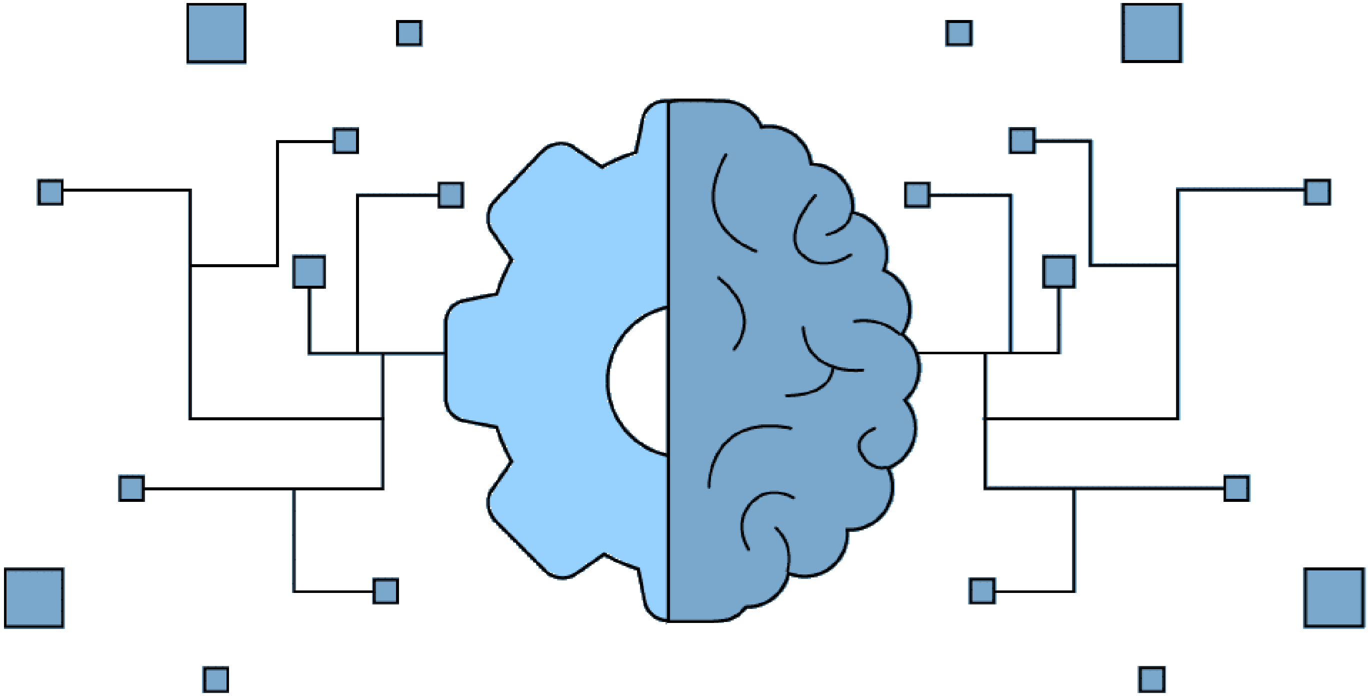
# MACHINE LEARNING

(HEP oriented)



Arantza Oyanguren, Jiahui Zhuo (IFIC - Valencia)

# Contents

- Introduction & Concepts
- Supervised Learning:
  - Regression
  - Classification
- Unsupervised Learning
  - Clustering
  - Dimensionality reduction
- Neural Networks
- Reinforcement learning
- Underlying hardware & libraries
- Hands-on

# Introduction & concepts

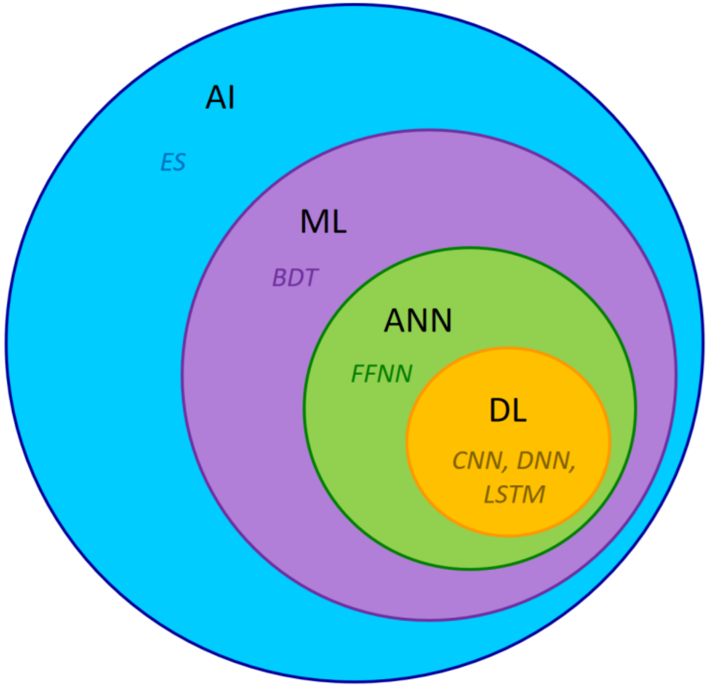
OpenArt

# Introduction

- What Machine Learning is ?

A subset of artificial intelligence (AI): machine learning (ML) deals with the study and use of **data and algorithms that mimic how humans learn**. This helps machines gradually improve their accuracy.
It estimates new output values by using historical data as input.

- **Artificial Intelligence (AI)**
  - *Expert Systems (ES)*
- **Machine learning (ML)**
  - *Boosted Decision Trees (BDT)*
- **Artificial Neural Network (ANN)**
  - *Feedforward Neural Network (FFNN)*
- **Deep Learning (DL)**
  - *Convolutional Neural Network (CNN),*
  - *Deep Neural Network (DNN),*
  - *Long short-term memory (LSTM)*



# Introduction

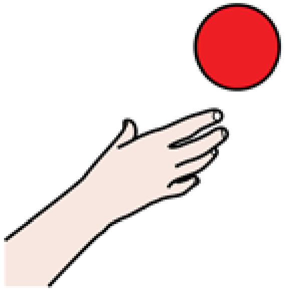
- Why Machine Learning ?

The problem

- Identification
- Classification
- Anomaly detection
- Selection
- Generation

The tools

- Big data
- Computing power
- New architectures
- Improved algorithms

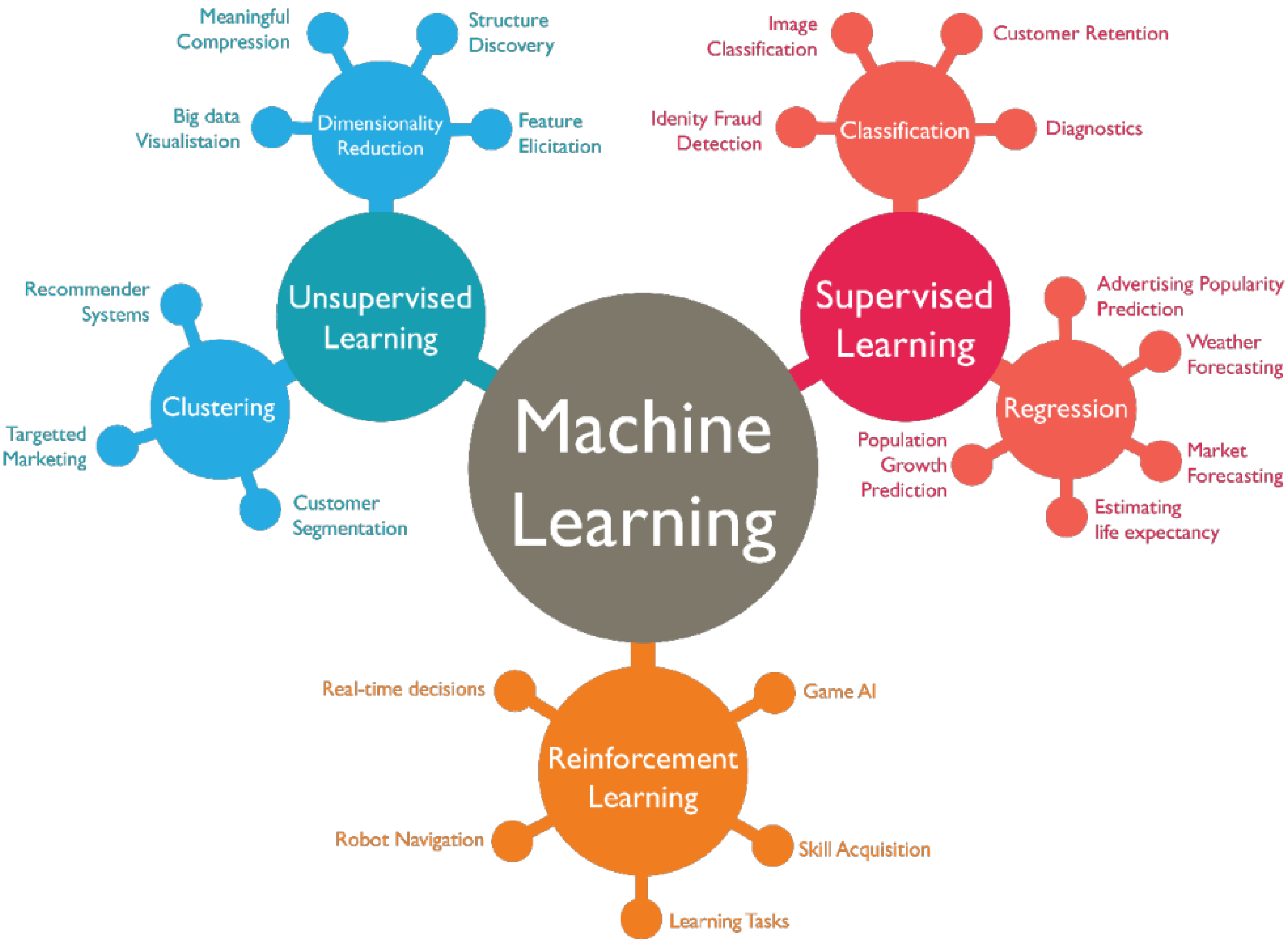


The wish

- Fast response
- High performance
- Unbiased
- Sustainable

# Introduction

- How to learn without explicitly being programed ?

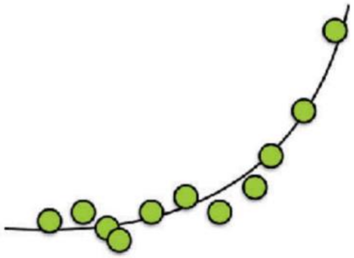


# Machine Learning

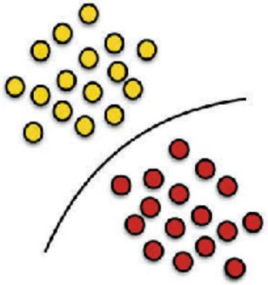
- How to learn without explicitly being programed ?

**Supervised learning:** the data is labeled in the way that we give inputs to the learning system and tell it which specific outputs should be associated to the inputs. I.e., the goal is to learn a mapping from inputs to outputs.

Examples:
*Regression* (weather prediction)  and *classification* (image classification)



The output comes in a large finite ordered or continuous set

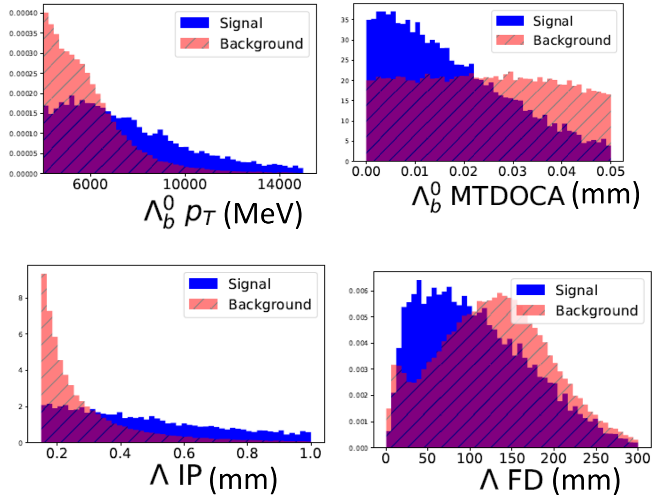


The output comes in a small finite set

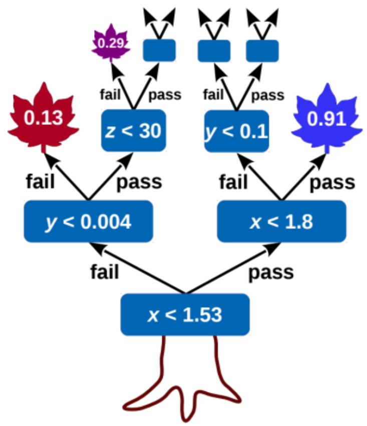
# Machine Learning

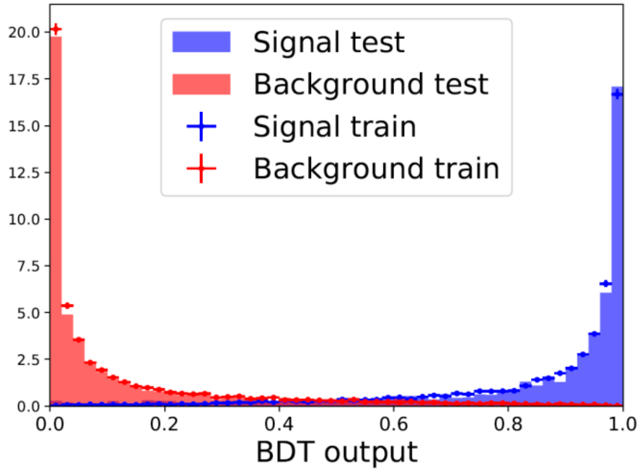
- Supervised learning

***Boosted Decision Trees (BDT)*** to separate signal and background sources

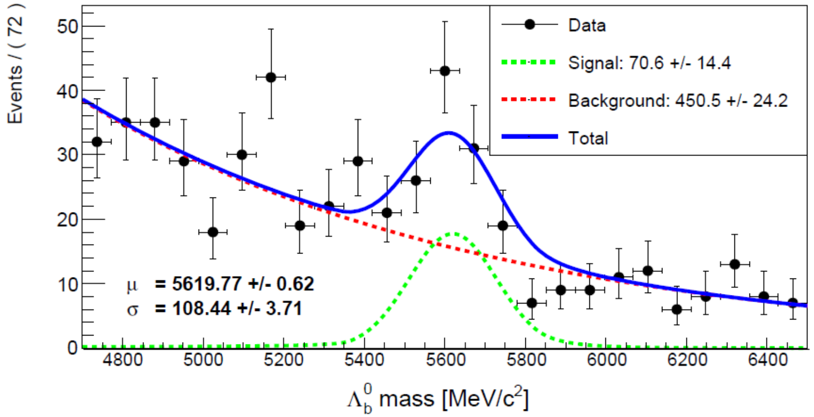


arXiv:2206.09645





Observation of the $\Lambda_b \rightarrow \Lambda\gamma$ decay channel:



Ex: XGBoost

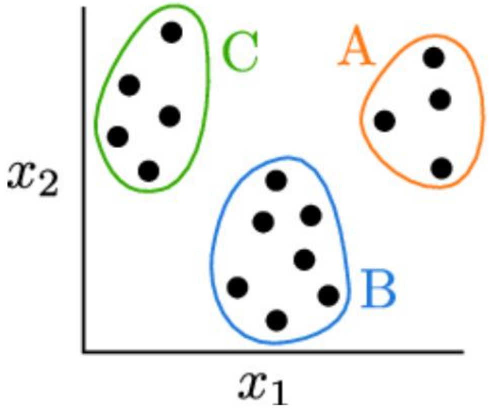
[CERN-THESIS-2020-404]

# Machine Learning

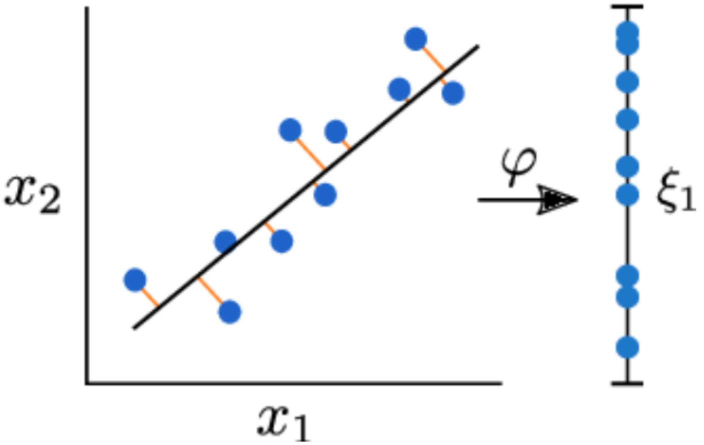
- How to learn without explicitly being programed ?

**Unsupervised learning:** the input data is not labelled, the learning system has to find some patterns or structures inherent in it.

Examples:
*Clustering* (recommendation systems) and *dimensionality reduction* (structure discovery)

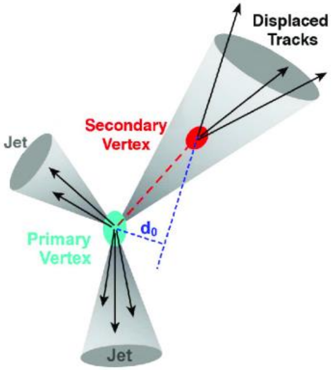


The output is characterized by group-labels and centroids



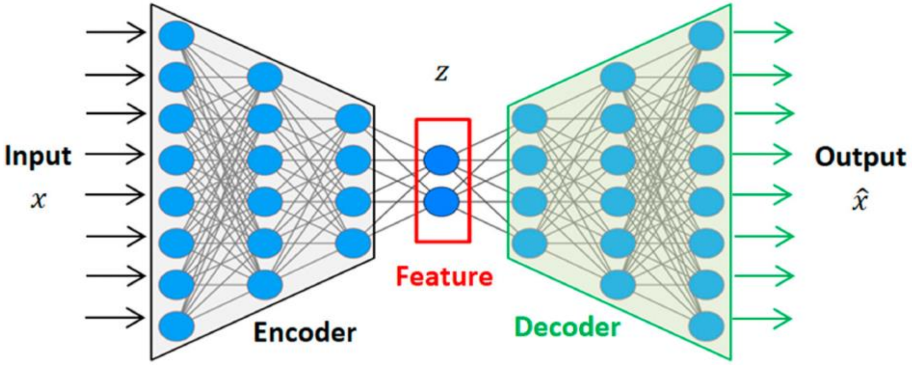
Simplified output with the most important patterns and information from the original data

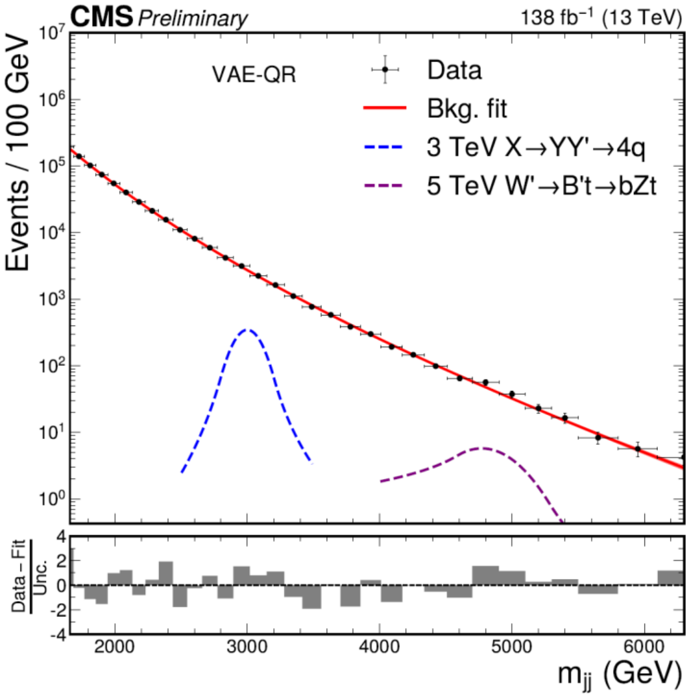
# Machine Learning



- Unsupervised learning

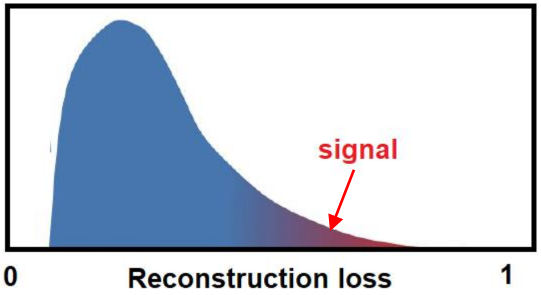
***Autoencoders*** for Anomaly Detection (Ex: VAE at CMS experiment)
Searching anomalies in di-jet resonances.





Training on background-dominated samples.

The difference between the original and reconstructed data can be used as an effective anomaly score.



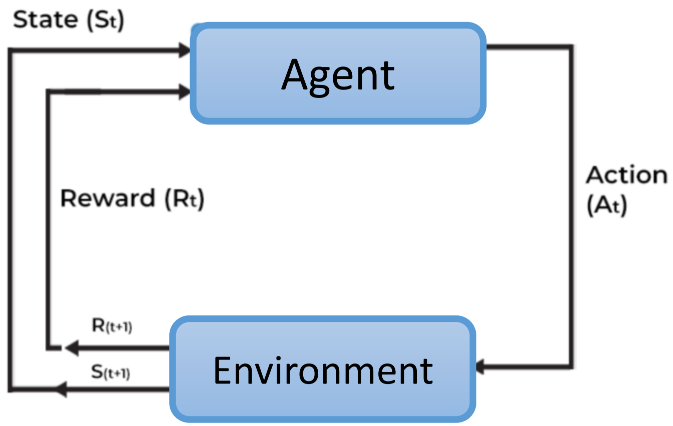
[CMS-PAS-EXO-22-026]

# Machine Learning

- How to learn without explicitly being programed ?

**Reinforcement learning:** the learning system has to learn a mapping from input values to output values without being supervised. The idea is that an agent learns to make decisions by interacting with the environment. The agent takes actions and receives feedback in the form of rewards or penalties.

