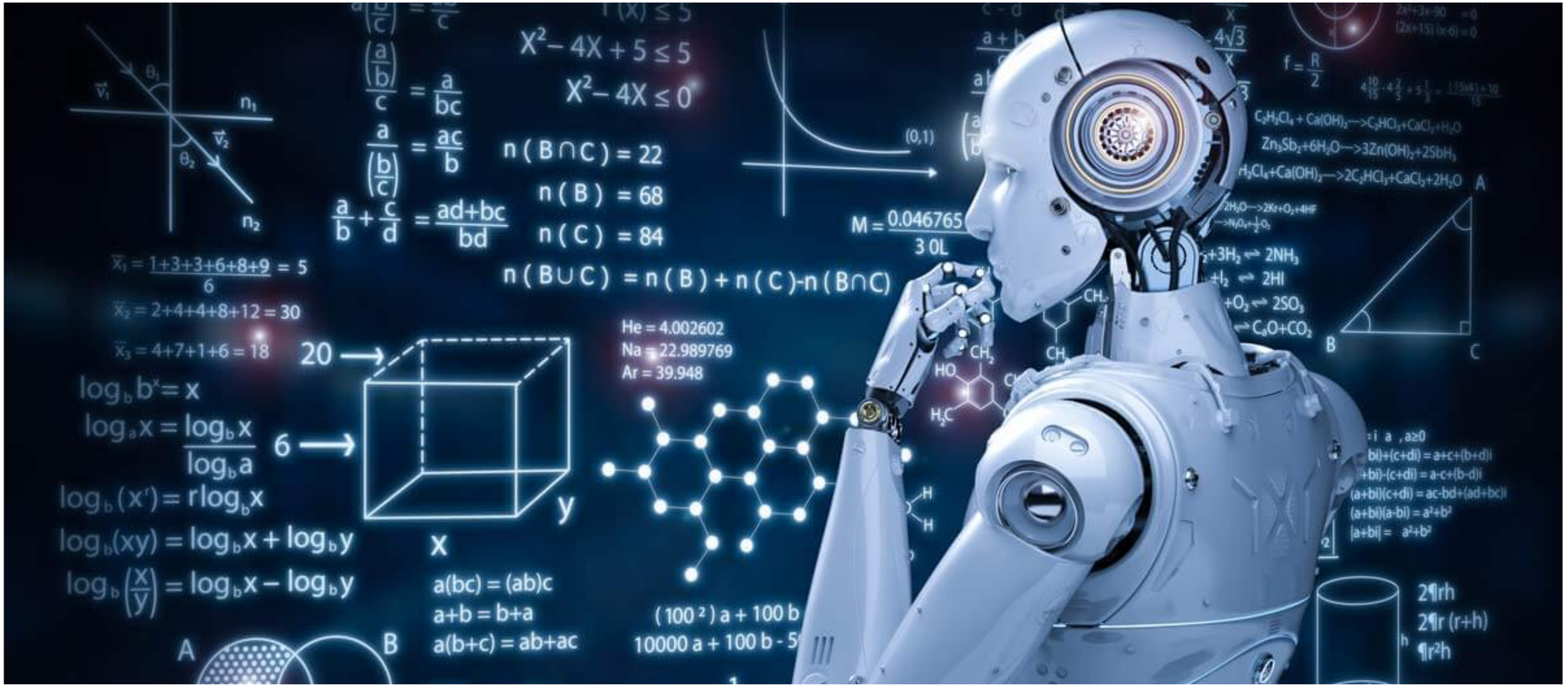
# Deep Learning
# Part I: Introduction

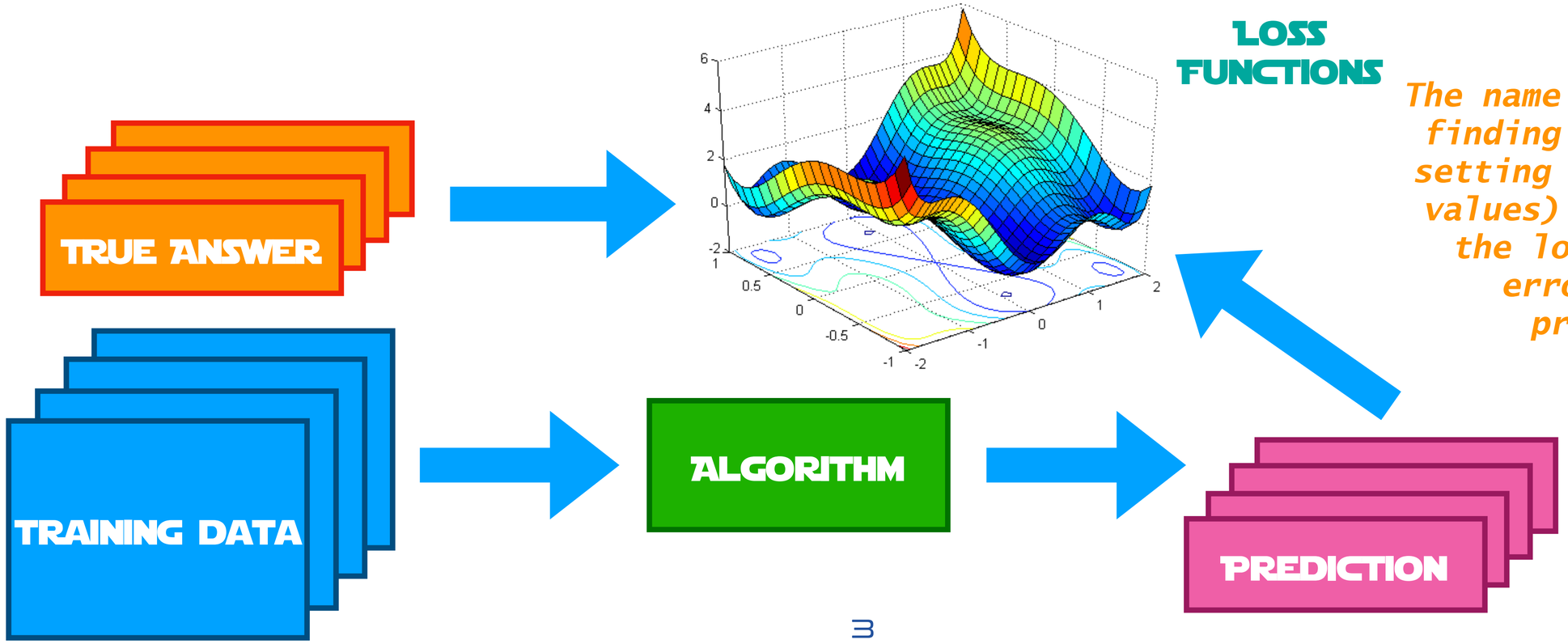
Maurizio Pierini


# What is Machine Learning ?

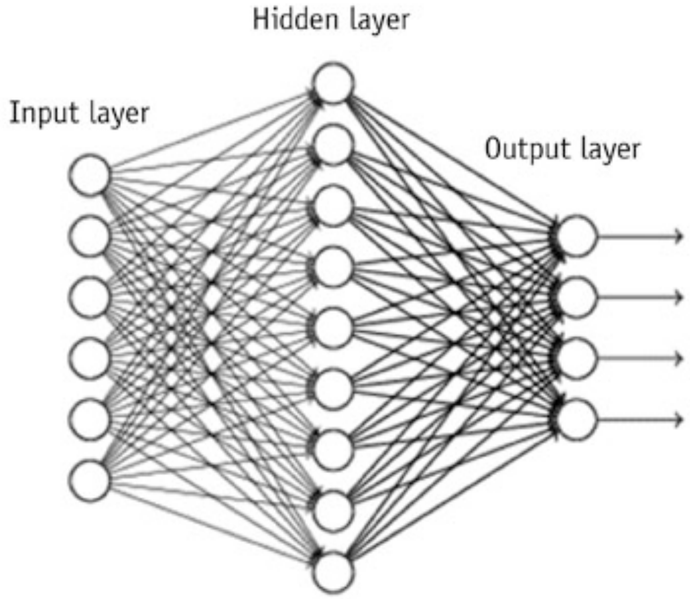
# A definition (Wikipedia)

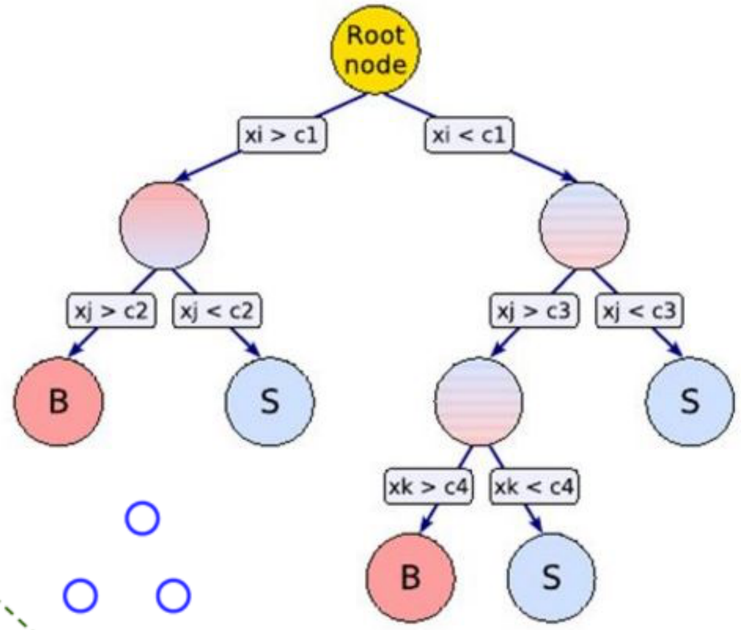
*Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to progressively improve their performance on a specific task. Machine learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.*

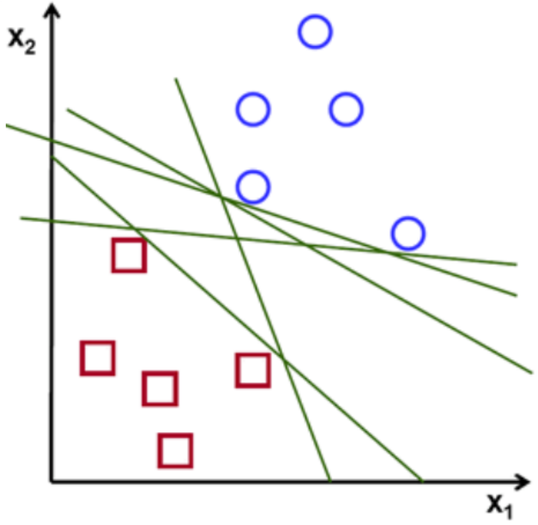


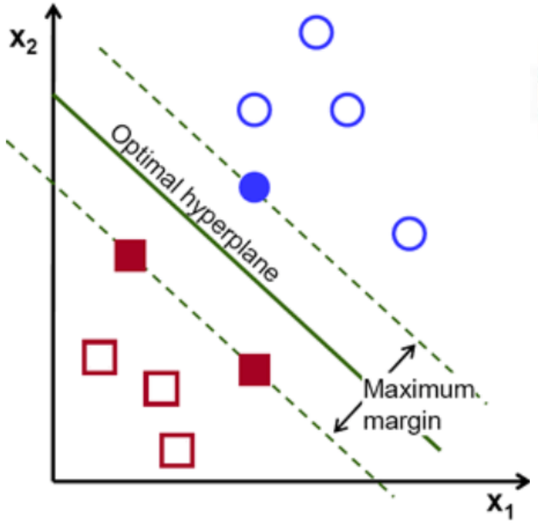
# Many flavors of ML

- *Different ML algorithms had their moment of glory*
  - *(Shallow) neural networks dominated in the 80's*
  - *Alternatives emerged in the 90's*
    - *Support vector machine*
    - *Boosting of decision trees*



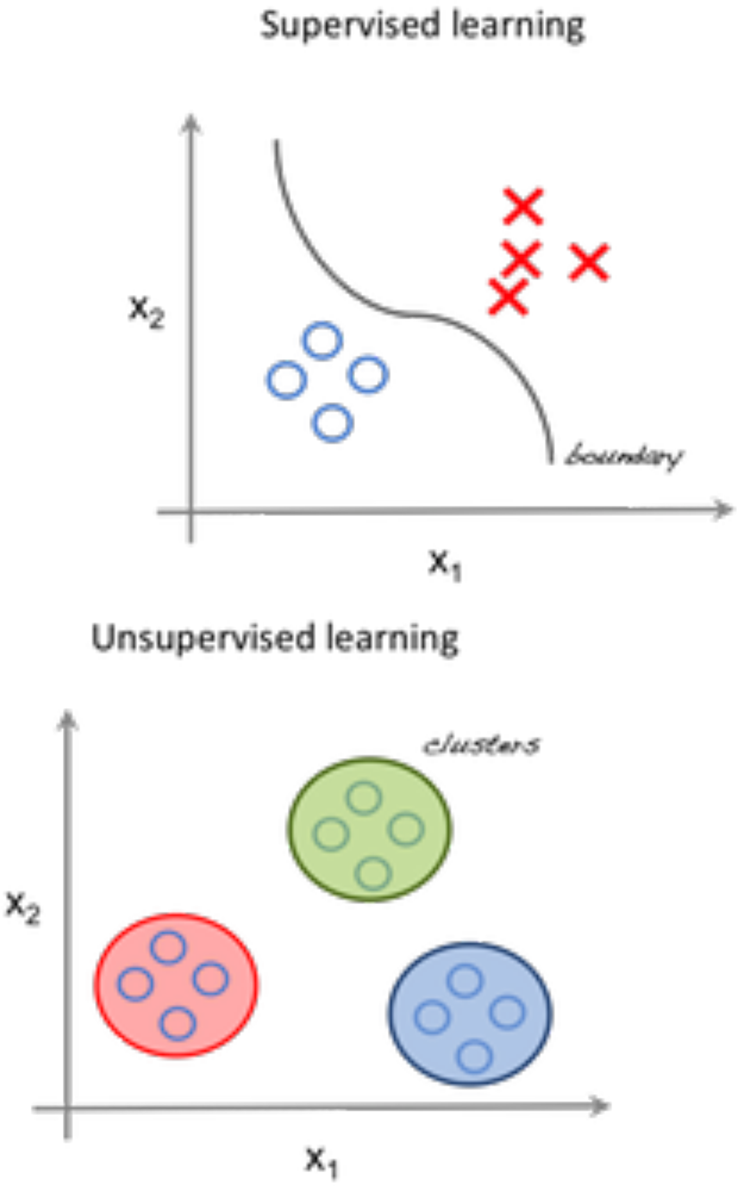



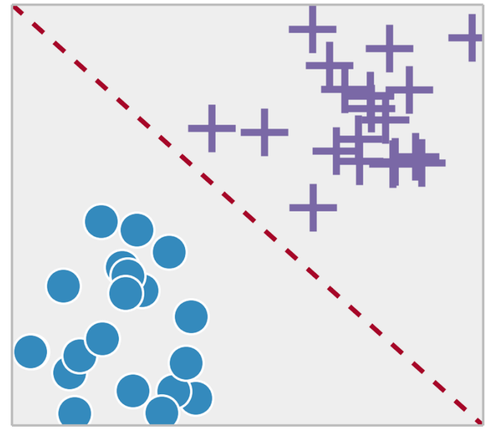
# A two-steps process

- ***Learning:*** *train the algorithm on a provided dataset*
  - ***Supervised:*** *the dataset X comes with the right answer y (right class in a classification problem). The algorithm learns the function*
  - ***Unsupervised****: the dataset X comes with no label. The algorithm learns structures in the data (e.g., alike events in a clustering algorithm)*
  - ***Reinforcement:*** *learn a series of actions and develop a decision-taking algorithm, based on some action/reward model*
  - …
- ***Inference****: once trained, the model can be applied to other datasets*

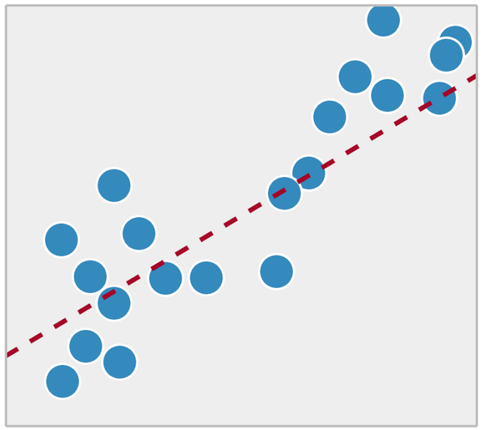


# Typical problems

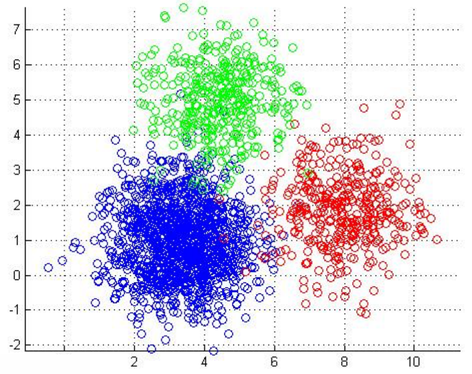
- ***Classification***: *associate a given element of a dataset to one of N exclusive classes*



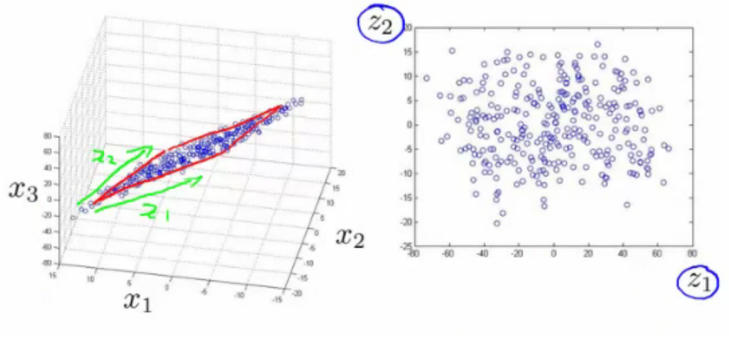
- ***Regression***: *determine a continuous value y from a set of inputs x*



- ***Clustering***: *group elements of a dataset because of their similarity according to some learned metric*

