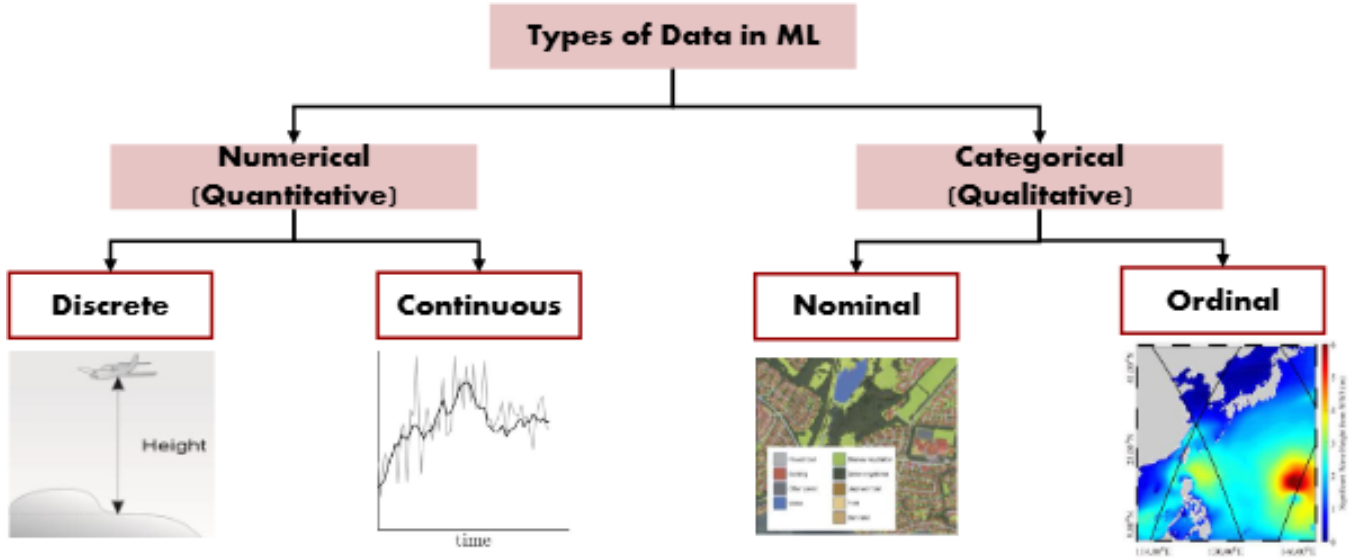
Κεντρική ιδέα: Το ίδιο δίκτυο μπορεί να αλλάξει ‘ρόλο’ ανάλογα με το πώς ορίζουμε την έξοδο y και το loss function.

Τα δεδομένα (Data)

The Machine Learning Pipeline

- **Data Collection**
  - ✓ Συλλογή δεδομένων από πραγματικές πηγές (Sensors, satellites, logs, databases, web scraping, κλπ.)
  - ✓ Η ποιότητα των δεδομένων επηρεάζει άμεσα το τελικό μοντέλο

- **Preprocessing / Cleaning**
  - ✓ Διαχείριση missing values (NaN)
  - ✓ Αφαίρεση outliers και θορύβου
  - ✓ Διόρθωση ασυνεπειών και normalization
  - ✓ Μετατροπή δεδομένων σε κατάλληλη μορφή



- **Feature Engineering**
  - ✓ Επιλογή σημαντικών χαρακτηριστικών (features)
  - ✓ Δημιουργία νέων μεταβλητών από τα δεδομένα
  - ✓ Μείωση διάστασης και redundancy
  - ✓ Βελτίωση της απόδοσης και γενίκευσης του μοντέλου

- **Data Splitting**

Διαχωρισμός δεδομένων για αξιόπιστη αξιολόγηση:

- ✓ **Training Set** χρησιμοποιείται για την εκπαίδευση του μοντέλου
- ✓ **Validation Set** χρησιμοποιείται για tuning υπερπαραμέτρων και την επιλογή καλύτερου μοντέλου
- ✓ **Test Set** για τελική αξιολόγηση σε unseen δεδομένα και εκτίμηση πραγματικής γενίκευσης

Tutorials / Educational Resources

•Google ML Crash Course

για train/test split, feature engineering, overfitting.

•Scikit-Learn Tutorials

Ίσως το καλύτερο practical ML resource.

•Machine Learning Mastery

Εξαιρετικό για preprocessing και pipelines.

•Towards Data Science

Πολύ χρήσιμο για visual explanations.

•Kaggle Learn

Mini courses με notebooks.

•DataCamp ML Tutorials

Πολύ καθαρά explainers.

Lecture Slides / Πανεπιστημιακά Μαθήματα

•Stanford CS229 (Andrew Ng)

Κλασικό reference.

•CMU Machine Learning Course

•Berkeley CS189

Πολύ καλά slides για generalization και evaluation.

•MIT Introduction to Deep Learning

•Fast.ai Course

Πολύ μοντέρνα ML/DL προσέγγιση.

•Feature Engineering Lecture Slides

Πολύ καλό ειδικά για preprocessing και feature engineering.

•Scikit-Learn Intro Slides

Περιέχει πολύ καθαρά diagrams για train/test transforms.

Datasets / Practice

•Kaggle Datasets

•UCI Machine Learning Repository

Κλασικά ML datasets.

•Google Dataset Search

•OpenML

Πολύ χρήσιμο για benchmarking.

•EarthData NASA

Για EO / remote sensing examples.

Papers / Advanced Reading

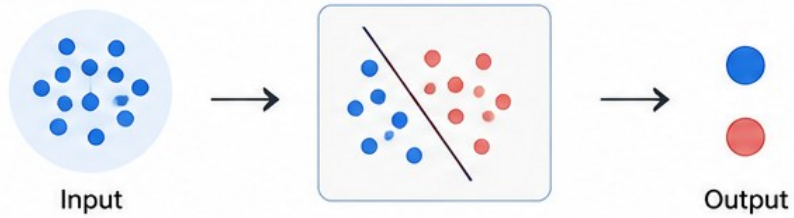
•A Critical Look at the Current Train/Test Split in Machine Learning

Πολύ ωραίο paper για limitations του κλασικού split.

•Comparative Analysis of Data Preprocessing Methods

Πολύ σχετικό με πραγματικά datasets και preprocessing effects.

Shallow Learning



Χαρακτηριστικά
Χειροκίνητη μηχανική χαρακτηριστικών

Μοντέλα
Απλά μοντέλα με λίγα επίπεδα
π.χ. Γραμμική Παλινδρόμηση, SVM, Decision Trees

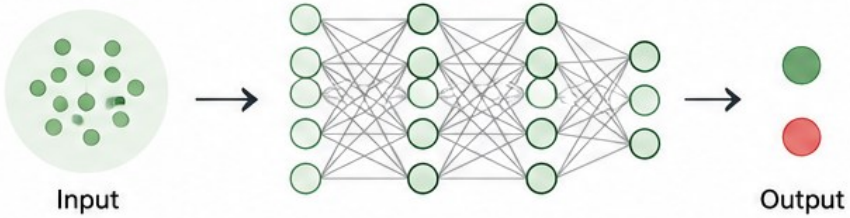
Δεδομένα
Απαιτεί μικρότερο όγκο δεδομένων

Ερμηνευσιμότητα
Πιο εύκολα ερμηνεύσιμα

Κατάλληλο για
Μικρά/μεσαία δεδομένα, δομημένα δεδομένα,
όταν η ερμηνευσιμότητα είναι κρίσιμη