Examples: Natural language processing, robot navigation, autonomous driving… (algorithms can be model-free or model-based)



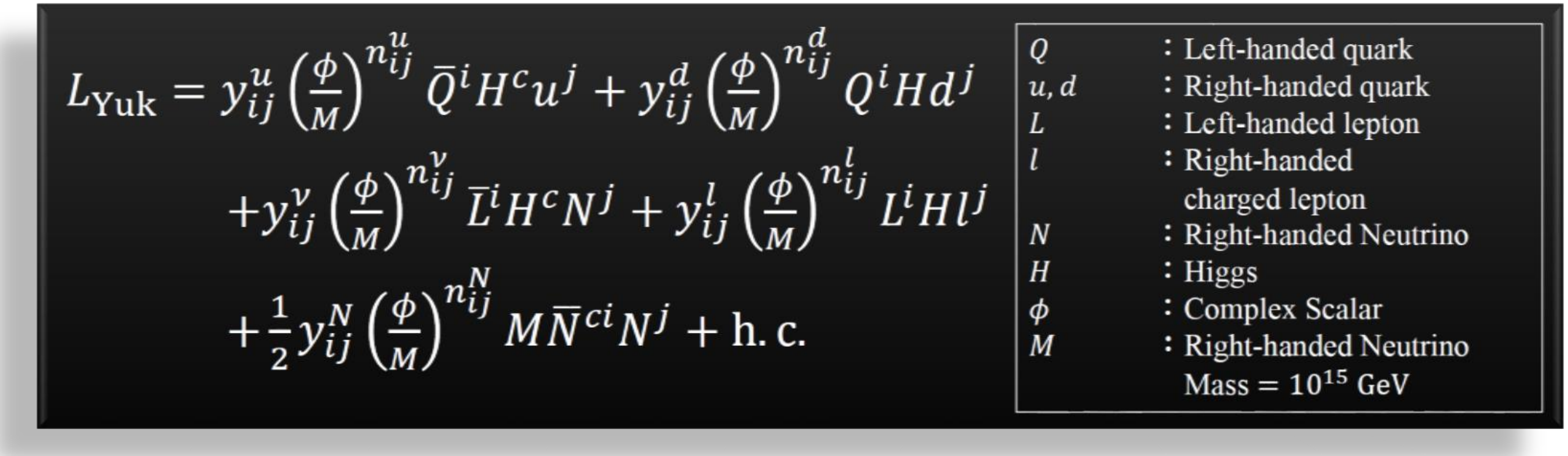
The output is a policy, i.e., a set of rules or a strategy to maximize cumulative rewards over time.

# Machine Learning

- Reinforcement learning

**Deep Q-Network (DQN)** to explore the flavor structure of quarks and leptons

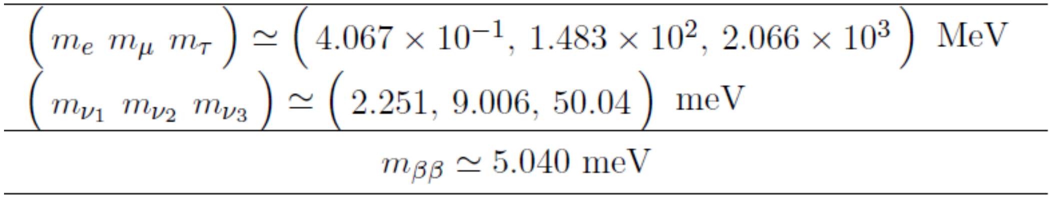
*Yukawa lagrangian with Froggatt-Nielsen Model:* [JHEP12(2023)021]



$$
\begin{aligned}
L_{\text{Yuk}} = & \, y^u_{ij}\left(\frac{\phi}{M}\right)^{n^u_{ij}} \bar{Q}^i H^c u^j + y^d_{ij}\left(\frac{\phi}{M}\right)^{n^d_{ij}} Q^i H d^j \\
& + y^\nu_{ij}\left(\frac{\phi}{M}\right)^{n^\nu_{ij}} \bar{L}^i H^c N^j + y^l_{ij}\left(\frac{\phi}{M}\right)^{n^l_{ij}} L^i H l^j \\
& + \frac{1}{2} y^N_{ij}\left(\frac{\phi}{M}\right)^{n^N_{ij}} M \bar{N}^{ci} N^j + \text{h.c.}
\end{aligned}
$$

| | |
|---|---|
| $Q$ | : Left-handed quark |
| $u, d$ | : Right-handed quark |
| $L$ | : Left-handed lepton |
| $l$ | : Right-handed charged lepton |
| $N$ | : Right-handed Neutrino |
| $H$ | : Higgs |
| $\phi$ | : Complex Scalar |
| $M$ | : Right-handed Neutrino Mass $= 10^{15}$ GeV |

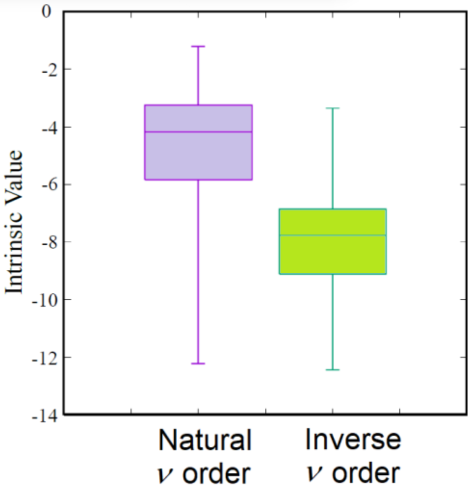
→ The agent gets points when the masses of particles and the mixing matrix are close to the experimental values.



$$\begin{pmatrix} m_e & m_\mu & m_\tau \end{pmatrix} \simeq \begin{pmatrix} 4.067 \times 10^{-1}, & 1.483 \times 10^2, & 2.066 \times 10^3 \end{pmatrix} \text{ MeV}$$

$$\begin{pmatrix} m_{\nu_1} & m_{\nu_2} & m_{\nu_3} \end{pmatrix} \simeq \begin{pmatrix} 2.251, & 9.006, & 50.04 \end{pmatrix} \text{ meV}$$

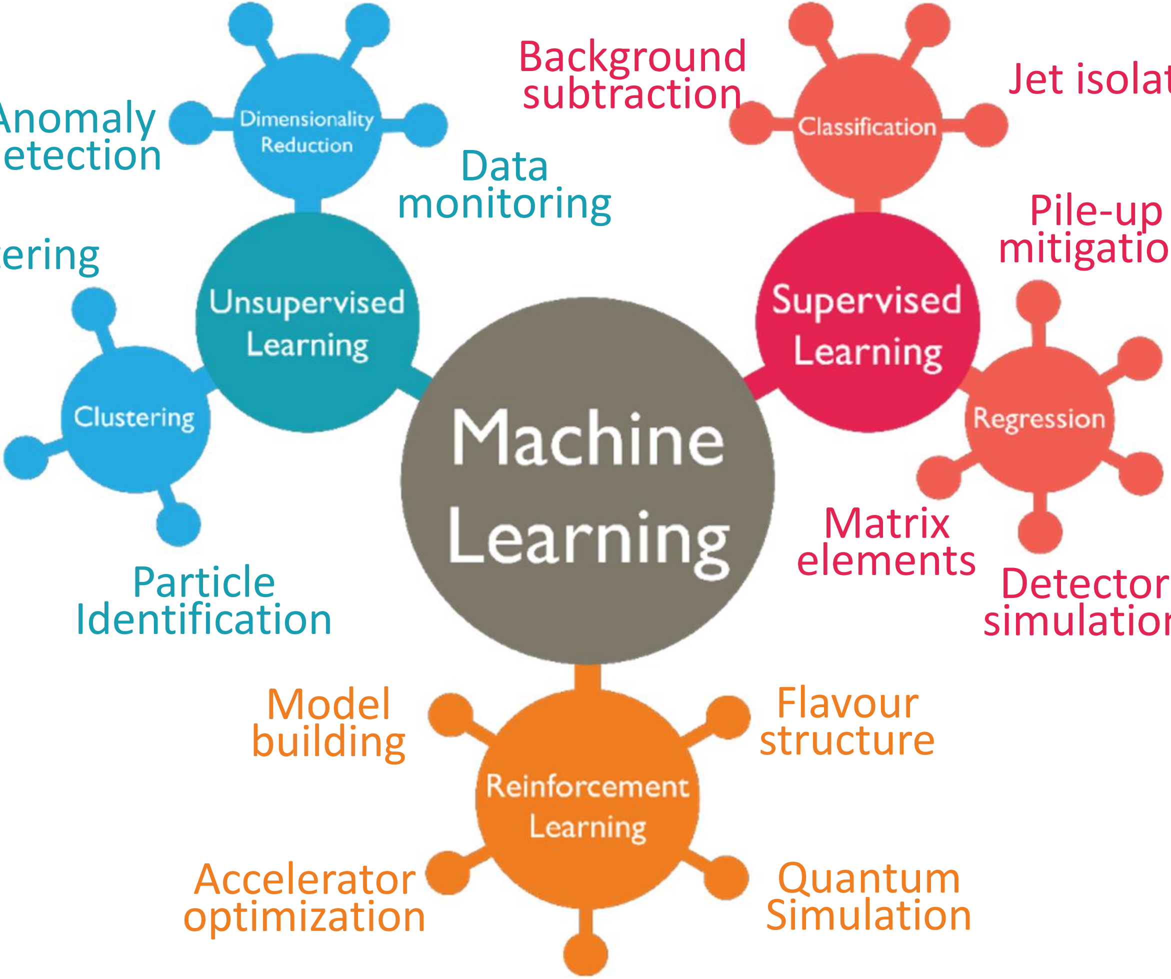
$$m_{\beta\beta} \simeq 5.040 \text{ meV}$$



RL predicts a natural ordering for neutrino masses!

# Machine Learning

- In HEP:



Jet substructure

Noise reduction

Anomaly detection

Data monitoring

Jet clustering

Hadronic showers

Particle Identification

Calibration

Pattern recognition

Background subtraction

Jet isolation

Pile-up mitigation

Event reconstruction

Detector calibration

Matrix elements

Detector simulation

Model building

Flavour structure

Accelerator optimization

Quantum Simulation

Detector design

See https://iml-wg.github.io/HEPML-LivingReview/

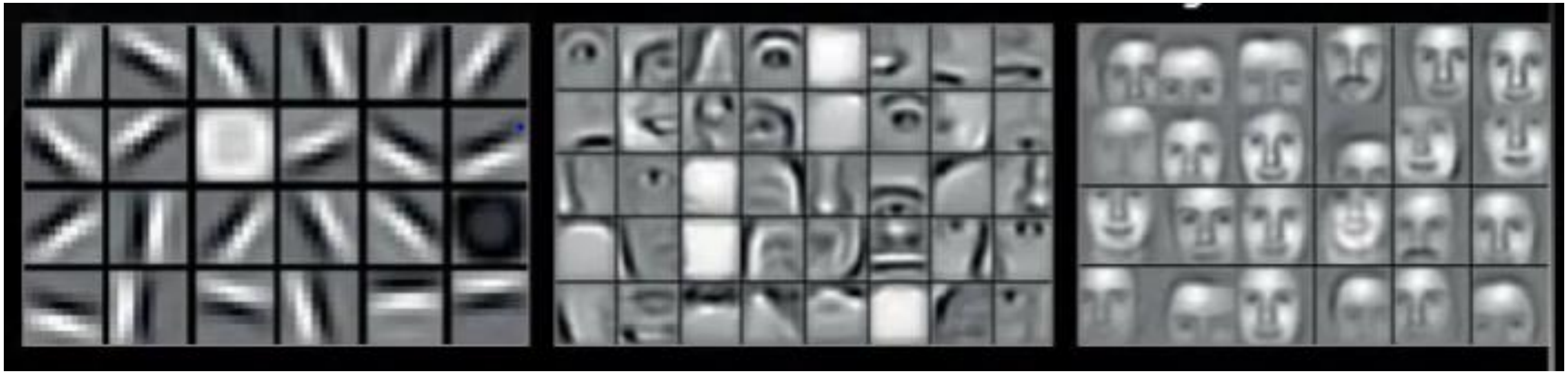
# Concepts

- UNDERLYING FEATURES

Data, features and labels

Raw data

Low level attributes ------------------------------------------------------ High level attributes



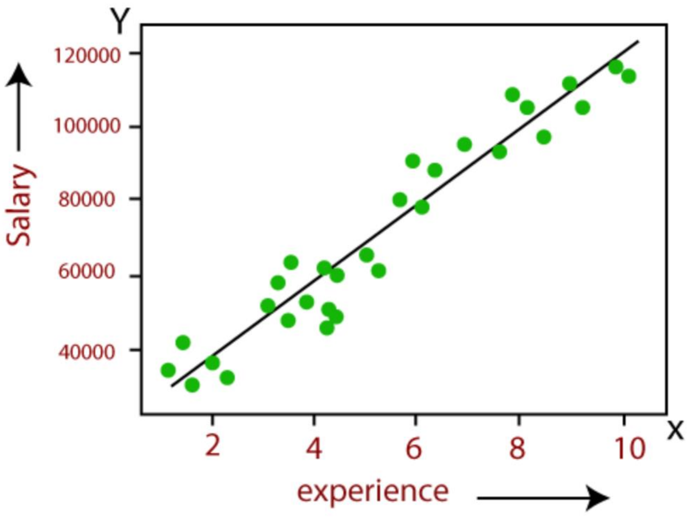
Lines & edges    Noses, mouths & ears    Facial structure

# Concepts

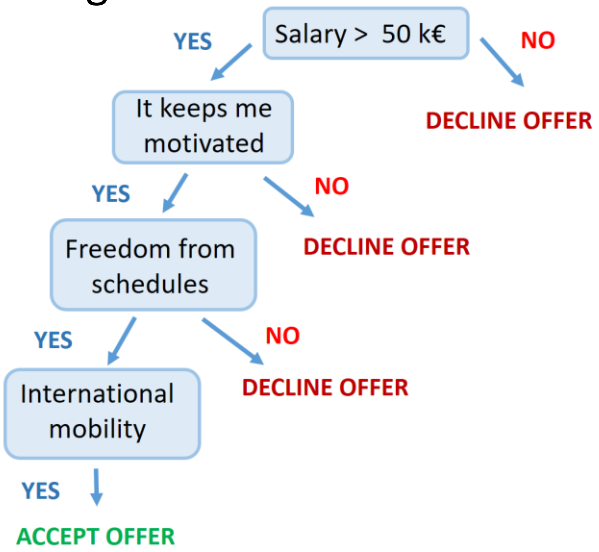
- MODELS

A model is a mathematical representation of the relationship between features and labels. It's created by applying a machine learning algorithm to the *training data*. The model is able to make predictions or decisions based on new, unseen data.

Ex: *a linear regression model* predicts a continuous output.



Ex: *a decision tree* classifies inputs into different categories.



Salary > 50 k€

YES

NO

DECLINE OFFER

It keeps me motivated

YES

NO

DECLINE OFFER

Freedom from schedules

YES

NO

DECLINE OFFER

International mobility

YES

ACCEPT OFFER

**Parameters:** internal variables that are learned from data (ex: slope)
**Hyperparameters:** characterize the learning process (ex: number of nodes)

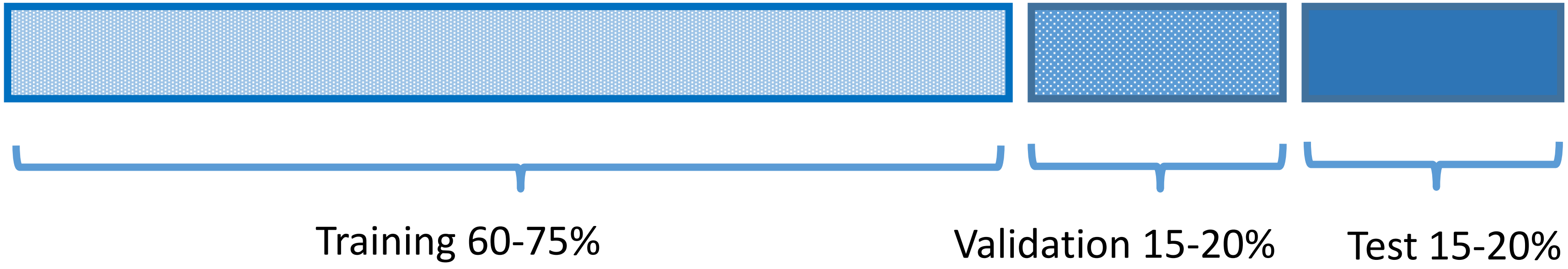
# Concepts

## TRAINING, VALIDATION AND TEST DATA

- **Training Data:** The portion of the dataset used to fit the model, allowing it to learn patterns and relationships within the data.

- **Validation Data:** A data subset to tune model *hyperparameters* and assess model performance during training, helping to prevent *overfitting*.

- **Test Data:** The final subset of the dataset used to evaluate the model's performance after training and validation, providing an unbiased assessment of its generalization to new, unseen data.



Training 60-75%

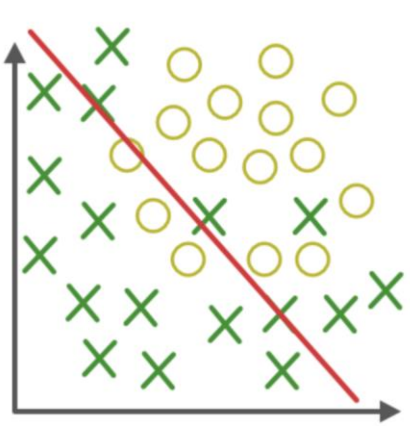
Validation 15-20%

Test 15-20%

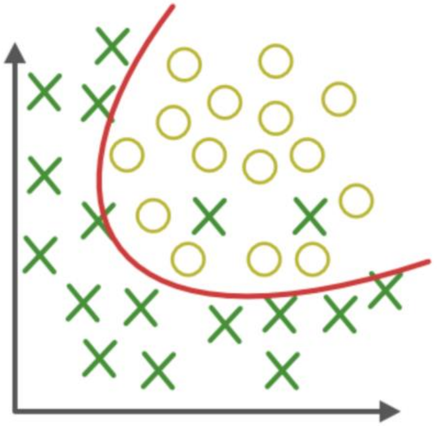
# Concepts

## GENERALIZATION

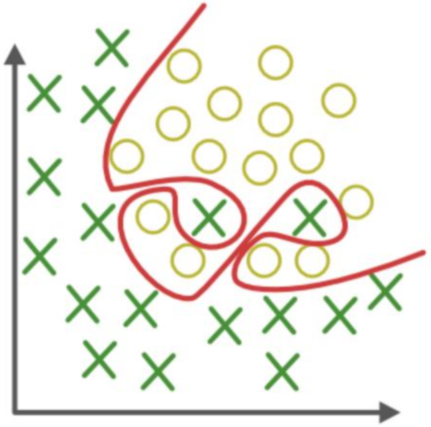
The ability of a model to perform well on new, unseen data.
We want a model that avoids both ***underfitting*** (failing to capture the data's structure) and ***overfitting*** (being too tailored to the training data).



***underfitting***



***proper fitting***



***overfitting***

- Too simple model
- Insufficient training
- Poor feature selection
- High *regularization* (penalty term)
- Incorrect model assumptions

- Too complex model
- Excessive training
- Too many features
- Low *regularization* (penalty term)
- Insufficient training data

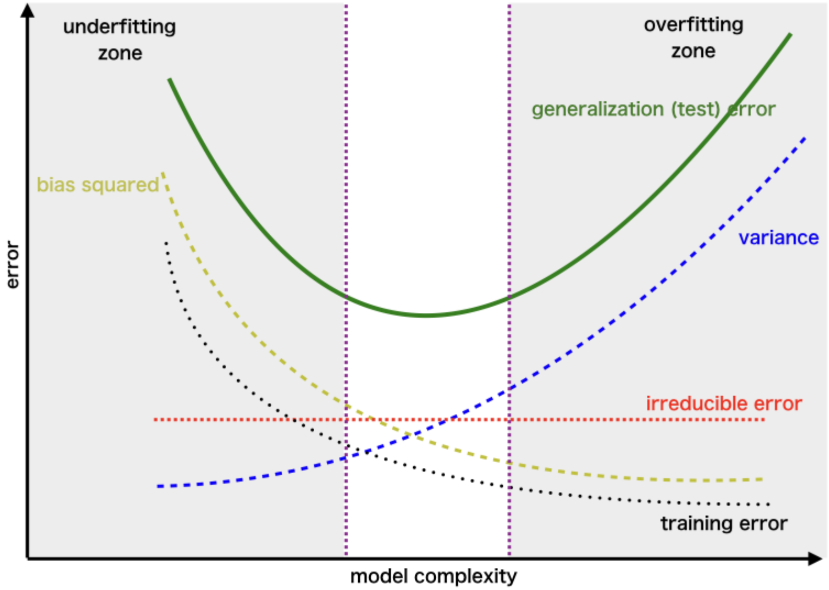
# Concepts

- INTERPRETABILITY & EXPLAINABILITY

We want to comprehend the decisions or predictions made by a model (cause-and-effect), and provide, for complex models, a clear explanation for why the model made them.

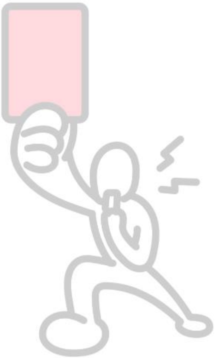
**Bias:** deviation from the real value introduced by approximating a problem, which may be complex, with a simplified model.
→ High bias implies that the model is too simple and fails to capture the underlying patterns in the data, leading to *underfitting*.

**Variance:** deviation from the real value introduced due to the model's sensitivity to small fluctuations in the training data.
→ High variance implies that the model is too complex and captures noise along with the underlying patterns, leading to *overfitting*.



$$\boxed{\text{Expected Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}}$$

# Concepts



## SCORES & PENALTIES

Models are evaluated not just on how well they score in making predictions, but also on how they're penalized for errors.

