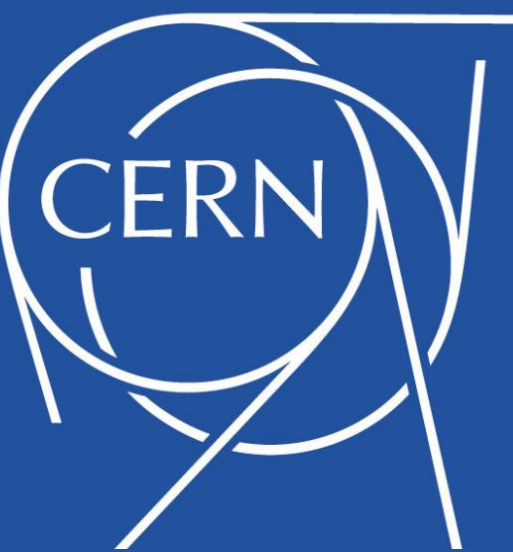
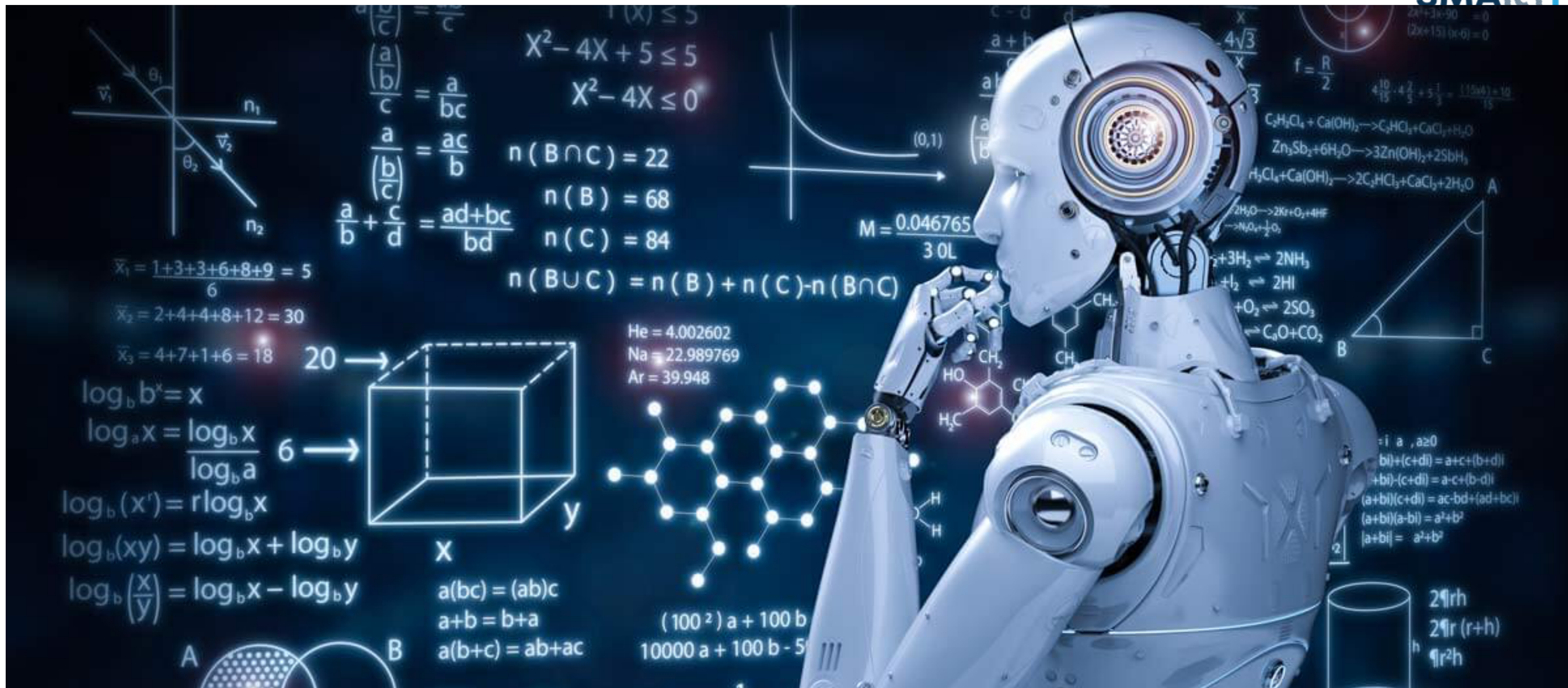


Lecture 2: Introduction to Machine Learning



Maurizio Pierini

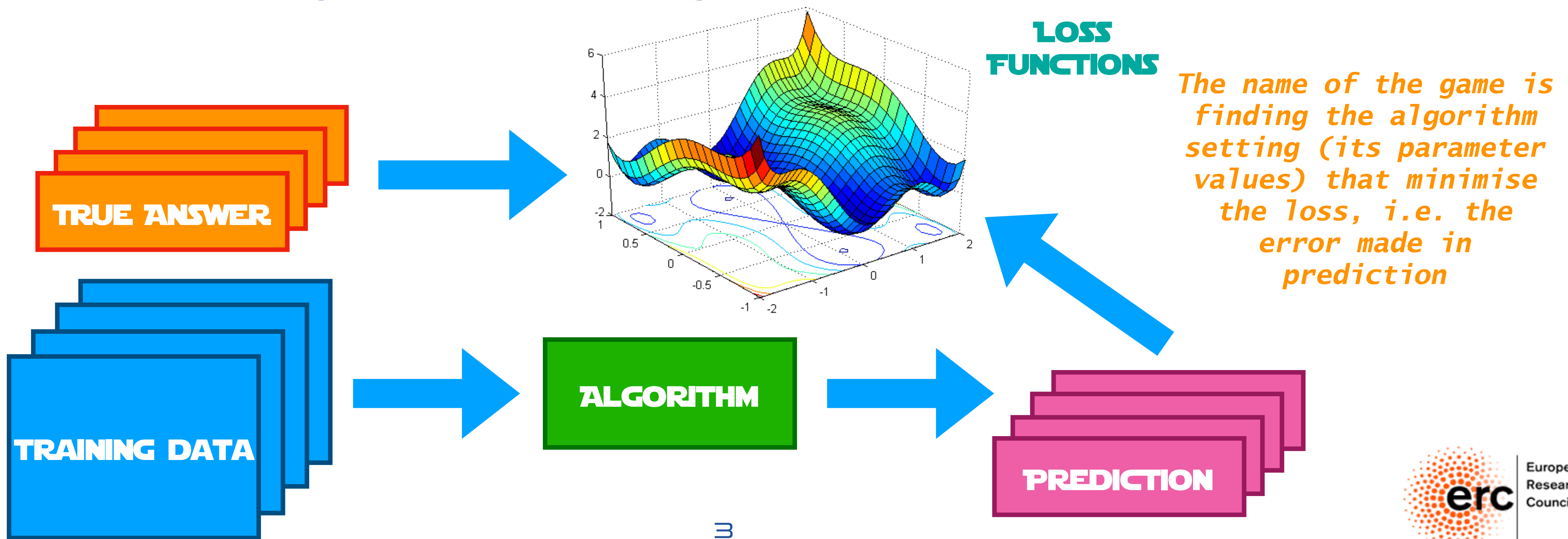




What is Machine Learning ?