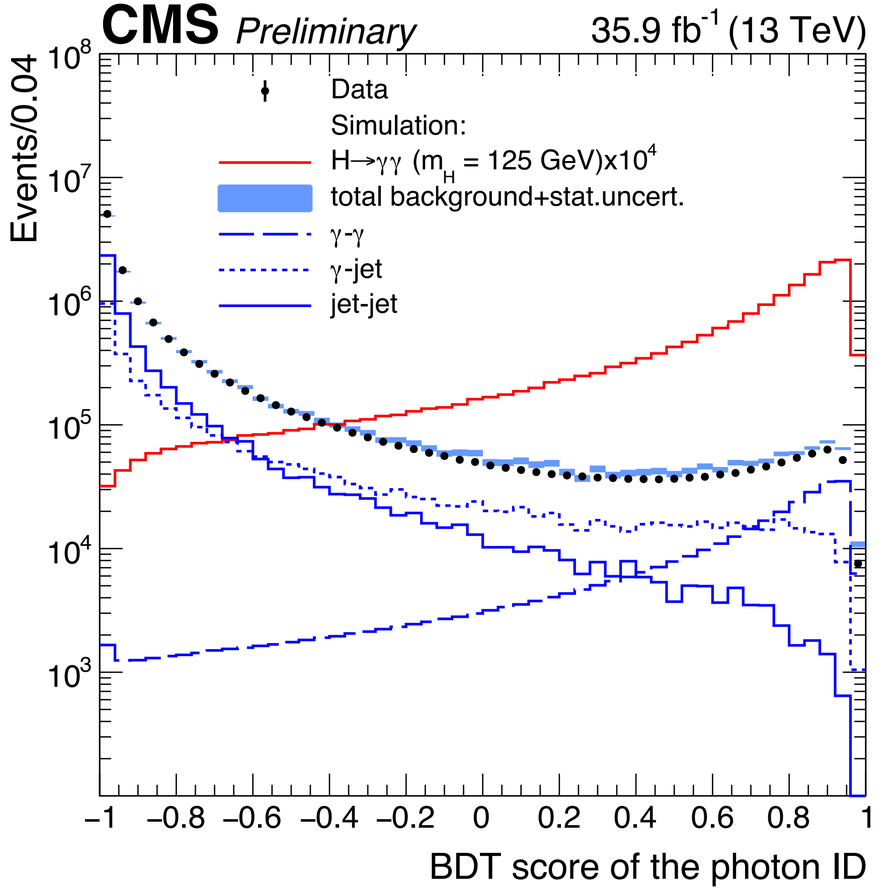



- ***Dimensionality reduction***: *find the k quantities of the N inputs (with k<N) that incorporate the relevant information (e.g., principal component analysis)*



# Machine Learning in HEP

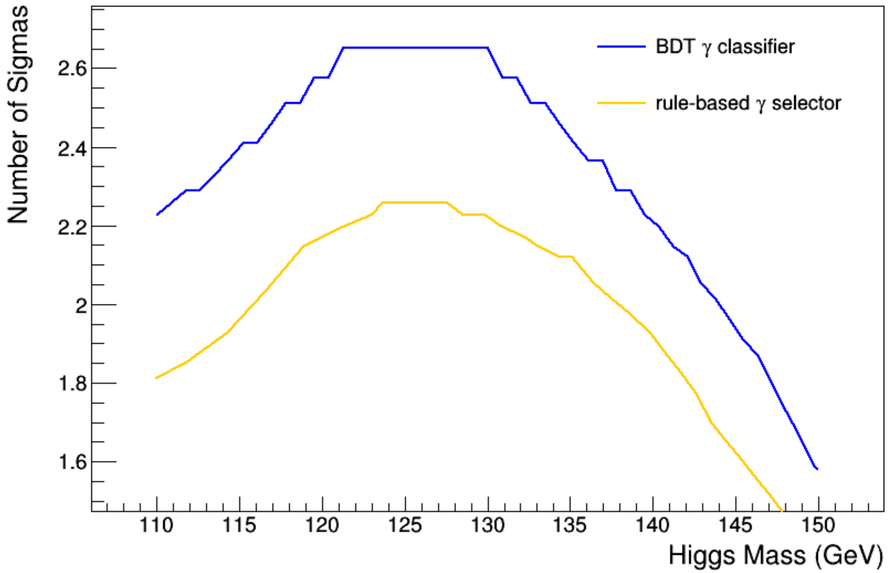
- *Classification:*
  - *identify a particle & reject fakes*
  - *identify signal events & reject background*
- *Regression:*
  - *Measure energy of a particle*
- *Up to now, these task mainly solved with BDTs*
  - *moved to Deep Learning for analysis-specific tasks*
  - *same will happen for centralised tasks (eventually)*

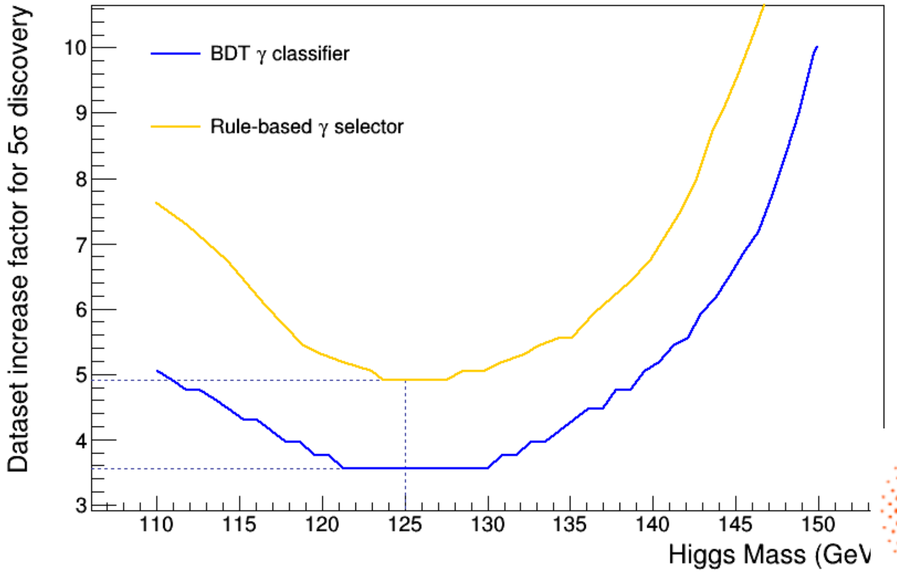


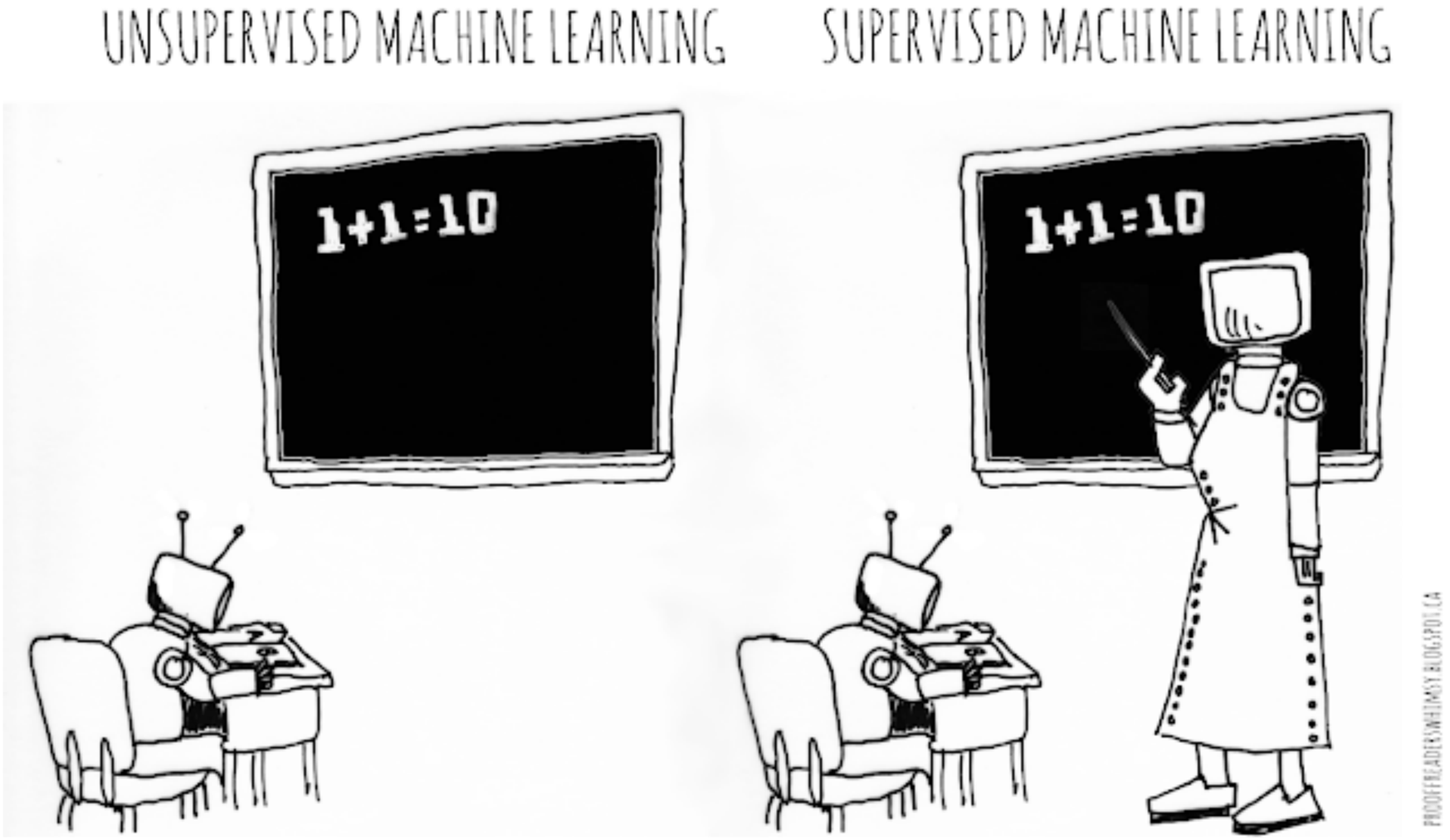

**Centralised task (in online or offline reconstruction)**
**Analysis-specific task (by users on local computing infrastructures)**

# Machine Learning in HEP

- *Long tradition*
  - *Neural networks used at LEP and the Tevatron*
  - *Boosted Decision Trees introduced by MiniNooNE and heavy used at BaBar*
  - ***BDTs ported to LHC and very useful on Higgs discovery***
  - *Now Deep Learning is opening up many new possibilities*



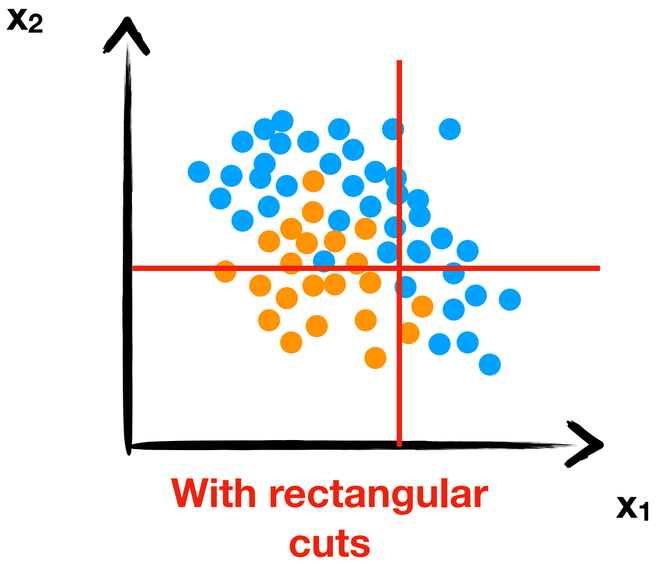



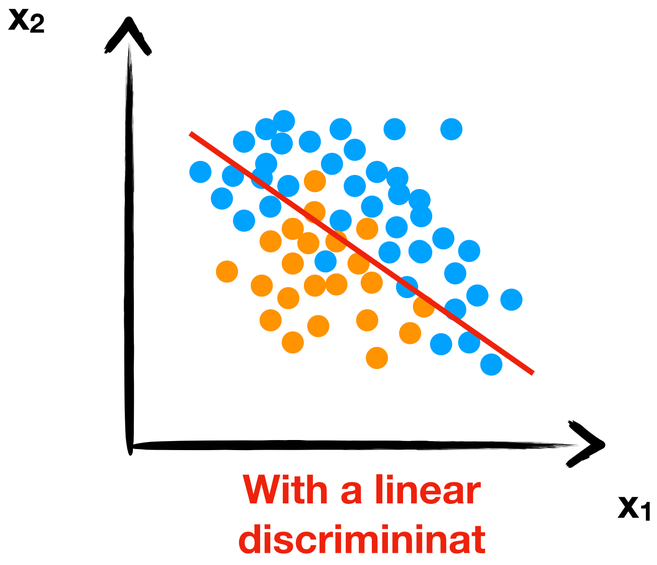


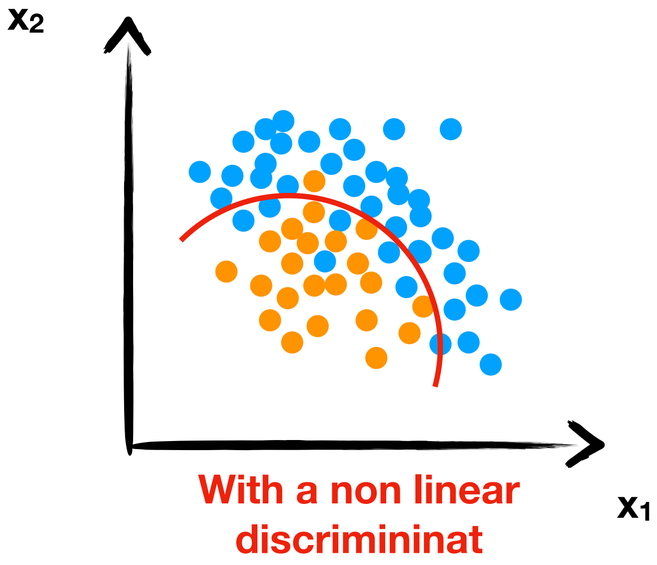

# Supervised Learning

# A simple example: S vs B selection

- *Define a selection to separate the signal from the background*

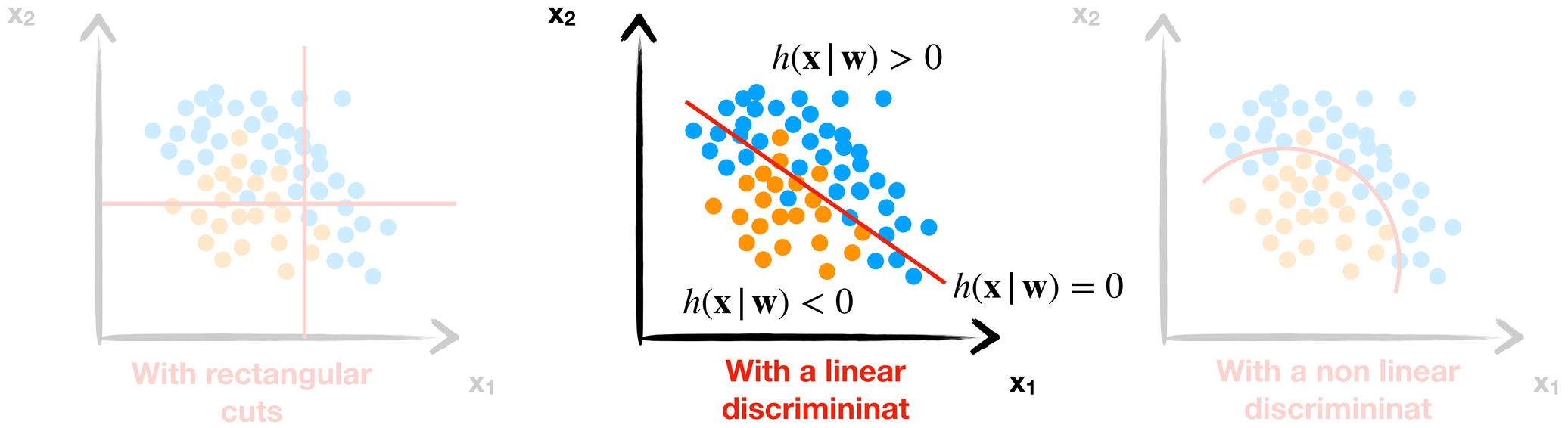


# A simple example: S vs B selection

- *Define a selection to separate the signal from the background*




- *Define a decision boundary which gives optimal separation*

$$h(\mathbf{x}\,|\,\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} = 0$$

(Signed) distance between x and the boundary plane

# Logistic Regression

- Give as input pairs of inputs and outputs:

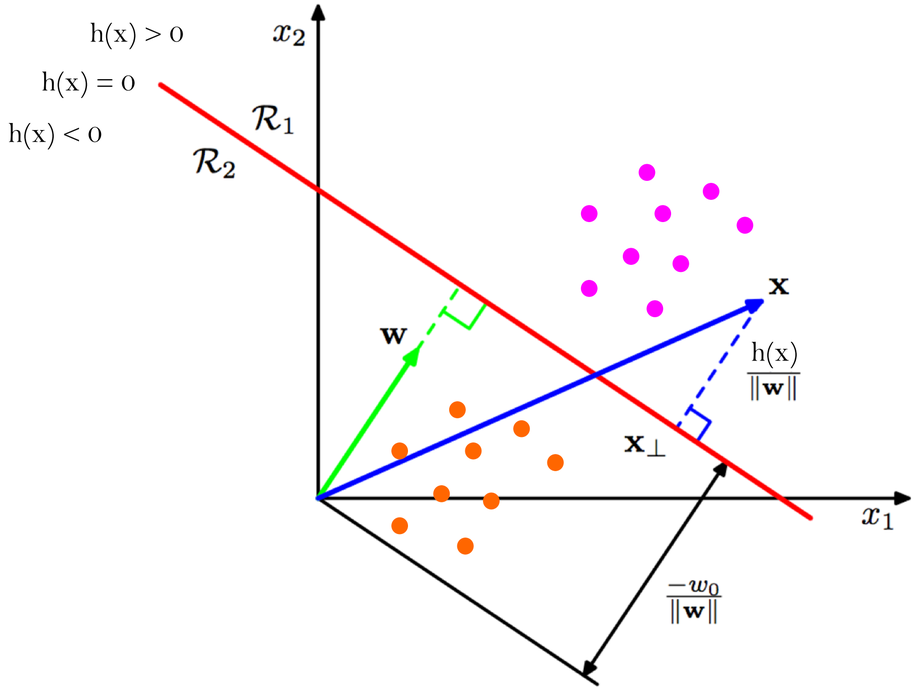
$$x_i \in \mathbb{R}^n \qquad y_i = \{0,1\}$$