***Loss function*** *$\mathcal{L}$(g,t)* : penalizes a wrong decision g (*guess*) when the answer is *t (true)*.

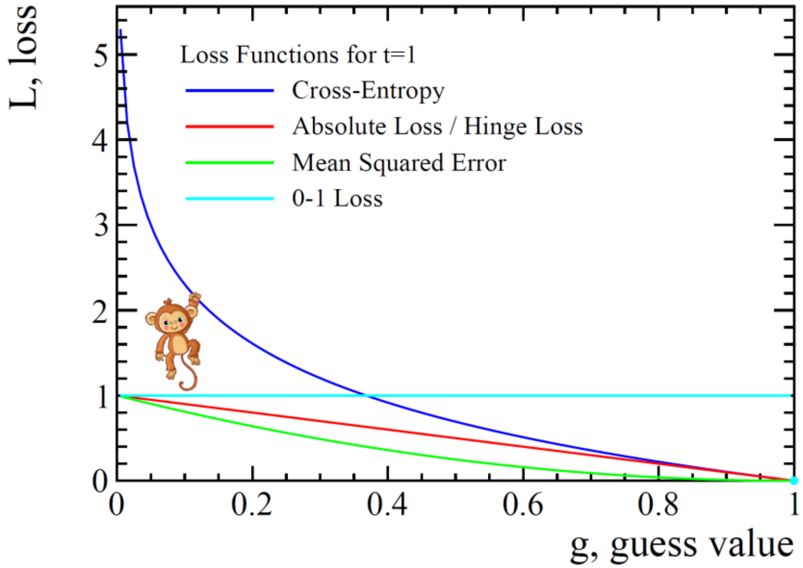
$$\text{0-1 Loss} = \begin{cases} 0 & \text{if } t_i = g_i \\ 1 & \text{if } t_i \neq g_i \end{cases}$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(t_i - g_i)^2$$

$$\text{Absolute Loss} = \frac{1}{n}\sum_{i=1}^{n}|t_i - g_i|$$

$$\text{Hinge Loss} = \sum_{i=1}^{n}\max(0, 1 - t_i \cdot g_i)$$

$$\text{Cross-Entropy} = -\frac{1}{n}\sum_{i=1}^{n}[t_i \log(g_i) + (1 - t_i)\log(1 - g_i)]$$



# Concepts

- GRADIENT DESCENT

It is an optimization procedure to minimize the error or *loss* of machine learning algorithms

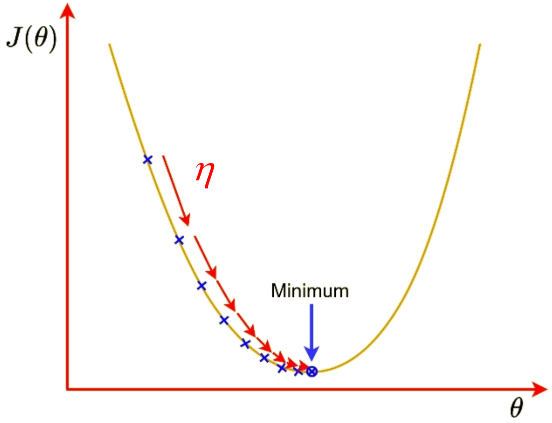
1- Initialize the parameters ($\theta$) of the model

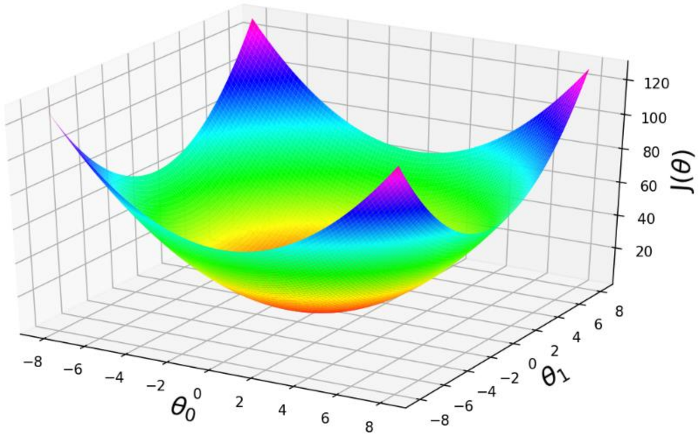
2- Compute the gradient (i.e., partial derivatives of the loss ($J$) with respect to each parameter)

3- Update the parameters adjusting them through the *learning rate* $\eta$ (a fraction of the gradient)

$$\theta_{new} = \theta_{old} - \eta \cdot \nabla_\theta J(\theta)$$

4- Iterate until the algorithm converges in a minimum



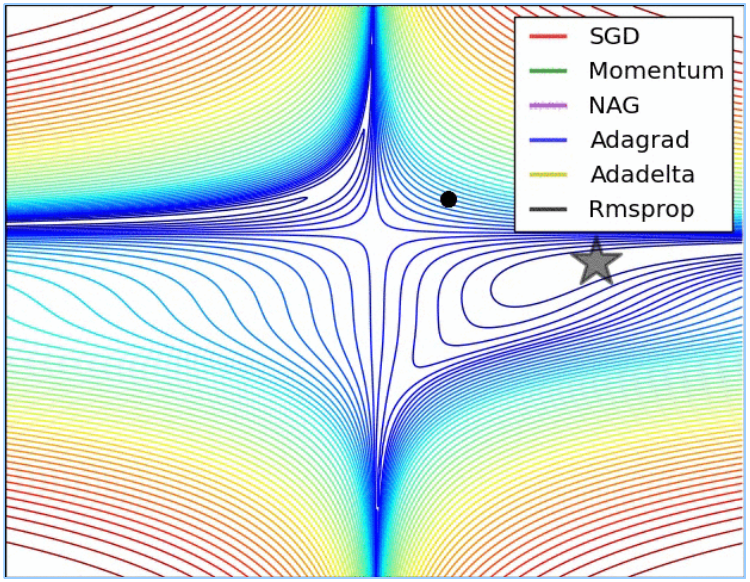


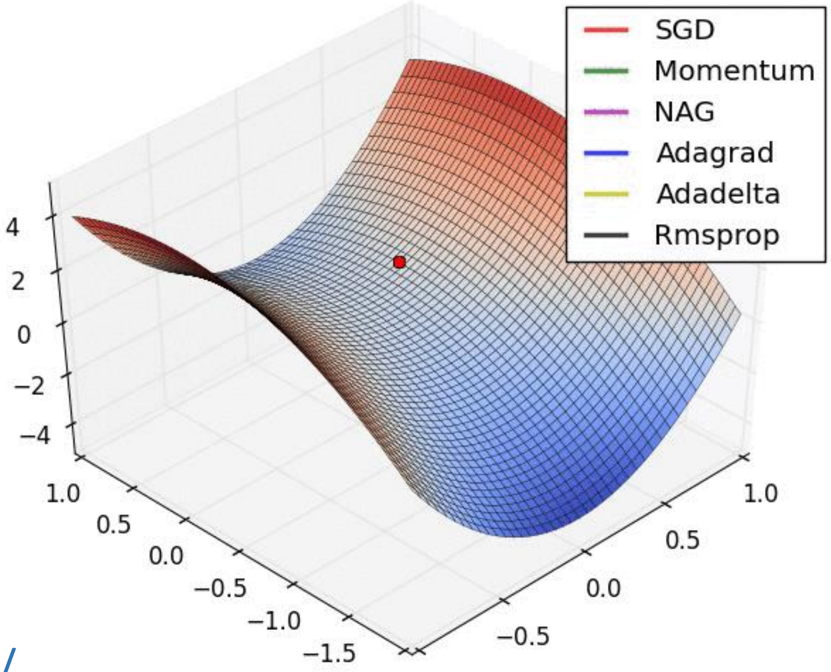
Note that the learning rate $\eta$ governs the convergence:
$\rightarrow$ if it is too small, the optimization will be too slow
$\rightarrow$ if it is too large, it can skip the minimum

# Concepts

→ Some examples of Gradient Descent-based algorithms:

- **Batch Gradient Descent (BGD)**: Uses the entire dataset to compute the gradient at each step.
- **Stochastic Gradient Descent (SGD)**: Uses one sample at a time, updating parameters more frequently.
- **Momentum; NAG**: Use a cumulative function "velocity" of the gradient to improve SGD.
- **Adagrad; Adadelta**: Adaptive learning rate depending of some parameters.
- **RMSprop** (Root Mean Square Propagation) also adaptive, but simpler way to compute gradients.

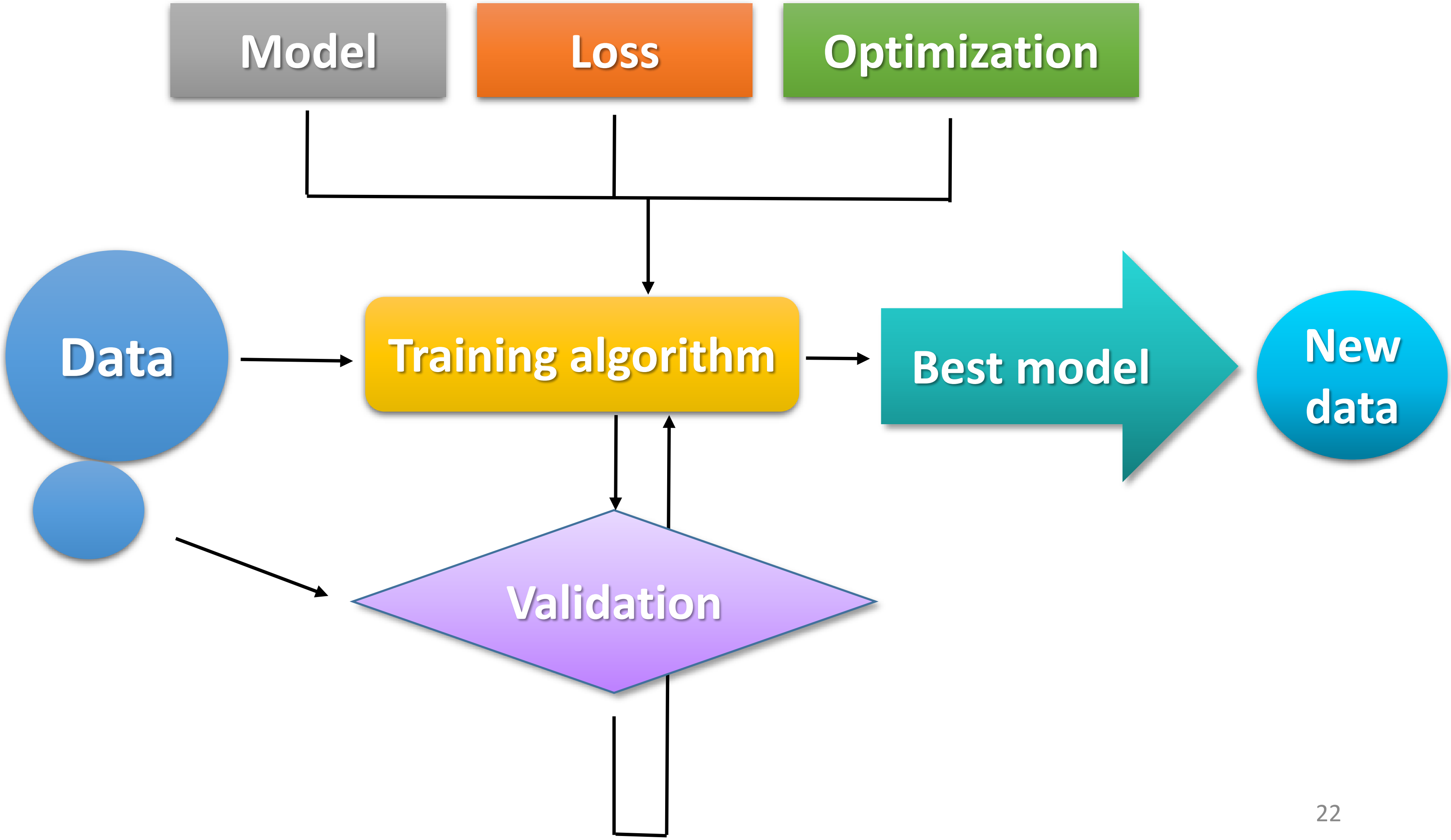




https://cs231n.github.io/neural-networks-3/
[arXiv:1609.04747v2]

# Concepts

- In summary: machine learning workflow



# Supervised Learning

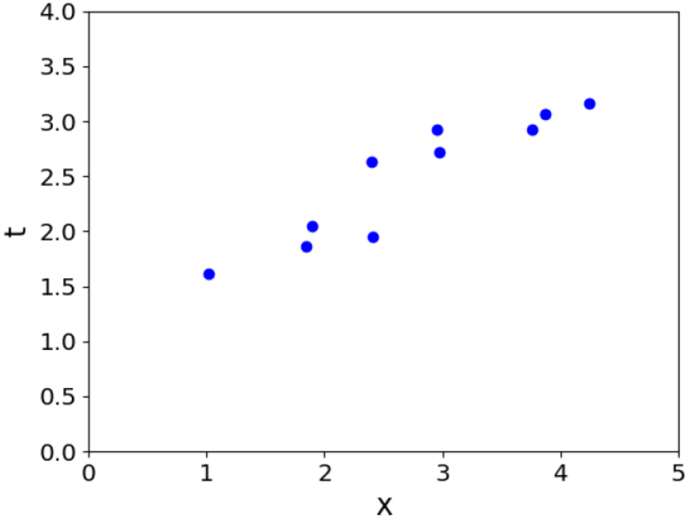
# Supervised learning

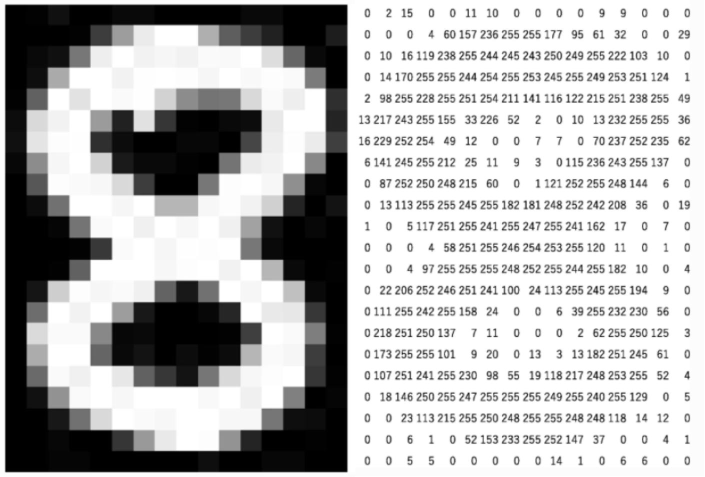
- The training set consists of inputs and corresponding labels.

- REGRESSION

1) The training set consists of a collection of pairs of an input vector **x** ∈ R$^d$ and its corresponding target, or label, t (note that **x** can be any input vector including songs, images, etc…). The output is a continuous or finite set.

$$\{(\mathbf{x}^{(1)}, t^{(1)}), \ldots, (\mathbf{x}^{(N)}, t^{(N)})\}$$





| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 15 | 0 | 0 | 11 | 10 | 0 | 0 | 0 | 0 | 9 | 9 | 0 | 0 | 0 |
| 0 | 0 | 0 | 4 | 60 | 157 | 236 | 255 | 255 | 177 | 95 | 61 | 32 | 0 | 0 | 29 |
| 0 | 10 | 16 | 119 | 238 | 255 | 244 | 245 | 243 | 250 | 249 | 255 | 222 | 103 | 10 | 0 |
| 0 | 14 | 170 | 255 | 255 | 244 | 254 | 255 | 253 | 245 | 255 | 249 | 253 | 251 | 124 | 1 |
| 2 | 98 | 255 | 228 | 255 | 251 | 254 | 211 | 141 | 116 | 122 | 215 | 251 | 238 | 255 | 49 |
| 13 | 217 | 243 | 255 | 155 | 33 | 226 | 52 | 2 | 0 | 10 | 13 | 232 | 255 | 255 | 36 |
| 16 | 229 | 252 | 254 | 49 | 12 | 0 | 0 | 7 | 7 | 0 | 70 | 237 | 252 | 235 | 62 |
| 6 | 141 | 245 | 255 | 212 | 25 | 11 | 9 | 3 | 0 | 115 | 236 | 243 | 255 | 137 | 0 |
| 0 | 87 | 252 | 250 | 248 | 215 | 60 | 0 | 1 | 121 | 252 | 255 | 248 | 144 | 6 | 0 |
| 0 | 13 | 113 | 255 | 255 | 245 | 255 | 182 | 181 | 248 | 252 | 242 | 208 | 36 | 0 | 19 |
| 1 | 0 | 5 | 117 | 251 | 255 | 241 | 255 | 247 | 255 | 241 | 162 | 17 | 0 | 7 | 0 |
| 0 | 0 | 0 | 4 | 58 | 251 | 255 | 246 | 254 | 253 | 255 | 120 | 11 | 0 | 1 | 0 |
| 0 | 0 | 4 | 97 | 255 | 255 | 255 | 248 | 252 | 255 | 244 | 255 | 182 | 10 | 0 | 4 |
| 0 | 22 | 206 | 252 | 246 | 251 | 241 | 100 | 24 | 113 | 255 | 245 | 255 | 194 | 9 | 0 |
| 0 | 111 | 255 | 242 | 255 | 158 | 24 | 0 | 0 | 6 | 39 | 255 | 232 | 230 | 56 | 0 |
| 0 | 218 | 251 | 250 | 137 | 7 | 11 | 0 | 0 | 0 | 2 | 62 | 255 | 250 | 125 | 3 |
| 0 | 173 | 255 | 255 | 101 | 9 | 20 | 0 | 13 | 3 | 13 | 182 | 251 | 245 | 61 | 0 |
| 0 | 107 | 251 | 241 | 255 | 230 | 98 | 55 | 19 | 118 | 217 | 248 | 253 | 255 | 52 | 4 |
| 0 | 18 | 146 | 250 | 255 | 247 | 255 | 255 | 255 | 249 | 255 | 240 | 255 | 129 | 0 | 5 |
| 0 | 0 | 23 | 113 | 215 | 255 | 250 | 248 | 255 | 255 | 248 | 248 | 118 | 14 | 12 | 0 |
| 0 | 0 | 6 | 1 | 0 | 52 | 153 | 233 | 255 | 252 | 147 | 37 | 0 | 0 | 4 | 1 |
| 0 | 0 | 5 | 5 | 0 | 0 | 0 | 0 | 0 | 14 | 1 | 0 | 6 | 6 | 0 | 0 |

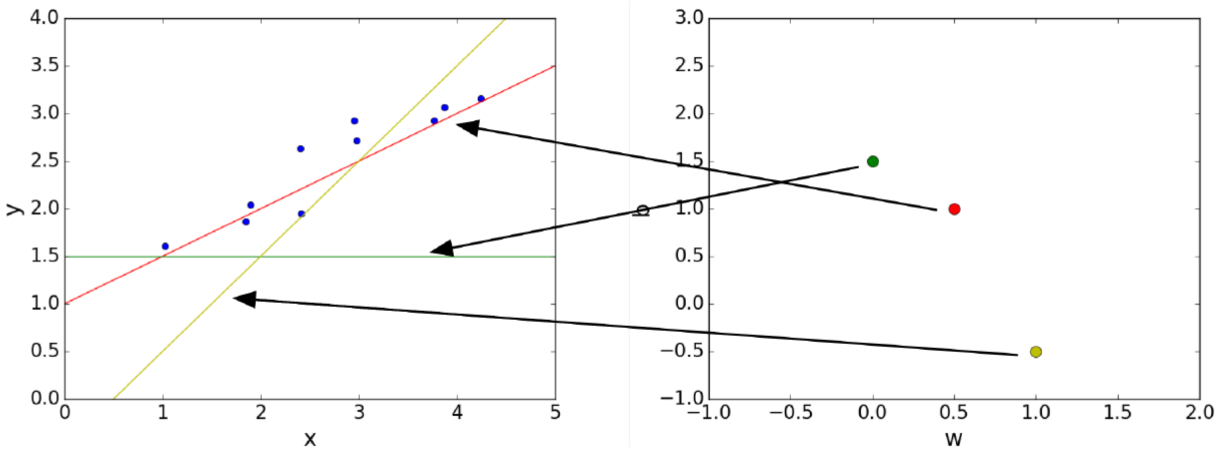
2) Our model can be of the type $\boxed{y = wx + b}$

w ≡ weight
b ≡ bias

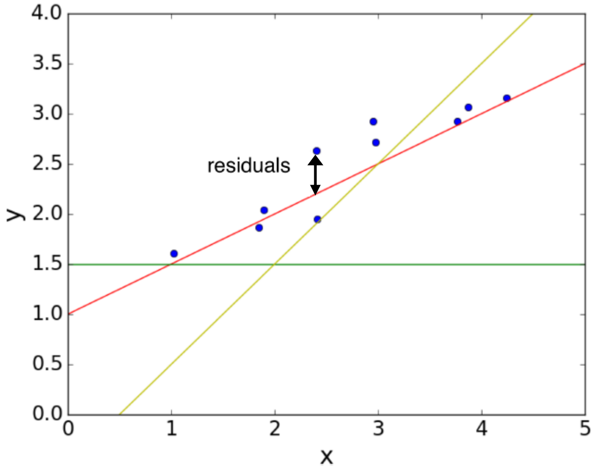
Parameters of the model

# Supervised learning

3) We define a space of weights and bias, which give us *y* predictions



4) We aim to find the best model parameters by minimizing the lost function (or cost function ≡ averaged lost function over all training samples)



Using a MSE (Minimum Squared Error) $\mathcal{L}(y, t)$:

$$\mathcal{J}(w, b) = \frac{1}{2N} \sum_{i=1}^{N} \left( y^{(i)} - t^{(i)} \right)^2 =$$

$$\frac{1}{2N} \sum_{i=1}^{N} \left( wx^{(i)} + b - t^{(i)} \right)^2$$

# Supervised learning

5) Since we usually have an input vector with multiple variables, we can use vectorial notation:

$$\mathbf{y} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix} = \begin{pmatrix} \mathbf{w}^\top \mathbf{x}^{(1)} + b \\ \vdots \\ \mathbf{w}^\top \mathbf{x}^{(N)} + b \end{pmatrix} = \mathbf{X}\mathbf{w} + b\mathbf{1}$$

With the cost function:

$$\mathcal{J} = \frac{1}{2N} \|\mathbf{y} - \mathbf{t}\|^2$$

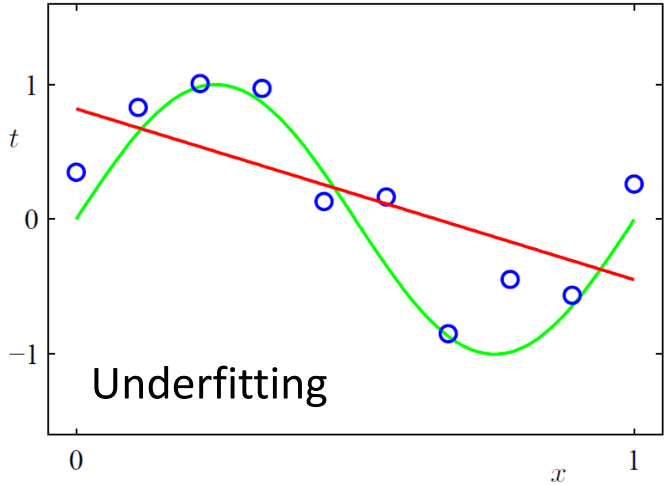
6) We minimize the gradient $\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_D} \end{pmatrix}$ through the learning rate $\eta$ to extract the model parameters.