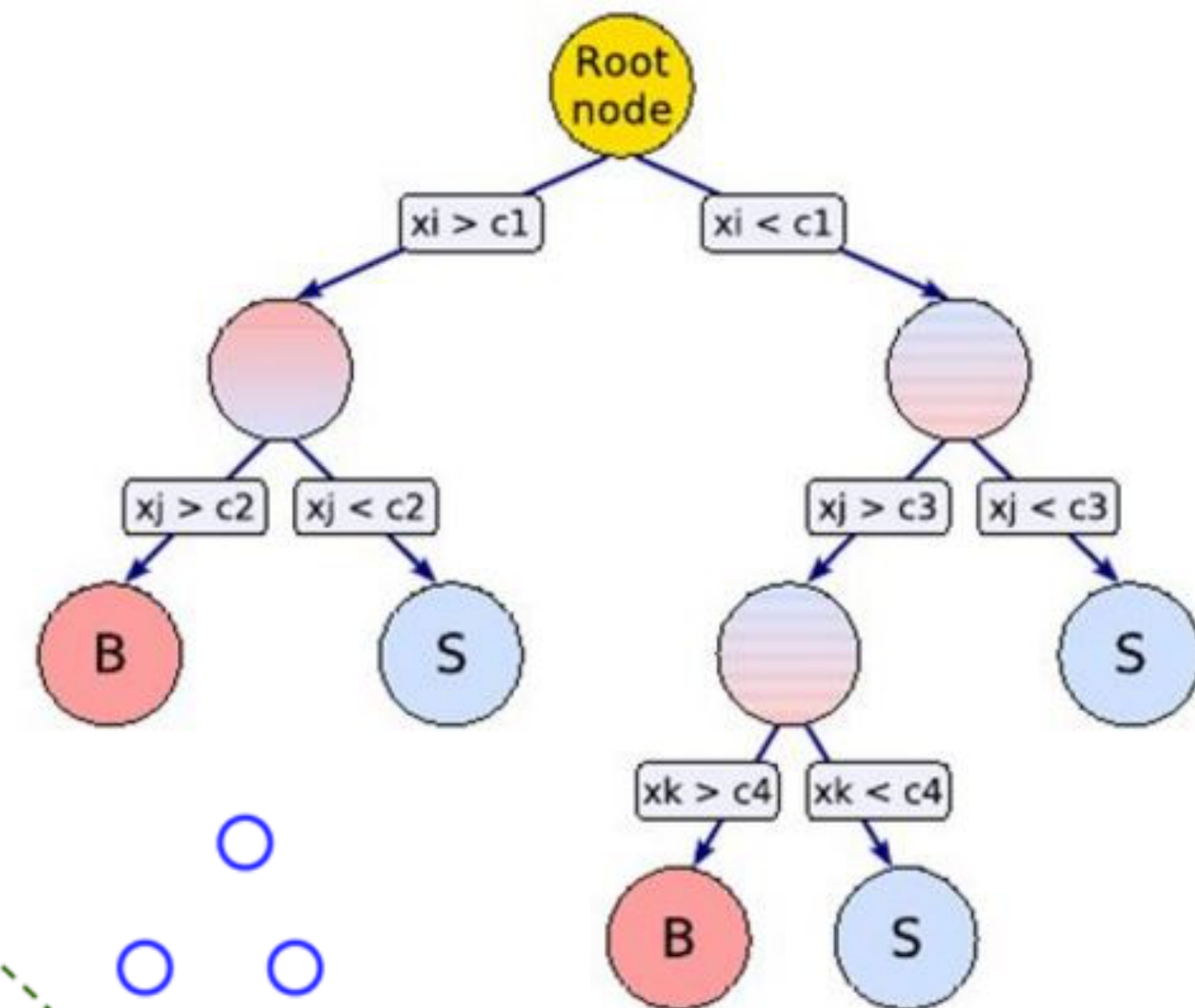
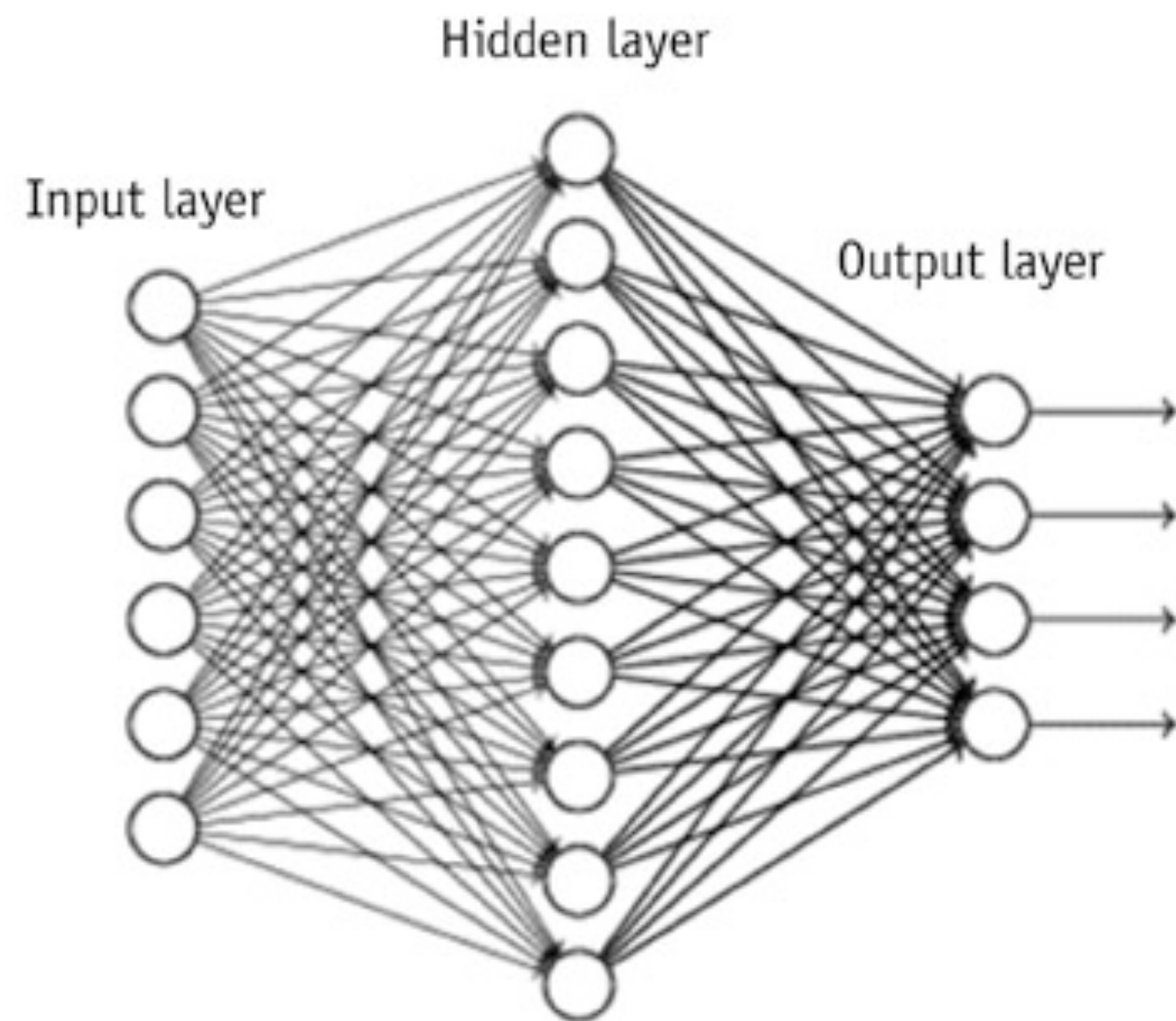
A definition (Wikipedia)

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.