VS

Deep Learning



Χαρακτηριστικά
Αυτόματη εκμάθηση χαρακτηριστικών από τα δεδομένα

Μοντέλα
Πολύπλοκα μοντέλα με πολλά επίπεδα
π.χ. CNN, RNN, Transformers

Δεδομένα
Απαιτεί μεγάλο όγκο δεδομένων

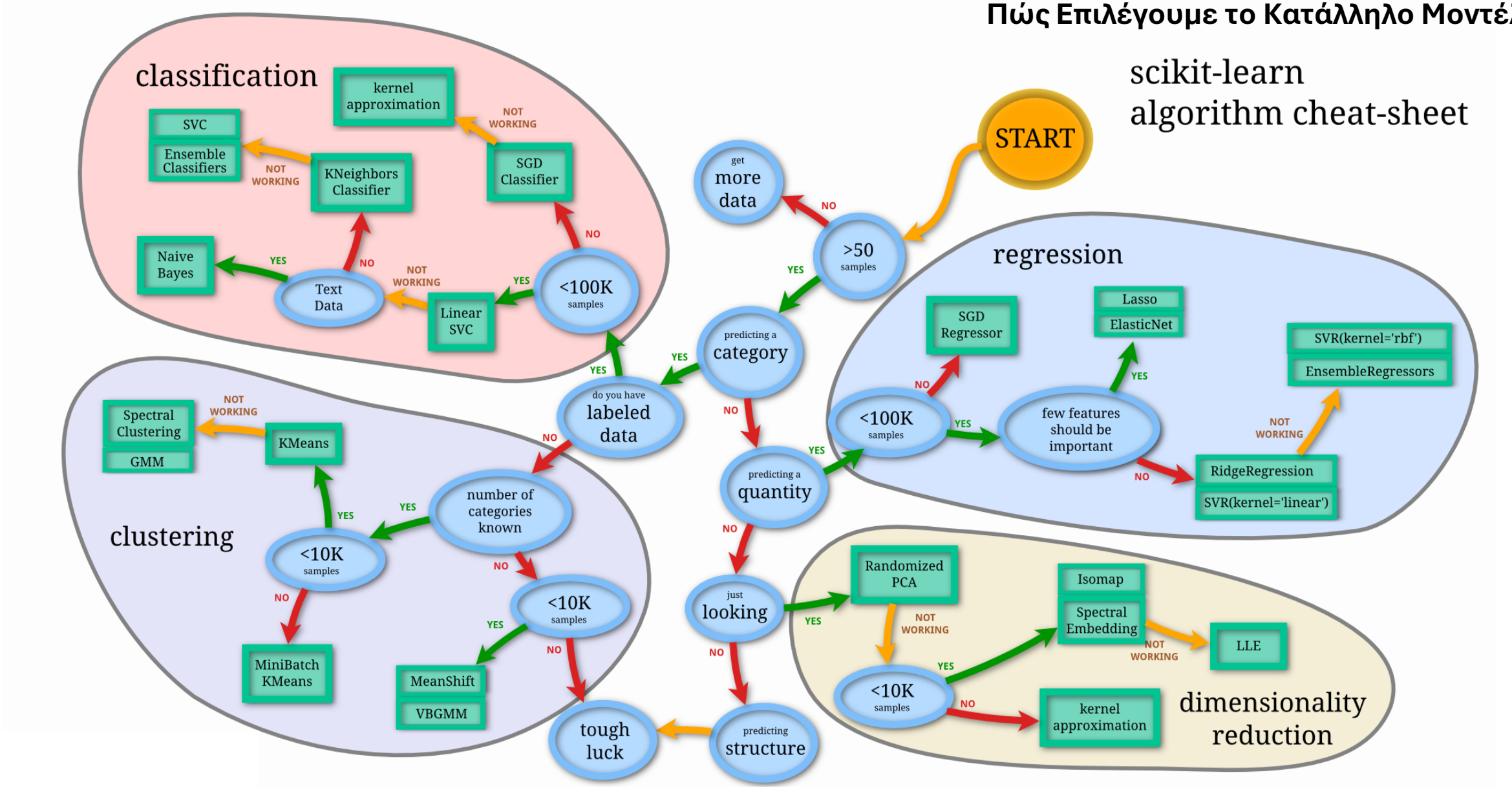
Ερμηνευσιμότητα
Δύσκολη ερμηνεία (black box)

Κατάλληλο για
Μεγάλα δεδομένα, μη δομημένα δεδομένα
(εικόνες, ήχος, κείμενο)

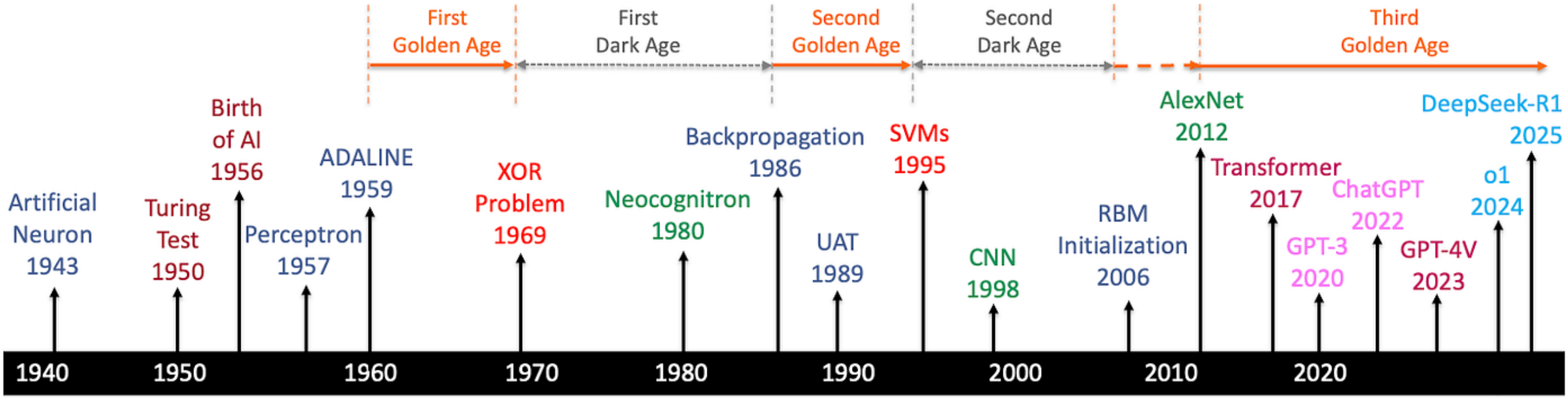
Συμπέρασμα

Δεν υπάρχει μία λύση για όλα.
Η επιλογή εξαρτάται από τα δεδομένα, το πρόβλημα και τις απαιτήσεις εφαρμογής.

Πώς Επιλέγουμε το Κατάλληλο Μοντέλο;



A Brief History of AI with Deep Learning



McCulloch-Pitts

Rosenblatt

Widrow-Hoff

Minsky-Papert

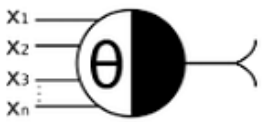
Rumelhart, Hinton et al.