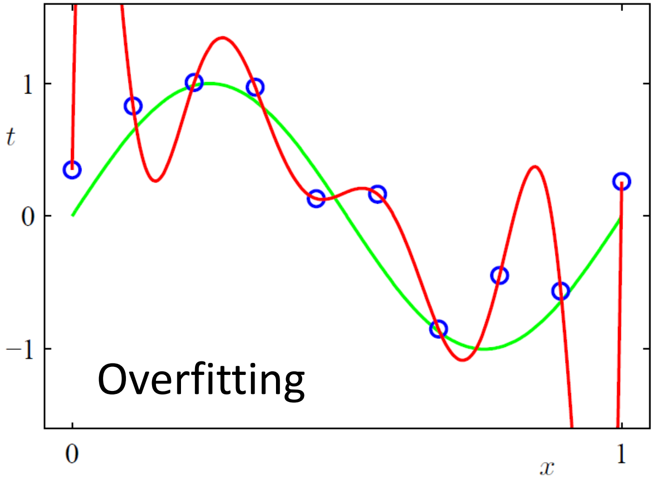
(over all data sample)

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \mathbf{w} - \frac{\eta}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) \, \mathbf{x}^{(i)}$$

# Supervised learning

7) Now we try with a new data sample … it is not just a question of minimization, but to choose the best weights and the best hyperparameters of the model.





**Regularization:** a penalty term, depending on the weights, which helps to find the better model description (it aims to keep small the squared norm of weights):

$$\mathcal{R}(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{w}\|^2 = \frac{1}{2}\sum_j w_j^2$$

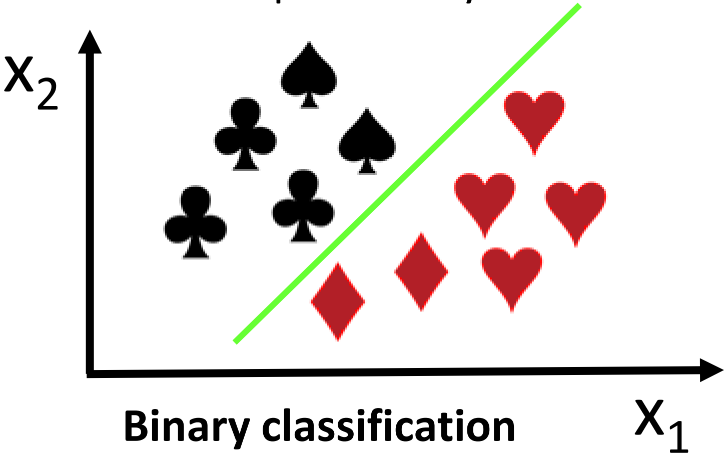
8) The cost function becomes:
( λ is an hyperparameter,
to be tuned)

$$\boxed{\mathcal{J}_{\text{reg}} = \mathcal{J} + \lambda\mathcal{R} = \mathcal{J} + \frac{\lambda}{2}\sum_j w_j^2}$$

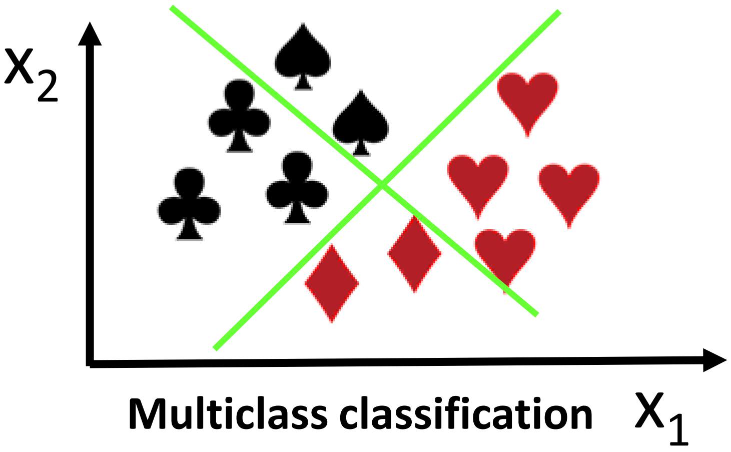
# Supervised learning

- CLASSIFICATION

1) The training set consists of a collection of pairs of an input vector **x** ∈ $R^d$ and its corresponding target, or label, t (note that **x** can be any input vector including songs, images, etc…). The target variable is discrete (*classes*).
Classes are separated by *decision boundaries*.



**Binary classification**



**Multiclass classification**

**Multilabel classification**

**Imbalanced classification**

# Supervised learning

2) Classification models:

- **Logistic Regression**: A linear model for binary classification.

- **Decision Trees**: It splits the feature space into regions by making a series of decisions.

- **Gradient Boosting Machines (GBMs):** It builds decision trees sequentially.

- **Support Vector Machines (SVM)**: It finds the hyperplane that best separates the classes.

- **k-Nearest Neighbors (k-NN)**: A non-parametric method that classifies instances based on the majority class among the k-nearest neighbors.

- **Naive Bayes**: A probabilistic classifier based on applying Bayes' theorem with strong independence assumptions between the features.

- **Neural Networks**: Models inspired by the human brain, particularly powerful in handling complex classification tasks.

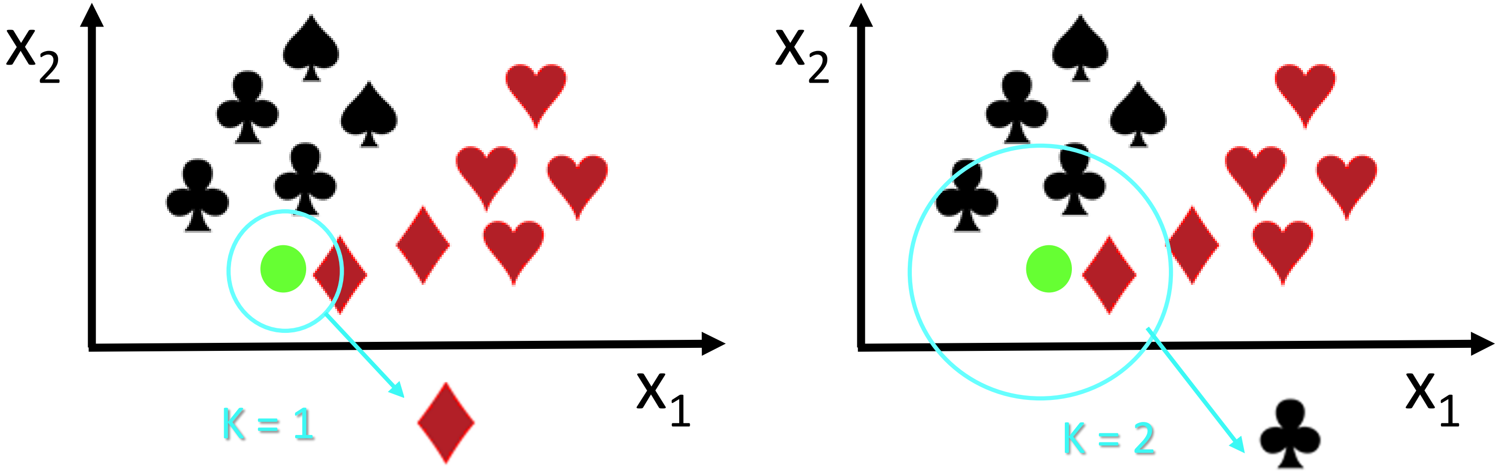
- **Random Forests**: An ensemble method using multiple decision trees to improve classification accuracy.

# Supervised learning

- ***K-Nearest Neighbors (NN):*** it is a non-parametric method:

Given a vector **x** to classify →
We need to find the nearest input vector to **x** in the training set and copy its label.

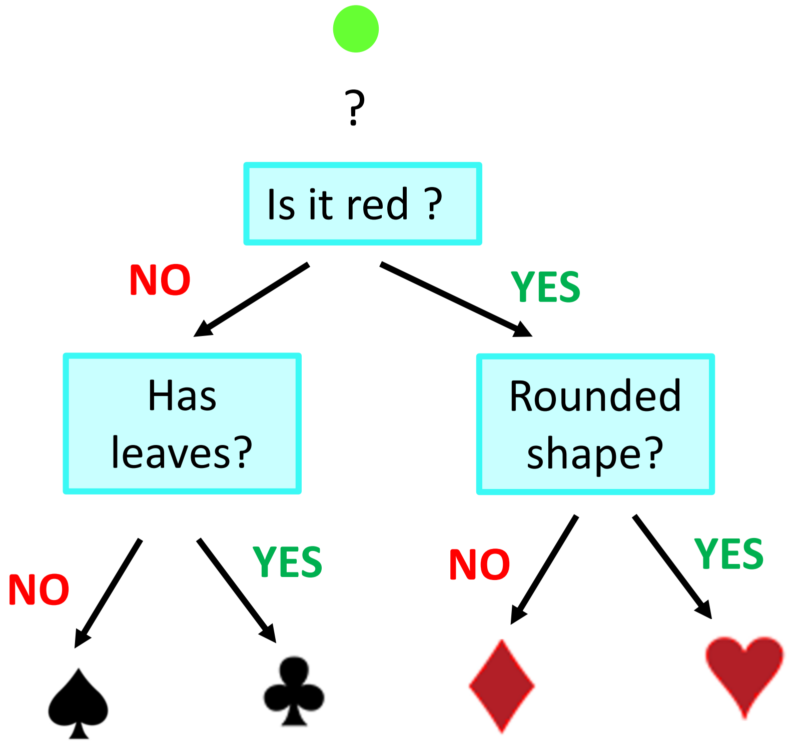
$$||\mathbf{x}^{(a)} - \mathbf{x}^{(b)}|| = \sqrt{\sum_{j=1}^{d} (x_j^{(a)} - x_j^{(b)})^2}$$



→ Small k gives fine tuning but can cause overfitting
→ Large k implies coarse tuning but can led to underfitting

# Supervised learning

- **Decision tree models:** make predictions by recursively splitting on different attributes according to a tree structure.



?

Is it red ?

NO

YES

Has leaves?

Rounded shape?

NO

YES

NO

YES

- Internal *nodes* test the attributes
- Attribute values are separated by *branches*
- *Leaf* nodes are output values

★ One can combine multiple models or decision trees in an ***ensemble*.**

***Bootstrapping:*** creation of multiple data subsets by randomly sampling with replacements from the original dataset.

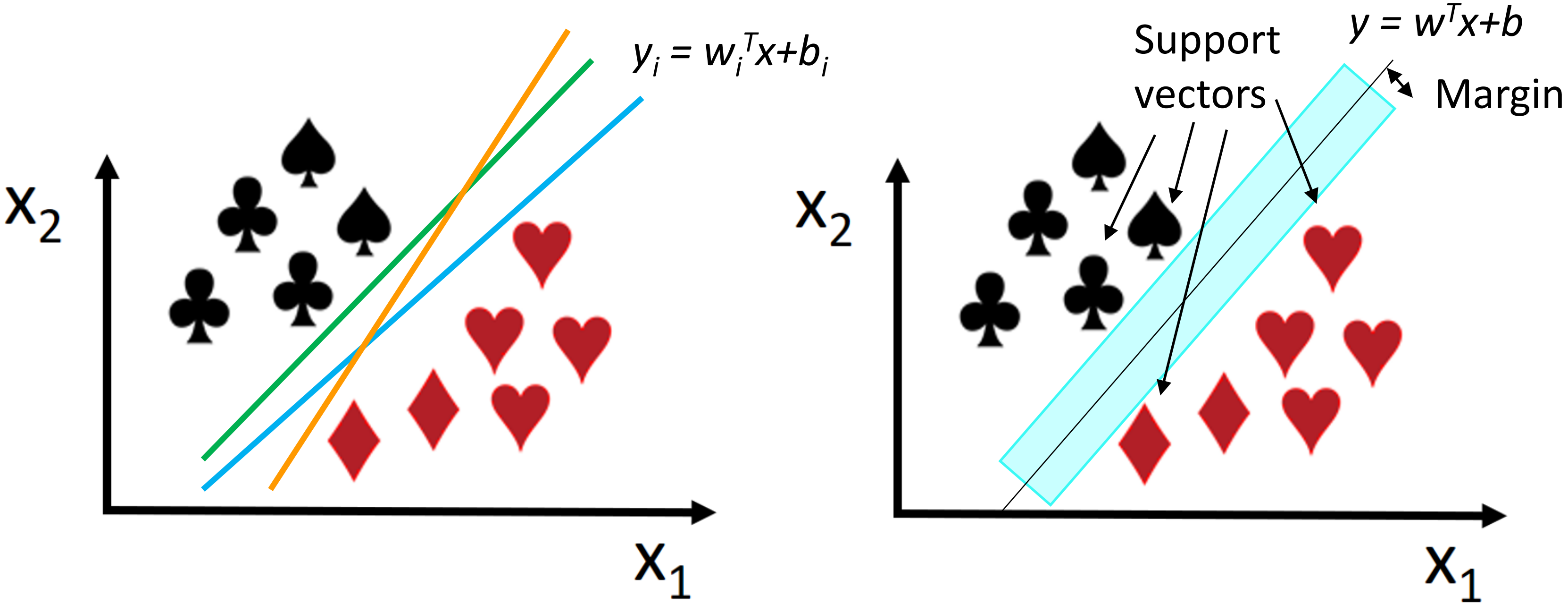
***Random Splits:*** division of data randomly into different parts or subsets.

***Bagging (Bootstrap Aggregating):*** multiple models are trained independently on different bootstrapped data subsets.

***Random forests: bagging + random splits***

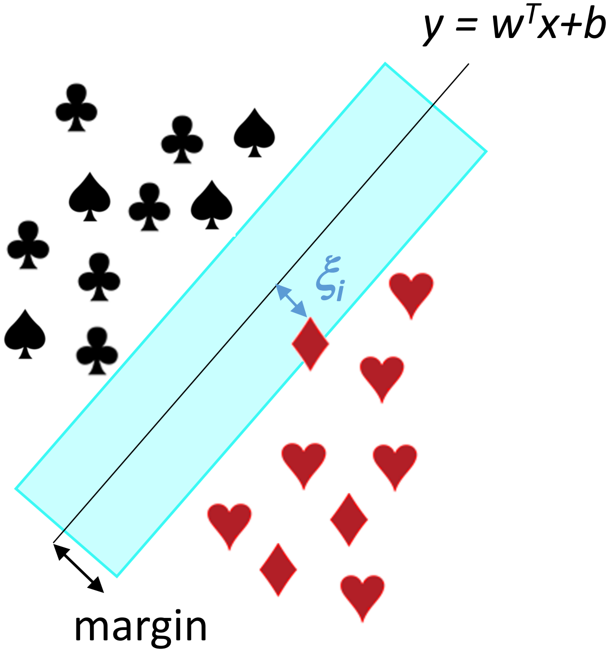
# Supervised learning

- **Support Vector Machines (SVM):** The idea is to find the best hyperplane that separates two classes by maximizing the distance to the closest point from either class, i.e., maximize the margin of the classifier.



**Hyperplane:** decision boundaries aiming to classifying the data points.
**Support Vectors:** nearest data points to the hyperplane.
**Margin:** the gap between the hyperplane and the support vectors.
**Kernel function:** functions used to determine the shape of the hyperplane and decision boundary.

# Supervised learning



$$C = \frac{1}{\|\mathbf{w}\|}$$

The hyperparameter C adjusts the margin:

→ a large C value narrows the margin for minimal misclassification

→ a small C value widens it, allowing for more misclassified data

One can use the *slack variable* $\xi$, including off-margin points and letting the classification to be more flexible.

$$\frac{t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|} \geq C(1 - \xi_i)$$

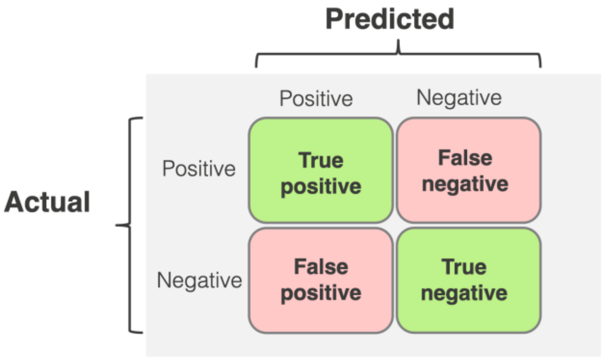
One can then constrain or penalize the total amount of slack.

Penalty term : $\sum_{i=1}^{N} \xi_i = \sum_{i=1}^{N} \max\{0, 1 - t^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + b)\}$ (Hinge loss)

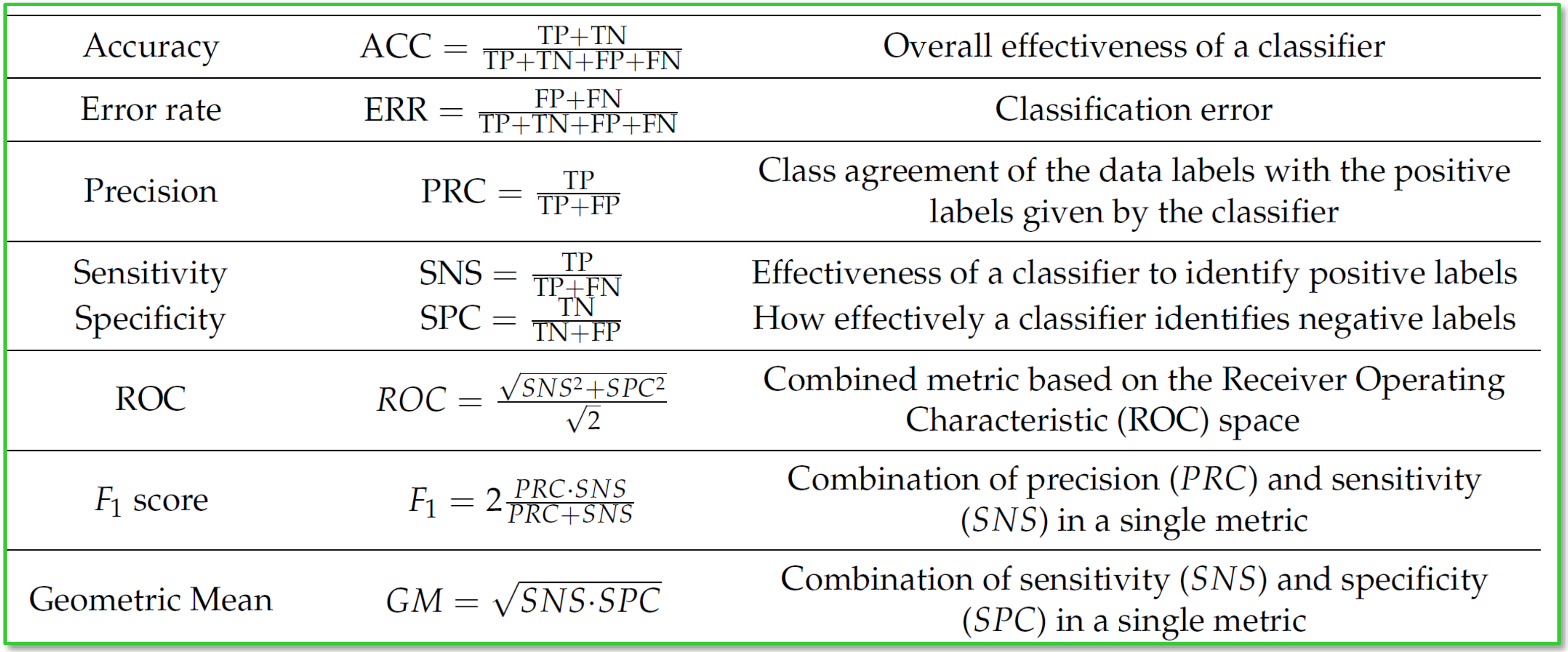
# Supervised learning

- <u>Metrics for Classification:</u>

Several figures of merit are defined according to the *confusion matrix*



|  |  | Predicted |  |
|---|---|---|---|
|  |  | Positive | Negative |
| Actual | Positive | True positive | False negative |
|  | Negative | False positive | True negative |



|  |  |  |
|---|---|---|
| Accuracy | $\mathrm{ACC} = \frac{\mathrm{TP+TN}}{\mathrm{TP+TN+FP+FN}}$ | Overall effectiveness of a classifier |
| Error rate | $\mathrm{ERR} = \frac{\mathrm{FP+FN}}{\mathrm{TP+TN+FP+FN}}$ | Classification error |
| Precision | $\mathrm{PRC} = \frac{\mathrm{TP}}{\mathrm{TP+FP}}$ | Class agreement of the data labels with the positive labels given by the classifier |
| Sensitivity<br>Specificity | $\mathrm{SNS} = \frac{\mathrm{TP}}{\mathrm{TP+FN}}$<br>$\mathrm{SPC} = \frac{\mathrm{TN}}{\mathrm{TN+FP}}$ | Effectiveness of a classifier to identify positive labels<br>How effectively a classifier identifies negative labels |
| ROC | $ROC = \frac{\sqrt{SNS^2+SPC^2}}{\sqrt{2}}$ | Combined metric based on the Receiver Operating Characteristic (ROC) space |
| $F_1$ score | $F_1 = 2\frac{PRC \cdot SNS}{PRC+SNS}$ | Combination of precision ($PRC$) and sensitivity ($SNS$) in a single metric |
| Geometric Mean | $GM = \sqrt{SNS \cdot SPC}$ | Combination of sensitivity ($SNS$) and specificity ($SPC$) in a single metric |

[DOI: 10.3390/s18061803]

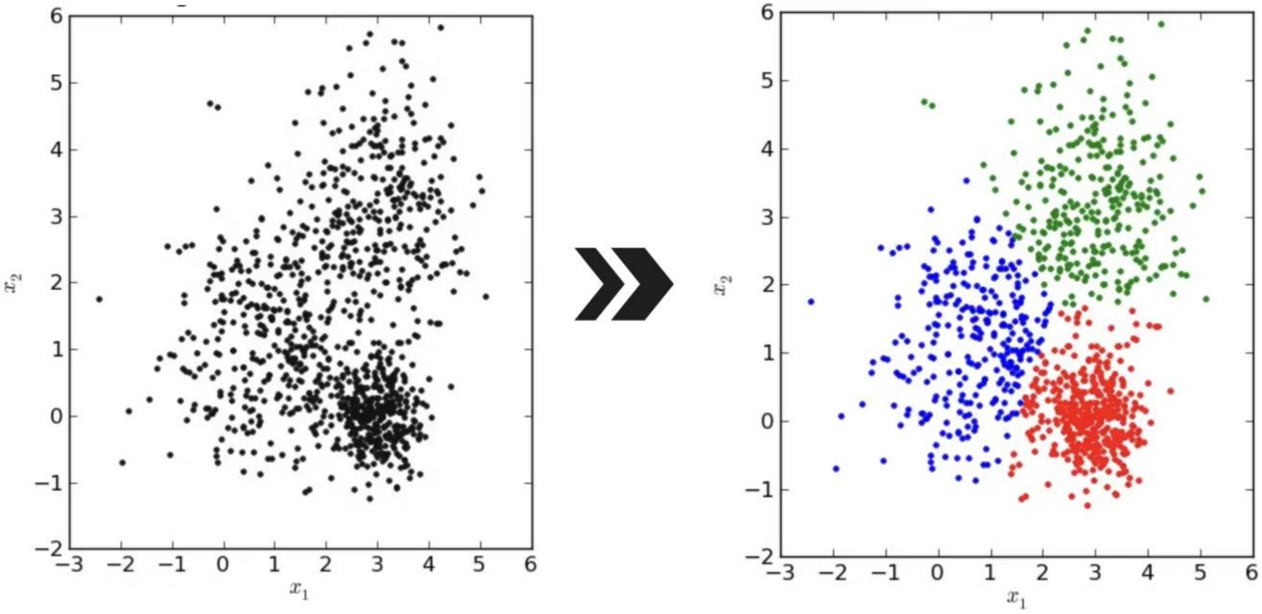
# Unsupervised Learning

# Unsupervised learning

- The training sample is not labelled, the model has to find the latent structure underlying the data.