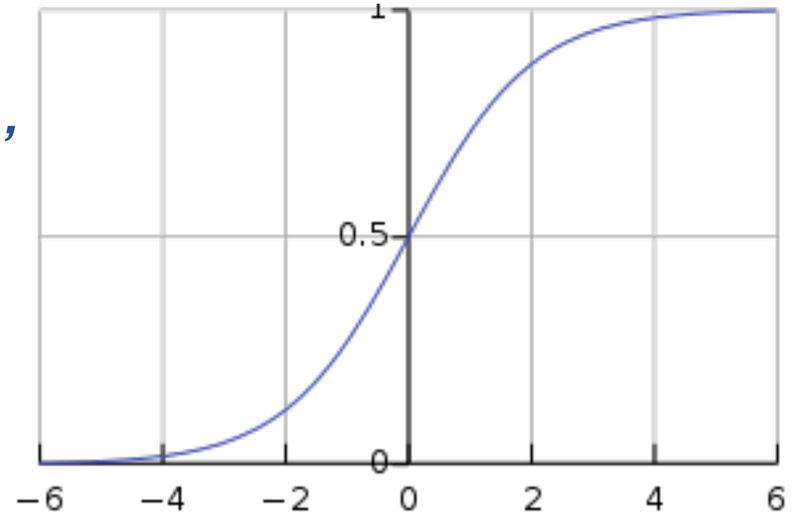
- Model the probability of x to be signal (y=1) as

$$p(y=1\,|\,x) = \frac{1}{1+e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}}$$

  - The larger (and positive) the distance the closer p to 1
  - The larger (and negative) the distance, the closer p to 0

- We can choose the plane such that we maximise the probability of the signal and minimise that of the background



# Bernoulli's problem

- *Bernoulli's problem: probability of a process that can give 1 or 0*
- *The corresponding likelihood is (as usual) the product of the probabilities across the events*
- *Maximizing the likelihood corresponds to minimizing the -logL*
- *Minimizing the -logL corresponds to minimizing the binary cross entropy*
- *How do we minimise it?*

$$\mathscr{L} = \prod_i p_i^{x_i}(1-p_i)^{1-x_i}$$

$$-\log \mathscr{L} = -\log[\prod_i p_i^{x_i}(1-p_i)^{1-x_i}]$$

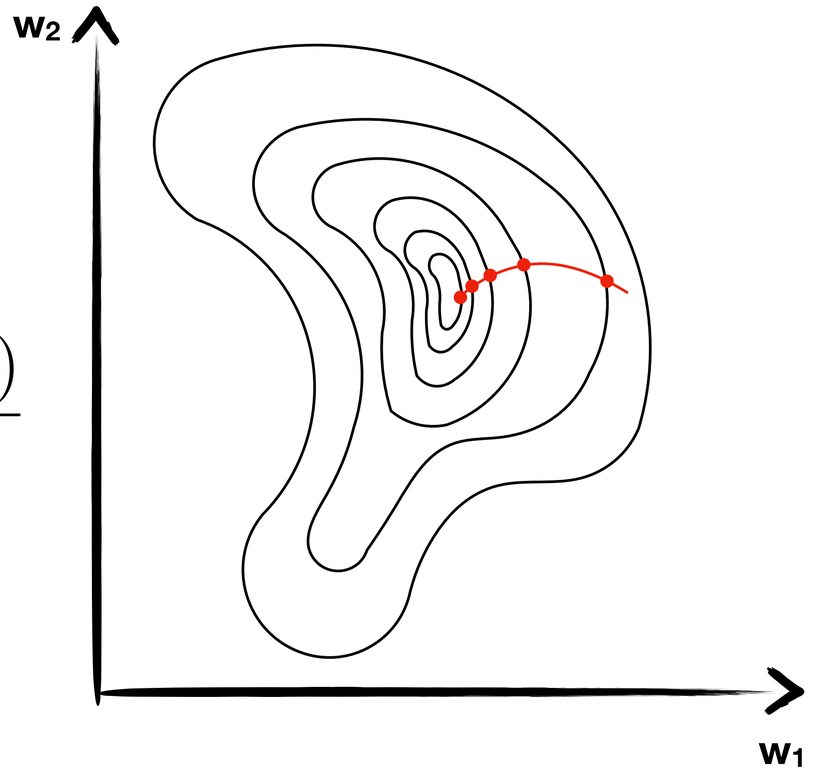
$$= -\sum_i \left[x_i \log p_i + (1-x_i)\log(1-p_i)\right]$$



European Research Council

# Gradient Descent

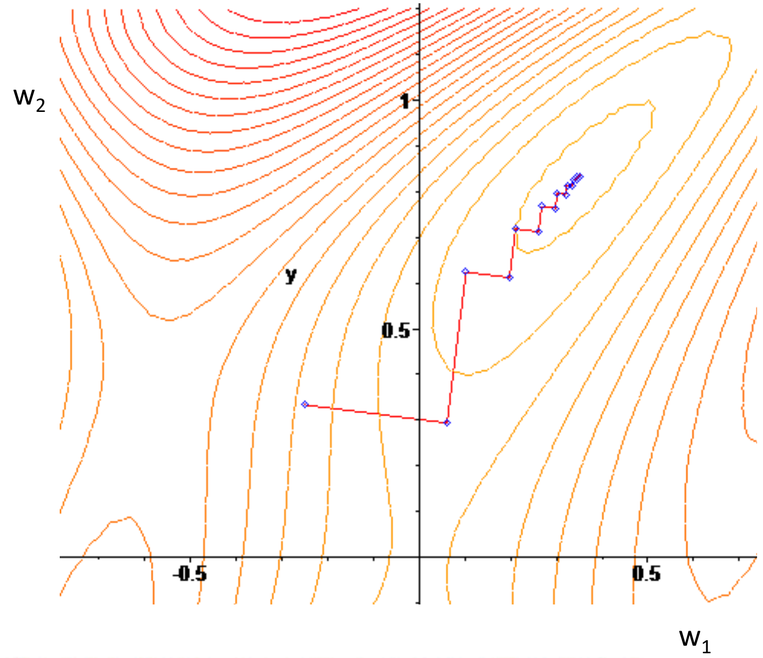
- *Gradient Descent is a popular minimisation algorithm*
  - *Start from a random point*
  - *Compute the gradient wrt the model parameters* $\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$
  - *Make a step of size η (the **learning rate**) towards the gradient direction*
  - *Update the parameters of the mode accordingly*
  - *Effective, but computationally expensive (gradient over entire dataset)*

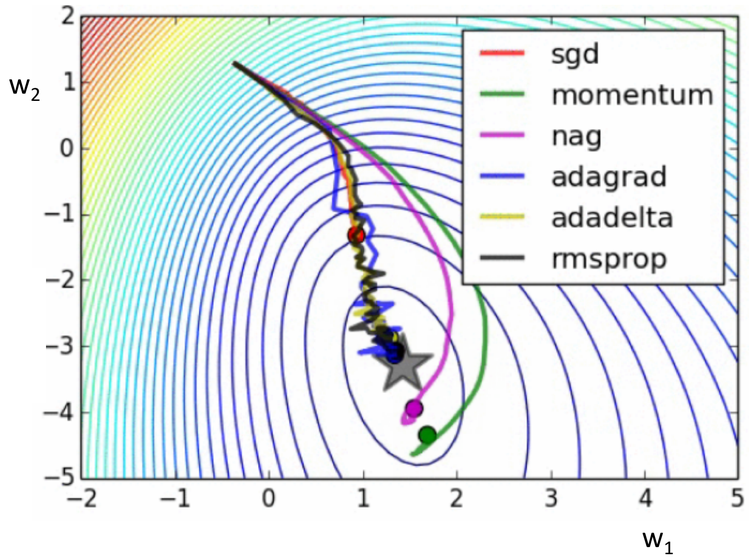



$$\mathbf{w}' \leftarrow \mathbf{w} - \eta \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$$

# Stochastic Gradient Descent

- *Make the minimisation more computationally efficient*
  - *Compute gradient on a small batch of events (faster & parallelizable, but noisy)*
  - *Average over the batches to reduce noise*
  - *BEWARE: better scalability come at the cost of (sometimes) not converging*
  - *Many recipes exist to help convergence, by playing with the algorithm setup (e.g., adapting learning rate)*



# Example: regression & MSE

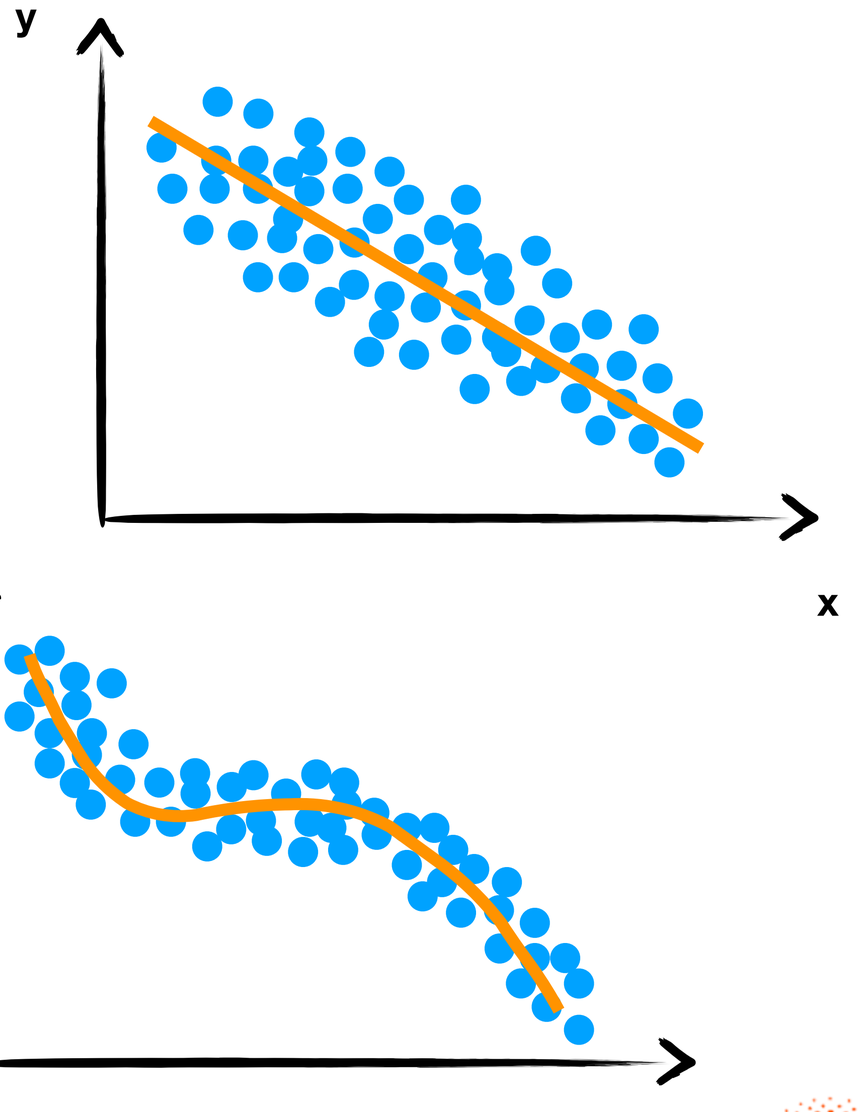
- *Given a set of points, find the curve that goes through them*
  - *Can be a linear model*

$$y_i = ax_i + b$$

  - *Can be a linear function of non-linear kernel of the x. For instance, a polynomial basis*

$$y_i = a\,\phi(x_i) + b$$

New feature, “engineered” from the input features



# Example: regression & MSE

- Take some model (e.g., linear)

$$h(x_i \mid a, b) = ax_i + b$$

- Consider the case of a Gaussian dispersion of y around the expected value

$$y_i = h(x_i) + e_i \qquad p(e_i) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{e_i^2}{2\sigma^2}}$$

- Assume that the resolution σ is fixed

- Write down the likelihood

$$\mathscr{L} = \prod_i \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{e_i^2}{2\sigma^2}} = \prod_i \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - h(x_i))^2}{2\sigma^2}}$$

# Example: regression & MSE

- *The maximisation of this likelihood corresponds to the minimisation of the mean square error (MSE)*

$$argmin[-2\log\mathscr{L}] = argmin\Big[-2\log[\prod_i \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - h(x_i))^2}{2\sigma^2}}]\Big]$$

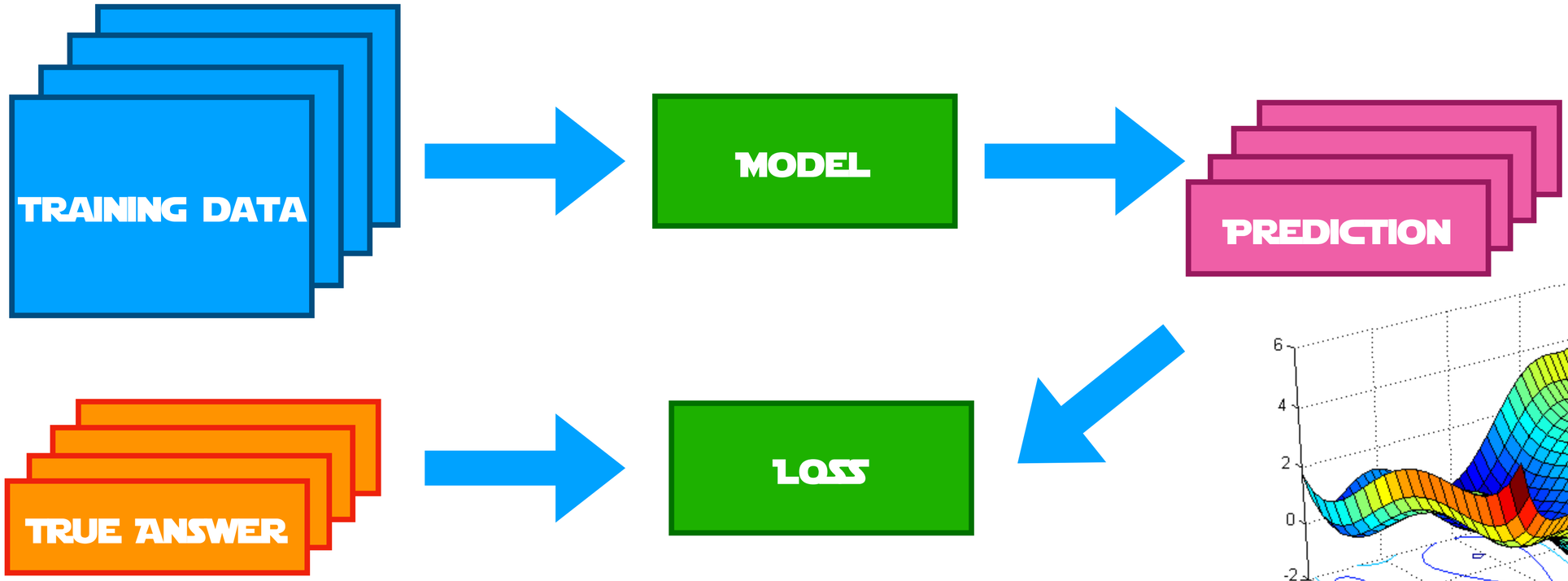
$$= argmin\Big[\sum_i \frac{(y_i - h(x_i))^2}{\sigma^2}\Big] = argmin\Big[\sum_i (y_i - h(x_i))^2\Big] = MSE$$

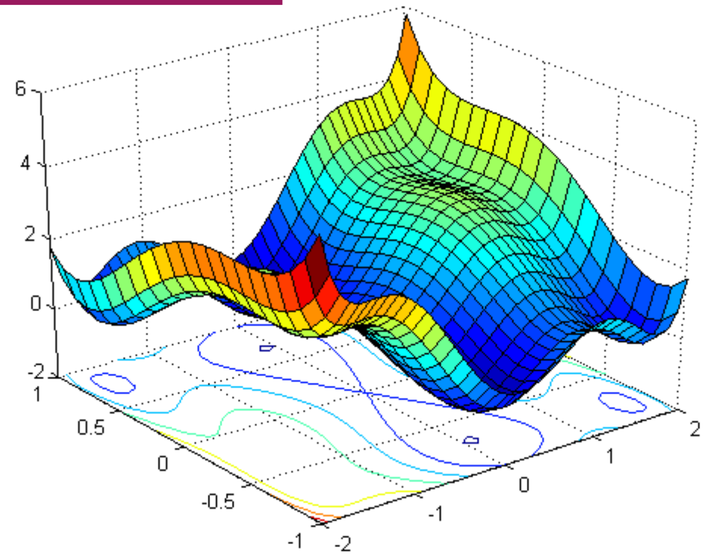
- *MSE is the most popular loss function when dealing with continuous outputs. We will use it a few times in the next days*

- ***BE AWARE OF THE UNDERLYING ASSUMPTION:*** *if you are using MSE, you are implicitly assuming that your y are Gaussian distributed, with fixed RMS*

- *What if the RMS is not a constant?*

# Supervised Learning in a nutshell



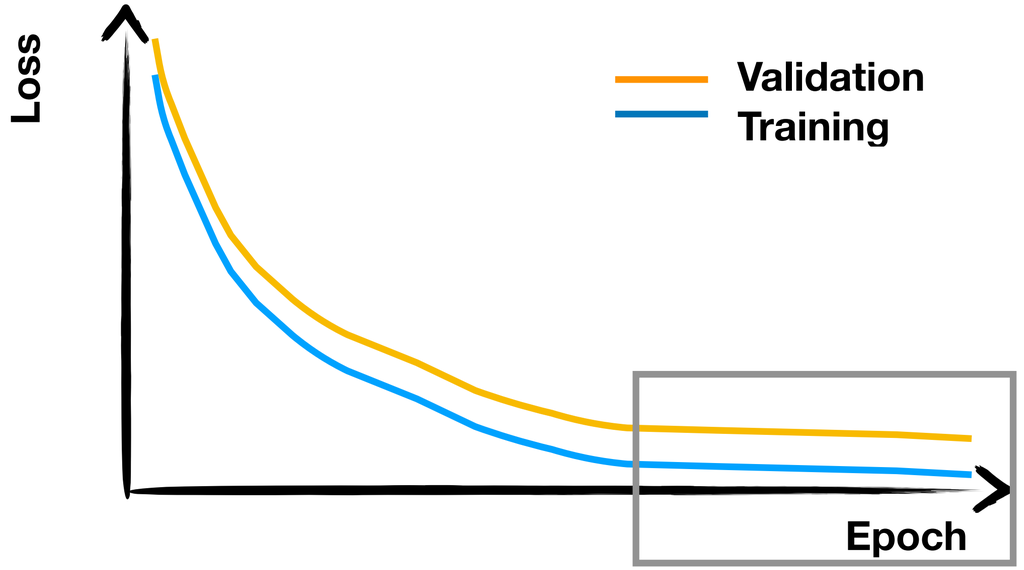

- *A training dataset* ***x***
- *A target y*
- *A model to go from* ***x*** *to y*
- *A loss function quantifying how wrong the model is*
- *A minimisation algorithm to find the model h that corresponds to the minimal loss*