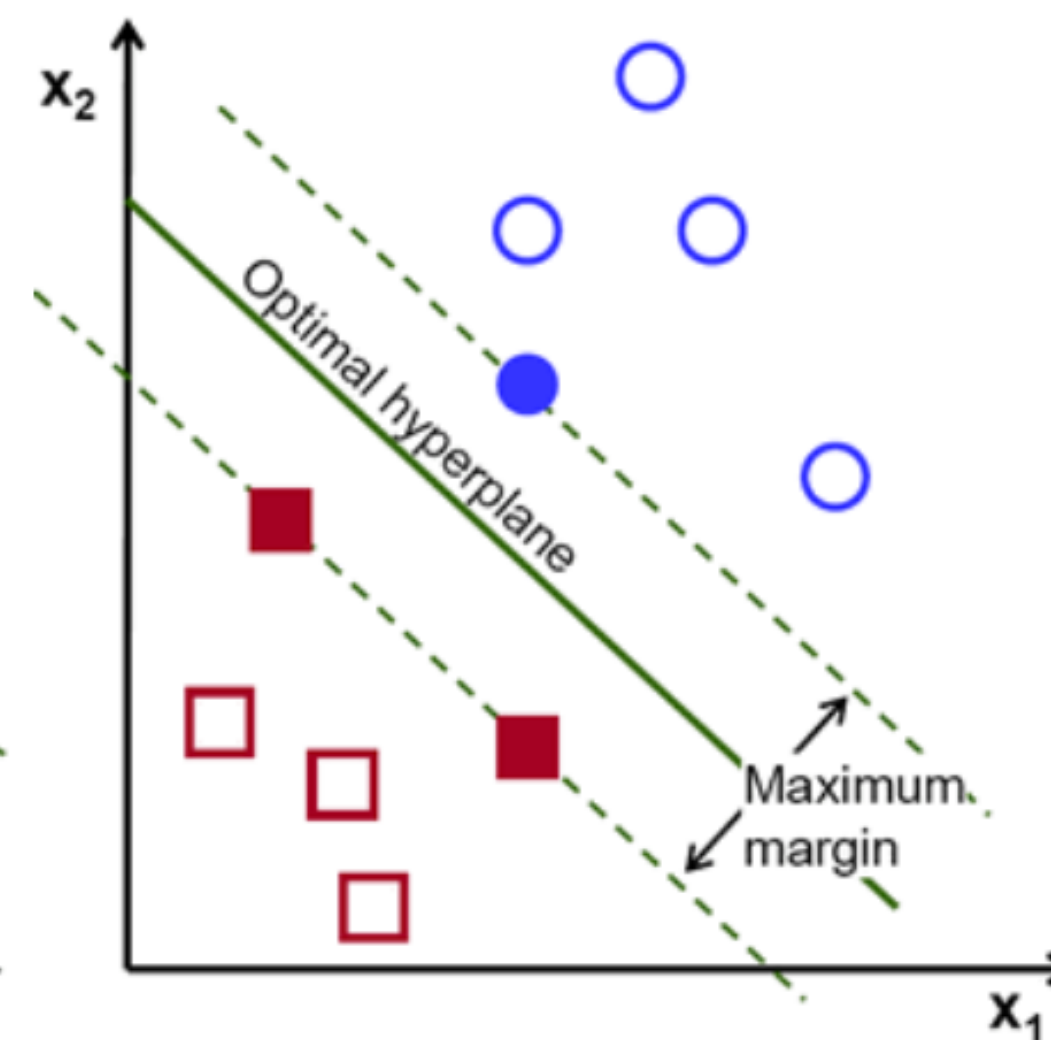
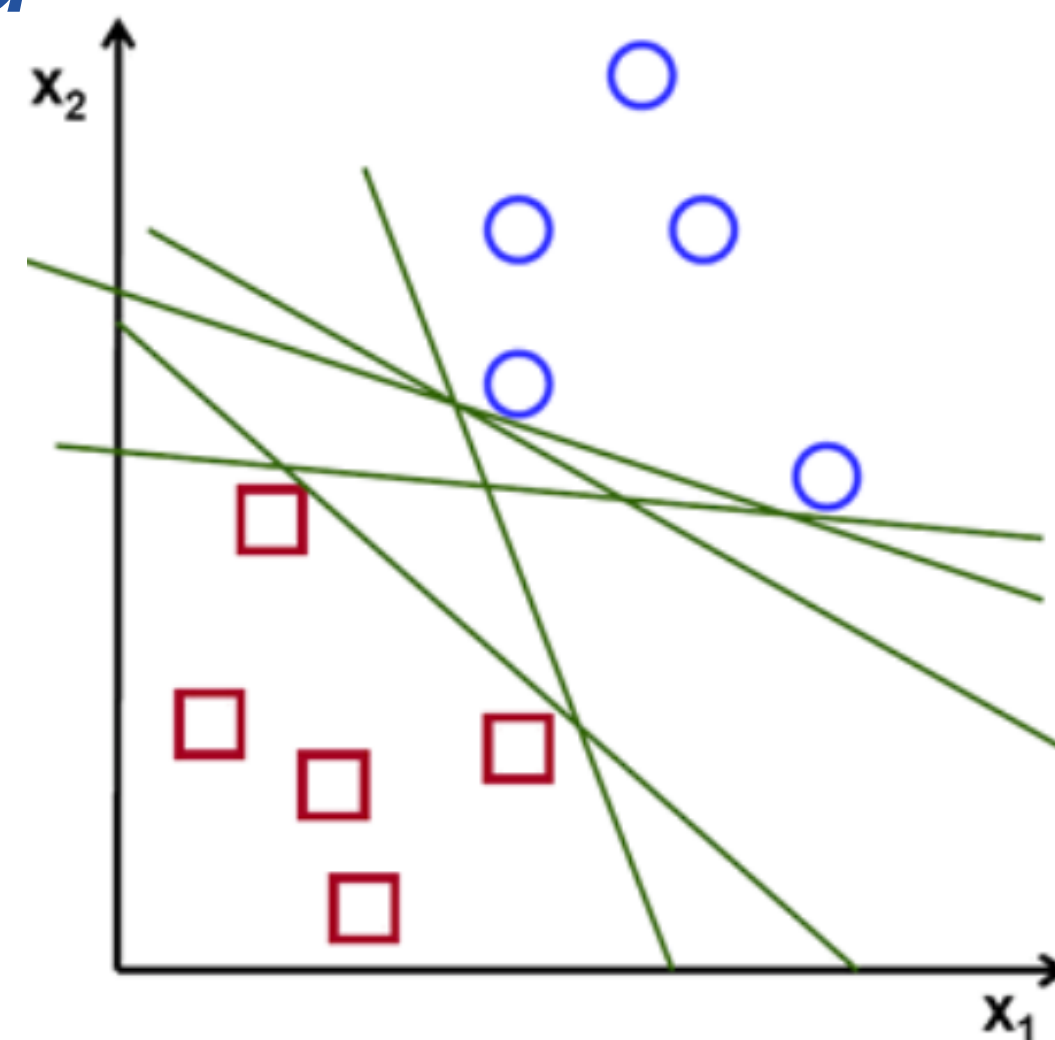
- *Different ML algorithms had their moment of glory*