- CLUSTERING

Meaningful grouping of data according to their similarities.

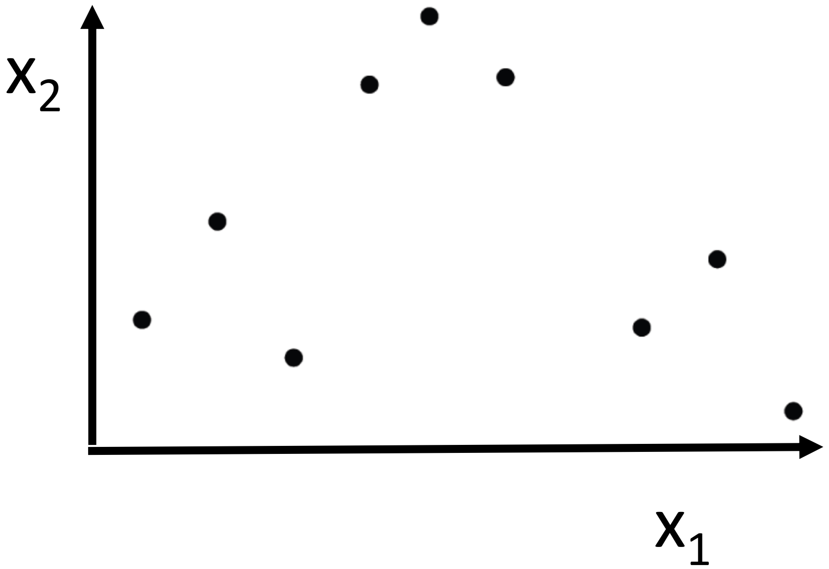


The goal: to find  the clusters that minimize or maximize an *objective function*.

# Unsupervised learning

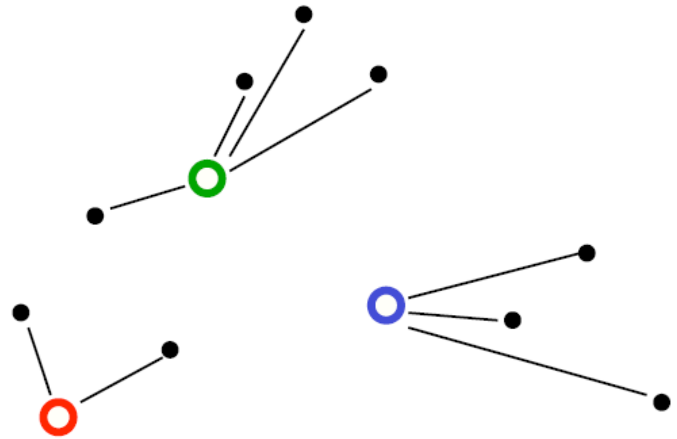
**K-means algorithm:**

The data belongs to K classes or patterns, in the way that the variance within them is as small as possible.

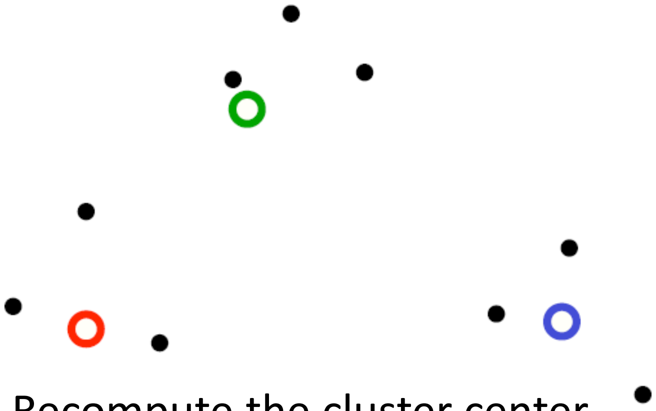


We have to find cluster centers (m) or *centroid* and make assignments ($r_K$) for each data point $x^{(n)}$

$$r_k^{(n)} \in \{0,1\} \qquad \sum_k r_k^{(n)} = 1$$



1- assign an initial cluster to each data

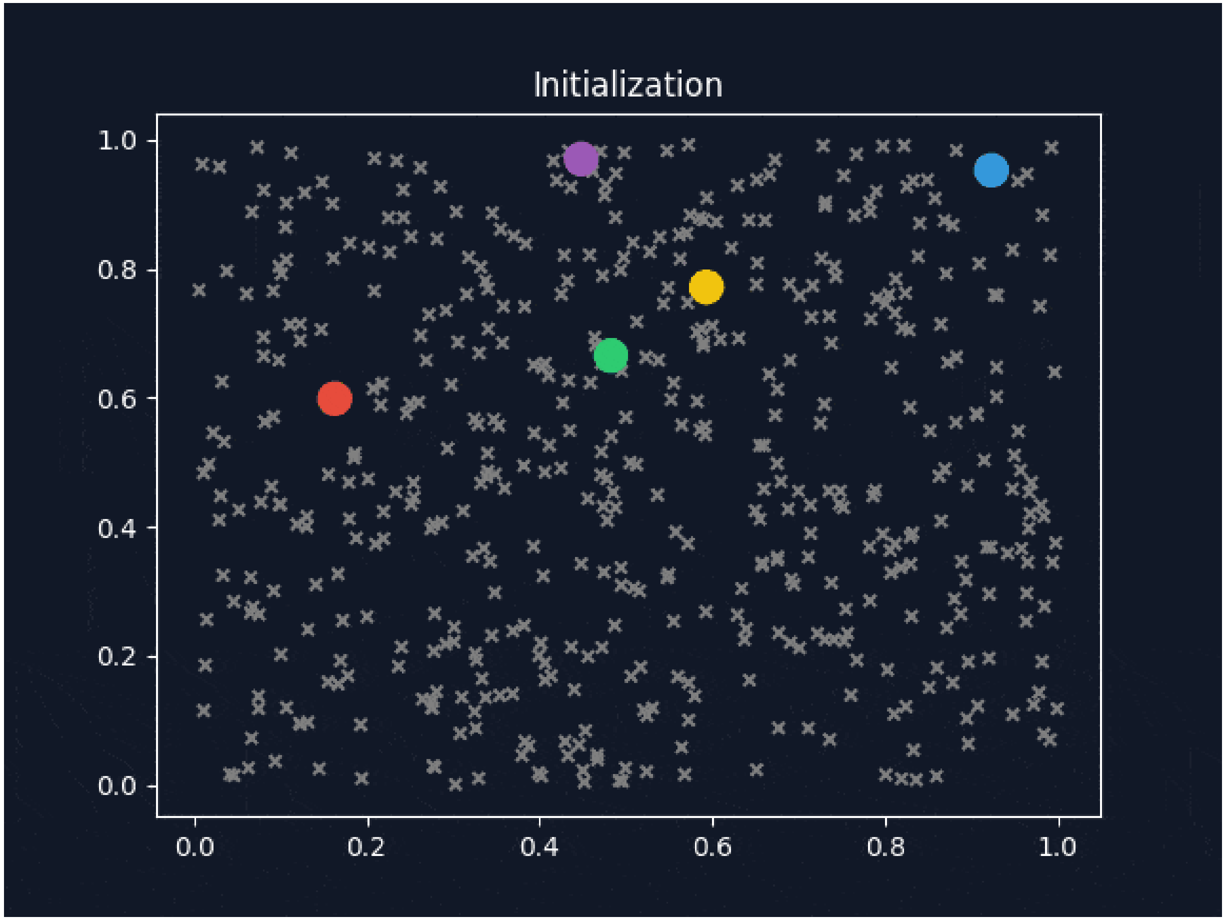


2- Recompute the cluster center according to the gravity center

# Unsupervised learning

**K-means algorithm:**

Optimization: $$\min_{\{\mathbf{m}\},\{\mathbf{r}\}} J(\{\mathbf{m}\},\{\mathbf{r}\}) = \min_{\{\mathbf{m}\},\{\mathbf{r}\}} \sum_{n=1}^{N} \sum_{k=1}^{K} r_k^{(n)} ||\mathbf{m}_k - \mathbf{x}^{(n)}||^2$$

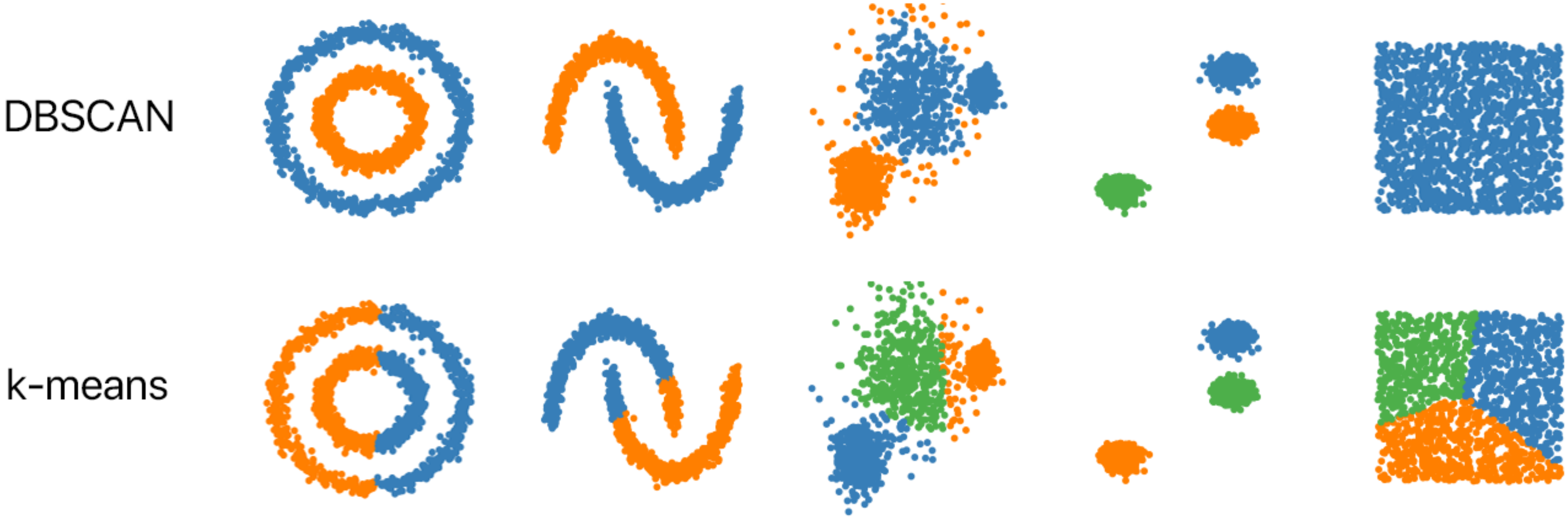


https://code-specialist.com/python/k-means-algorithm

# Unsupervised learning

**DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**

It groups data points based on their density. It allow to identify outliers/noise points that do not fit any cluster (good for anomaly detection).

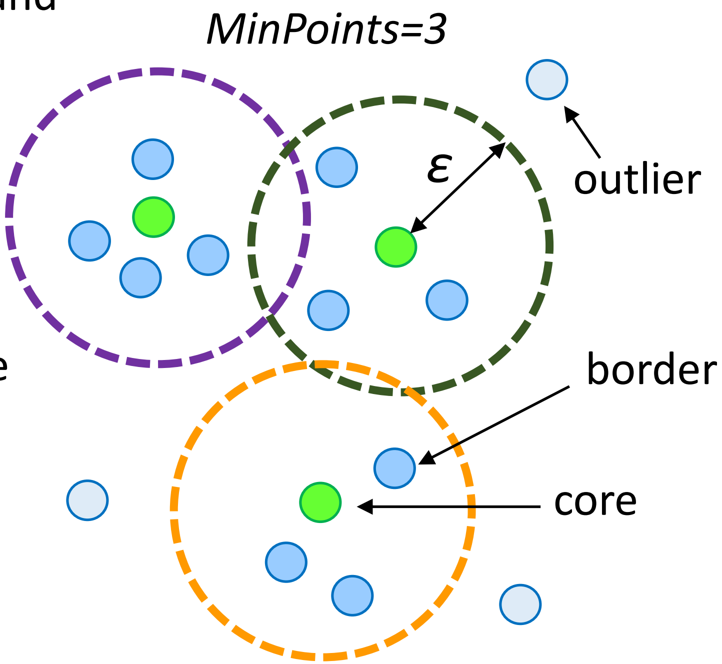


https://github.com/NSHipster/DBSCAN

As compare to k-means, DBSCAN can discover clusters of any arbitrary shape.

# Unsupervised learning

- Data points are searched for into dense regions and separated by not-so-dense regions.



•***Core Point:*** At least *MinPts* points in its neighborhood, defined by a radius *ε*.

•***Border Point:*** Its neighborhood contains less than *MinPts* data points, or it is within *ε*-distance from a core point.

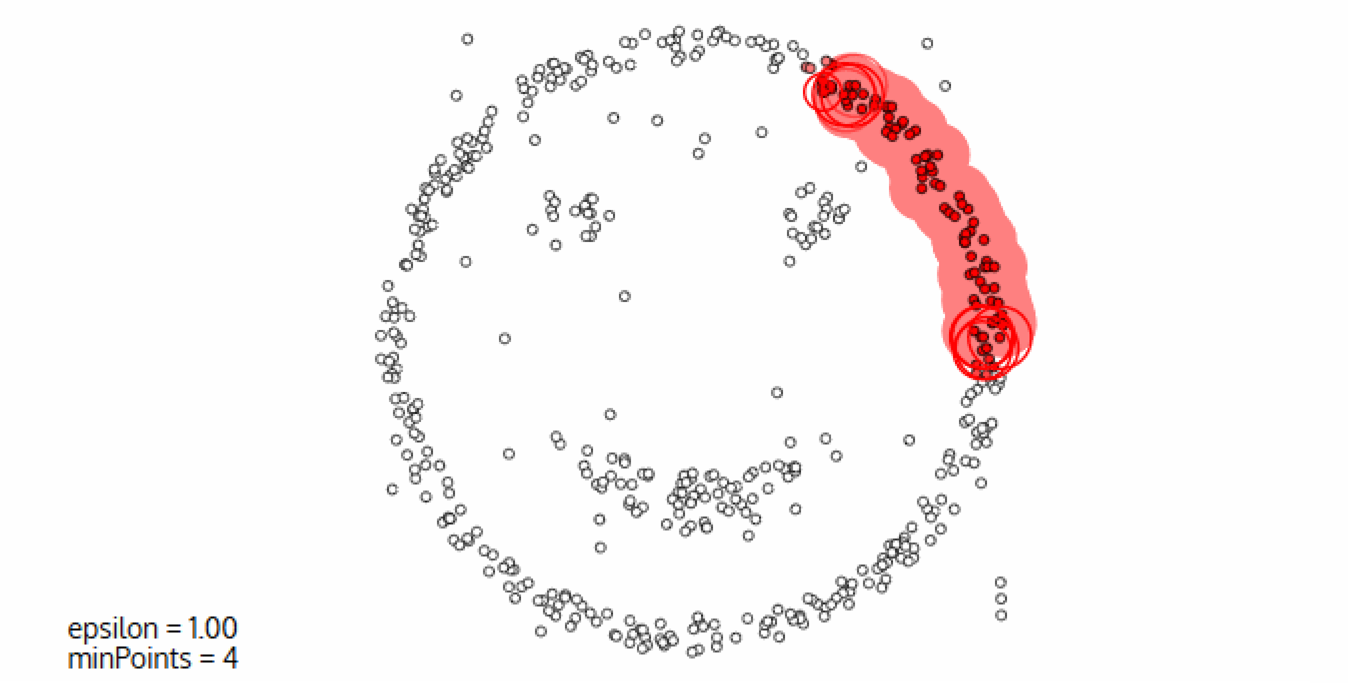
•***Outlier Point:*** It is not a core point, and is not close enough to be reachable from a core point.

Algorithm procedure:

1) Arbitrarily pick up a point in the dataset (until all points have been visited).
2) If there are at least *MinPts* points within a radius (*ε)* to the point then we consider all these points to be part of the same cluster.
3) The clusters are then expanded by recursively repeating the neighborhood calculation for each neighboring point.

# Unsupervised learning

- Grouping is usually made by using the Euclidean $\mathrm{distance}(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$

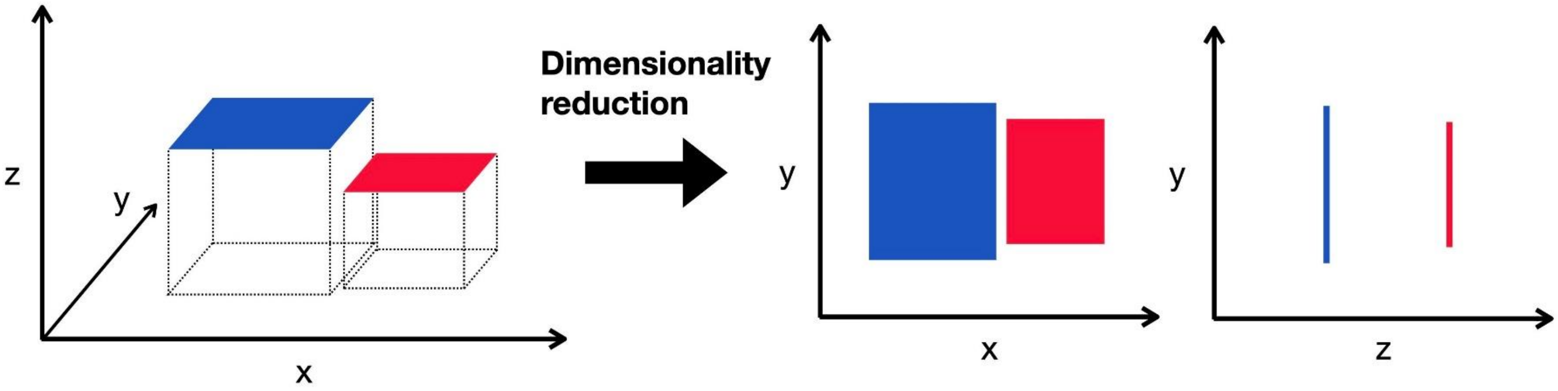


https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html

# Unsupervised learning

## ● DIMENSIONALITY REDUCTION

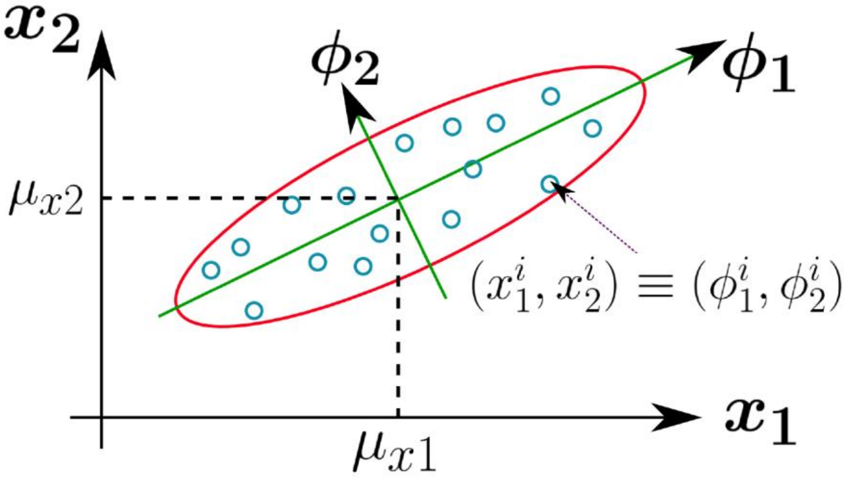
→ The idea is to reduce the number of features or variables in a dataset while retaining as much important information as possible. The goal of dimensionality reduction is to simplify the data while preserving the relevant information.



**Principal Component Analysis (PCA)**

Using the input variables and their covariance matrix, a new set of uncorrelated variables called **principal components** (PCs) is created via an orthogonal transformation of the original dataset → aiming for an easier learning and visualization.

# Unsupervised learning



1) Using all the data points we find the mean values of the variables ($\mu_{x1}$; $\mu_{x2}$) and the covariance matrix $\Sigma$

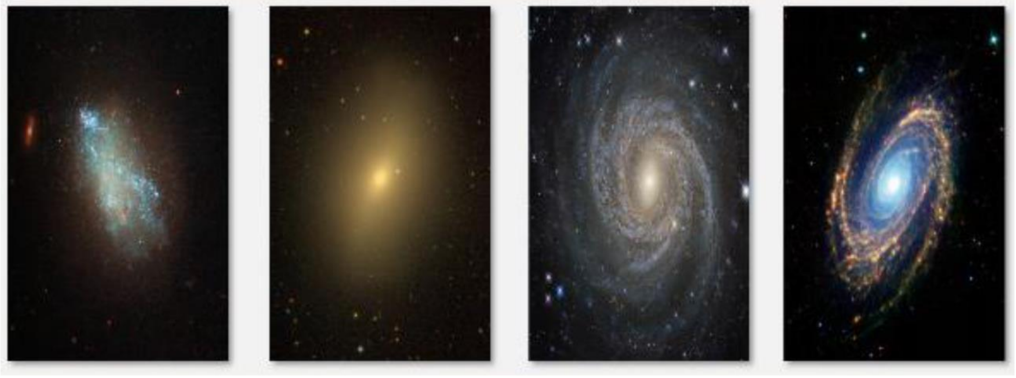
2) We calculate the eigenvectors of the coariance matrix, and get the direction vectors ($\phi_1$ and $\phi_2$).