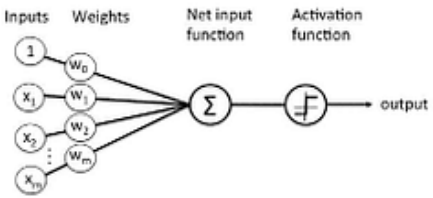
LeCun

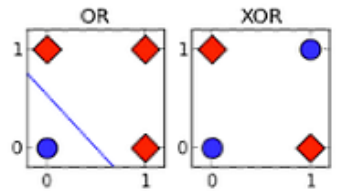
Hinton-Ruslan

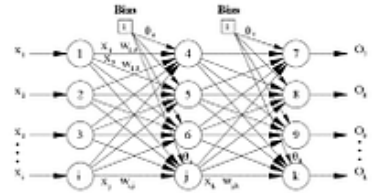
Krizhevsky et al.

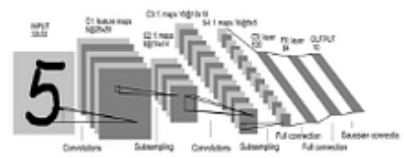
Vaswani

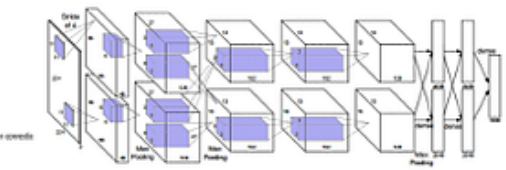












ΣΥΝΕΛΙΚΤΙΚΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ (CNNs)

Τα CNNs είναι ειδικά σχεδιασμένα για δεδομένα σε μορφή πλέγματος, όπως οι εικόνες.

Είσοδος
(Εικόνα)

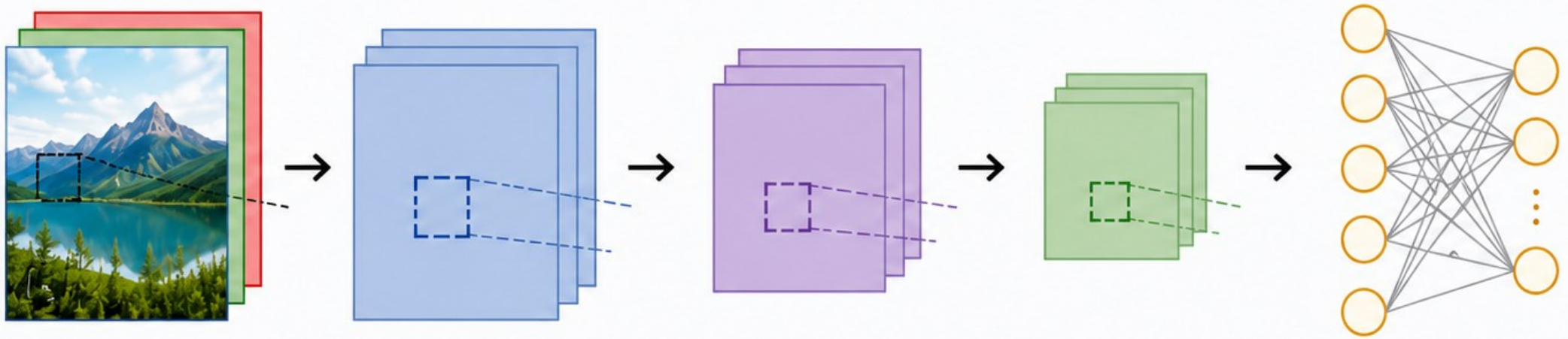
Συνέλιξη
(Εξαγωγή Χαρακτηριστικών)

ReLU
(Μη Γραμμικότητα)

Pooling
(Μείωση Διαστάσεων)

Πλήρως Συνδεδεμένο
(Ταξινόμηση)

Softmax
(Έξοδος)



Γάτα	0.05
Σκύλος	0.90
Αυτοκίνητο	0.03
Πουλί	0.02
...	

Τοπική εξαγωγή χαρακτηριστικών

Κοινά βάρη (λιγότερες παράμετροι)

Αναλλοίωτο σε μετατοπίσεις

Εξαιρετική απόδοση σε εικόνες & βίντεο

Η ΠΡΑΞΗ ΤΗΣ ΣΥΝΕΛΙΞΗΣ (CONVOLUTION)

Ο πυρήνας (kernel) «γλιστρά» πάνω στην εικόνα. Κάθε φορά, υπολογίζεται το σταθμισμένο άθροισμα των στοιχείων και παράγεται μία νέα τιμή στο χάρτη χαρακτηριστικών (feature map).

Εικόνα εισόδου
(6 x 6)

1	0	1	2	1	0
0	1	1	0	1	1
1	1	2	1	0	0
0	0	1	2	1	1
1	2	1	0	1	0
0	1	0	1	1	1

*

Πυρήνας (Kernel)
(2 x 2)

1	0
0	-1

=

Χάρτης χαρακτηριστικών
(5 x 5)

1	-1	0	3	0
0	2	-1	-1	0
2	0	0	0	1
-1	-1	1	1	1
1	-1	0	0	0

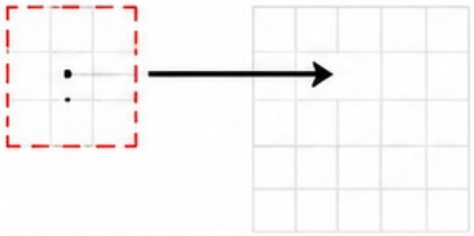
Υπολογισμός πρώτης θέσης

			⊙			=		
1	0			1	0		1	0
0	1			0	-1		0	-1

Άθροισμα: 1 + 0 + 0 + (-1) = **0**

⊙ Στοιχειο-προς-στοιχείο πολλαπλασιασμός

Συνέλιξη = Σταθμισμένο άθροισμα



Ο πυρήνας μετακινείται
με βήμα (**stride**) = 1
πάνω στην εικόνα

Βασικά σημεία

- Πολλαπλασιασμός στοιχείο-προς-στοιχείο και άθροιση.
- Ανιχνεύει τοπικά μοτίβα (π.χ. ακμές, γωνίες, υφές).
- Διαφορετικοί πυρήνες → διαφορετικά χαρακτηριστικά.
- Το μέγεθος του χάρτη χαρακτηριστικών εξαρτάται από: το μέγεθος της εικόνας, τον πυρήνα, το stride και το padding.

Σημαντικό: Το padding (προσθήκη μηδενικών γύρω από την εικόνα) ελέγχει το μέγεθος του χάρτη εξόδου και βοηθά στη διατήρηση των πληροφοριών στα σύνορα.