- *(Shallow) neural networks dominated in the 80's*



- *Alternatives emerged in the 90's*

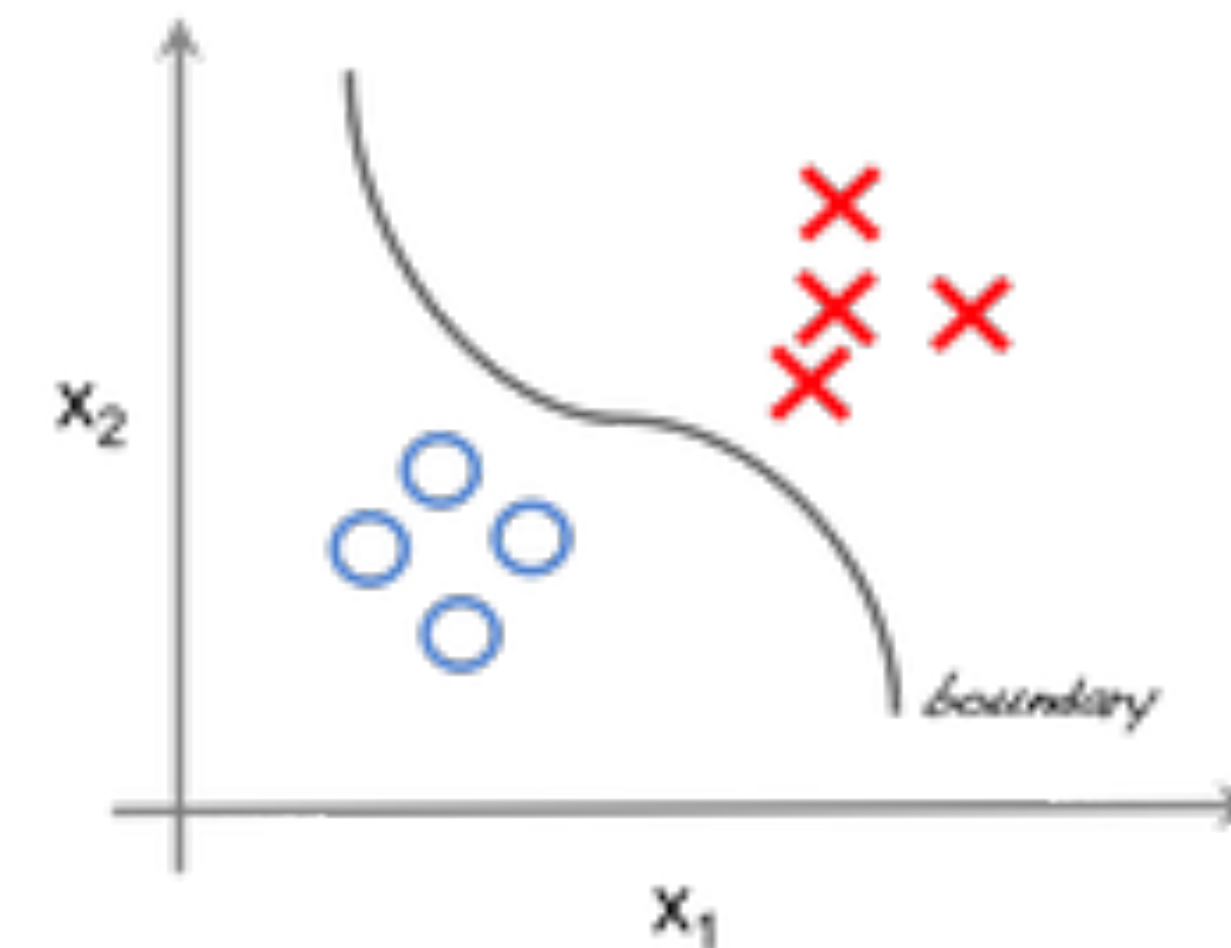
- Support vector machine



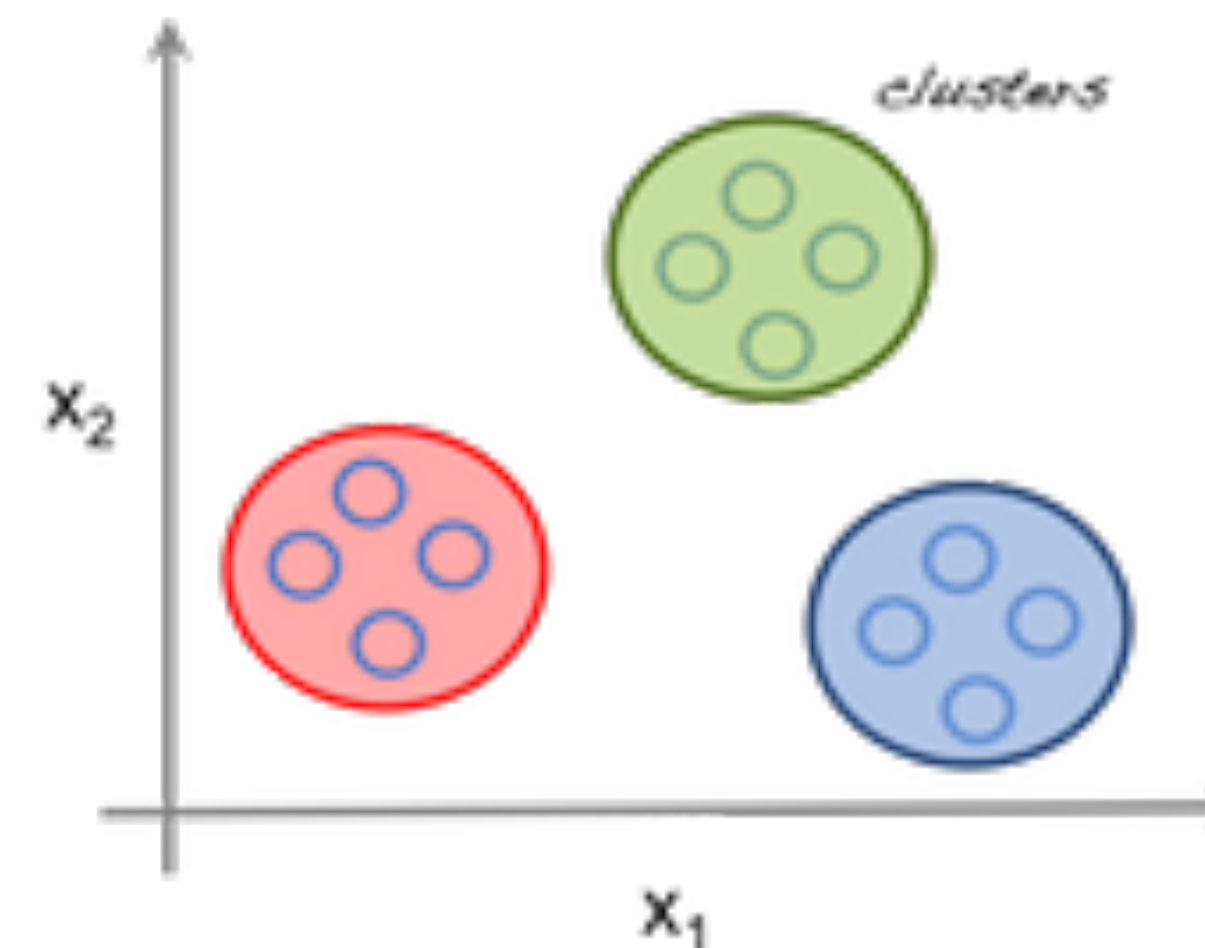
- Boosting of decision trees

- **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)