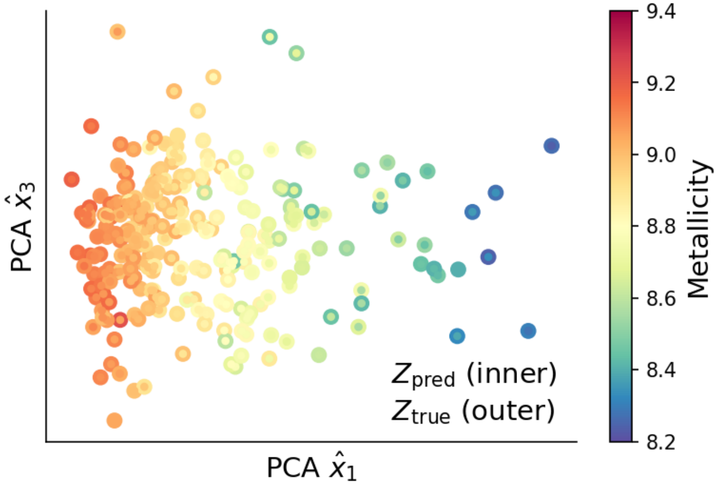
3) We create a transformation matrix of the type $\boldsymbol{p}_\phi = (\boldsymbol{p}_x - \boldsymbol{\mu}_x)\,\Phi$

4) The Principal Components are the k eigenvectors with the highest values.

[Ref]

Ex: 256 galaxies described 512-dimensional feature vectors





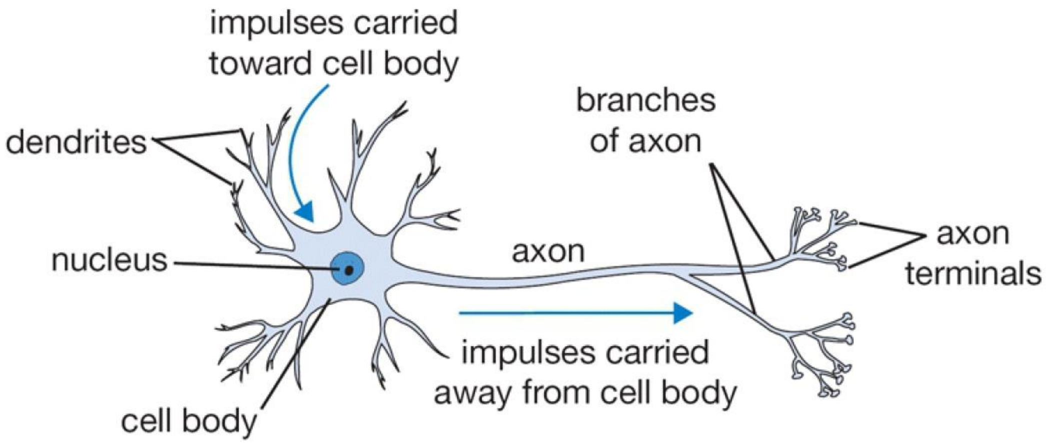
# Neural Networks

# Neural Networks

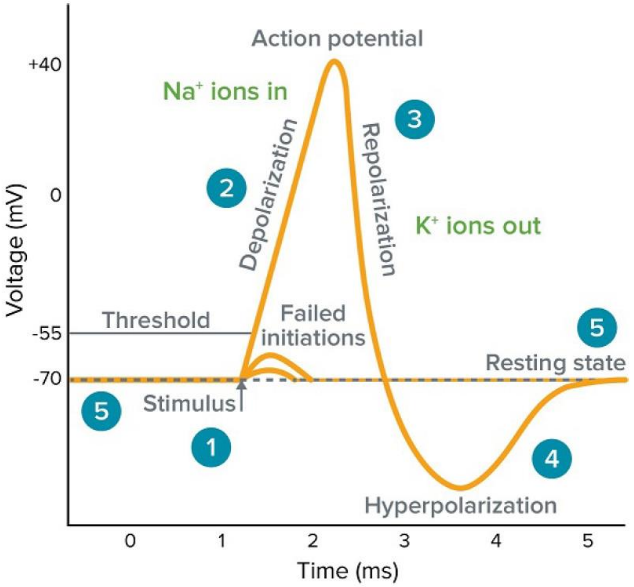
***Biological inspiration:***

There are approximately *100* billion *neurons* in a mature human brain, each of them connected and communicating with other 10K neurons.

The basic computational unit in a neural network is the ***neuron***



impulses carried toward cell body
dendrites
branches of axon
axon
axon terminals
nucleus
impulses carried away from cell body
cell body



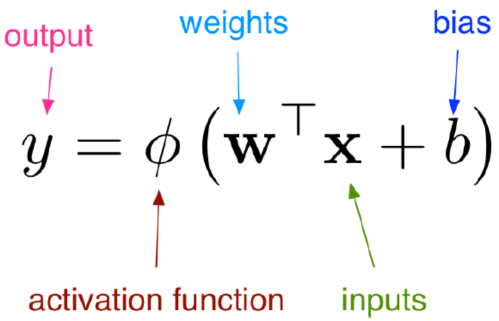
[www.moleculardevices.com]

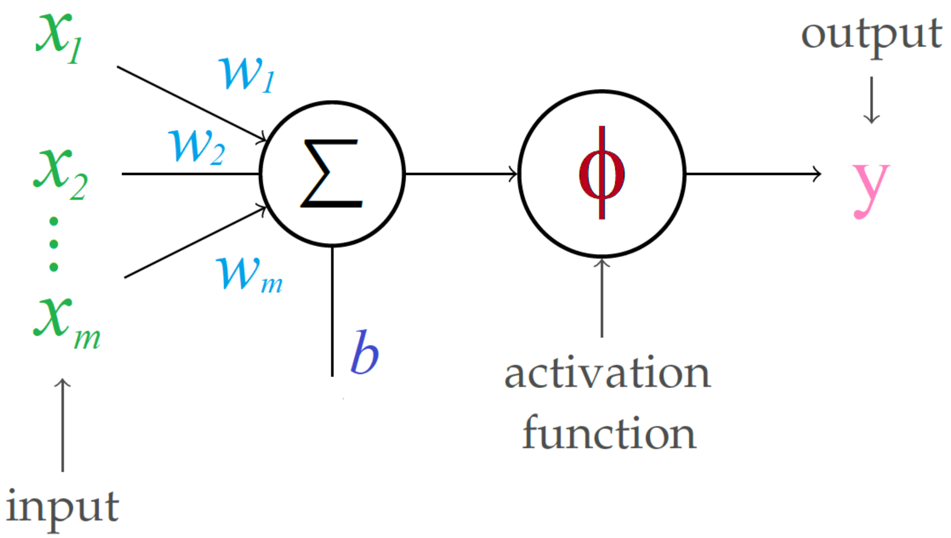
Neurons receive electric input signals and accumulate voltage, firing spiking responses after some threshold.

# Neural Networks

- PERCEPTRON

It is a mathematical model of a biological neuron. While in actual neurons the dendrite receives electrical signals from the axons of other neurons, in the perceptron these electrical signals are represented as numerical values.



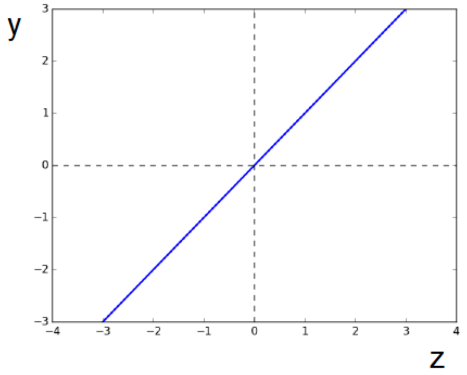


The ***activation function*** is responsible for making the nodes 'fire.' It processes the input signal (data features) through a mathematical transformation, checks if the value of the weighted sum of inputs crosses a threshold, and if it does, provides an output.

# Neural Networks

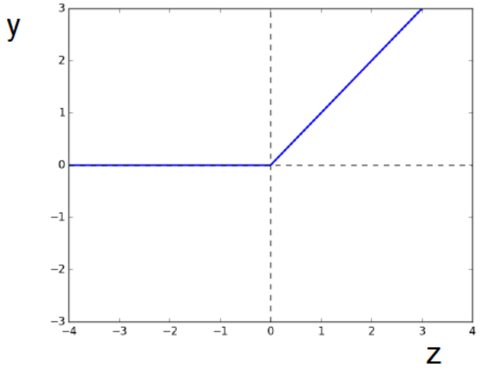
**Activation functions:**

## Linear



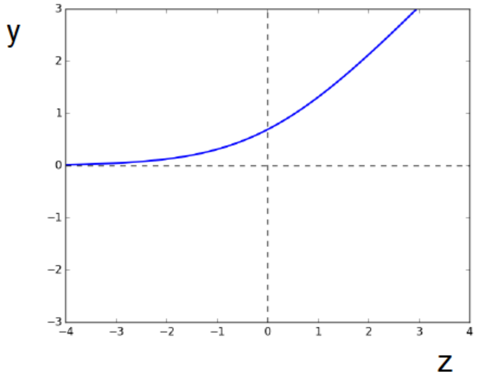
$$y = z$$

## Rectified Linear Unit (ReLU)



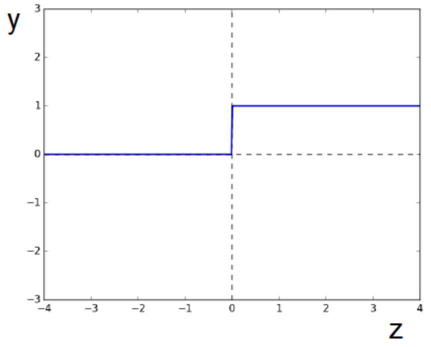
$$y = \max(0, z)$$

## Soft ReLU



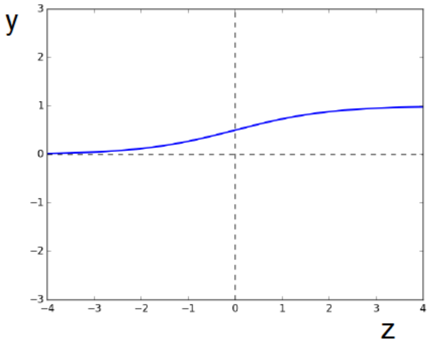
$$y = \log 1 + e^z$$

## Hard Threshold



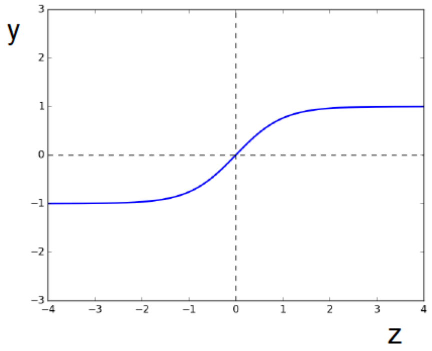
$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

## Logistic



$$y = \frac{1}{1 + e^{-z}}$$

## Hyperbolic Tangent (tanh)



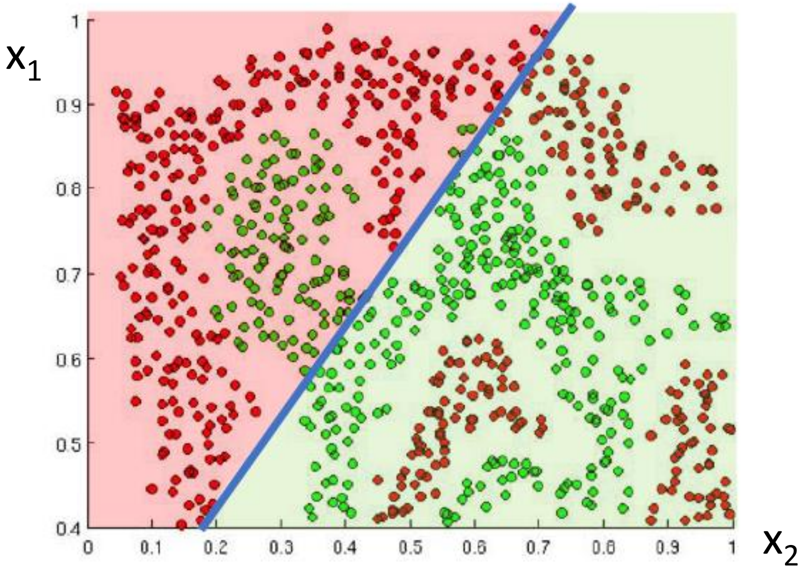
$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

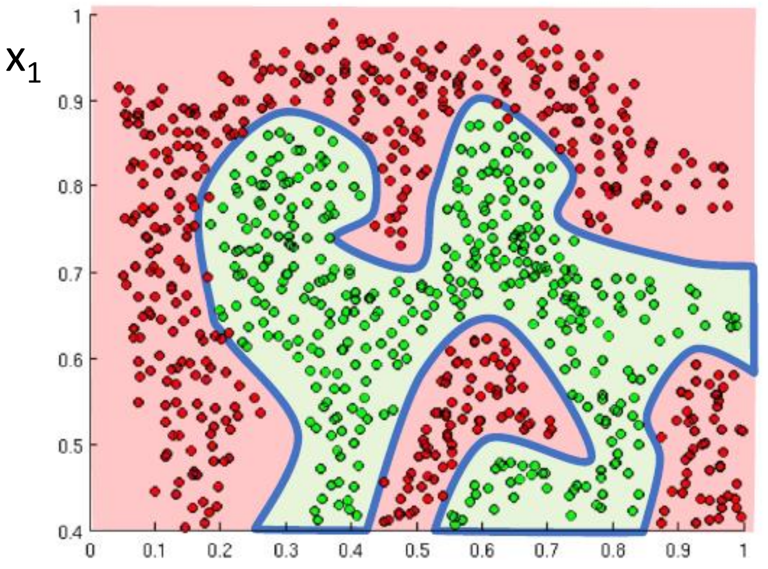
# Neural Networks

The purpose of the ***activation function*** is to introduce non-linearities, enabling the network to learn and model complex patterns:

Linear = coffee + sugar

Non-linear = yeast + dough





$x_2$

[A. Amini (MIT)]

# Neural Networks

The purpose of the ***activation function*** is to introduce non-linearities, enabling the network to learn and model complex patterns:

Linear = coffee + sugar

Non-linear = yeast + dough

- **Ridge Functions**: Linear transformations followed by non-linearity (ex: Sigmoid, ReLU).
- **Radial Functions**: Symmetric functions based on distance from a center.
- **Fold Functions**: Non-linear mappings with sharp transitions (ex: Step function).

They can be *saturating* or *non-saturating*

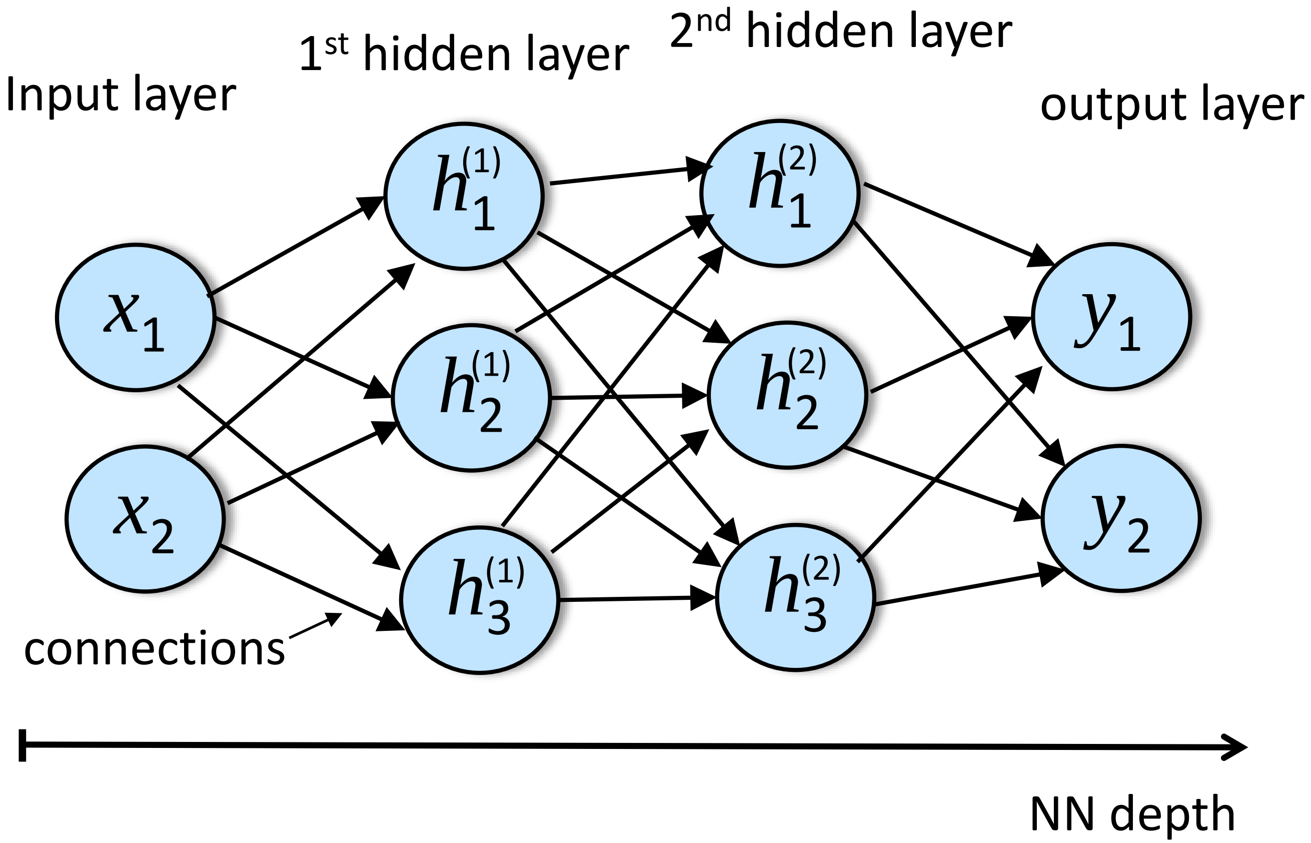
$$\lim_{x \to \pm\infty} \phi(x) = \text{finite value}$$

$$\lim_{x \to +\infty} \phi(x) = +\infty$$

$$\lim_{x \to -\infty} \phi(x) = -\infty \text{ or finite value}$$

# Neural Networks

- Neurons are grouped together into layers:



→ This gives a *feed-forward neural* network (FFNN).

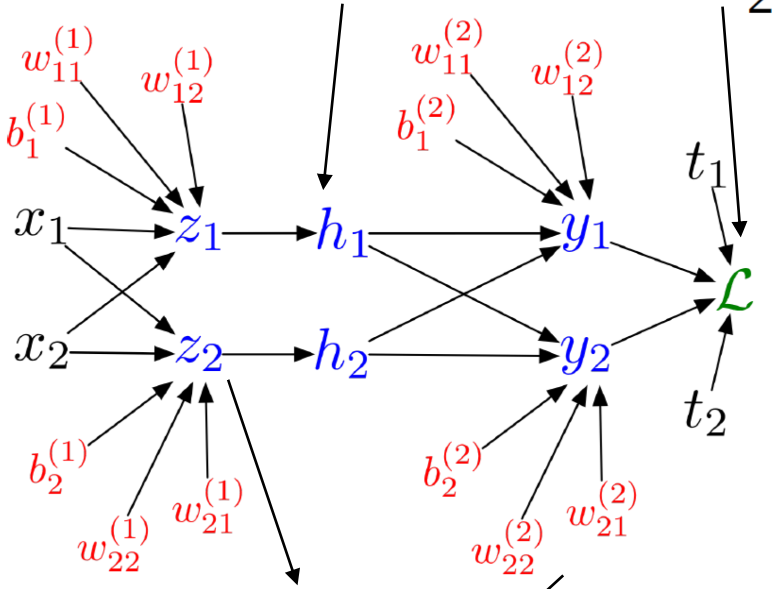
→ If all input units are connected to all output units: *fully connected FCNN* (*multilayer perceptron*)

# Neural Networks

- ***Backpropagation:*** in the training period try to find the network weights $w_i$ that achieve the lowest loss by using the gradient descent

$$h_i = \sigma(z_i)$$

$$\mathcal{L} = \frac{1}{2}\sum_k (y_k - t_k)^2$$



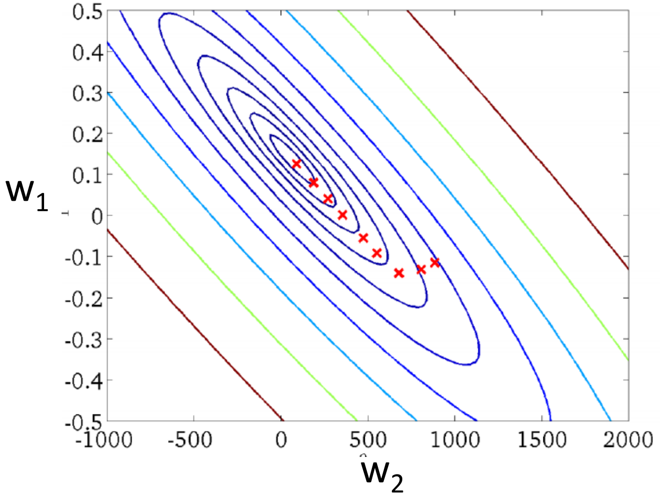
$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(2)}$$