CNNs ΣΤΗΝ ΠΡΑΞΗ

Παράδειγμα σε Python (TensorFlow / Keras)

```python
import tensorflow as tf
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu',
                  input_shape=(128,128,3)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# Compile και εκπαίδευση
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Που χρησιμοποιούνται τα CNNs;

Ταξινόμηση Εικόνων
(π.χ. γάτα ή σκύλος)

Ανίχνευση Αντικειμένων
(π.χ. εντοπισμός ανθρώπων)

Τμηματοποίηση Εικόνας
(π.χ. ιατρικές εικόνες)

Αυτόνομη Οδήγηση
(αντίληψη περιβάλλοντος)

Ιατρική Απεικόνιση
(π.χ. ανίχνευση όγκων)

... και πολλά άλλα!

Τα CNNs μαθαίνουν πλούσια χαρακτηριστικά απευθείας από τα δεδομένα,
με ελάχιστη χειροκίνητη σχεδίαση.

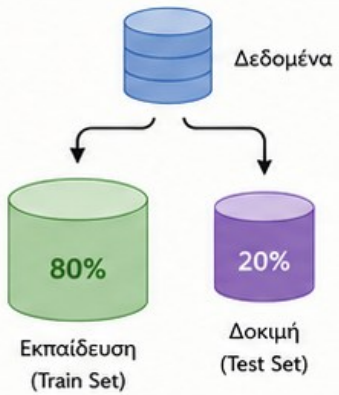
Αξιολόγηση Μοντέλων (Model Evaluation)

Η αξιολόγηση μοντέλων είναι η διαδικασία με την οποία ελέγχουμε πόσο καλά αποδίδει ένα μοντέλο μηχανικής μάθησης σε **νέα, αθέατα δεδομένα**. Στόχος είναι να διασφαλίσουμε ότι το μοντέλο δεν απομνημονεύει τα δεδομένα εκπαίδευσης, αλλά **γενικεύει σωστά** σε νέες περιπτώσεις.

1. Επικύρωση με Διασταυρούμενη Αξιολόγηση (Cross-Validation)

Μέθοδοι που χρησιμοποιούμε για να αξιολογήσουμε την απόδοση του μοντέλου και να μειώσουμε τον κίνδυνο υπερπροσαρμογής.

α Holdout Method (Διαίρεση σε Train/Test)



- Τα δεδομένα χωρίζονται σε σύνολο εκπαίδευσης και σύνολο δοκιμής (συνήθως 7:3 ή 8:2).
- Το μοντέλο εκπαιδεύεται στο train set και αξιολογείται στο test set.

```
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

iris = load_iris()
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.20, random_state=42)
)

print("Training set size:", len(X_train))
print("Testing set size:", len(X_test))
```

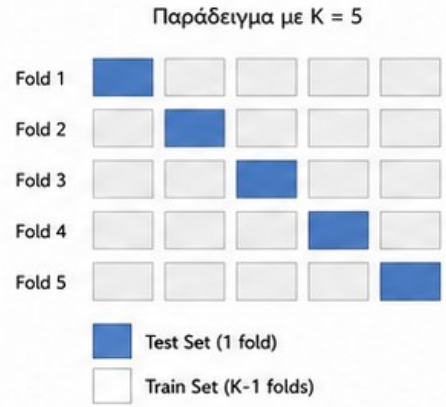
Έξοδος:

```
Training set size: 120
Testing set size: 30
```

Πότε το χρησιμοποιούμε;

Όταν έχουμε αρκετά δεδομένα και θέλουμε μια απλή και γρήγορη εκτίμηση απόδοσης.

β K-Fold Cross-Validation (Κ-πτυχια Επικύρωση)



- Τα δεδομένα χωρίζονται σε k μέρη (folds).
- Κάθε fold χρησιμοποιείται μία φορά ως test set και τα υπόλοιπα k-1 ως train set.
- Ο μέσος όρος των επιδόσεων δίνει πιο αξιόπιστη εκτίμηση.

```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()

kfold = KFold(n_splits=5, shuffle=True, random_state=42)

scores = cross_val_score(model, X, y, cv=kfold)

print("Cross-validation scores:", scores)
print("Average CV Score:", scores.mean())
```

Πότε το χρησιμοποιούμε;

Όταν τα δεδομένα είναι λίγα ή θέλουμε πιο σταθερή εκτίμηση της απόδοσης του μοντέλου.

Έξοδος:

```
Cross-validation scores: [1.        1.        0.93333333 0.93333333 0.93333333]
Average CV Score: 0.9600000000000002
```

Συνήθεις Μετρικές Αξιολόγησης

Ακρίβεια (Accuracy)
Ποσοστό σωστών προβλέψεων.

Ακρίβεια (Precision)
Πόσα από τα θετικά που προβλέψαμε είναι πραγματικά θετικά.

Ανάκληση (Recall)
Πόσα από τα πραγματικά θετικά καταφέραμε να βρούμε.

F1-Score
Συνδυάζει Precision και Recall σε μία ισορροπημένη μετρική.

Μετρικές Αξιολόγησης για Προβλήματα Ταξινόμησης

Οι μετρικές αξιολόγησης μας βοηθούν να ποσοτικοποιήσουμε την απόδοση ενός μοντέλου ταξινόμησης **σε δεδομένα που δεν έχει ξαναδεί** (test set).

Ροή Αξιολόγησης

1 Φόρτωση & Διαχωρισμός

Φορτώνουμε το dataset και το διαχωρίζουμε σε train (80%) και test (20%).

→

2 Εκπαίδευση Μοντέλου

Εκπαιδεύουμε το μοντέλο (π.χ. Decision Tree Classifier) στο training set.

→

3 Προβλέψεις

Το μοντέλο προβλέπει τις κλάσεις για το test set.

→

4 Υπολογισμός Μετρικών

Υπολογίζουμε μετρικές όπως accuracy, precision, recall, F1 score, confusion matrix, AUC-ROC.

Ακρίβεια (Accuracy)

Ποσοστό σωστών προβλέψεων στο σύνολο.

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

- **TP:** True Positives (Σωστά Θετικά)
- **TN:** True Negatives (Σωστά Αρνητικά)
- **FP:** False Positives (Ψευδώς Θετικά)
- **FN:** False Negatives (Ψευδώς Αρνητικά)

Ακρίβεια Θετικών (Precision)

Από τα δείγματα που προβλέφθηκαν θετικά, πόσα ήταν όντως θετικά.

$$Precision = \frac{TP}{TP + FP}$$

Εστιάζει στην ορθότητα των θετικών προβλέψεων.

Ανάκληση (Recall)

Από τα πραγματικά θετικά δείγματα, πόσα αναγνωρίστηκαν σωστά.

$$Recall = \frac{TP}{TP + FN}$$

Εστιάζει στην κάλυψη όλων των θετικών.

F1 Score

Αρμονικός μέσος όρος της precision και της recall.

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Εξισορροπεί precision και recall σε μία τιμή.

Πίνακας Σύγχυσης (Confusion Matrix)

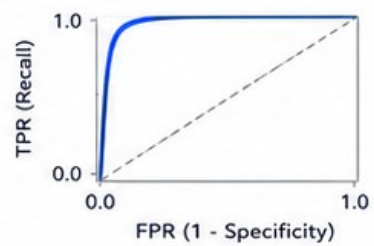
Πίνακας που δείχνει τις σωστές και λανθασμένες προβλέψεις ανά κλάση.

Actual \ Predicted	Positive	Negative
Positive	TP	FN
Negative	FP	TN

- Οι διαγώνιες τιμές είναι οι σωστές προβλέψεις.
- Οι εκτός διαγωνίου τιμές είναι οι λάθος προβλέψεις.

Καμπύλη ROC & AUC

Η καμπύλη ROC δείχνει την ικανότητα διάκρισης του μοντέλου μεταξύ κλάσεων.



- $TPR = \frac{TP}{TP + FN}$ (True Positive Rate)
- $FPR = \frac{FP}{FP + TN}$ (False Positive Rate)
- AUC: Εμβαδό κάτω από την καμπύλη ROC.
- Τιμές: 0.5 = τυχαίο μοντέλο, 1 = τέλειο μοντέλο.
- Όσο μεγαλύτερο το AUC, τόσο καλύτερο το μοντέλο.

Συμπέρασμα: Καμία μετρική από μόνη της δεν αρκεί. Η σωστή επιλογή μετρικής εξαρτάται από το πρόβλημα και το κόστος των λαθών (FP vs FN).

Μετρικές Αξιολόγησης για Προβλήματα Παλινδρόμησης (Regression)

Τα μοντέλα παλινδρόμησης προβλέπουν συνεχείς τιμές (π.χ. θερμοκρασία). Χρησιμοποιούμε μετρικές σφάλματος για να μετρήσουμε την ακρίβεια των προβλέψεων.

1 Mean Absolute Error (MAE)

Μέσο απόλυτο σφάλμα μεταξύ πραγματικών και προβλεπόμενων τιμών.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

y_i : Πραγματική τιμή
$\hat{y}_i$: Προβλεπόμενη τιμή
n : Αριθμός δειγμάτων

Μικρότερη τιμή → καλύτερες προβλέψεις.

2 Mean Squared Error (MSE)

Μέσο τετραγωνικό σφάλμα μεταξύ πραγματικών και προβλεπόμενων τιμών.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

- Τετραγωνίζει τα σφάλματα → δίνει μεγαλύτερη βαρύτητα σε μεγάλα σφάλματα.
- Χρησιμοποιείται συχνά ως συνάρτηση απώλειας.

3 Root Mean Squared Error (RMSE)

Τετραγωνική ρίζα του MSE.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

- Επαναφέρει το σφάλμα στην αρχική κλίμακα της μεταβλητής στόχου.
- Πιο εύκολη ερμηνεία από το MSE.
- Μικρότερη τιμή → καλύτερες προβλέψεις.

4 Mean Absolute Percentage Error (MAPE)

Εκφράζει το σφάλμα ως ποσοστό της πραγματικής τιμής.

$$MAPE = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

- Πόσο «εκτός» είναι οι προβλέψεις σε ποσοστιαίους όρους.
- Χρήσιμο σε επιχειρηματικά προβλήματα (π.χ. πρόβλεψη πωλήσεων).
- Ευαίσθητο όταν οι πραγματικές τιμές είναι πολύ κοντά στο μηδέν.

Συμπέρασμα: Όσο μικρότερη είναι η τιμή των μετρικών σφάλματος (MAE, MSE, RMSE, MAPE), τόσο καλύτερη είναι η απόδοση του μοντέλου παλινδρόμησης.

Συνοπτικά:
MAE → εύκολη ερμηνεία | MSE → τιμωρεί μεγάλα σφάλματα
RMSE → στην ίδια μονάδα μέτρησης | MAPE → σε ποσοστό (%)

‘Ενα απλό παράδειγμα συνελικτικών δικτύων (1/3)