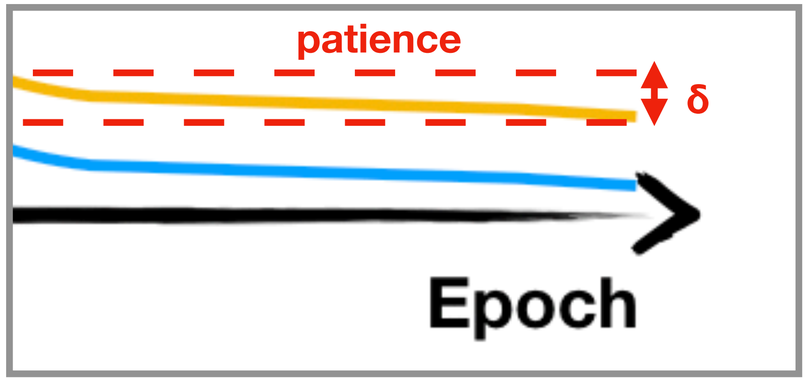
# Training in practice

- *Split your sample in three:*
  - *Training: the biggest chunk, where you learn from*
  - *Validation: an auxiliary dataset to verify generalization and prevent overtraining*
  - *Test: the dataset for the final independent check*

# Training in practice

- Train across multiple epochs
  - 1 epoch = going once through the full dataset
- Use small batches (64, 128, etc)
- Check your training history
  - on the training data (training loss)
  - and the validation ones (validation loss)
- Use an objective algorithm to stop (e.g., early stopping)

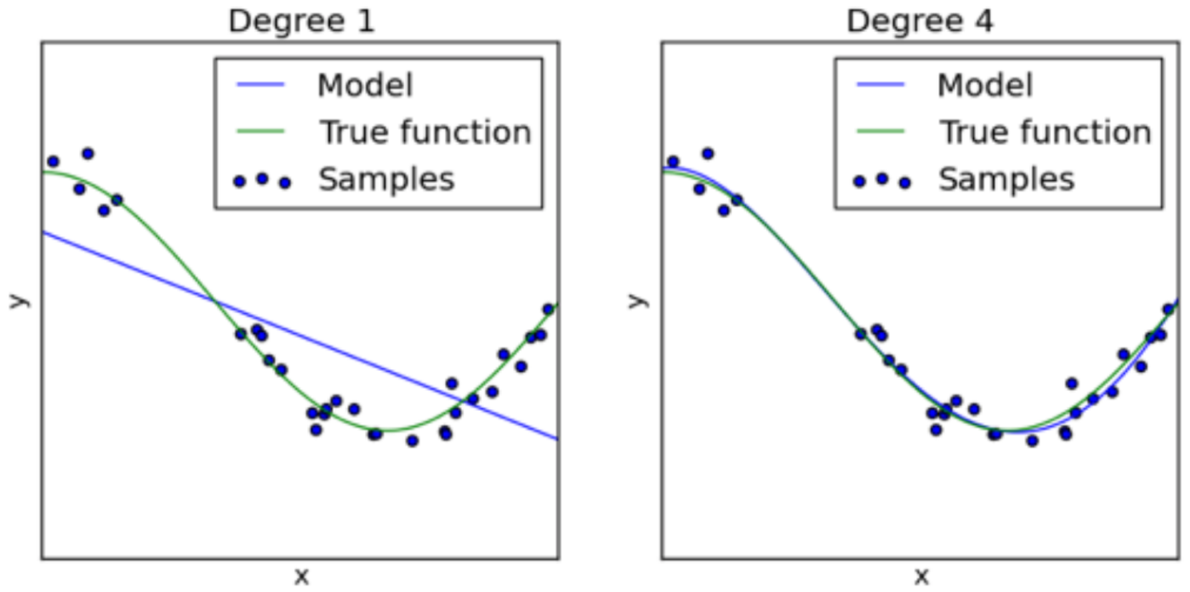


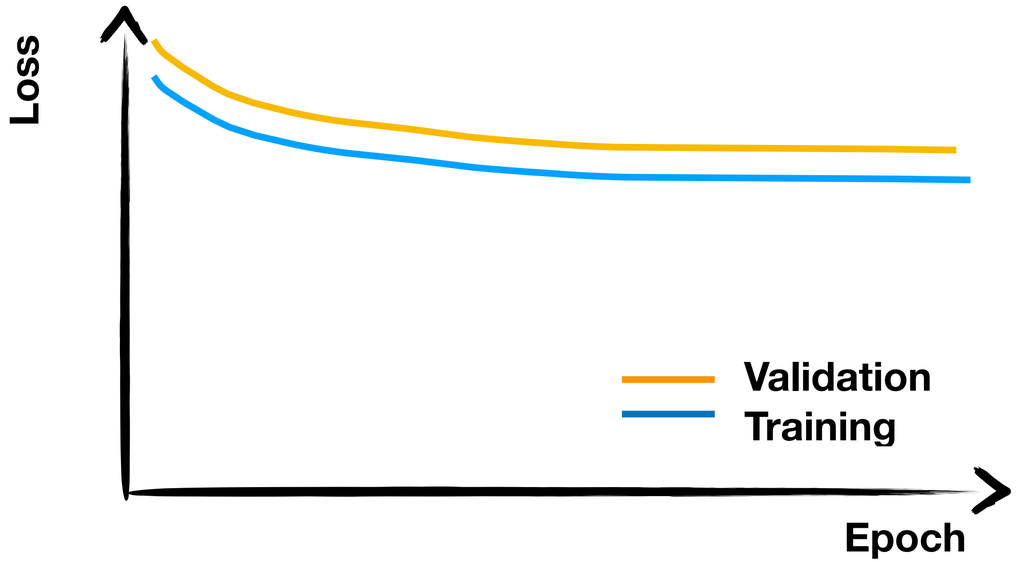




**EARLY TOPPING: stop the train if the validation loss didn't change more than δ in the last n epochs (patience)**

# What can go wrong: underfitting

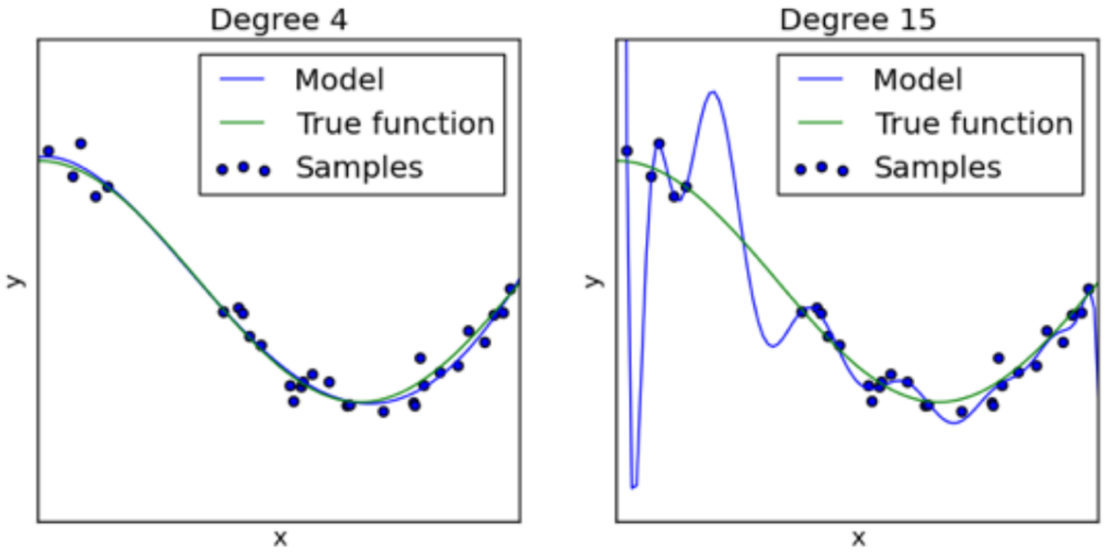
- *If your model has not enough flexibility, it will not be able to describe the data*
- *The training and validation loss will be close, but their value will not decrease*
- *The model is said to be underfitting, or being **biased***

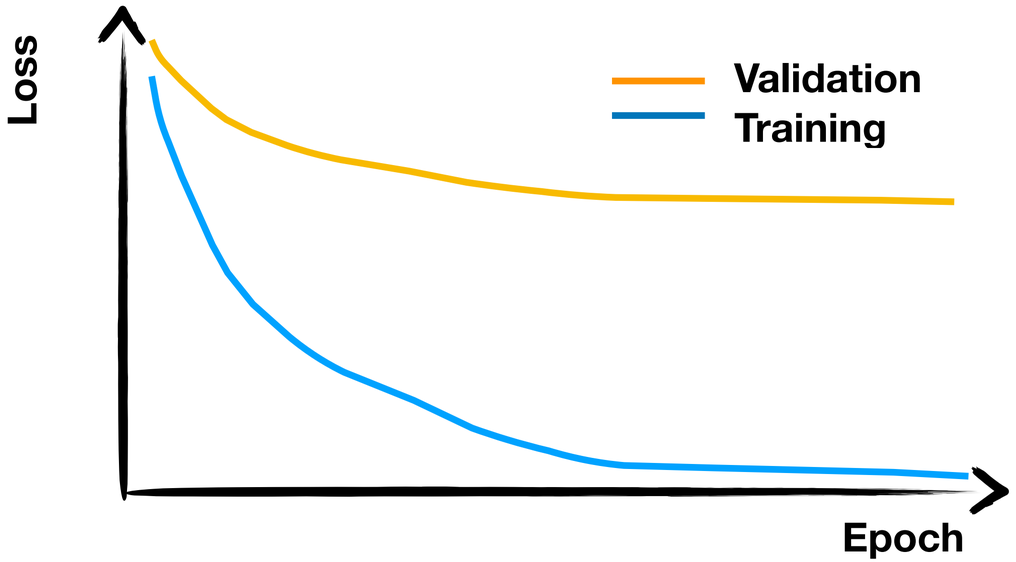


# What can go wrong: overfitting

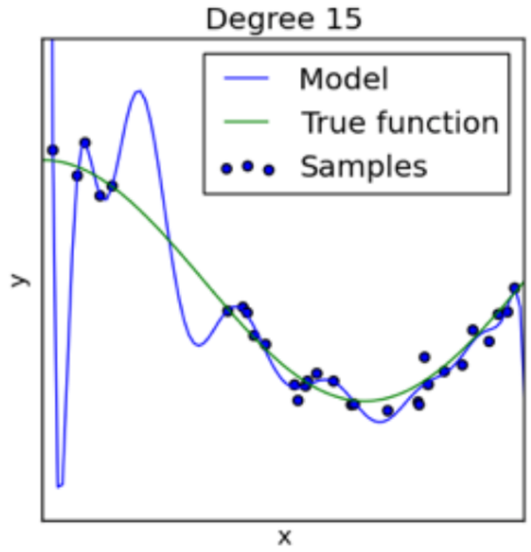
- *Your model can learn too much of your training dataset*
  - *e.g., its statistical fluctuations*
- *Such an overfitted model would not generalise*
- *So, its description of the validation dataset will be bad (i.e., **the mode doesn't generalise**)*
- *This is typically highlighted by a divergence of the training and validation loss*

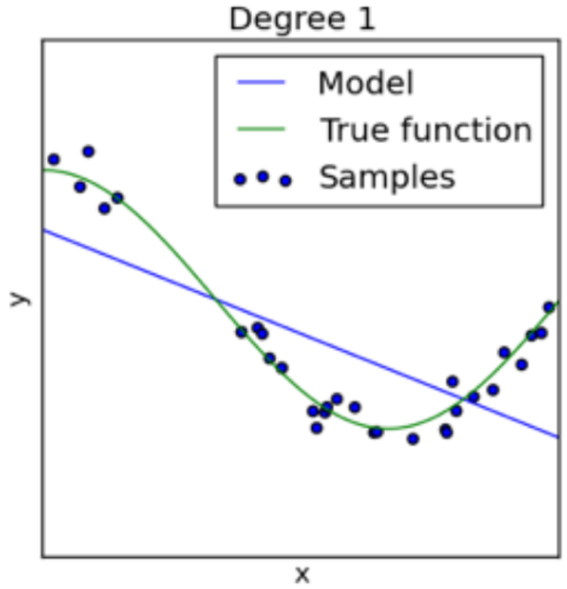


# The Bias vs Variance tradeoff

- ***A model would underfit if too simple:*** *it will not be able to model the mean value*
- ***A model would overfit if too complex:*** *it will reproduce the mean value, but it will underestimate the variance of the data*
- *The* ***generalization error*** *is the error made going from the training sample to another sample (e.g., the test sample)*



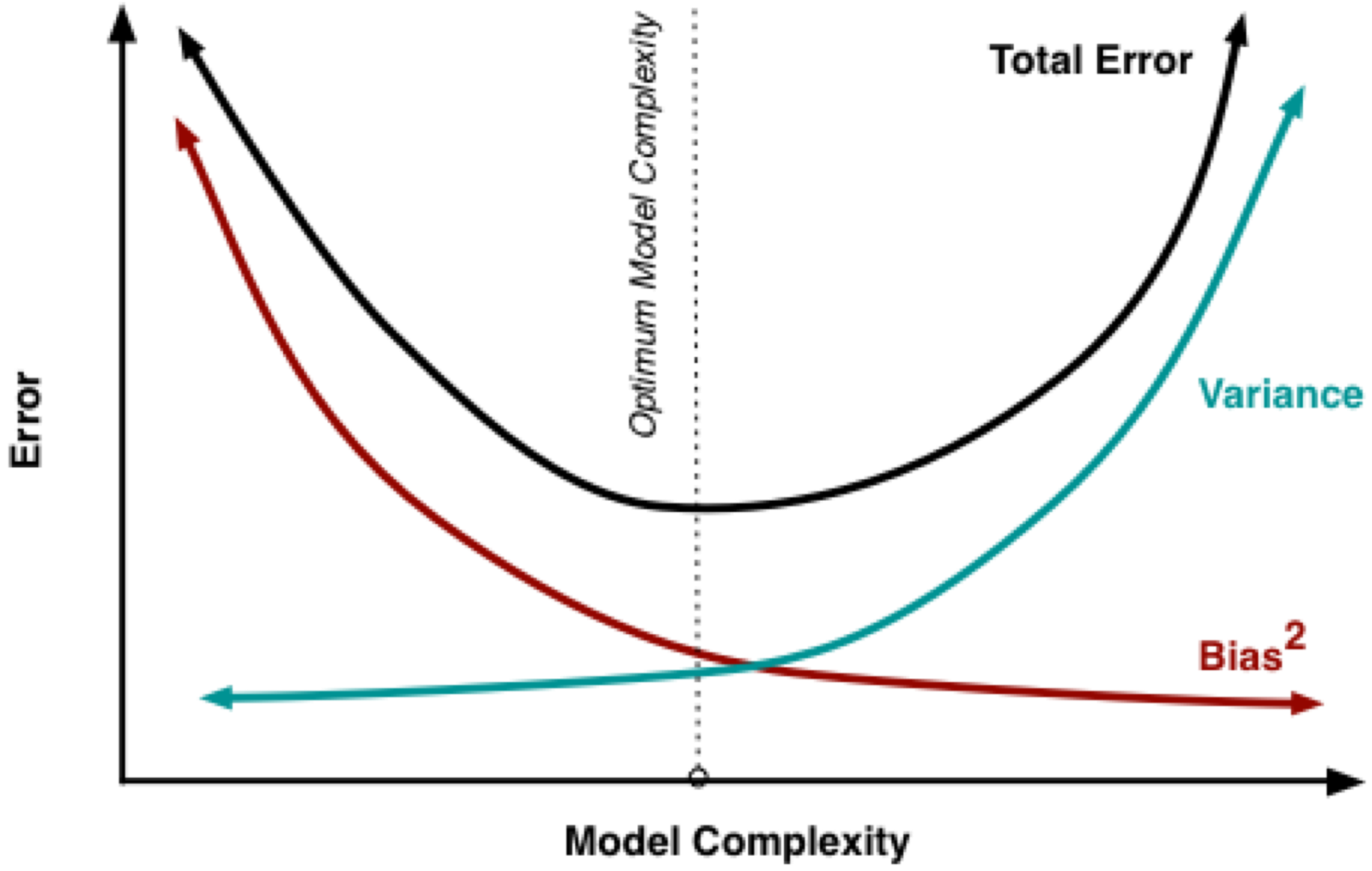
# The Bias vs Variance tradeoff

- *Generalization error can be written as the sum of three terms:*
  - *The intrinsic statistical noise in the data*
  - *the bias wrt the mean*
  - *the variance of the prediction around the mean*

$$E[(y - h(x))^2] = E[(y - \bar{y})^2] + (\bar{y} - \bar{h}(x))^2 + E[(h(x) - \bar{h}(x))^2]$$