 - ...
- **Inference:** once trained, the model can be applied to other datasets

Supervised learning



Unsupervised learning



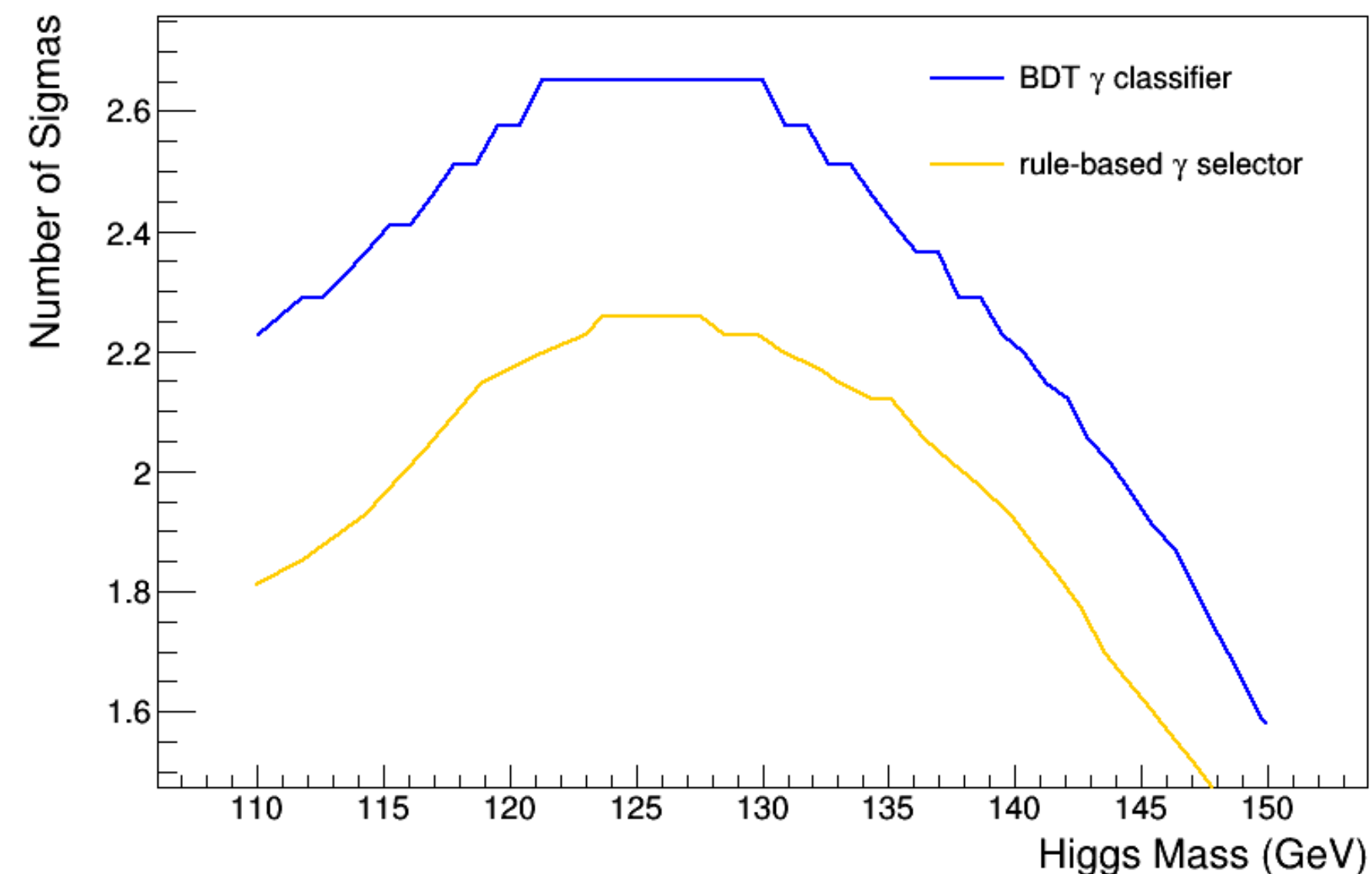
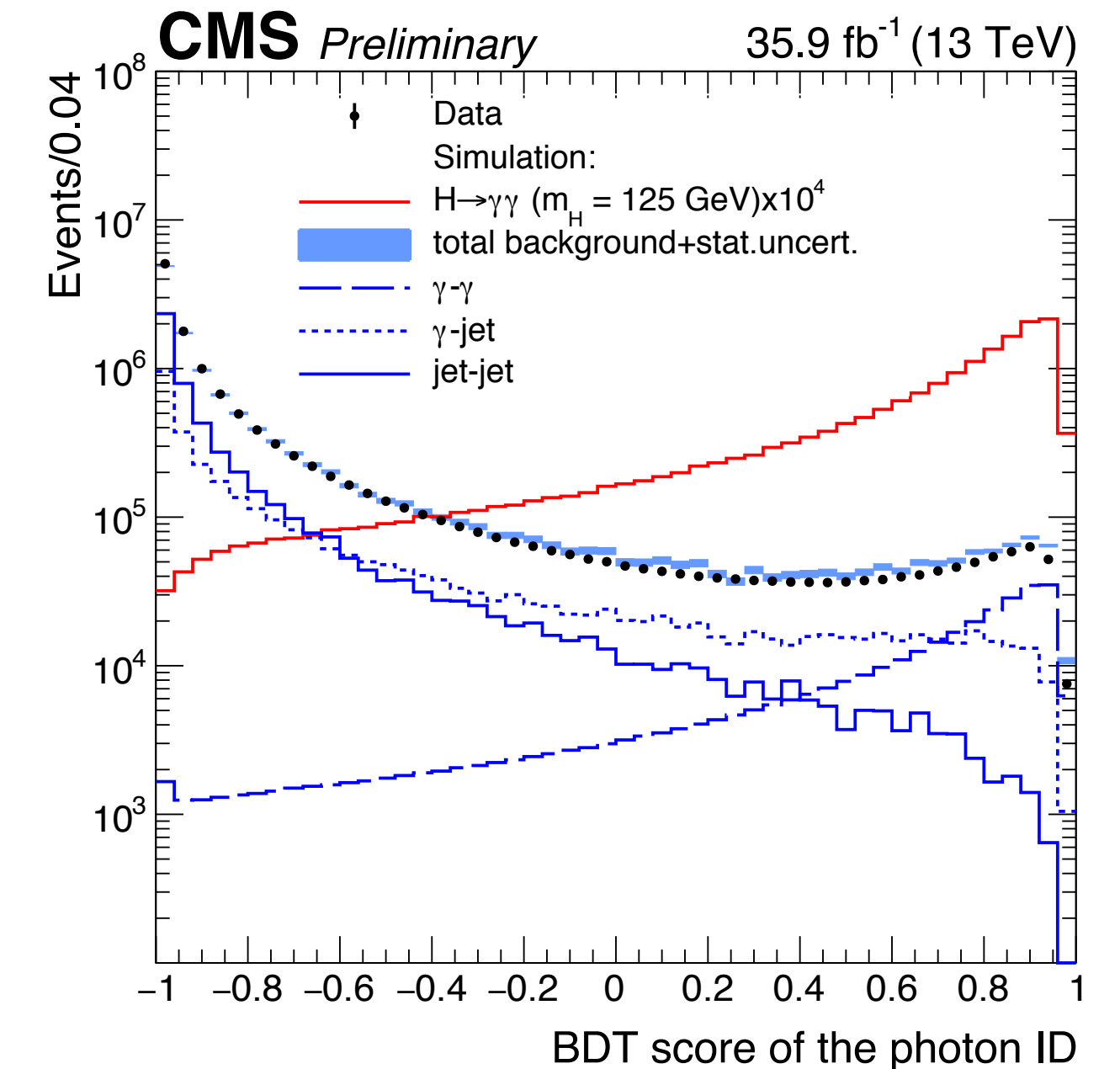
● *Long standing tradition*