```
[ ]
################################################################################
###                          KERAS IMPLEMENTATION                            ###
```

```
[ ]
# A. CUSTOM NETWORK 1/2
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.optimizers import Adam
from keras.utils import to_categorical

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize pixel values to the range [0, 1]
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
print('x_train',x_train.shape)
# Convert class vectors to binary class matrices
# The y_train and y_test variables initially contain integer values that represent class labels (e.g., 0, 1, 2, ..., 9 if there are 10 classes).
# The function to_categorical(y_train, num_classes) converts each label into a binary matrix (one-hot vector) of length num_classes.
print(y_train.shape)
print(y_test.shape)
num_classes = 10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
print('one_hot_encodded', y_train.shape)
print('one_hot_encodded', y_test.shape)
```

‘Ενα απλό παράδειγμα συνελικτικών δικτύων (2/3)

```
# Build custom CNN model

######## MAXPOOLING
# Before Pooling (4×4 feature map):
# [[1, 3, 2, 1],
#  [4, 6, 5, 2],
#  [3, 7, 9, 8],
#  [2, 4, 6, 5]]
# After MaxPooling2D(pool_size=(2,2)):
# [[6, 5],
#  [7, 9]]
# It takes the maximum value from each 2×2 block.

####### DROPOUT
#The model randomly "drops" some neurons (i.e., sets their output to 0) at each forward pass.
#Dropout is mainly used to prevent overfitting and improve generalization in deep learning models.

####### SOFTMAX
# softmax activation function Logits are the raw output values of the last layer in a neural network before applying softmax.
#To convert these logits into probabilities, we use the softmax function.

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])
```

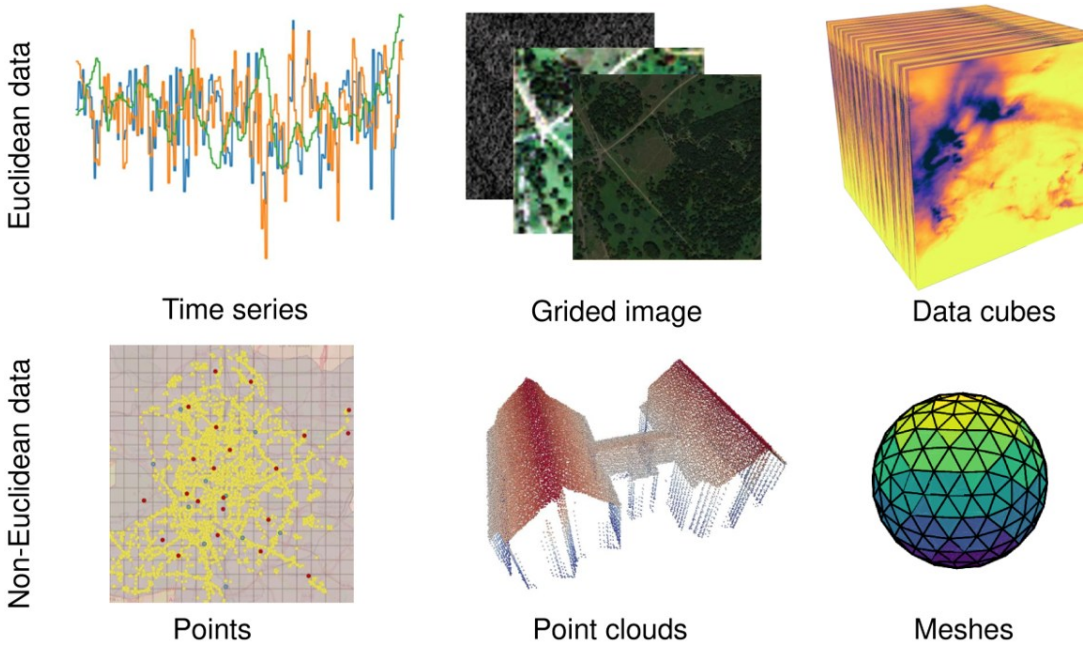
‘Ενα απλό παράδειγμα συνελικτικών δικτύων (3/3)