**Noise** **Bias Squared** **Variance**

# The Bias vs Variance tradeoff



# Regularization

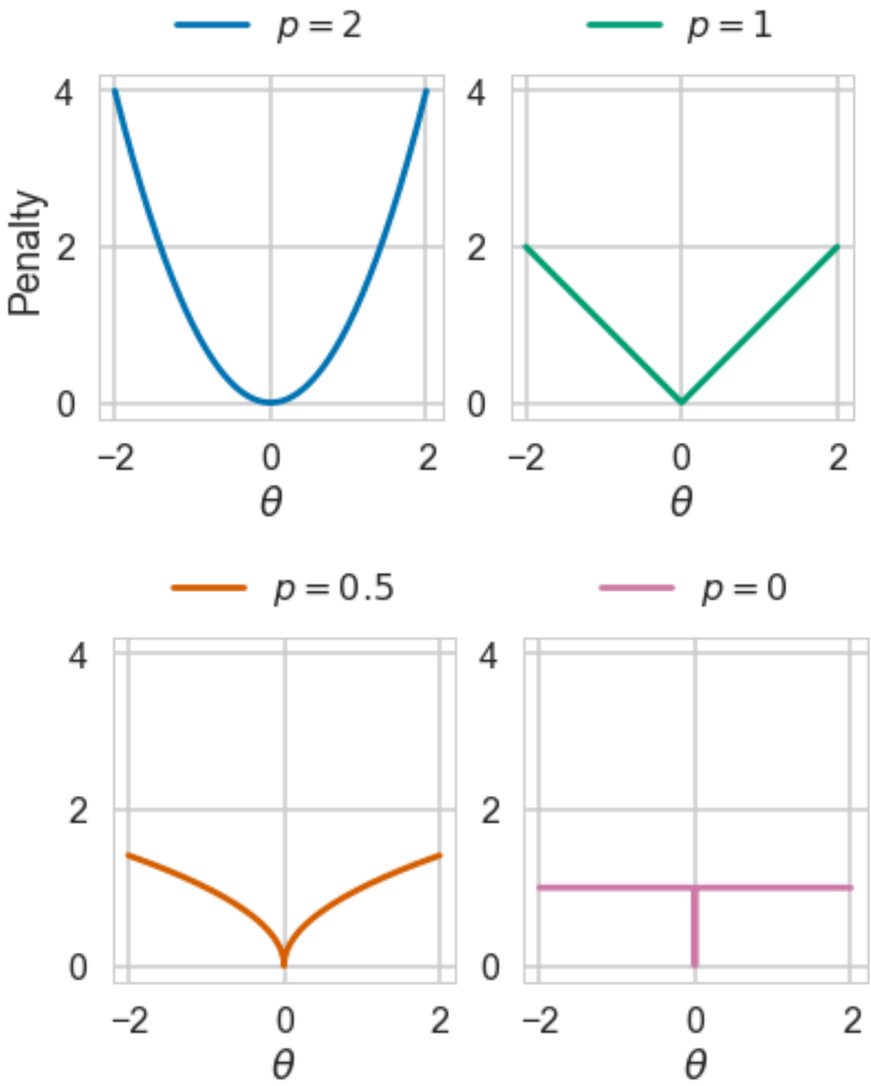
- Model complexity can be "optimized" when minimizing the loss
  - A modified loss is introduced, with a penalty term attache to each model parameter

$$L_{reg} = L + \Omega(w)$$

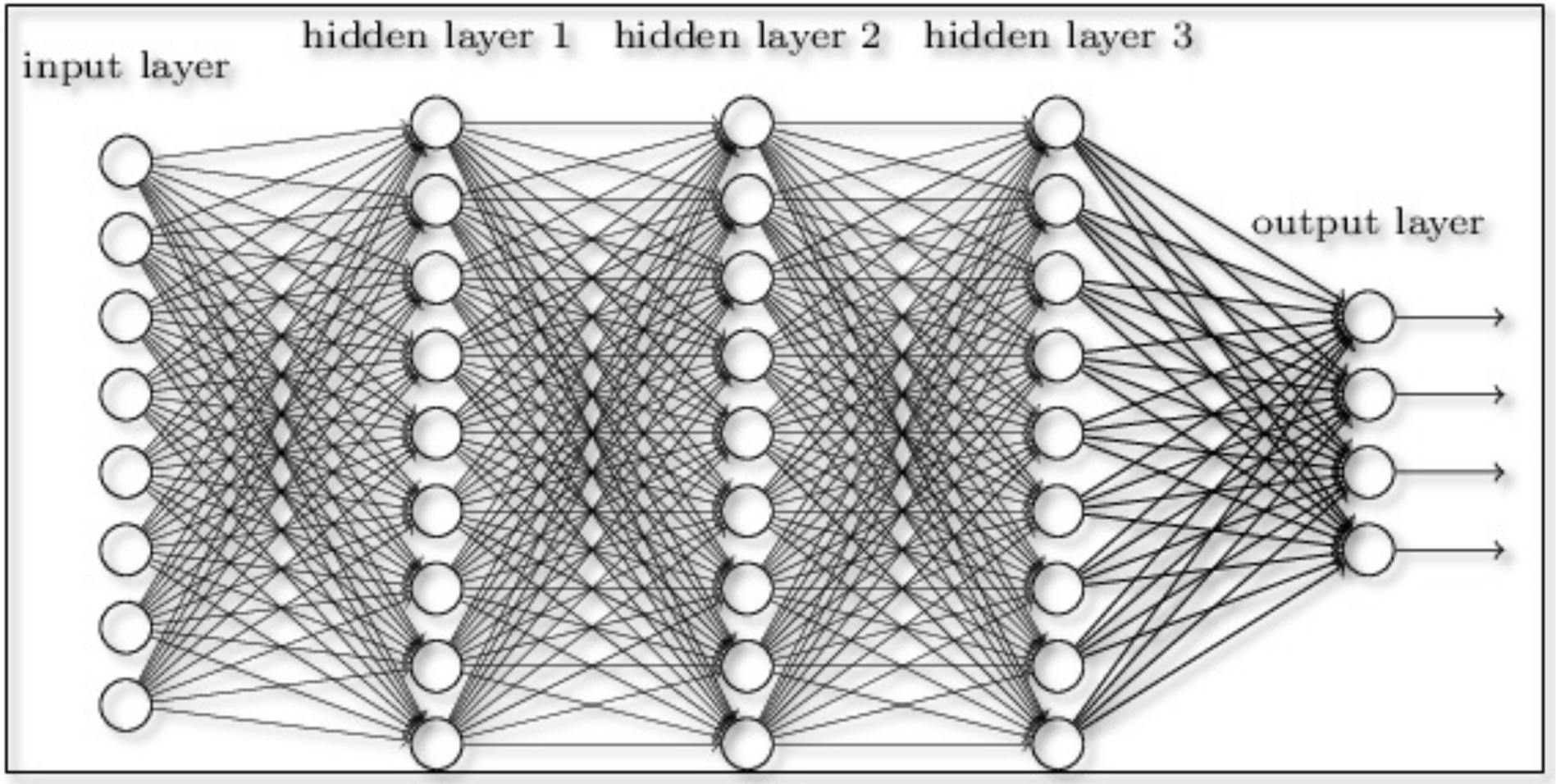
- For instance, Lp regularisation

$$L_p = \|\mathbf{w}\|^p = \sum_i |w_i|^p$$

- The minimisation is a tradeoff between:
  - pushing down the 1st term by taking advantage of the parameters
  - pushing down the 2nd term by switching off the parameters



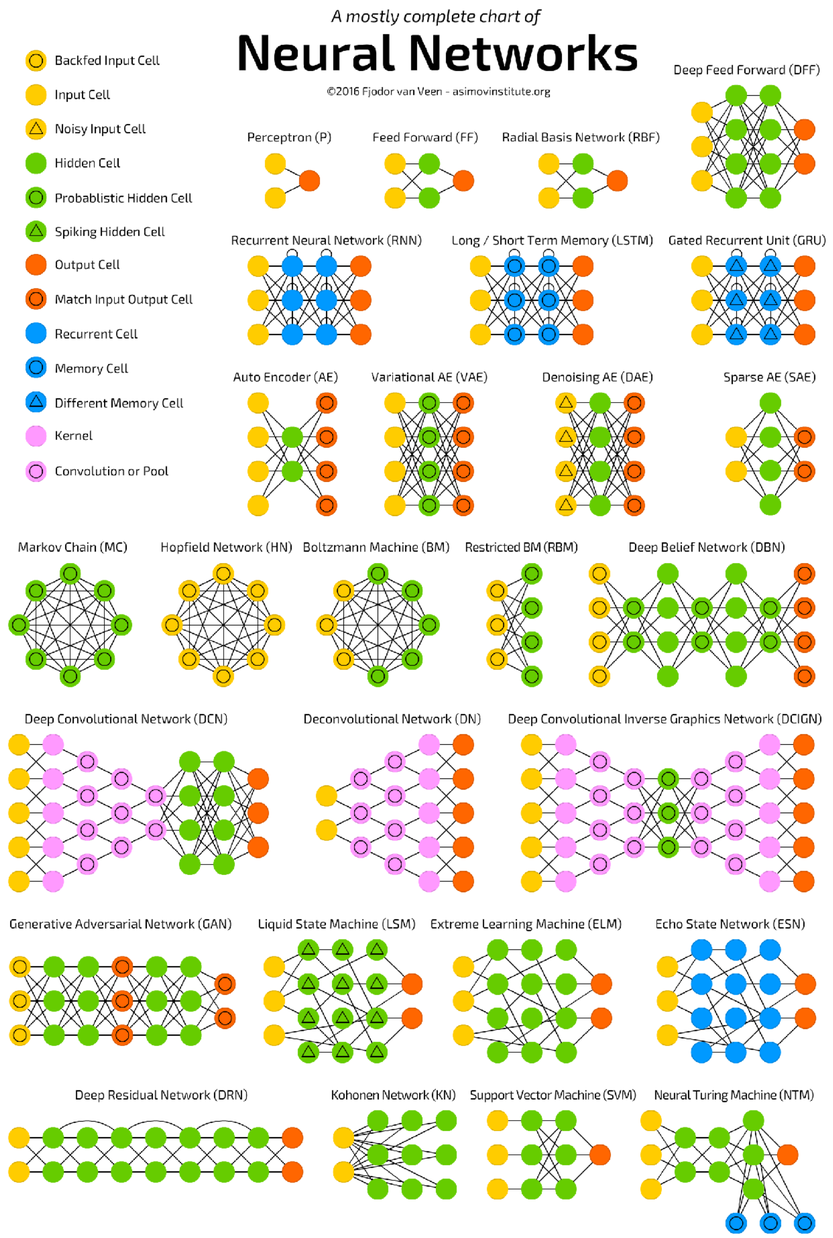

https://openreview.net/pdf?id=H1Y8hhg0b




# Deep Learning

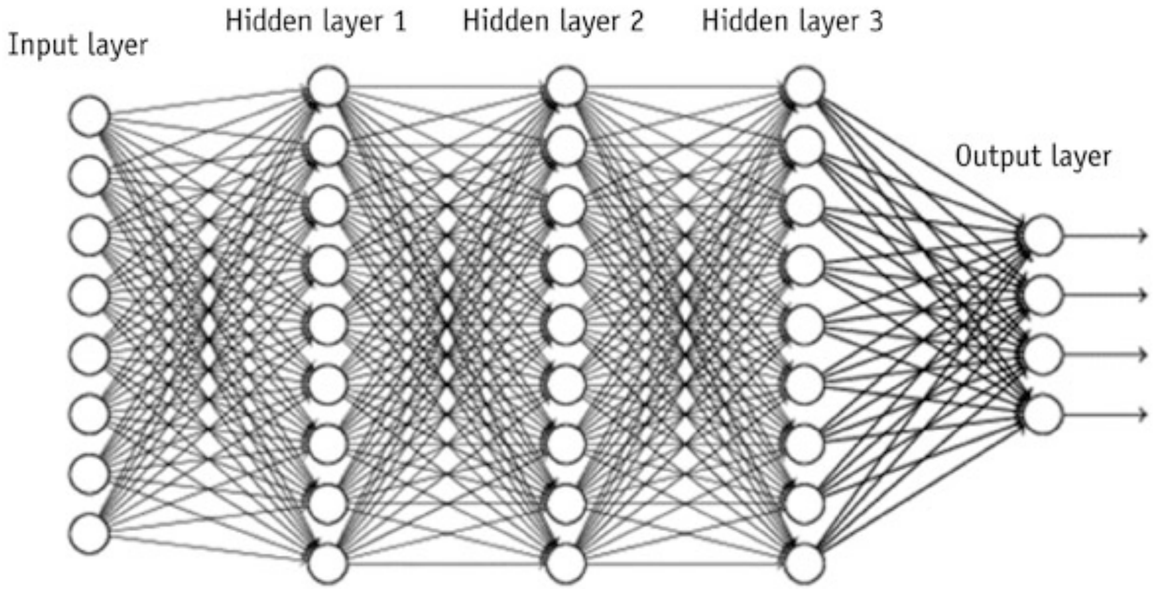
# Neural Networks in a nutshell

- NNs are (as of today) the best ML solution on the market
- NNs are usually structured in nodes connected by edges
  - each node performs a math operation on the input
  - edges determine the flow of neuron's inputs & outputs



# Deep Neural Networks

- *Deep neural networks are those with >1 inner layer*
- *Thanks to GPUs, it is now possible to train them efficiently, which boosted the revival of neural networks in the years 2000*
- *In addition, new architectures emerged, which better exploit the new computing power*



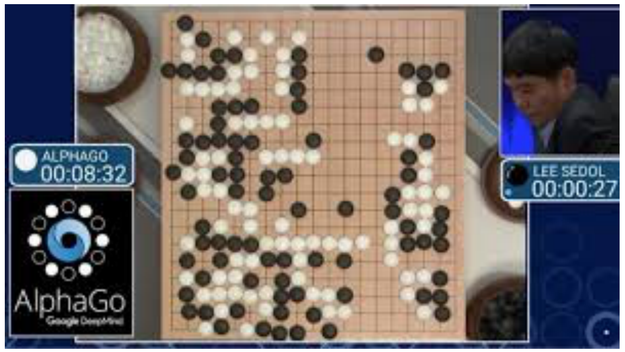

**Large-scale Deep Unsupervised Learning using Graphics Processors**

**Rajat Raina** RAJATR@CS.STANFORD.EDU
**Anand Madhavan** MANAND@STANFORD.EDU
**Andrew Y. Ng** ANG@CS.STANFORD.EDU
Computer Science Department, Stanford University, Stanford CA 94305 USA

# What is DL used for

## Image processing


## text/sound processing


## Reinforcement Learning

**Everything is a Recommendation**


**Over 75%** of what people watch comes from our recommendations

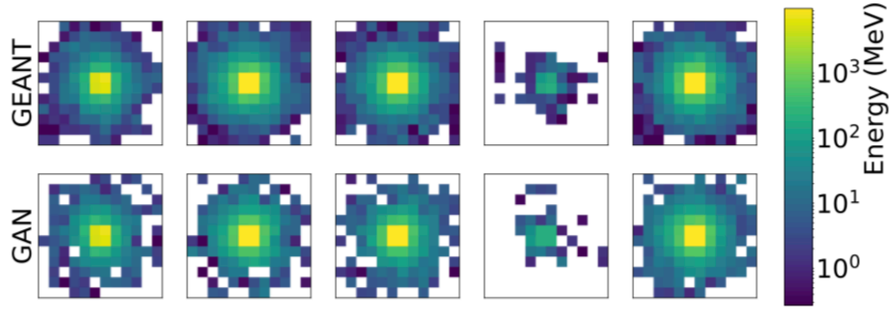
Recommendations are driven by **Machine Learning**

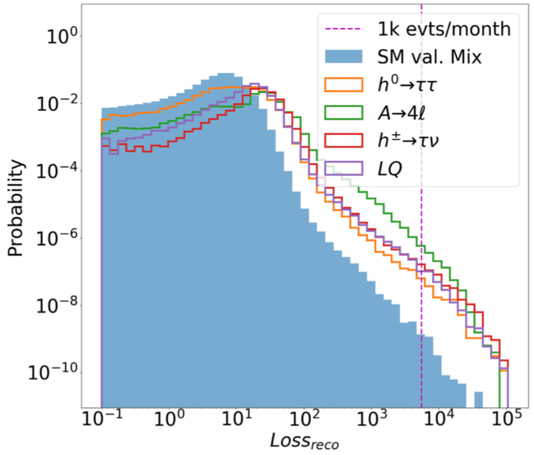
NETFLIX

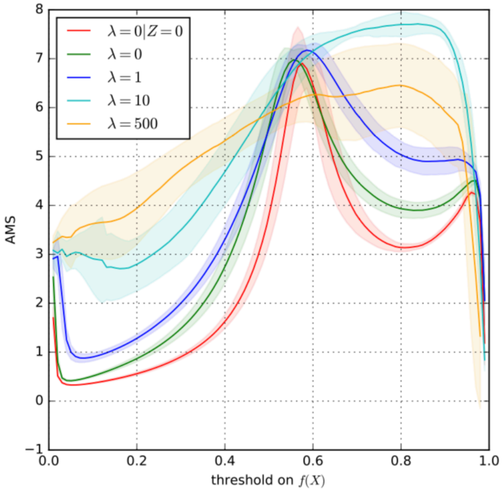
## Clustering

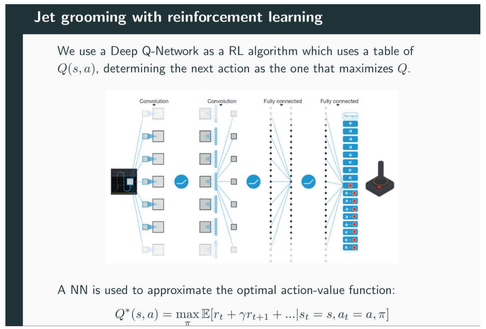
# DL, HEP, and new opportunities

- *Event Generation with generative models*
- *Anomaly Detection to search for new Physics*
- *Adversarial training for systematics*
- *Reinforcement learning for jet grooming*
- …



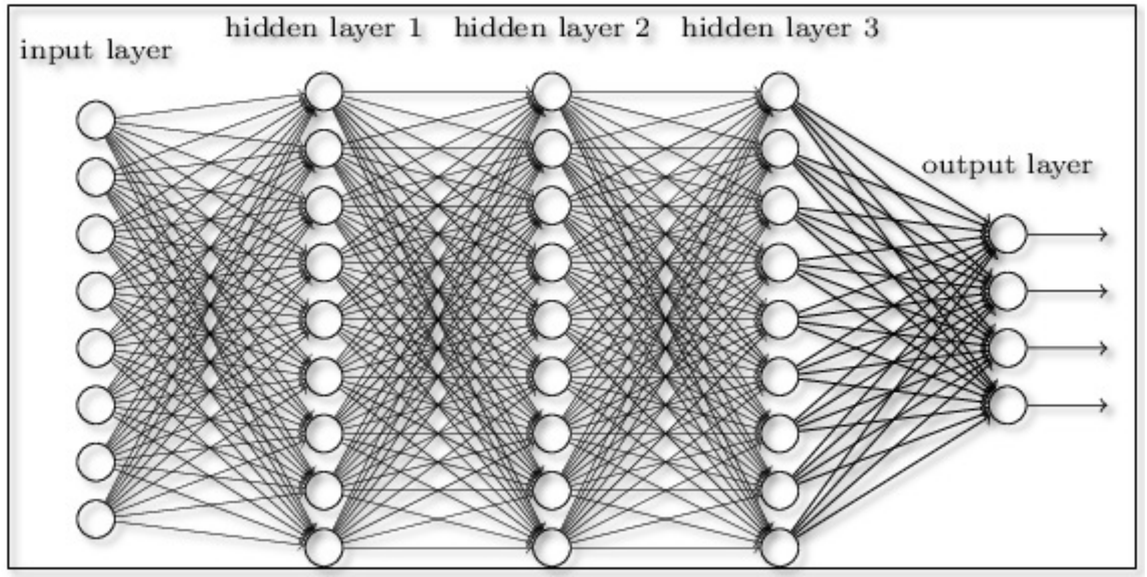



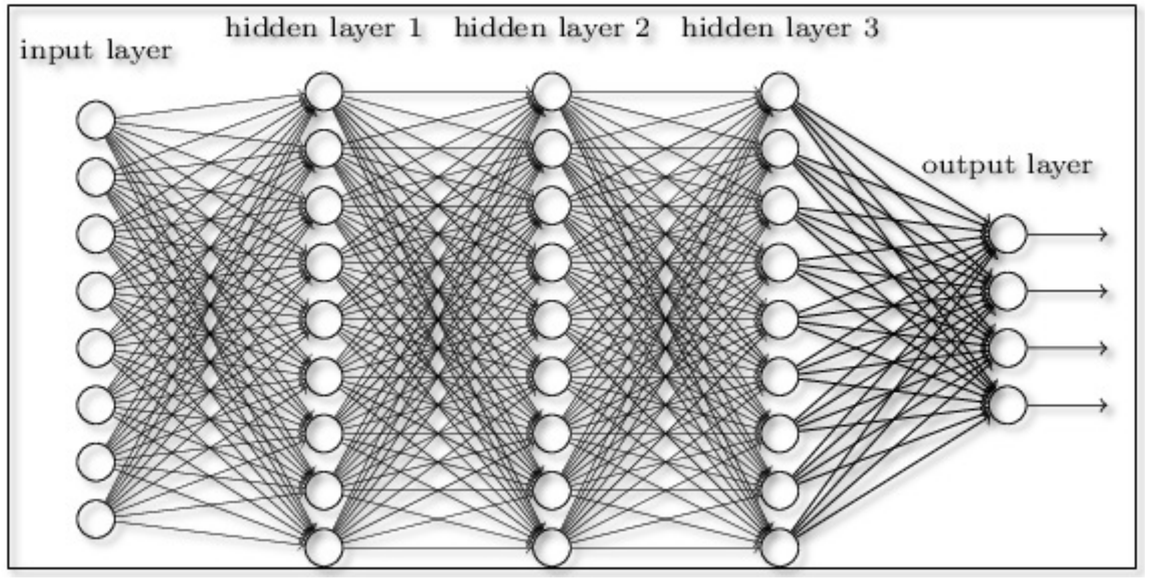
# Feed-Forward NNs

- Feed-forward neural networks have hierarchical structures:
  - inputs enter from the left and flow to the right
  - no closed loops or circularities
- Deep neural networks are FF-NN with more than one hidden layer
- Out of this "classic idea, new architectures emerge, optimised for computing vision, language processing, etc