● *Neural networks used at LEP and the Tevatron*

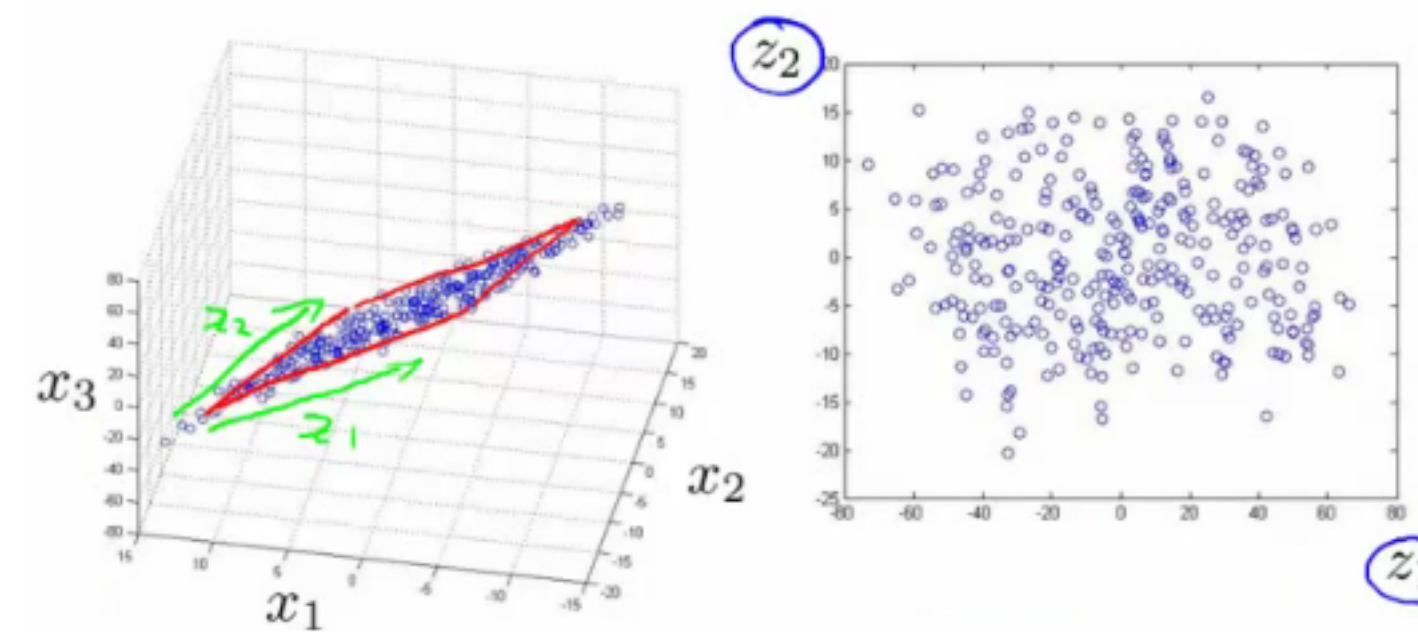
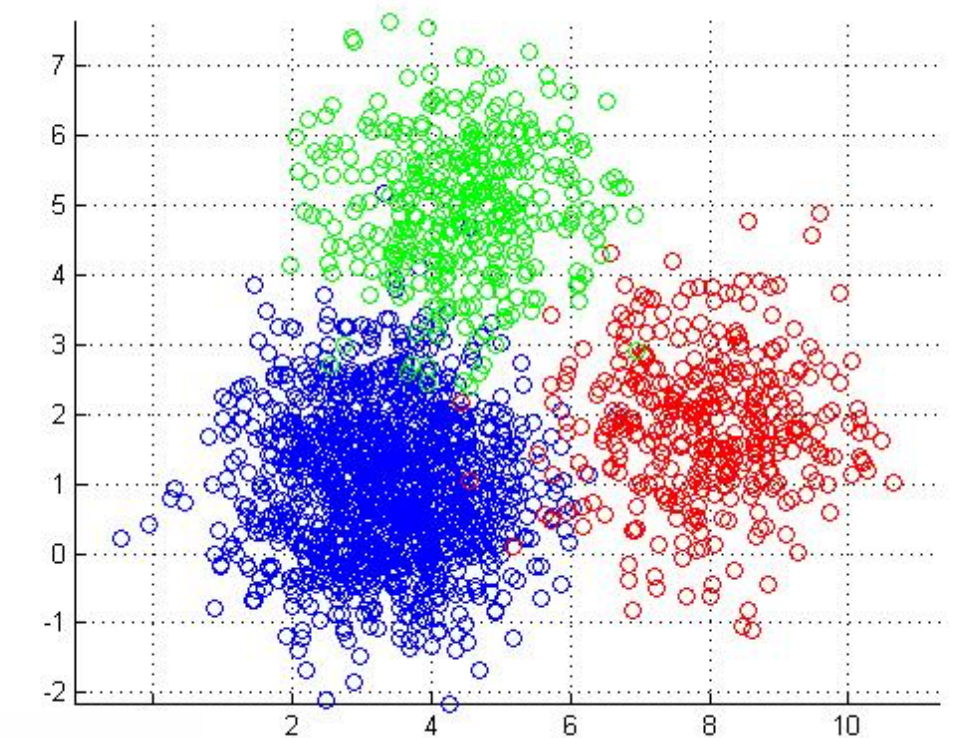