```
# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train, batch_size=128, epochs=1, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test)
print('\nTest accuracy:', test_acc)
```

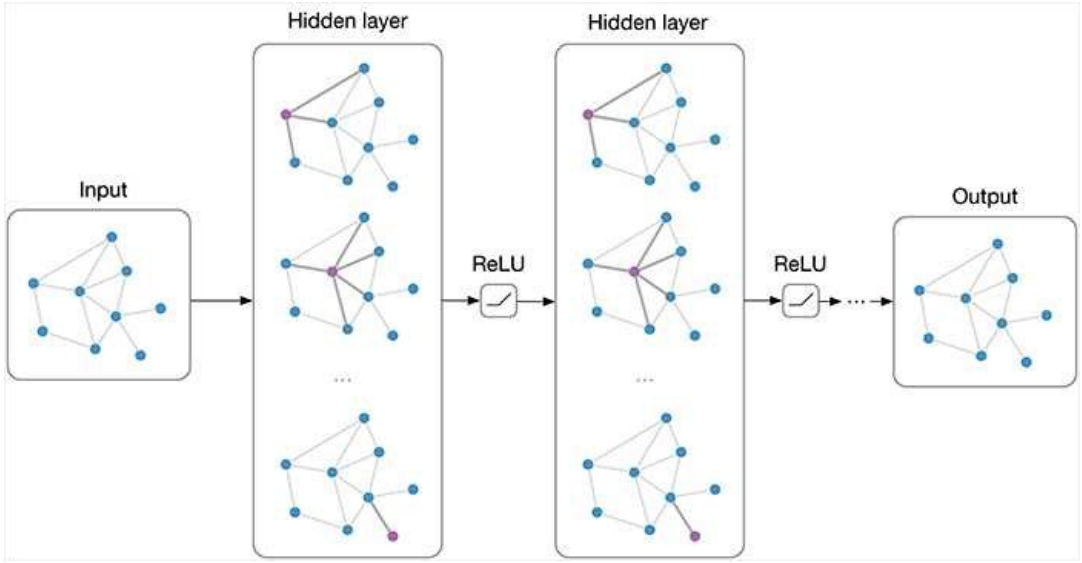
Python

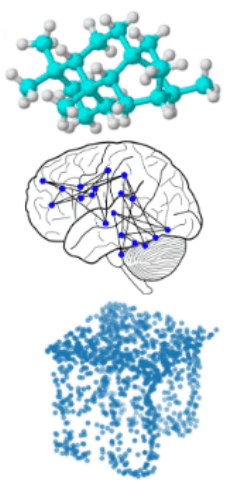
python + colab is a free, cloud-based platform that lets you write and execute Python code directly in your web browser.

Traditional Machine Learning vs Geometric Machine Learning



Graph Neural Networks & Geometric Deep Learning





Structure =
Adjacency matrix

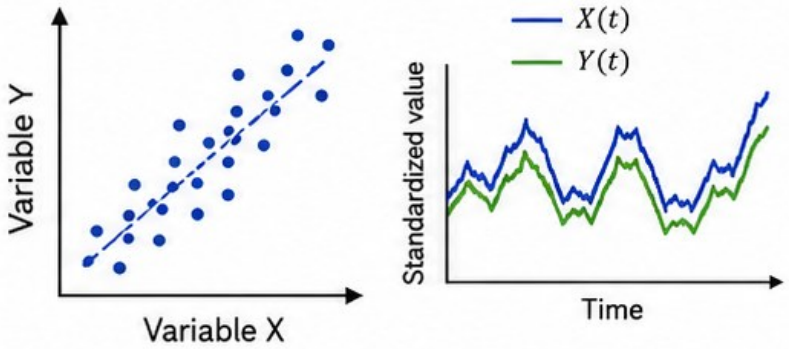
$$A \in R^{N \times N}$$

1	0	1	0
0	1	0	0
1	0	1	1
0	0	1	1

An adjacency matrix is a square grid (or 2D array) used to represent a graph mathematically

Correlation vs Causation and the role of interconnected systems

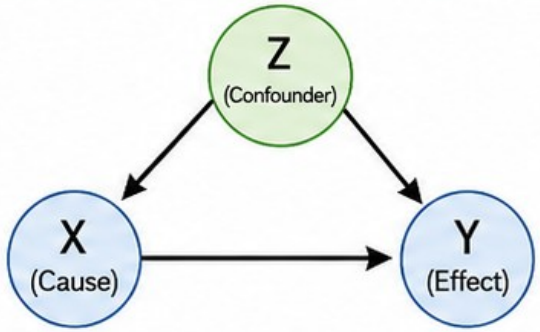
Correlation



- Association between variables
- Does not reveal direction or mechanism

Correlation ≠ causation

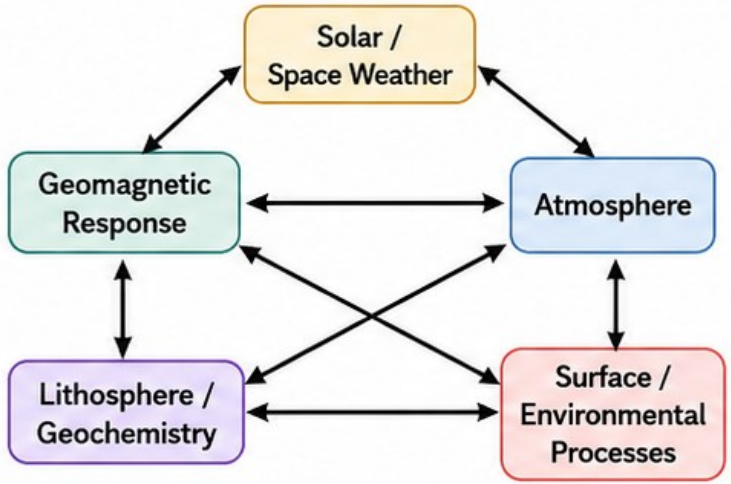
Causation



$$P(Y \mid do(X)) \neq P(Y \mid X)$$

- Focuses on cause–effect relationships
- Accounts for confounders, interventions, and directionality

Interconnected systems



Observed links may emerge through indirect pathways across coupled subsystems.

Causal learning ideas

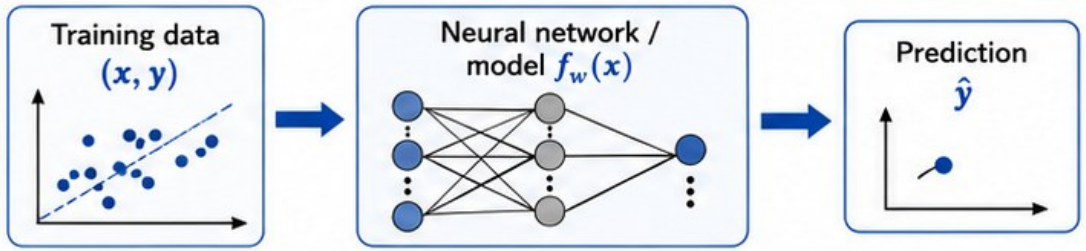
1. Structural Causal Models (SCMs) & DAGs
2. Confounder control
3. Temporal causal discovery (e.g. lagged dependencies, PCMCI/Granger)
4. Interventions & counterfactual reasoning
5. Domain knowledge / physics-informed constraints

In complex Earth and environmental systems, causal learning helps move from observed associations to interpretable mechanisms.

Traditional Machine Learning vs Simulation-Based Inference (SBI)

Prediction from labeled data vs Bayesian inference from simulations

1. Traditional Machine Learning



Learn: $\hat{y} = f_w(x)$

Train by minimizing a loss:

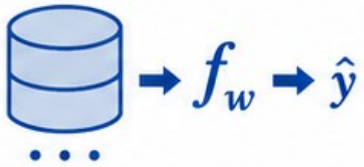
$$w^* = \arg\min_w \sum_i L(y_i, \hat{y}_i)$$

Common examples

Regression: $\text{MSE} = \frac{1}{N}\sum_i (y_i - \hat{y}_i)^2$

Classification:

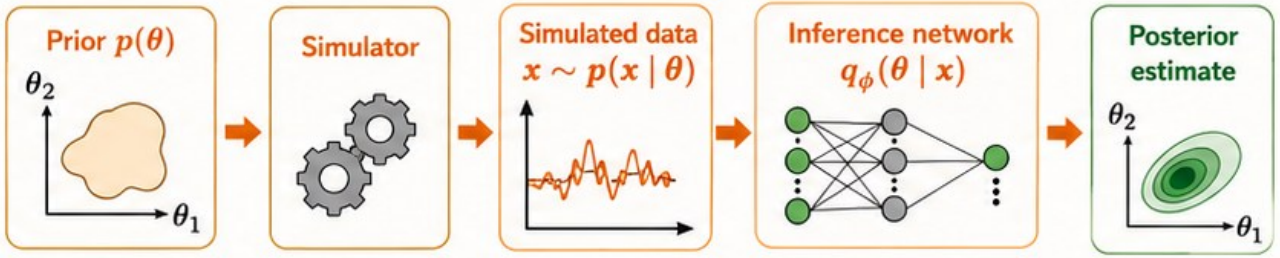
Cross-entropy $= -\sum_i \log p(y_i \mid x_i, w)$



$\rightarrow f_w \rightarrow \hat{y}$

- **Input:** labeled real data
- **Goal:** predict outputs y from inputs x
- **Output:** a point prediction or class probability
- **Typical question:** "Given x, what is y?"

2. Simulation-Based Inference (SBI)



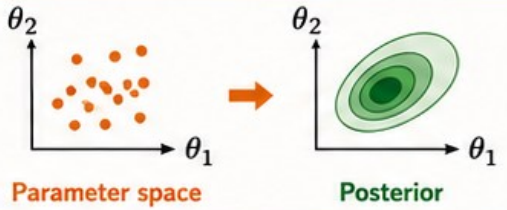
Goal: infer the posterior

$$p(\theta \mid x_{\text{obs}}) = \frac{p(x_{\text{obs}} \mid \theta)\, p(\theta)}{p(x_{\text{obs}})}$$

Train on simulated pairs (θ_i, x_i)

$$q_\phi(\theta \mid x) \approx p(\theta \mid x)$$

$$\phi^* = \arg\min_\phi \left[-\sum_i \log q_\phi(\theta_i \mid x_i) \right]$$



- **Input:** simulations + prior knowledge
- **Goal:** infer unknown parameters θ