# The role of a network node

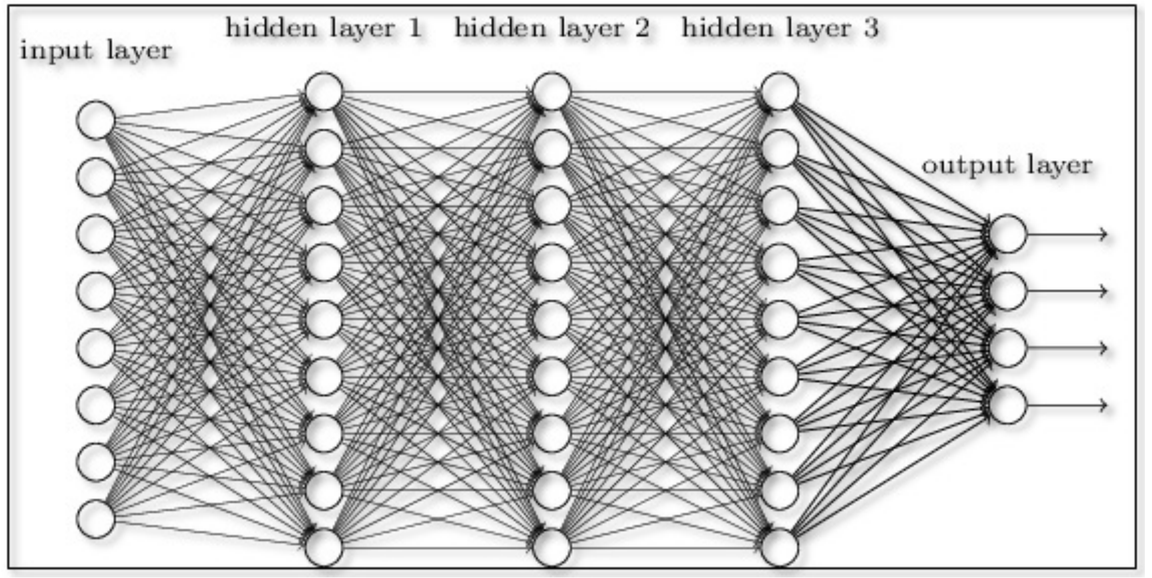
- **Each input is multiplied by a weight**
- The weighted values are summed
- A bias is added
- The result is passed to an activation function




$$w_{ij}x_j$$

# The role of a network node

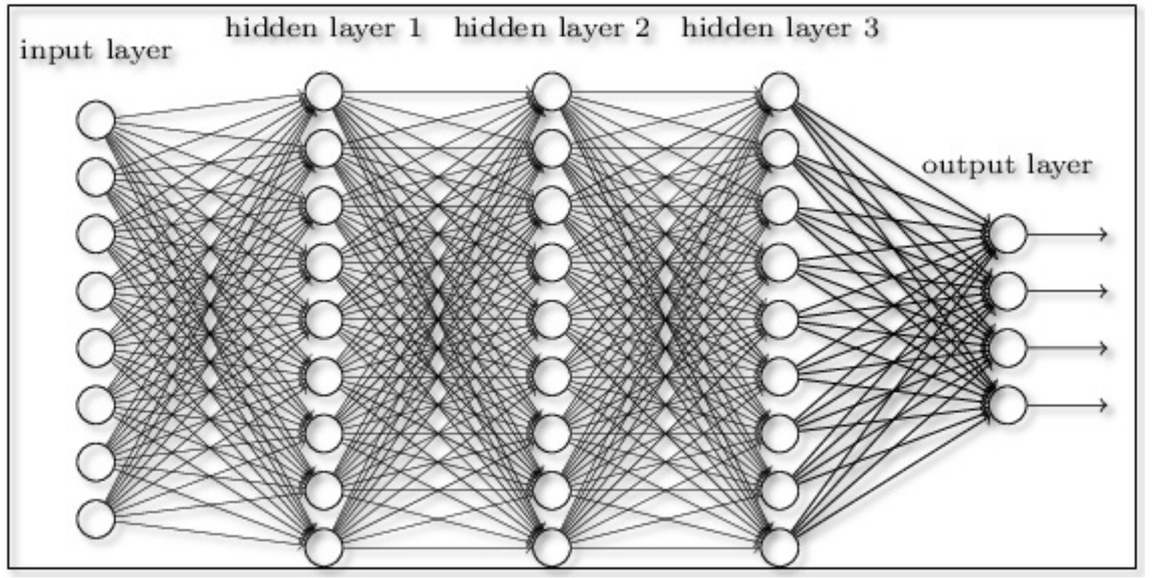
- Each input is multiplied by a weight
- **The weighted values are summed**
- A bias is added
- The result is passed to an activation function




$$\Sigma_j w_{ij} x_j$$

# The role of a network node

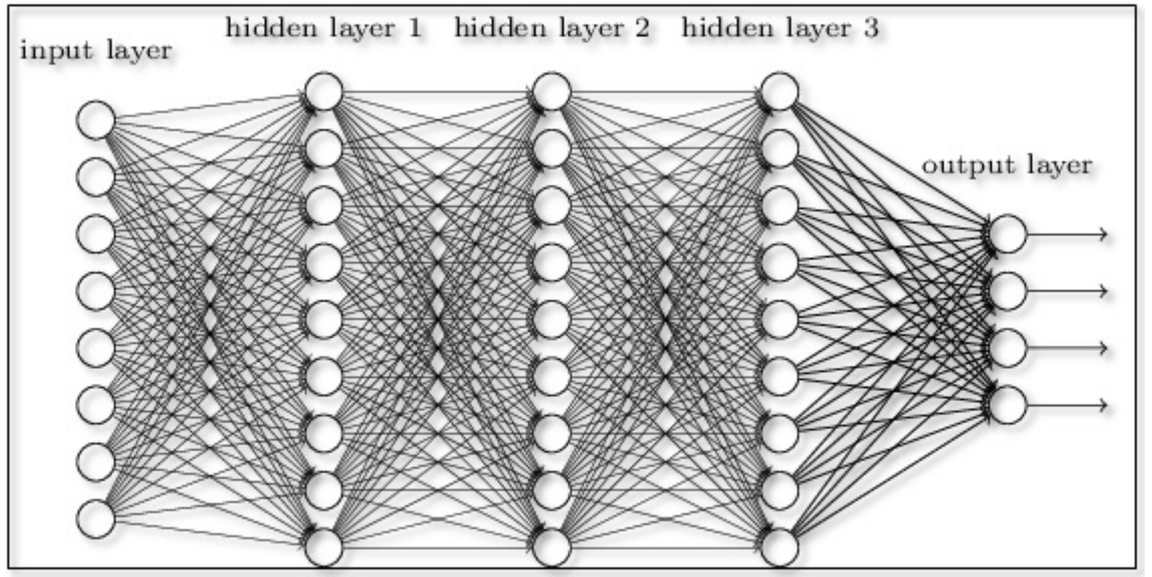
- Each input is multiplied by a weight
- The weighted values are summed
- **A bias is added**
- The result is passed to an activation function




$$\Sigma_j w_{ij} x_j + b_i$$

# The role of a network node

- Each input is multiplied by a weight
- The weighted values are summed
- A bias is added
- **The result is passed to an activation function**

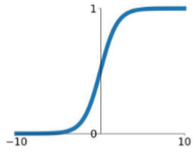



$$y_i = f(\Sigma_j w_{ij} x_j + b_i)$$

## Activation Functions

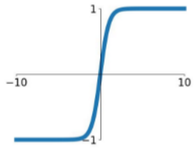
**Sigmoid**

$\sigma(x) = \frac{1}{1+e^{-x}}$



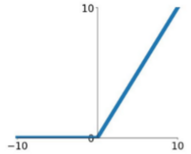
**tanh**

$\tanh(x)$



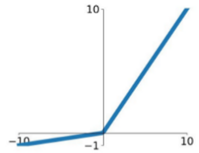
**ReLU**

$\max(0, x)$



**Leaky ReLU**

$\max(0.1x, x)$

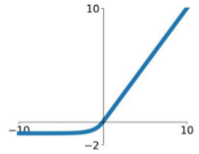


**Maxout**

$\max(w_1^T x + b_1, w_2^T x + b_2)$

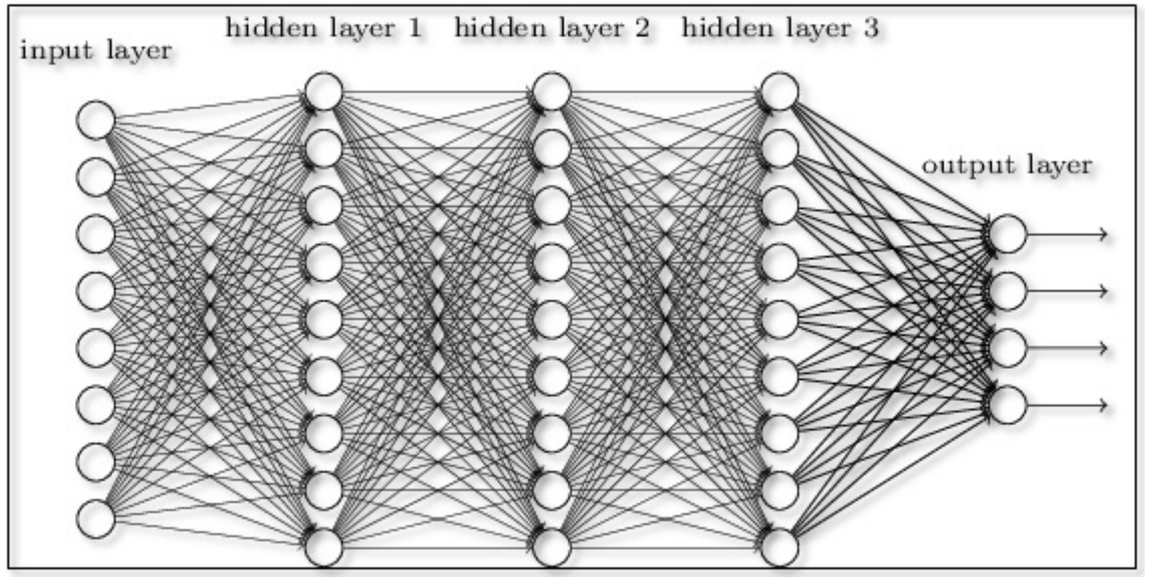
**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# The full picture

- *In a feed-forward chain, each node processes what comes from the previous layer*
- *The final result (depending on the network geometry) is K outputs, given N inputs*

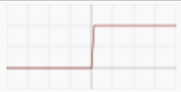
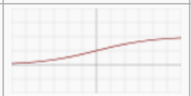
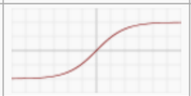
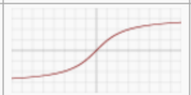
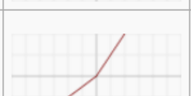



$$y_j = f^{(3)}(\Sigma_l w_{jl}^{(3)} f^{(2)}(\Sigma_k w_{lk}^{(2)} f^{(1)}(\Sigma_i w_{ki}^{(1)} x_i + b_k^{(1)}) + b_l^{(2)}) + b_j^{(3)})$$

- *One can show that such a mechanism allows to learn generic $\mathbb{R}^N \to \mathbb{R}^K$ functions*

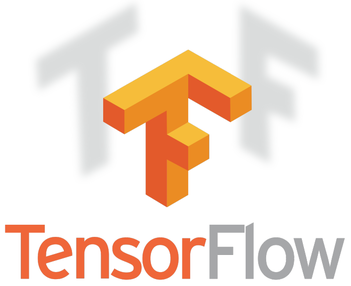
# Activation Functions

- *Activation functions are an example of network hyper parameters*
  - *they come from architecture choice, rather than from the training itself*
- *Activation output of the output layer play a special role:*
  - *it needs to return the output in the right domain*
  - *it needs to preserve the wanted features of the output (e.g., periodic, positive defined, etc.)*

| Name | Plot | Equation | Derivative |
|---|---|---|---|
| Identity |  | $f(x) = x$ | $f'(x) = 1$ |
| Binary step |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$ |
| Logistic (a.k.a Soft step) |  | $f(x) = \frac{1}{1+e^{-x}}$ | $f'(x) = f(x)(1-f(x))$ |
| TanH |  | $f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$ | $f'(x) = 1 - f(x)^2$ |
| ArcTan |  | $f(x) = \tan^{-1}(x)$ | $f'(x) = \frac{1}{x^2+1}$ |
| Rectified Linear Unit (ReLU) |  | $f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| Parameteric Rectified Linear Unit (PReLU)[2] |  | $f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| Exponential Linear Unit (ELU)[3] |  | $f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ | $f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$ |
| SoftPlus |  | $f(x) = \log_e(1 + e^x)$ | $f'(x) = \frac{1}{1+e^{-x}}$ |

# Training Libraries

- *Many solutions exist. Most popular softwares live in a python ecosystem*
  - *Google's TensorFlow*
  - *Facebook's Pytorch*
  - *Apache MXnet*
- *All of them integrated in a data science ecosystem*
  - *with numpy, scikit, etc.*
- *Convenient libraries built on top, with pre-coded ingredients*
  - *Keras for TF (this is what we will be using)*







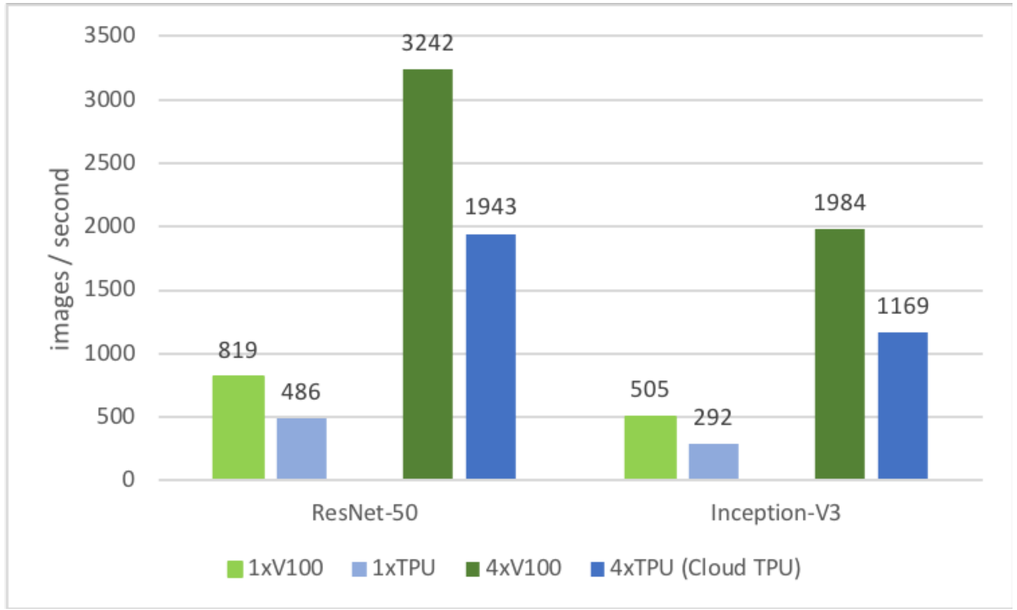


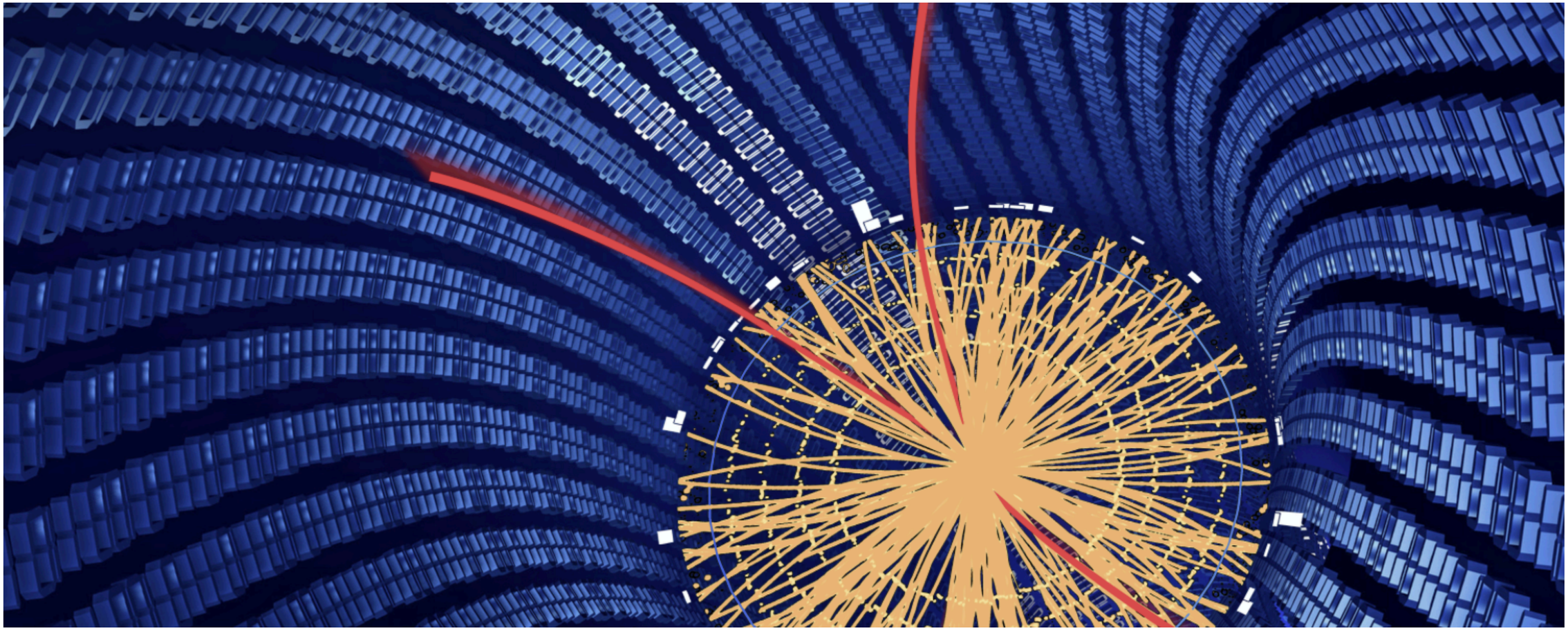


European Research Council

# GPUs & TPUs

- *All codes come with GPU support, through CUDA*
  - *They work on nVidia GPUs*
- *GPUs are very suitable to train neural networks*
  - *dedicated VRAM provides large memory to load datasets*
  - *architecture ideal to run vectorised operations on tensors*
  - *can also paralyse training tasks (e.g., processing in parallel multiple batches)*
- *A single-precision gaming card is good enough for standalone studies (200-1000 $, depending on model)*
- *Large tasks require access to clusters (with libraries for distributed training)*
- *Dedicated architectures (e.g., Google TPU) now emerging. Essentially, Deep Learning ASICs*