★ **We want to minimize *d𝓛/dw***

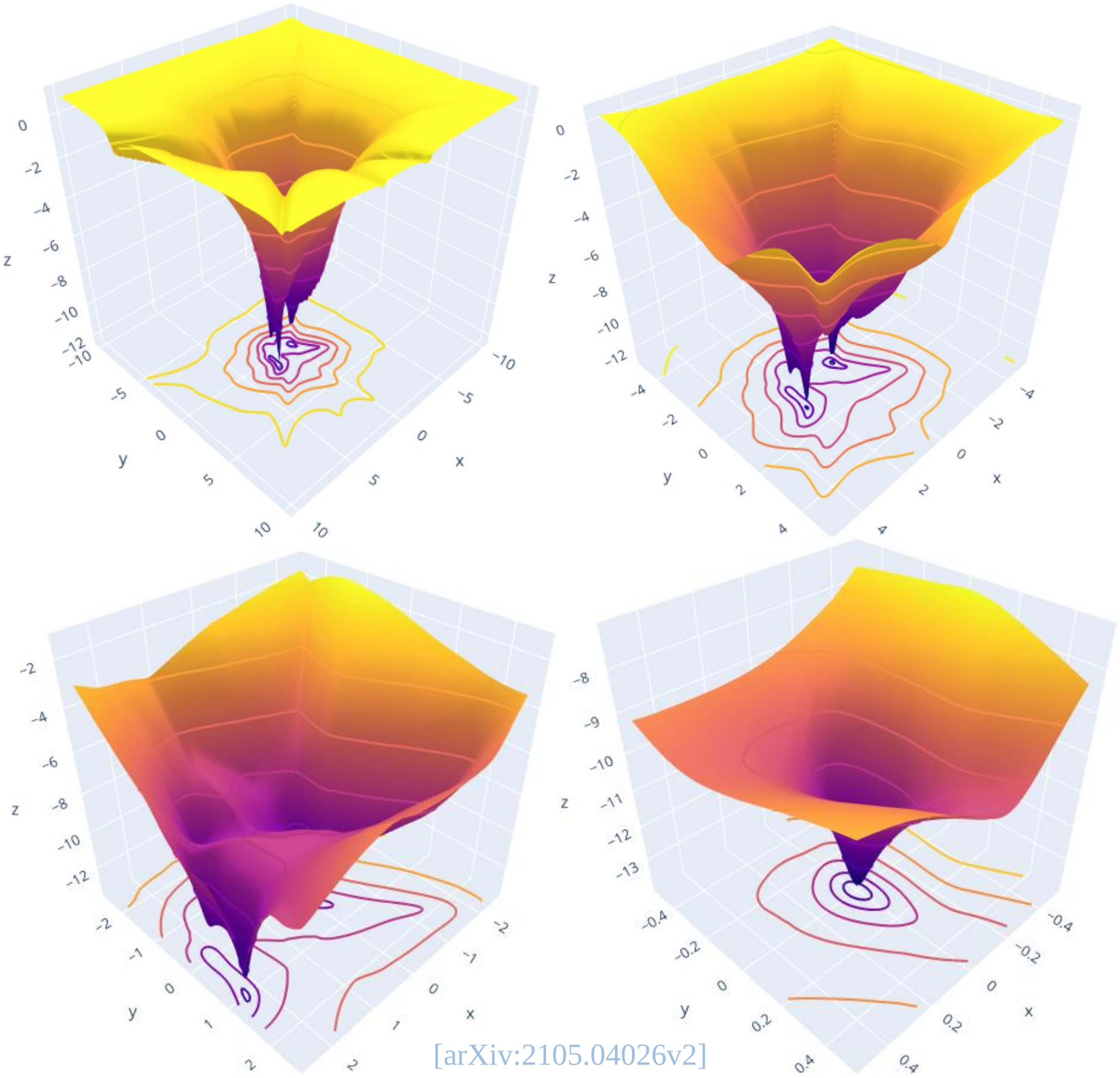
$$w_i^{\text{new}} = w_i^{\text{old}} - \eta \frac{\partial L}{\partial w_i}$$

$\eta \equiv$ learning rate



$$\boxed{\frac{\partial L}{\partial w_i} = (y_i - t_i) \cdot \frac{\partial h_i}{\partial z_i} \cdot x_i}$$

# Neural Networks



[arXiv:2105.04026v2]

Ex: 2D projection of the loss landscape of 4 layers-NN with a ReLU activation function

# Neural Networks

- ***Optimizers***: $w_t$ are the weights at time step t, $\eta$ the learning rate, $\nabla$ and $g$ represent the gradient, $\beta_1$ and $\beta_2$ are decay rate coefficients and $\varepsilon$ a protection parameter:

**Stochastic Gradient Descent (SGD)**

$$w_{t+1} = w_t - \eta \cdot \nabla_w J(w_t)$$

**Momentum**

$$v_t = \beta \cdot v_{t-1} + \\ + (1-\beta) \cdot \nabla_w J(w_t)$$

$$w_{t+1} = w_t - \eta \cdot v_t$$

**RMSProp**

$$E[g^2]_t = \beta \cdot E[g^2]_{t-1} + \\ + (1-\beta) \cdot g_t^2$$

$$w_{t+1} = \text{w\_t -} \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t$$

***ADAM***

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$$

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon} \cdot \hat{m}_t$$

***AMSGrad***

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$$

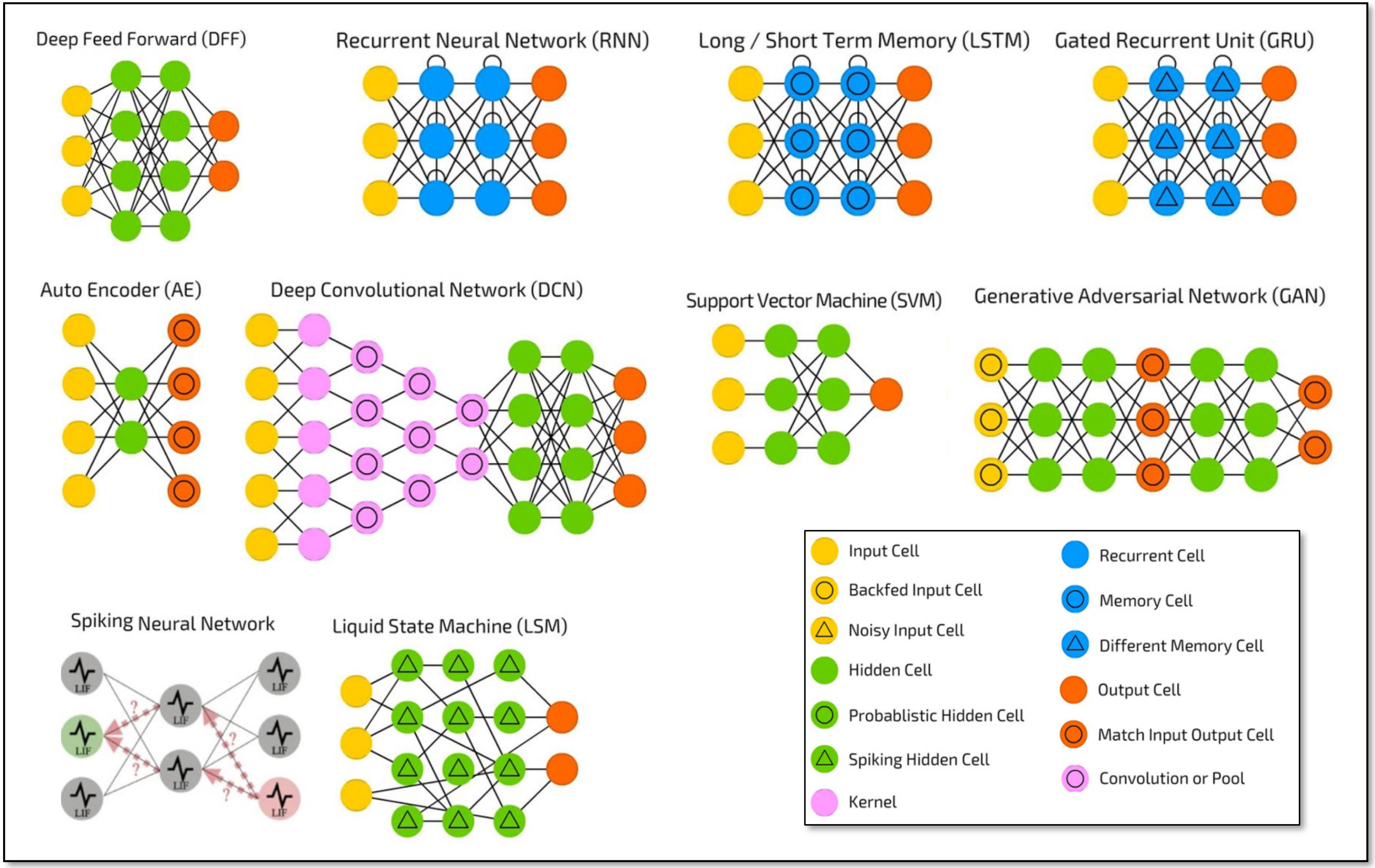
$$\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon} \cdot m_t$$

[arXiv:1904.09237]

# Neural Networks

- Types of neural networks:



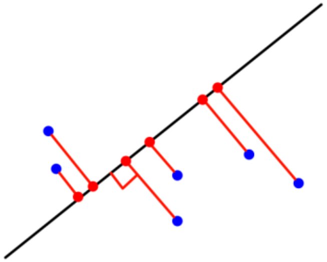
[10.20944/preprints202309.1149]

# Unsupervised learning + NN

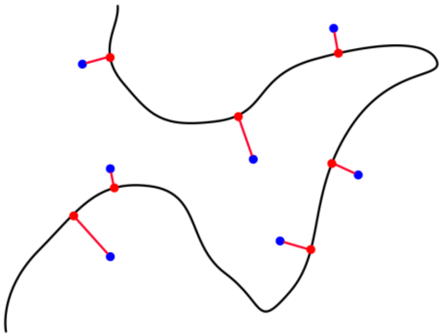
- AUTOENCODERS

A neural network architecture designed to efficiently compress (encode) input data down to its essential features, then reconstruct (decode) the original input from this compressed representation.

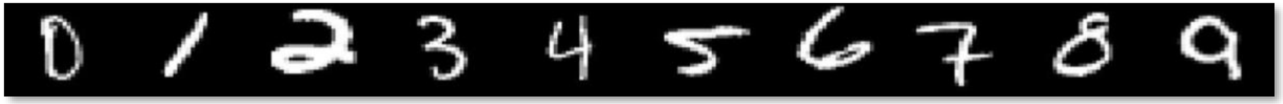
linear dimensional reduction



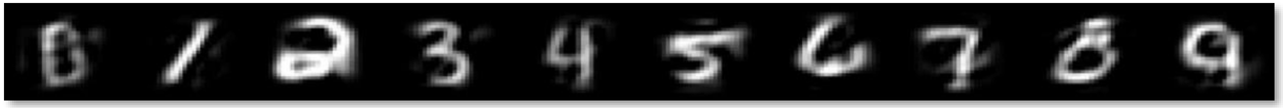
non-linear dimensional reduction



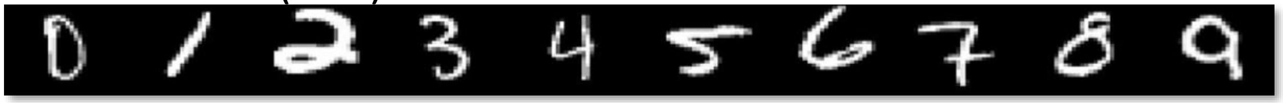
Real data



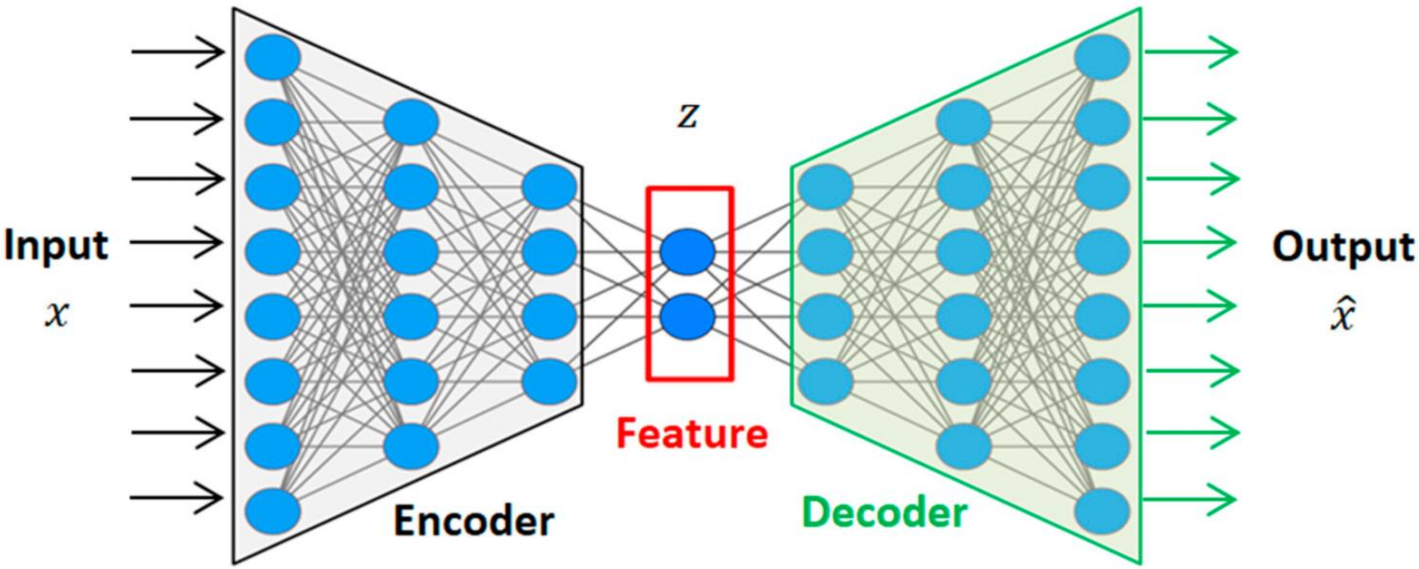
PCA (30D)



Autoencoder (30D)



# Unsupervised learning + NN



The learning objective is to minimize the difference between the original input data and the reconstructed data produced by the network.

data expectation

$$\min_{\theta} \mathcal{L}(x, \hat{x}) = \min_{\theta} \mathbb{E}_{x \sim p_{\text{data}}(x)} \left[ ||x - \hat{x}||^2 \right]$$

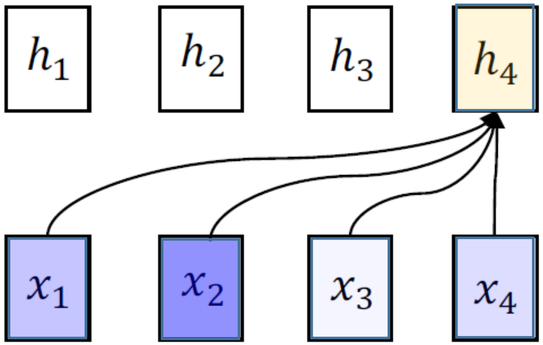
NN parameters ($w_i$,$b_i$)

$\hat{x} = D(E(x))$, with *D* and *E* the decoding and encoding functions

# Un/supervised learning + NN

- TRANSFORMERS

Transformers are made up of encoders and decoders, but have a key ingredient: the ***attention*** [arXiv:1706.03762]
→ They are a foundational model in natural language processing (NLP)



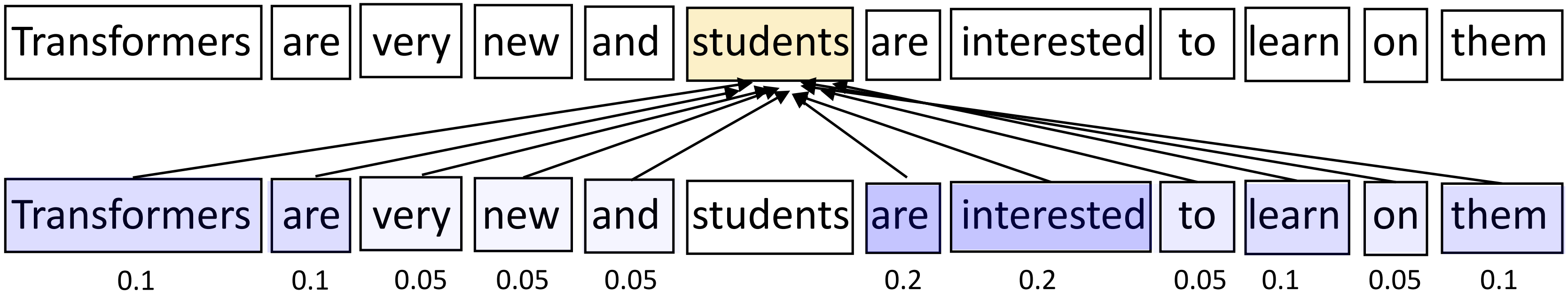
*Attention*

$$h_t = \sum_{i=1}^{t} s_i x_t$$

*NN states from previous layer*

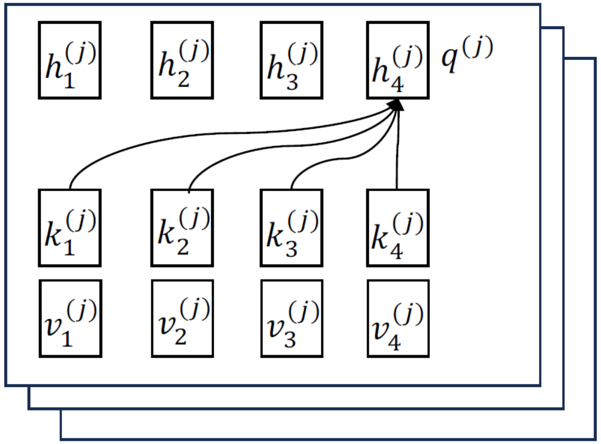
***attention score:***
computes how relevant is the *i* position of the input to this current hidden output

***Attention*** enables the model to dynamically prioritize relevant information by weighing the importance of different parts of the input, transformed in embedding vectors.



# Un/supervised learning + NN

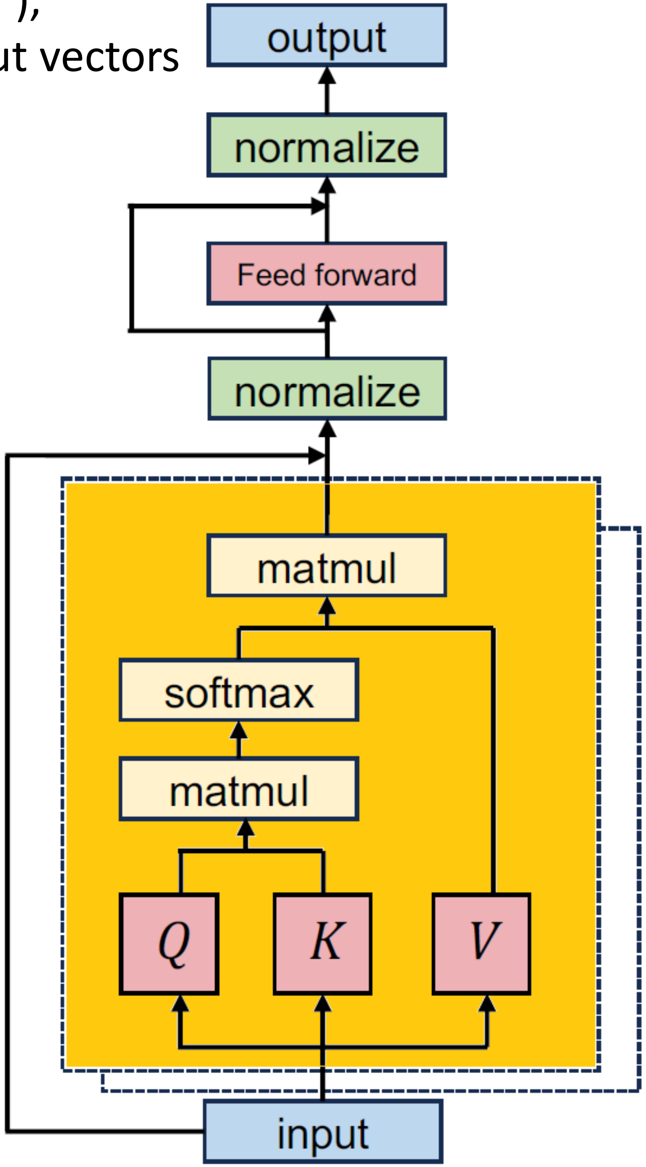
Having $Q$, $K$, $V$ inputs ∈ $\mathbb{R}^{\mathrm{Txd}}$) (“queries”, “keys”, “values”), we can have *multi-head attention (j)* projecting the input vectors in multiple subspaces:



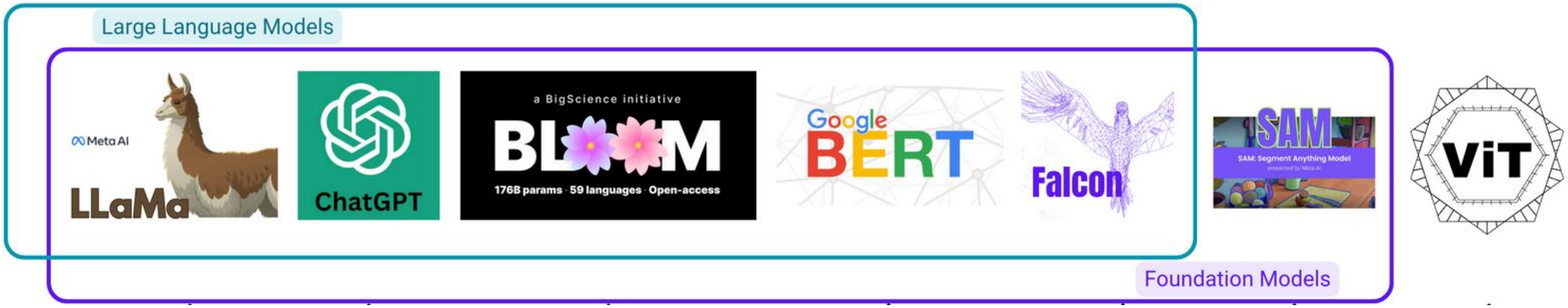
$$h = \sum_i \mathrm{softmax}(s)_i v_i = \frac{\sum_i \exp(s_i) v_i}{\sum_j \exp(s_j)}$$

$$\mathrm{SelfAttention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{d^{1/2}}\right)V$$

It provides a set of contextualized embeddings that can be used as input in a FNN for further processing



# Un/supervised learning + NN



**LLaMA** (Large Language Model Meta AI): Natural Language Processing (NLP) research, low-resource deployment, experimentation with large models.

ChatGPT (Generative Pre-trained Transformer): Chatbots, virtual assistants, code generation, tutoring, content creation, etc.

BLOOM (BigScience Large Open-science Open-access Multilingual Language Model): text generation, translation, code generation, language research across multiple languages.

BERT (Bidirectional Encoder Representations from Transformers): text classification, question answering, named entity recognition (NER), text summarization, and translation.

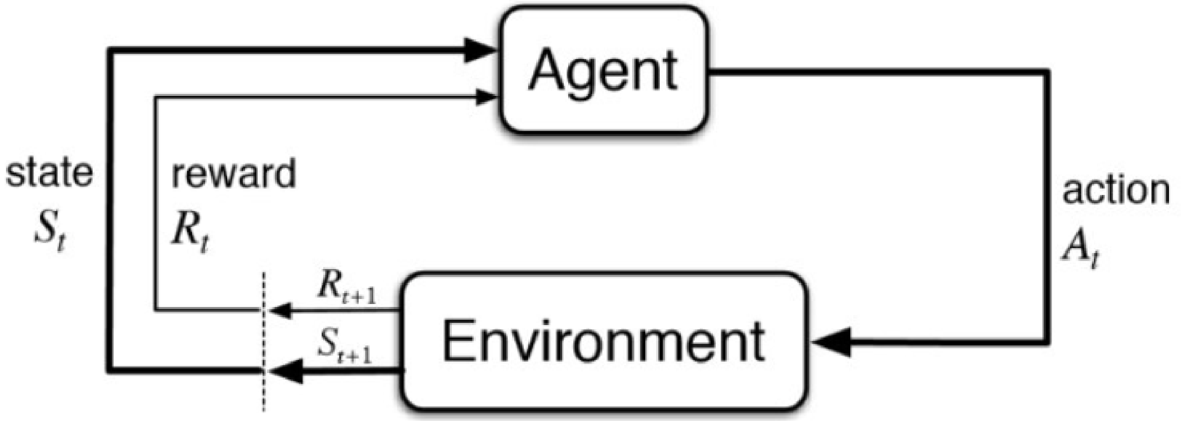
Falcon: text generation, text summarization, translation, sentiment analysis, question answering, conversational agents, and research and development in NLP

SAM (Segment Anything Model) and **ViT** (Vision Transformer): image segmentation and recognition tasks.