- **Output:** posterior distribution with uncertainty
- **Typical question:** "Given observed data x_{obs}, which parameters θ could have generated it?"

Main difference

1. Traditional ML learns $x \rightarrow y$ for prediction.
2. SBI learns $x \rightarrow p(\theta \mid x)$ for parameter inference.
3. Traditional ML uses labeled data; SBI uses simulations when the likelihood is difficult or unavailable.

SBI is not just prediction — it is Bayesian inference powered by simulations and neural networks.

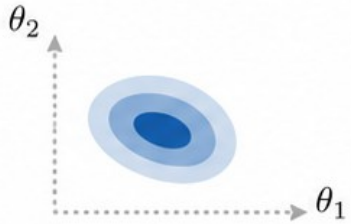
Simulation-Based Inference (SBI)

Bayesian inference with simulations and neural networks

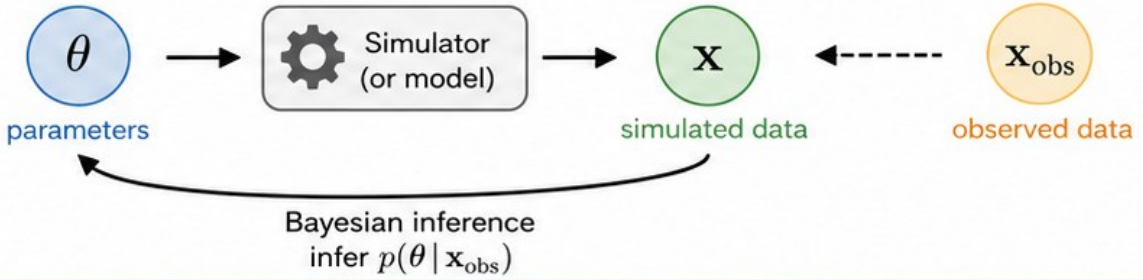
1 The Goal

We observe data $\mathbf{x}_{\text{obs}}$ and want to infer the unknown parameters θ that generated it.

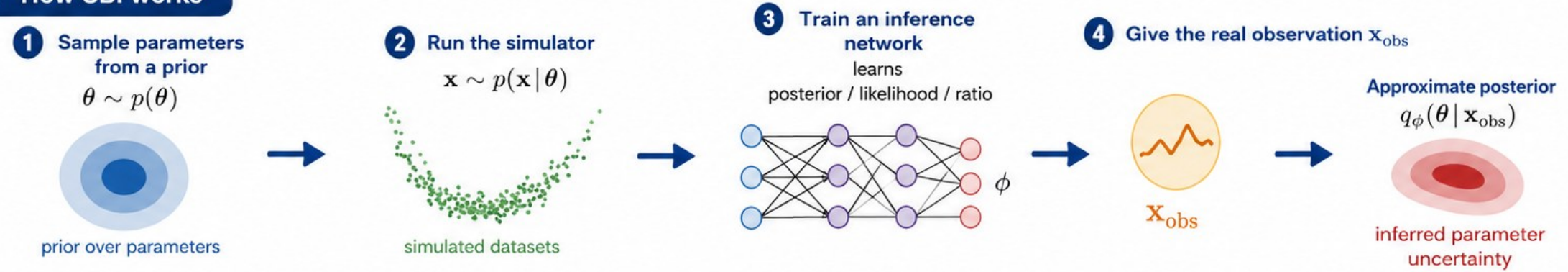
posterior: $p(\theta \mid \mathbf{x}_{\text{obs}})$



2 Classical Bayesian idea



3 How SBI works



4 Why SBI is useful

Works when the likelihood is hard or impossible to write down

Uses realistic simulations

Handles limited and noisy data

After training, inference is fast (amortized inference)

Common methods

- **SNPE** = Neural Posterior Estimation
- **SNLE** = Neural Likelihood Estimation
- **SNRE** = Neural Ratio Estimation

Inspired by the Jülich Supercomputing Centre course: Introduction to Simulation Based Inference: Enhancing Synthetic Models with Artificial Intelligence (2026).

https://github.com/smsharma/awesome-neural-sbi#cosmology-astrophysics-and-astronomy

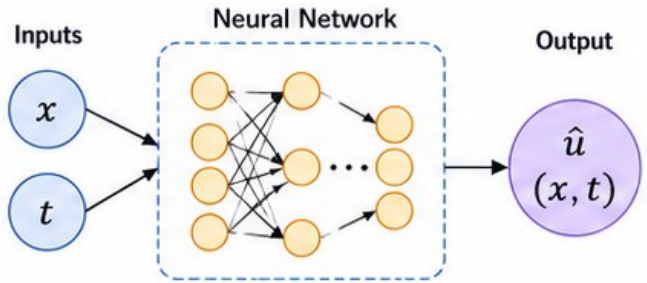
Ιδέες απο papers

Physics-Informed Neural Networks (PINNs)

Learning from data while enforcing known physical equations

1) Key Idea

- We know the governing physics (e.g. ODEs/PDEs, conservation laws, boundary/initial conditions).
- The neural network approximates the solution: $\hat{u}(x, t; \theta)$.
- Training is guided by **both observed data** and the **physical equations**.



The network outputs the solution approximation $\hat{u}(x, t; \theta)$.

2) From the physical equation to the residual

1 Known physical equation

$$\mathcal{N}[u(x,t)] = 0$$

$\mathcal{N}$: differential operator (e.g., PDE/ODE)

2 Network approximation

$$\hat{u}(x,t;\theta)$$

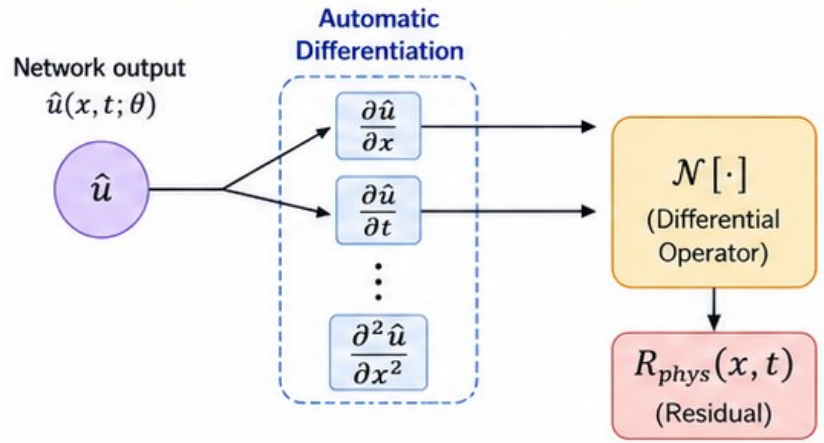
Neural network with parameters θ

3 Physics residual

$$R_{phys}(x,t) = \mathcal{N}[\hat{u}(x,t;\theta)]$$

If the network respects the known physics, the residual should be **close to zero**.

How derivatives are obtained and used



Automatic differentiation computes spatial/temporal derivatives of $\hat{u}$ and inserts them into the operator $\mathcal{N}$ to form the residual.

3) Training loss

$$\mathcal{L} = \lambda_{data}\mathcal{L}_{data} + \lambda_{phys}\mathcal{L}_{phys} + \lambda_{bc}\mathcal{L}_{bc}$$

Data loss

$$\mathcal{L}_{data} = \frac{1}{N}\sum_i \left|\hat{u}(x_i,t_i) - u_i^{obs}\right|$$