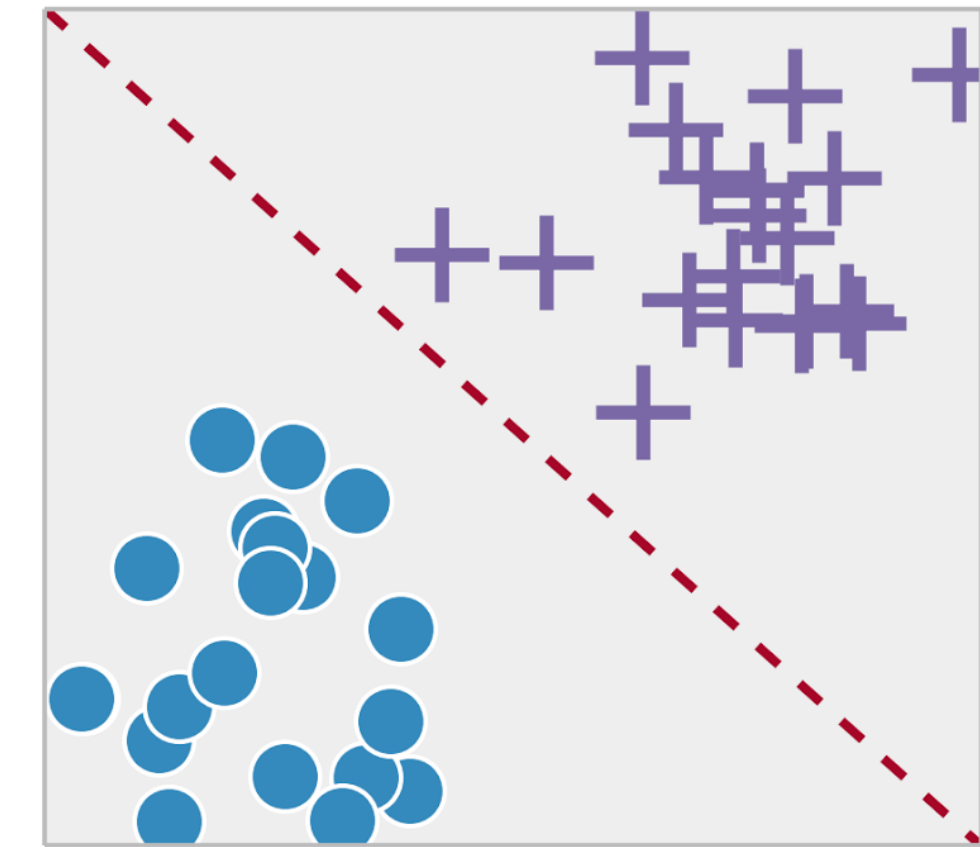
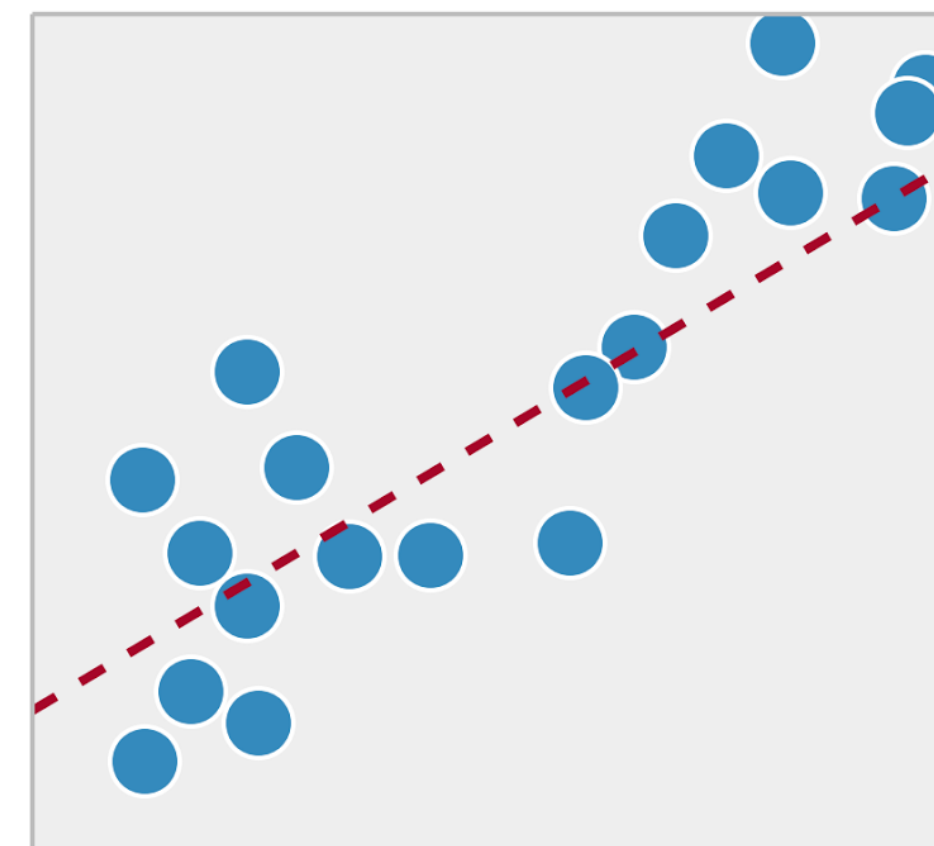
● *Boosted Decision Trees introduced by MiniNooNE and heavy used at BaBar*