# Reinforcement Learning

# Reinforcement learning

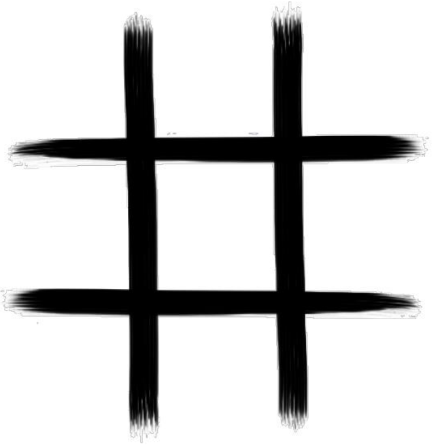
- It is not only a leaning process over time, but a taking decision system

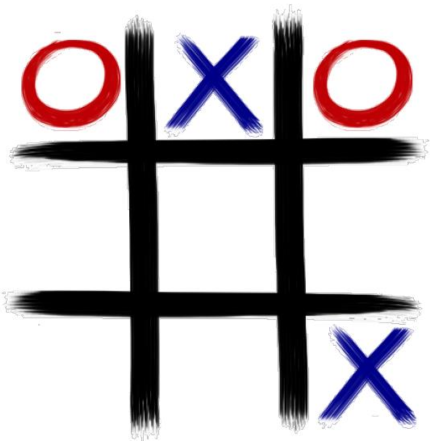


**Agent:** Entity learning about the environment and making decisions. We need to specify a learning algorithm for the agent that allows it to learn a policy (π).
**Environment**: Everything outside the agent, including other agents.
**Policy (π)**: it tells the agent what action to take in a given state.
**State**: A representation of the environment. At time step *t*, the agent is in state $S_t \in S$, where *S* is the set of all possible states.
**Action *(A)***: At time step *t*, an agent takes an action $A_t \in A(t_s)$ where $A(t_s)$ is the set of actions available in state $S_t$.
**Rewards:** Numerical quantities that represent feedback from the environment that an agent tries to maximize.

# Reinforcement learning

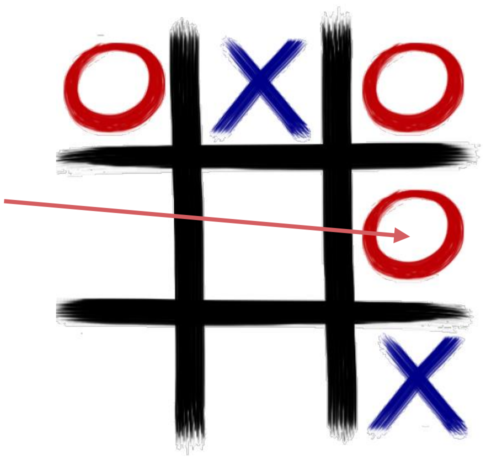
Environment:
tic tac toe board

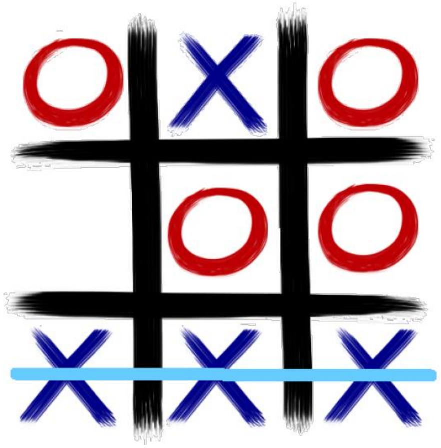




State $S_t$

Action $A(S_t)$





Reward $r_{t+1} = -1$

# Reinforcement learning

- The Q-value is the expected reward of taking action *A* in state *S* and then continuing according to the policy π.

reward
at t+1

Initial state and action

$$Q^{\pi}(S,A) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \mid S_0 = S, A_0 = A\right]$$

Discount factor ∈(0,1), which reduces the weight of future rewards

Expectation
(average over time/trials)

The learning process consists of finding the optimal policy by iteratively updating Q-values for each state-action pair:

best possible action

$$Q^{\pi}(S,A) \leftarrow Q^{\pi}(S,A) + \alpha \left(R + \gamma \max_{A'} Q(S',A') - Q(S,A)\right)$$

learning rate (controls how much new information overrides old information)

# Underlying hardware and libraries

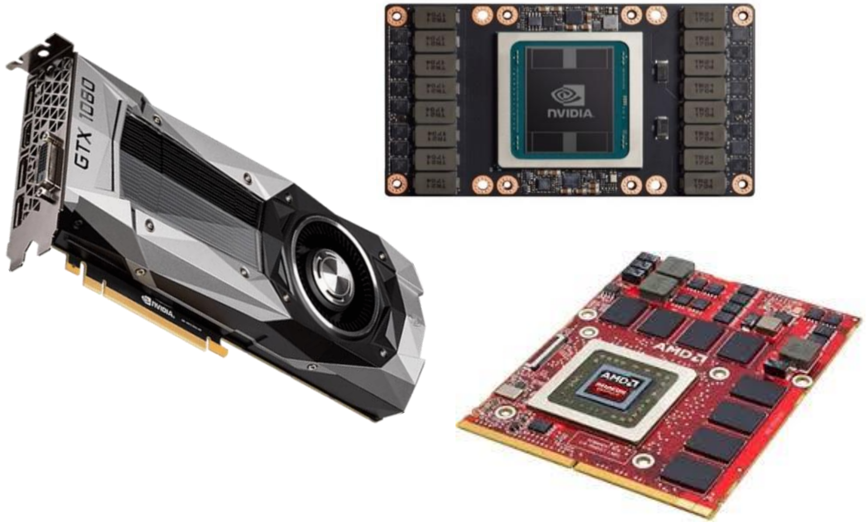
# Underlying hardware

**Any ML technique depends on the hardware platform where is executed:**

→ Use more than one kind of processor or cores to maximize performance or energy efficiency (specially important for training).

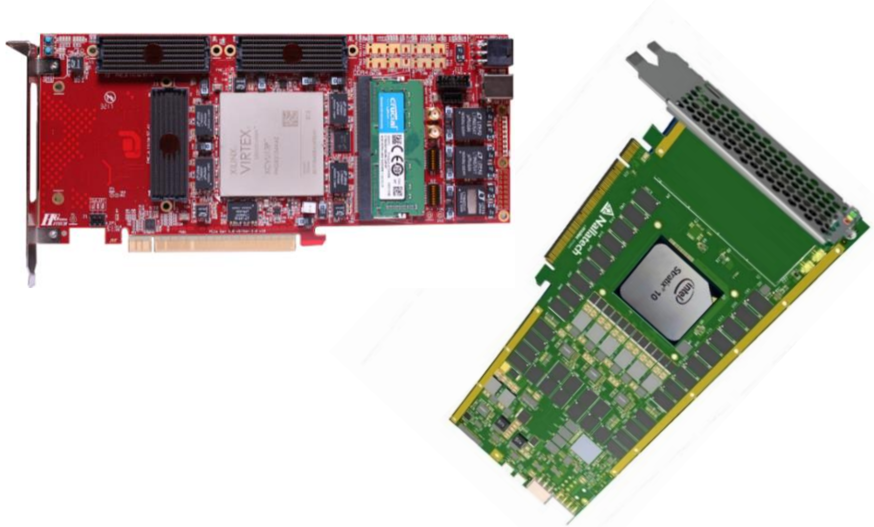
→ Exploit the high level of parallelism to handle particular tasks.

### Graphic Processor Units (GPUs)



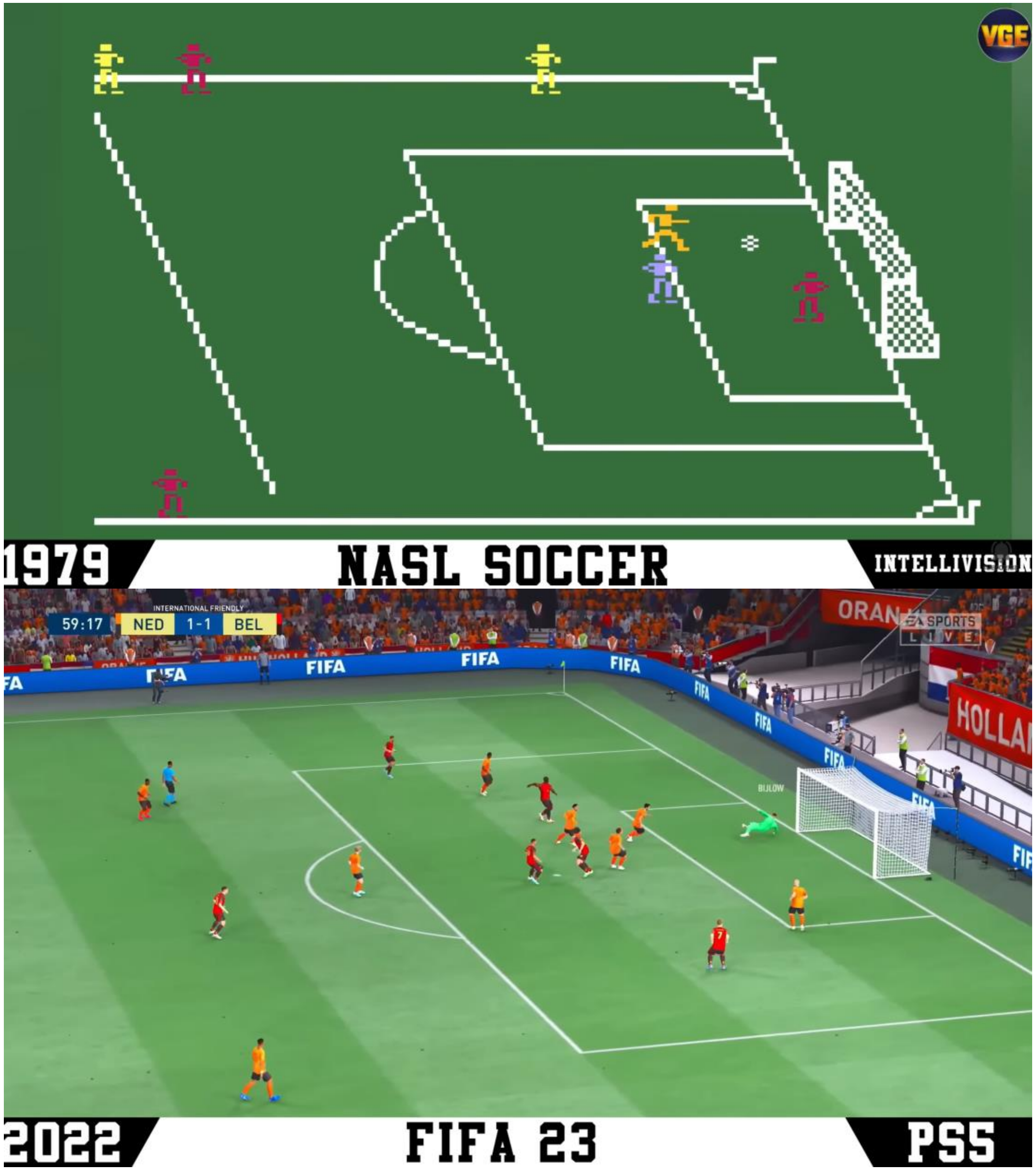
- Multicore processors, highly commercial
- High throughput
- Ideal for data –intensive parallelizable applications

### Field Programmable Gate Arrays (FPGAs)



- Programmable and flexible devices
- Low latency
- Low power consumption
- Ideal for compute- and data-intensive workloads

# Underlying hardware



Gen. Inst.
CP1600 +
Standard
Television
Interface Chip

AMD
Ryzen Zen 2 +
RDNA-2

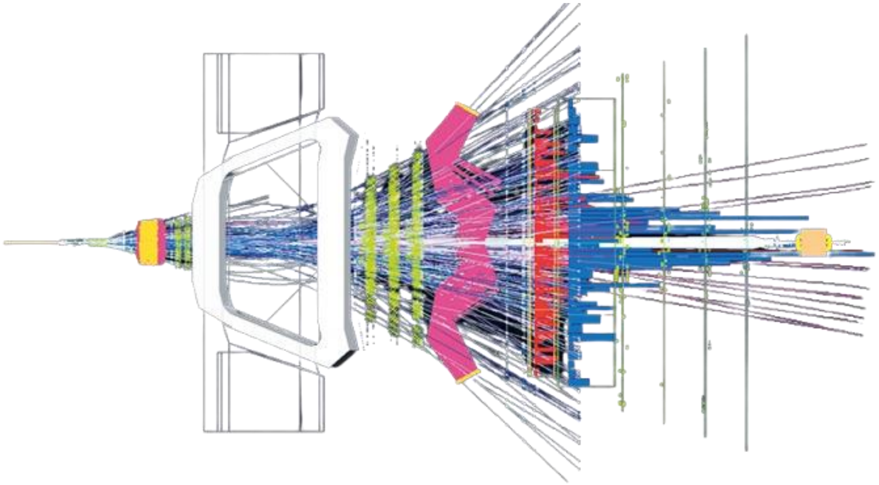
# Underlying hardware

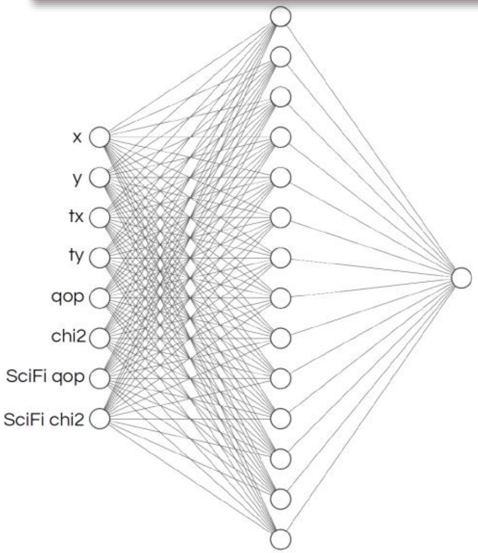
**Artemisa:** https://artemisa.ific.uv.es/web/
ARTificial Environment for ML and Innovation in Scientific Advanced Computing

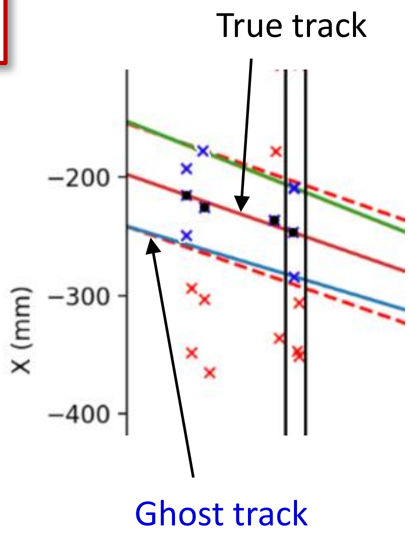
# Underlying hardware

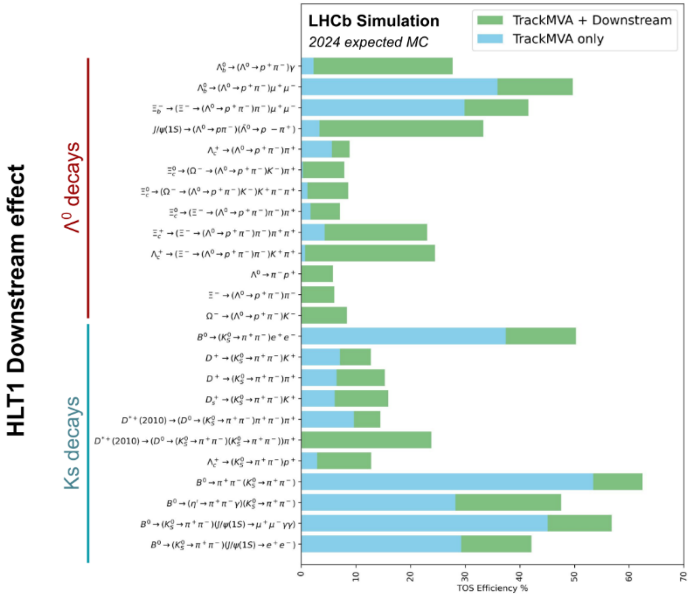
- A successful (high impact!) example at LHCb:

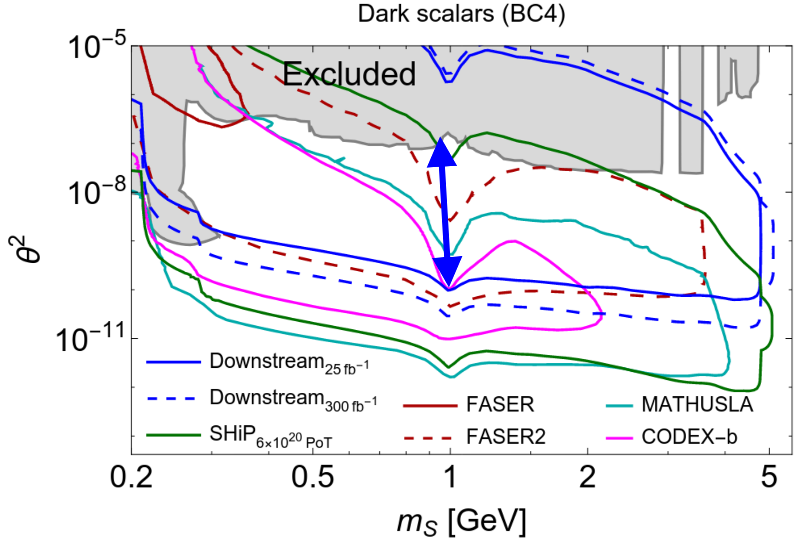
A fast FNN implemented on GPUs (30MHz rate)











[Eur.Phys.J.C 84 (2024)6, 608]

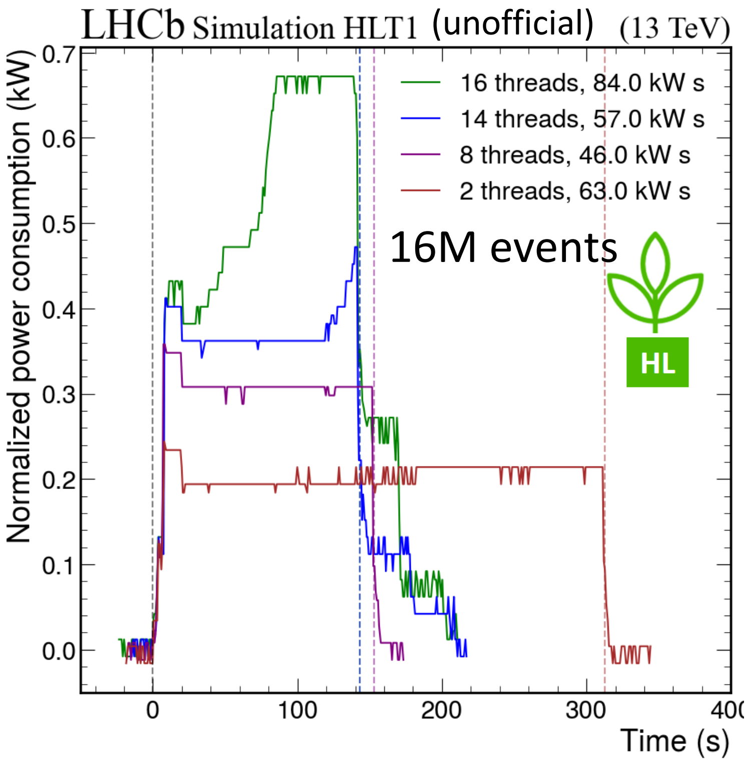
# Underlying hardware

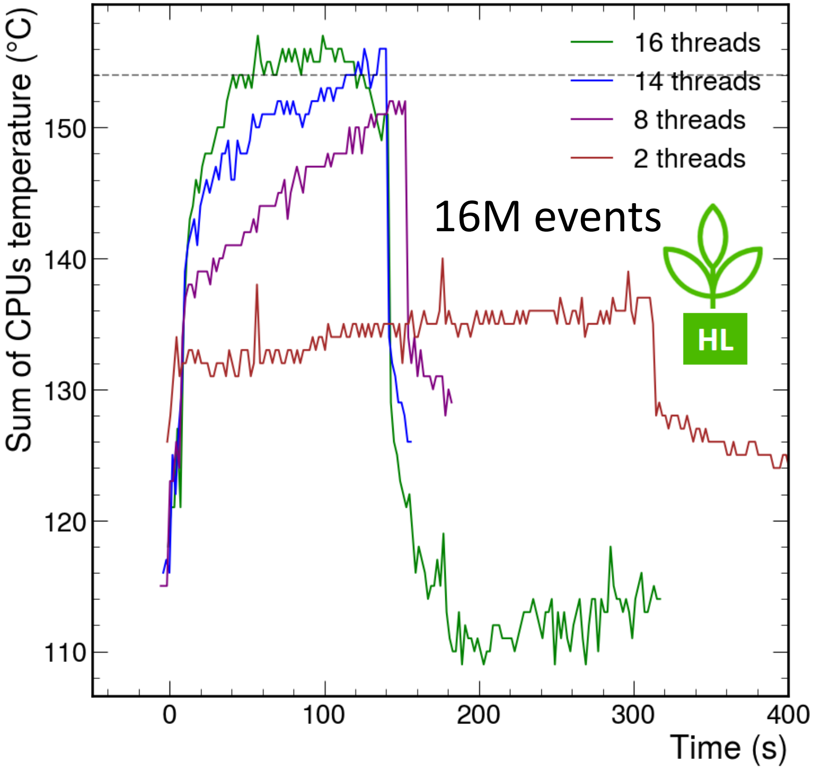
BE RESPONSIBLE

- Last words on power consumption…

Ex: A *random forest* consumes x 2 energy as compared to *AdaBoost*

Energy consumption is fully correlated to with *throughput* but it also depends on the hardware utilization:





# Machine Learning Libraries

- Data processing: pandas, NumPy

  https://pandas.pydata.org/

  https://numpy.org/

- Machine learning: scikit-learn

  https://scikit-learn.org

- Deep learning: PyTorch, TensorFlow, Keras

  https://pytorch.org/

  https://www.tensorflow.org

  https://keras.io/

- Visualization: seaborn, matplotlib

  https://seaborn.pydata.org/

  https://matplotlib.org/

# Hands on