Fit to observed data

Physics loss

$$\mathcal{L}_{phys} = \frac{1}{N_r}\sum_j \left|R_{phys}(x_j,t_j)\right|^2$$

Enforce the physical equation

BC/IC loss

$\mathcal{L}_{bc}$: boundary/initial condition loss

Enforce boundary/initial conditions

Unlike traditional neural networks, PINNs do not learn only from data — they also learn by **satisfying the known physical laws**.

Why PINNs?

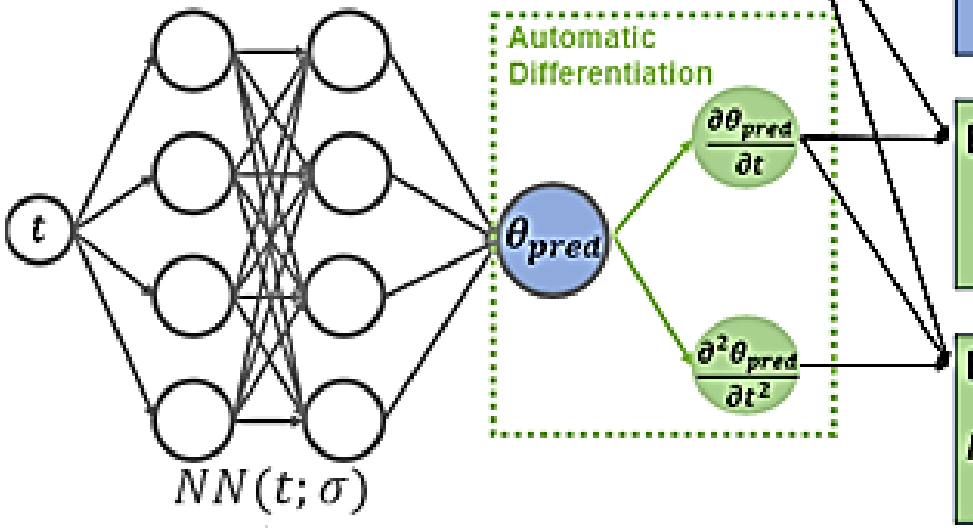
- Uses prior physical knowledge to guide learning.
- Helps when data are limited or expensive to obtain.
- Produces physically consistent solutions.

Physics-Informed Neural Networks (PINNs)

$$\theta''(t) + 2\beta\theta'(t) + \omega_0^2\theta(t) = 0 \quad \text{— Pendulum equation}$$

$$\theta(t_0) = \theta_0 \quad \text{— Initial conditions}$$

$$\theta'(t_0) = v_0 \quad \text{— Initial conditions}$$



Additional measurement data (optional)

$$L_{Data}(\sigma) = \frac{1}{N}\sum_{i=1}^{N}\left|\theta_{pred}(t_i;\sigma) - \theta_{meas}(t_i)\right|^2$$

+

Boundary/Initial Conditions (optional)

$$L_{Conds}(\sigma) = \frac{1}{2}\left|\theta_{pred}(t_0;\sigma) - \theta_0\right|^2 + \frac{1}{2}\left|\frac{\partial}{\partial t}\theta_{pred}(t_0;\sigma) - v_0\right|^2$$

+

Differential Equation

$$L_{Physics}(\sigma) = \frac{1}{M}\sum_{i=1}^{M}\left|\frac{\partial^2}{\partial t^2}\theta_{pred}(t_i;\sigma) + 2\beta\frac{\partial}{\partial t}\theta_{pred}(t_i;\sigma) + \omega_0^2\theta_{pred}(t_i;\sigma)\right|^2$$

Loss function

$$L(\sigma) = L_{Physics}(\sigma) + L_{Conds}(\sigma) + L_{Data}(\sigma)$$

Optimization Algorithm

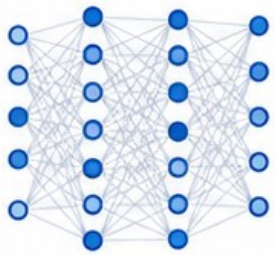
PINN = neural network trained to fit data and satisfy a known equation

NEURAL OPERATORS

From classical neural networks to neural operators

Classical Neural Networks

Learn a function $y = f(x)$



$Wx + b$

Input:
feature vector x

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

→ **Linear layer** → **Non-linearity** → • • • →

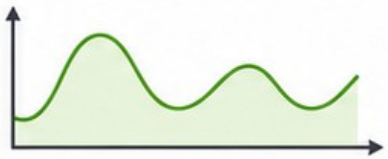
Output:
label / value y

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

ⓘ Input and output are finite-dimensional vectors.

Neural Operators

Learn an operator $u = G(a)$



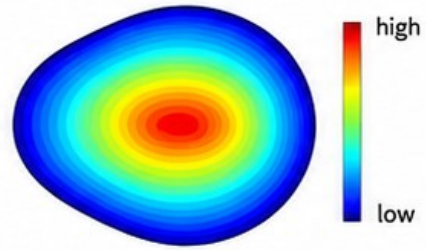
$\int \kappa(x,y)a(y)dy + W(a)(x) + b(x)$

Input:
function a

$a(x)$

→ **Integral / operator layer** → **Non-linearity** → • • • →

Output:
solution function u



ⓘ Input and output are functions / fields.

- Neural operators output functions, not just numbers or labels.
- They learn mappings between function spaces (e.g. initial condition -> solution field).
- They are useful for PDEs, surrogate modeling, and scientific machine learning.

Main idea: a standard neural network learns pointwise mappings $x \longrightarrow y$, while a neural operator learns mappings between functions $a(\cdot) \longrightarrow u(\cdot)$.

Thank you