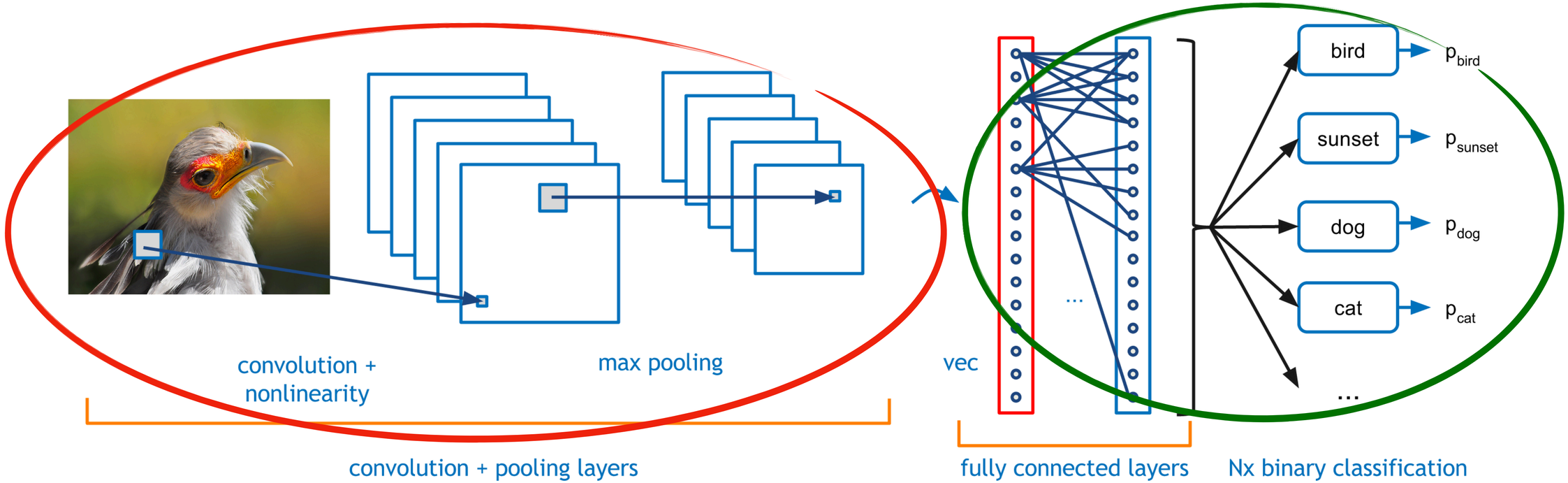




# Data Representation

# Deep Neural Network in a nutshell

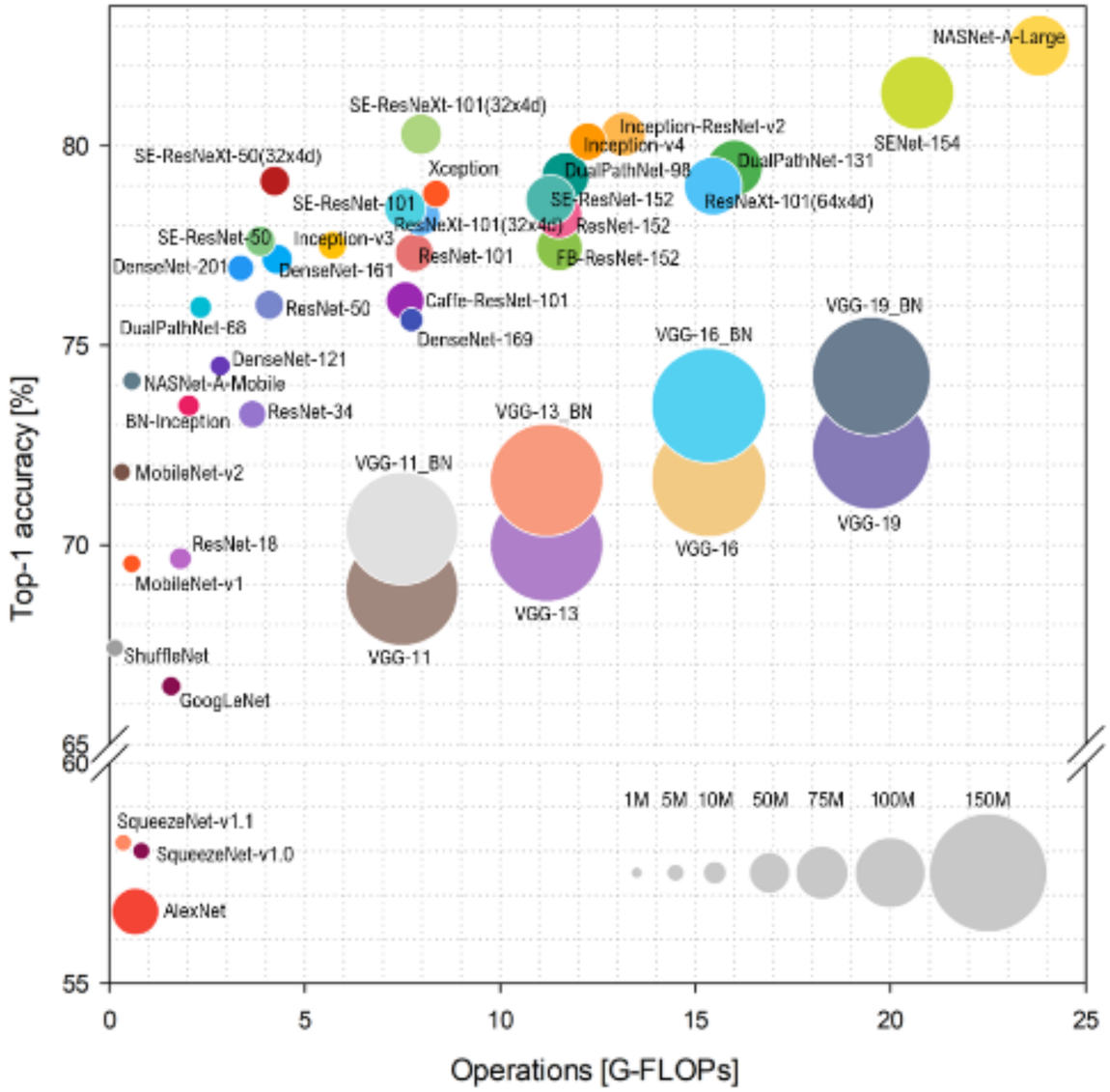
- *DNNs typically rely on two phases:*
  - ***Feature engineering*** *from Raw Data. This is where new & exotic architectures (depending on data type) take the best out of your data*




  - ***Task solving****: start from engineered features and solve the task (classification, regression, etc.)*

# Deep Learning & Computing Vision

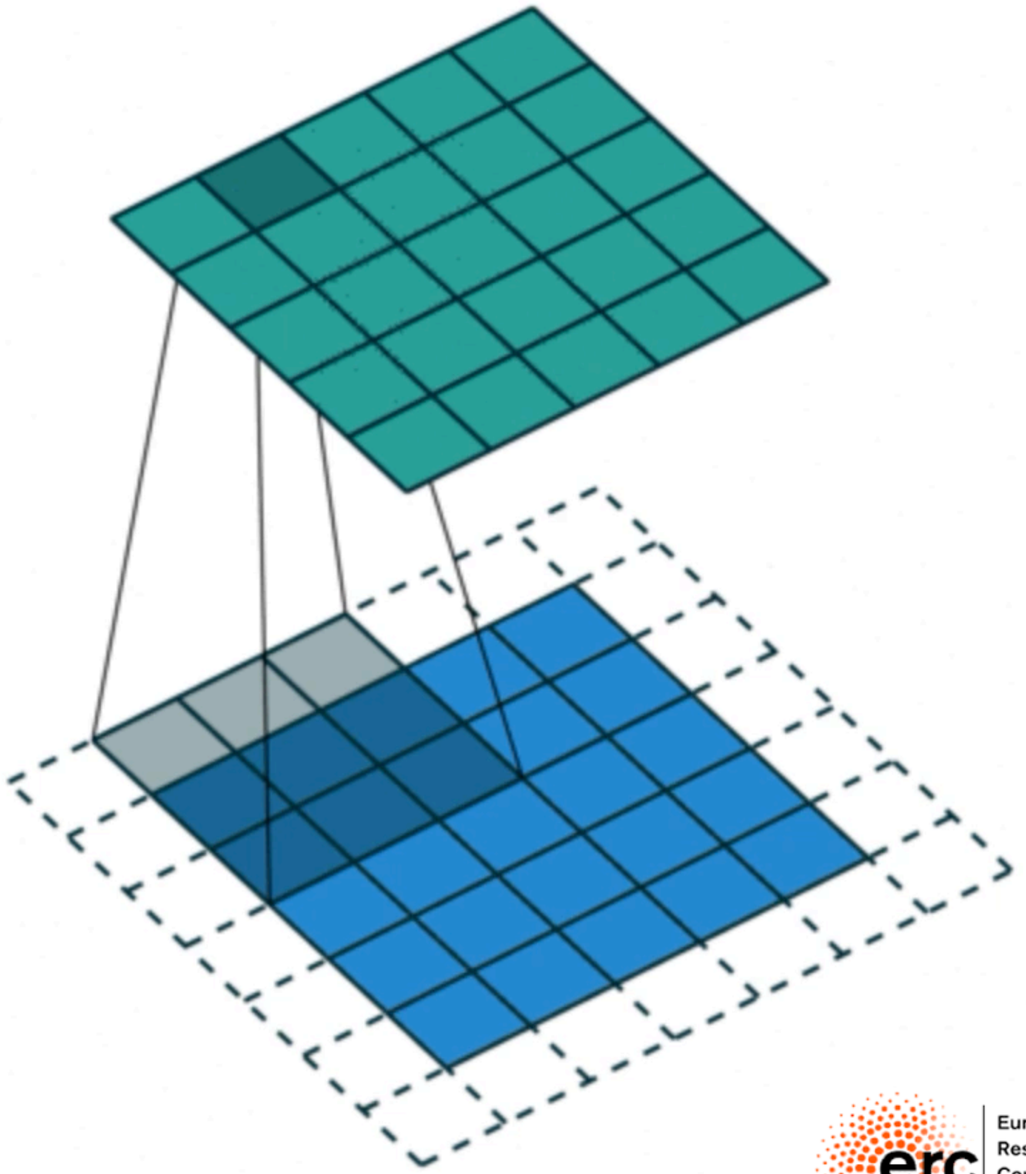
- *The most evident success of Deep Learning is computing vision with Convolutional NNs*
  - *A kernel scans an array of pixels*
  - *The network is translation invariant*
  - *The network knows which pixels are near each other and learns from there*




Research Council

# Deep Learning & Computing Vision

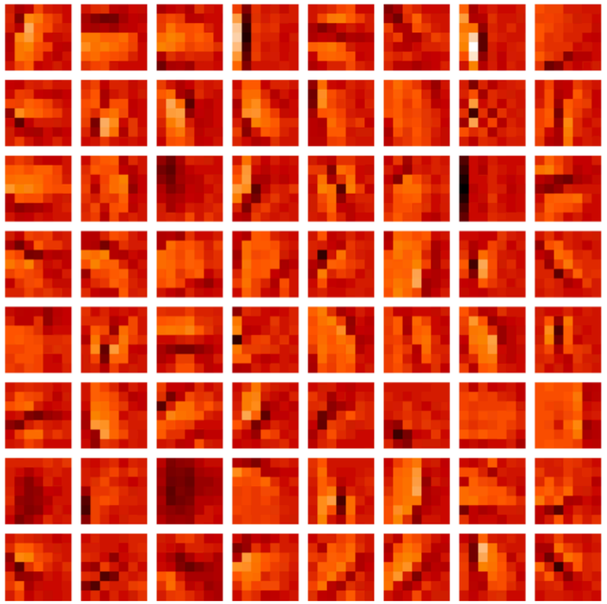
- *The most evident success of Deep Learning is computing vision with Convolutional NNs*
  - *A kernel scans an array of pixels*
  - *The network is translation invariant*
  - *The network knows which pixels are near each other and learns from there*



European Research Council

# CNN in Science

- *Paradigm applied successfully to many scientific problems*
  - *Exoplanet detection*
  - *Frequency-domain analysis of Gravitational Interferometer data*
  - *Neutrino detection*
  - *etc…*



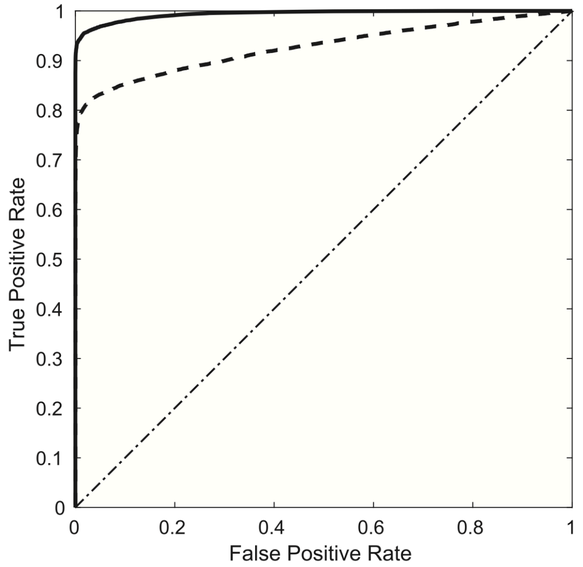
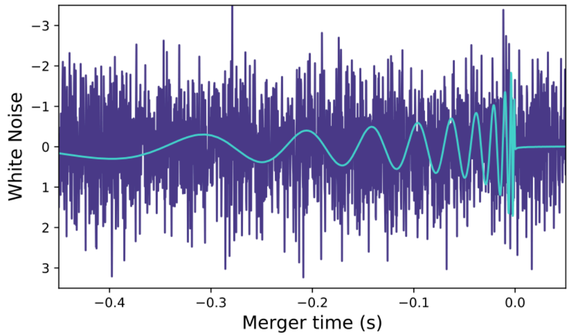
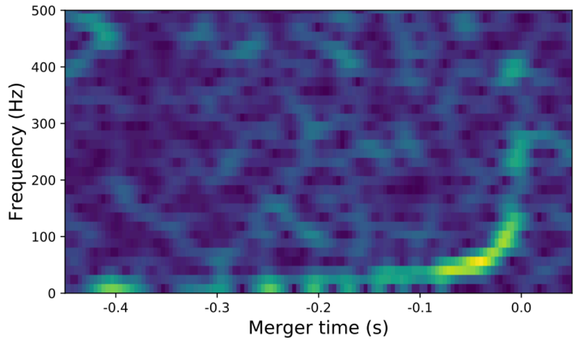



**Figure 2.** Receiver Operating Characteristic (ROC) curve for the neural network and the data set presented in this work. The dashed line represents the performance of the BLS preceded by a high-pass filter. The dotted–dashed line is the so-called "no-discrimination" line, corresponding to random guess.



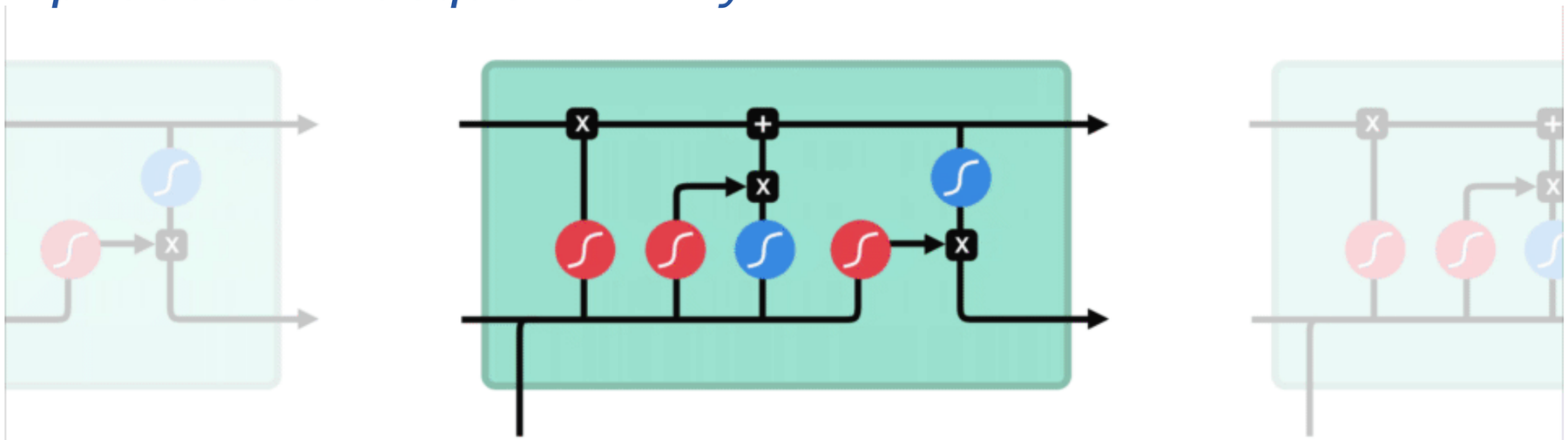

European Research Council

# Deep Learning & Natural Language

- *Natural language processing is another big success of Deep Learning*
- *Based on recurrent neural networks*
  - *data ordered (time sequence, words in sentence, etc.)*
  - *data processed sequentially*

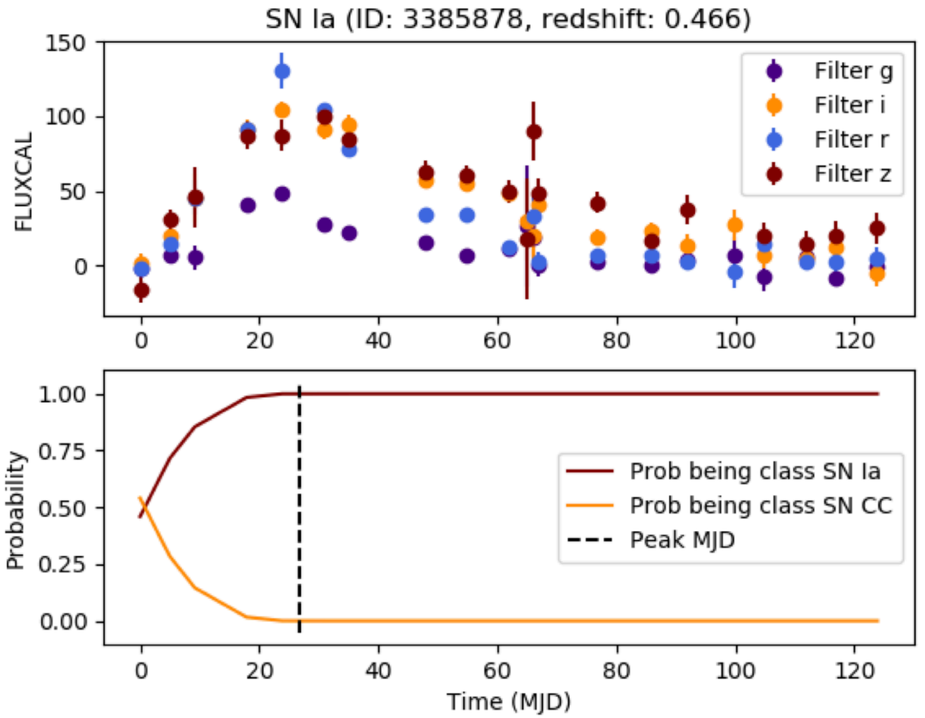
# Deep Learning & Natural Language

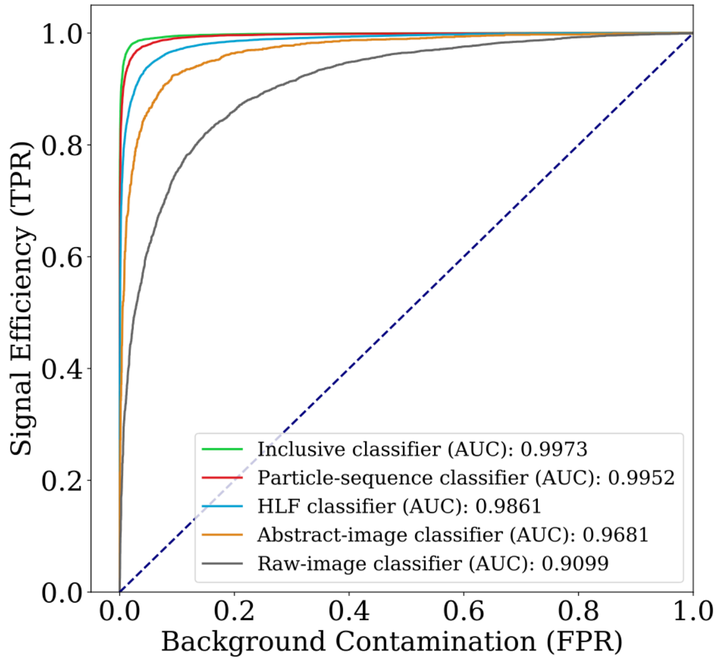
- *Natural language processing is another big success of Deep Learning*
- *Based on recurrent neural networks*
  - *data ordered (time sequence, words in sentence, etc.)*
  - *data processed sequentially*

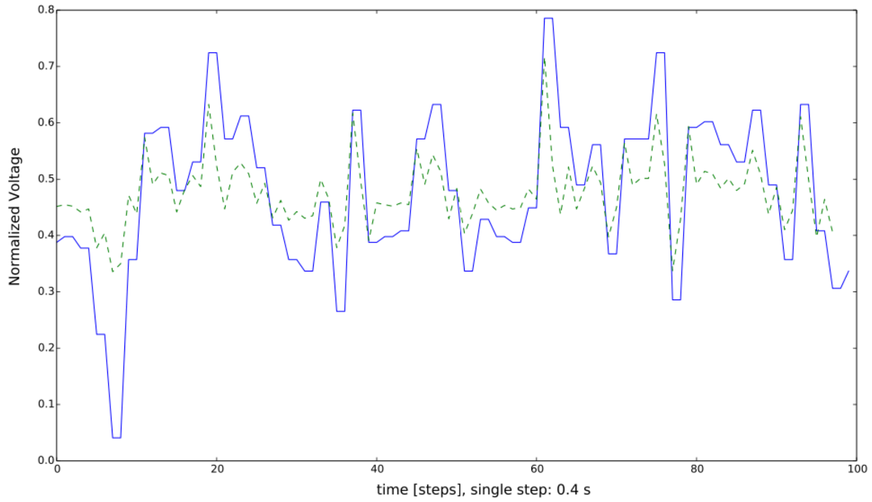


# RNN in Science

- *RNNs can be used to monitor time sequences and look for transient events*
  - *Supernovae detection*
  - *Monitoring LHC magnets*
  - *Event classification at LHC*
  - *…*



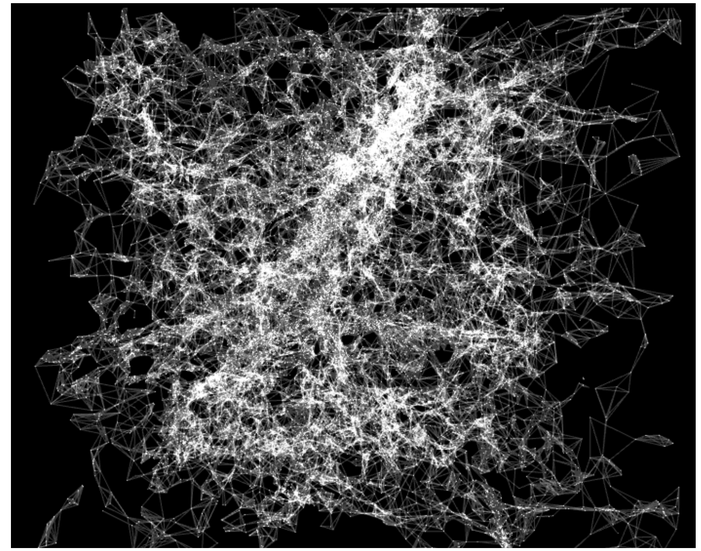



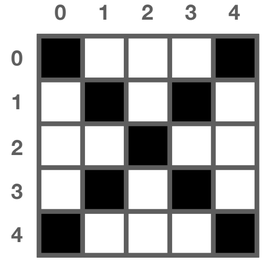




European Research Council

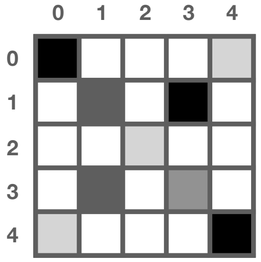
# What about irregular data?

- *Unfortunately, many scientific domains deal with data which are not regular arrays (neither images nor sequences)*
  - *Galaxies or star populations in sky*
  - *Sensors from HEP detector*
  - *Molecules in chemistry*
- *These data can all be seen as sparse sets in some abstract space*
  - *each element of the set being specified by some array of features*
  - *geometrical coordinates could be some of these features*

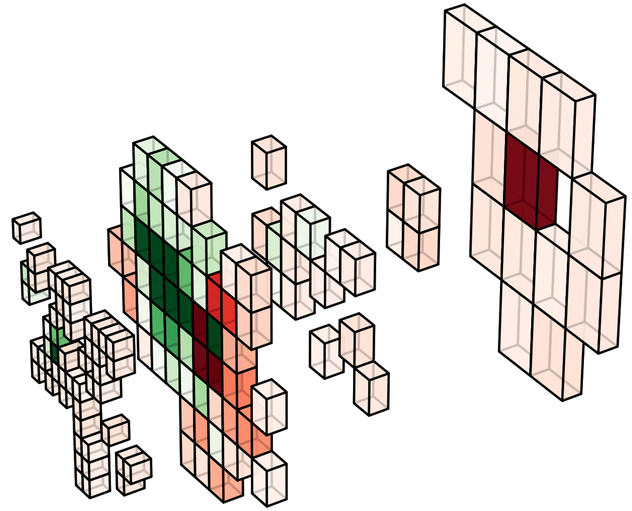





| 0 | 0 | 1 | 1 | 2 | 3 | 3 | 4 | 4 |
|---|---|---|---|---|---|---|---|---|
| 0 | 4 | 1 | 3 | 2 | 1 | 3 | 0 | 4 |



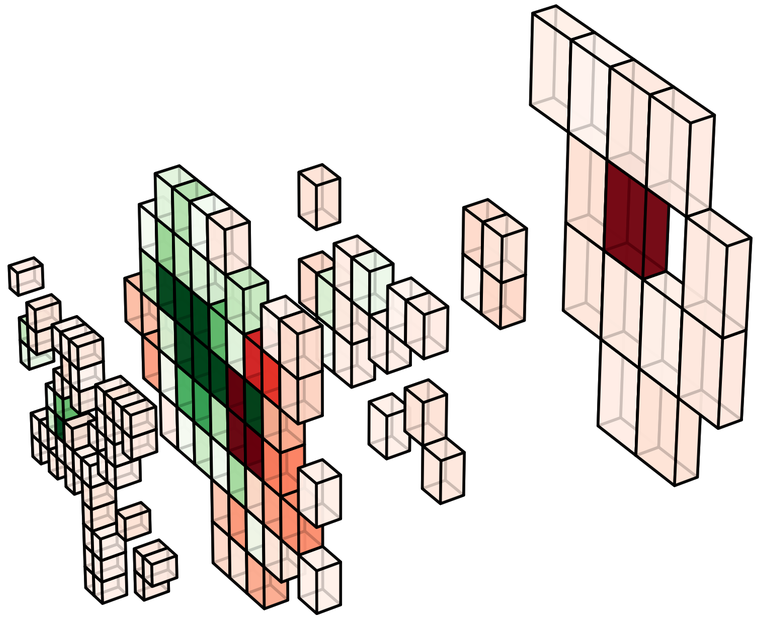

| 0 | 0 | 1 | 1 | 2 | 3 | 3 | 4 | 4 |
|---|---|---|---|---|---|---|---|---|
| 0 | 4 | 1 | 3 | 2 | 1 | 3 | 0 | 4 |
| 1.00 | 0.25 | 0.75 | 1.00 | 0.25 | 0.75 | 0.50 | 0.25 | 1.00 |

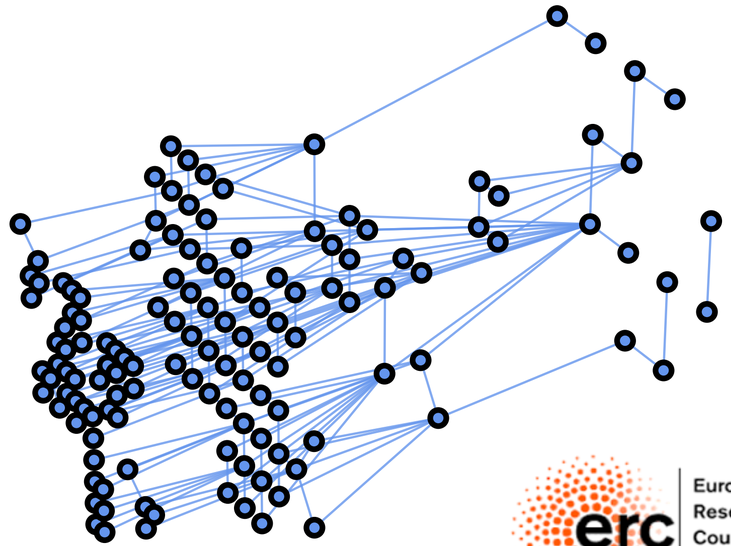


European Research Council

# From Sets to Graphs

- *Given such a set, we want to generalise the image representation as regular array that is fed to a CNN*
  - *Once that is done, we can generalise CNN itself*
- *For images, a lot of information is carried by pixels being next to each other. A metric is intrinsic in the data representation as image*
- *With a set, we need to specify a metric that tell us who is close to who in the abstract space of features that we have at hand*
  - *SOLUTION: connect elements of sets and learn (e.g., with a neural network) from data which connections are relevant*

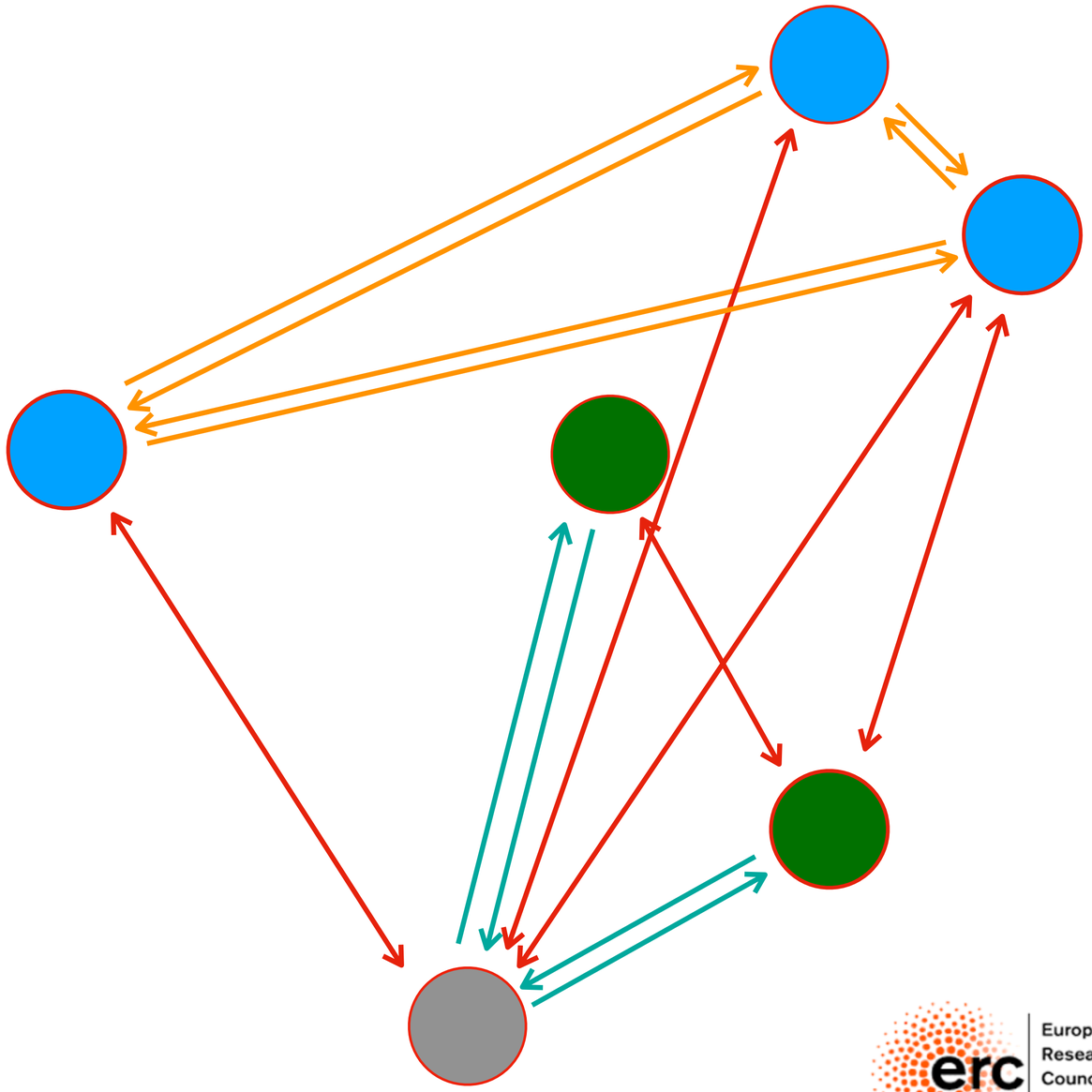




European Research Council

# Building the Graph

- *Each element of your set is a vertex V*
- *Edges E connect them*
  - *Edges can be made directional*
  - *Graphs can be fully connected ($N^2$)*
  - *Or you could use some criterion (e.g., nearest k neighbours in some space) to reduce number of connections*
  - *if more than one kind of vertex, you could connect only Vs of same kind, of different kind, etc*
- *The (V,E) construction is your graph. Building it, you could enforce some structure in your data*
  - *If you have no prior, then go for a directional fully connected graph*



# Summary of Part 1

- *ML models are adaptable algorithms that are trained (and not programmed) to accomplish a task*
- *The training happens minimizing a loss function on a given sample*
- *The loss function has a direct connection to the statistical properties of the problem*
- *Deep Learning is the most powerful class of ML algorithms nowadays*
- *Several architectures exist, tuned on different event representations*

# References


- Michael Kagan, CERN OpenLab classes on Machine Learning
  - Source of inspiration for this first lesson
- Pattern Recognition and Machine learning (Bishop)
- I. Goodfellow and Y. Bengio and A. Courville, "Deep Learning" MIT press


- Main reference for tutorial exercise: https://arxiv.org/abs/1908.05318
- All notebooks and classes are/will be on GitHub: https://github.com/pierinim/tutorials/tree/master/GGI_Jan2021
- Full dataset available at: https://zenodo.org/record/3602260