● *BDTs ported to LHC and very useful on Higgs discovery*

● *Now Deep Learning is opening up many new possibilities*

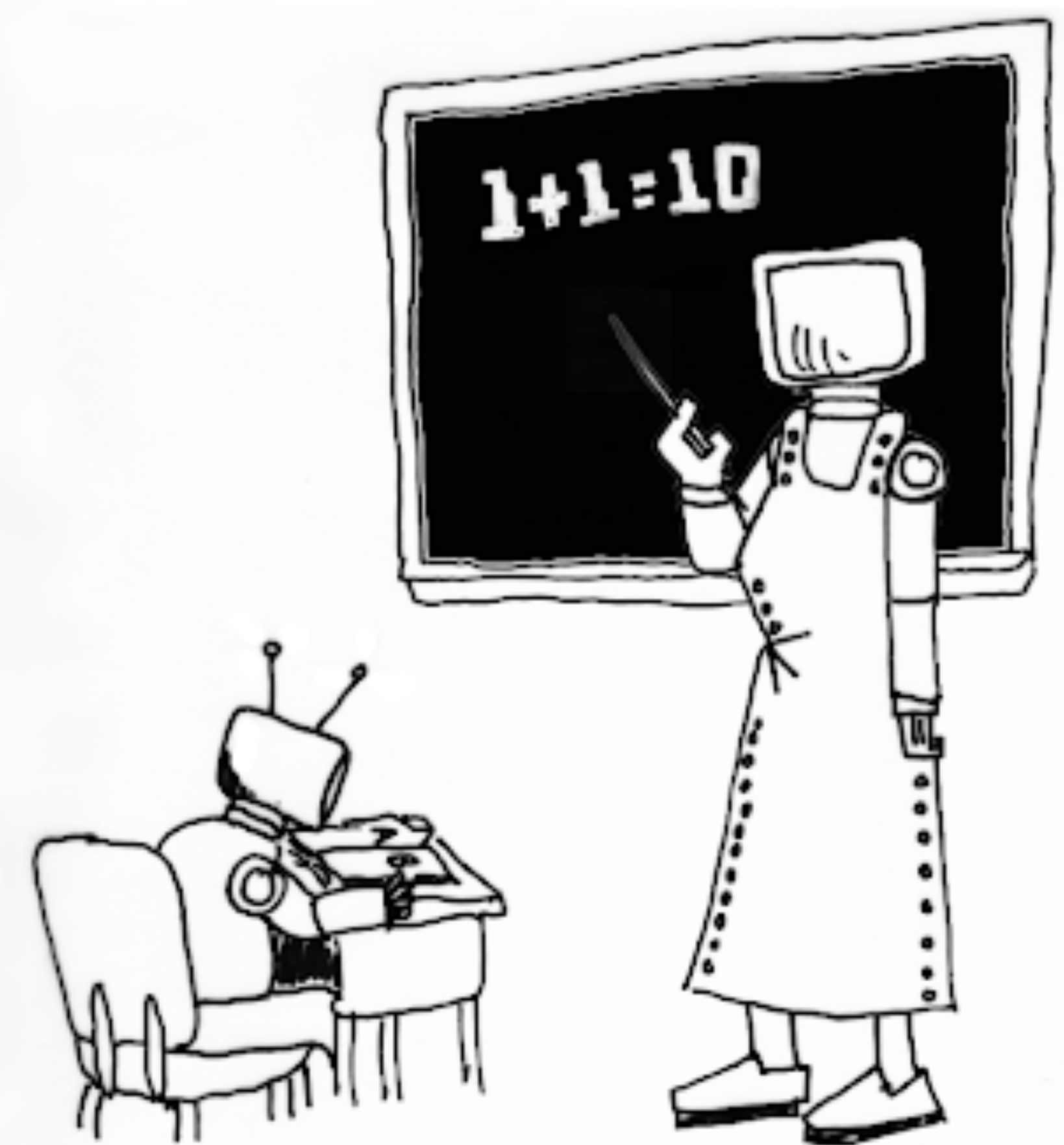
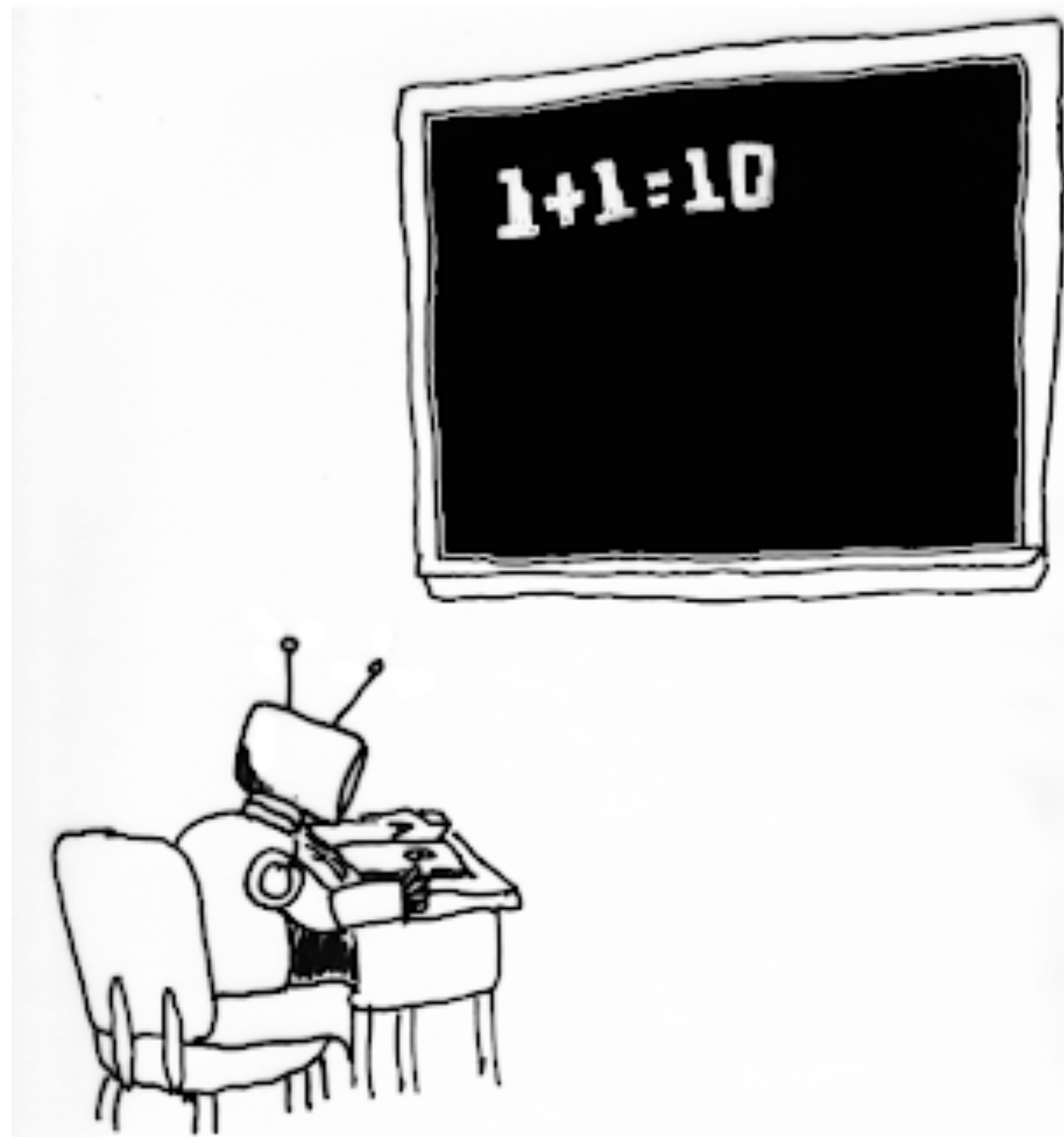


- ◎ **Classification:** associate a given element of a dataset to one of N exclusive classes
- ◎ **Regression:** determine a continuous value y from a set of inputs x
- ◎ **Clustering:** group elements of a dataset because of their similarity according to some learned metric
- ◎ **Dimensionality reduction:** find the k quantities of the N inputs (with $k < N$) that incorporate the relevant information (e.g., principal component analysis)



UNSUPERVISED MACHINE LEARNING

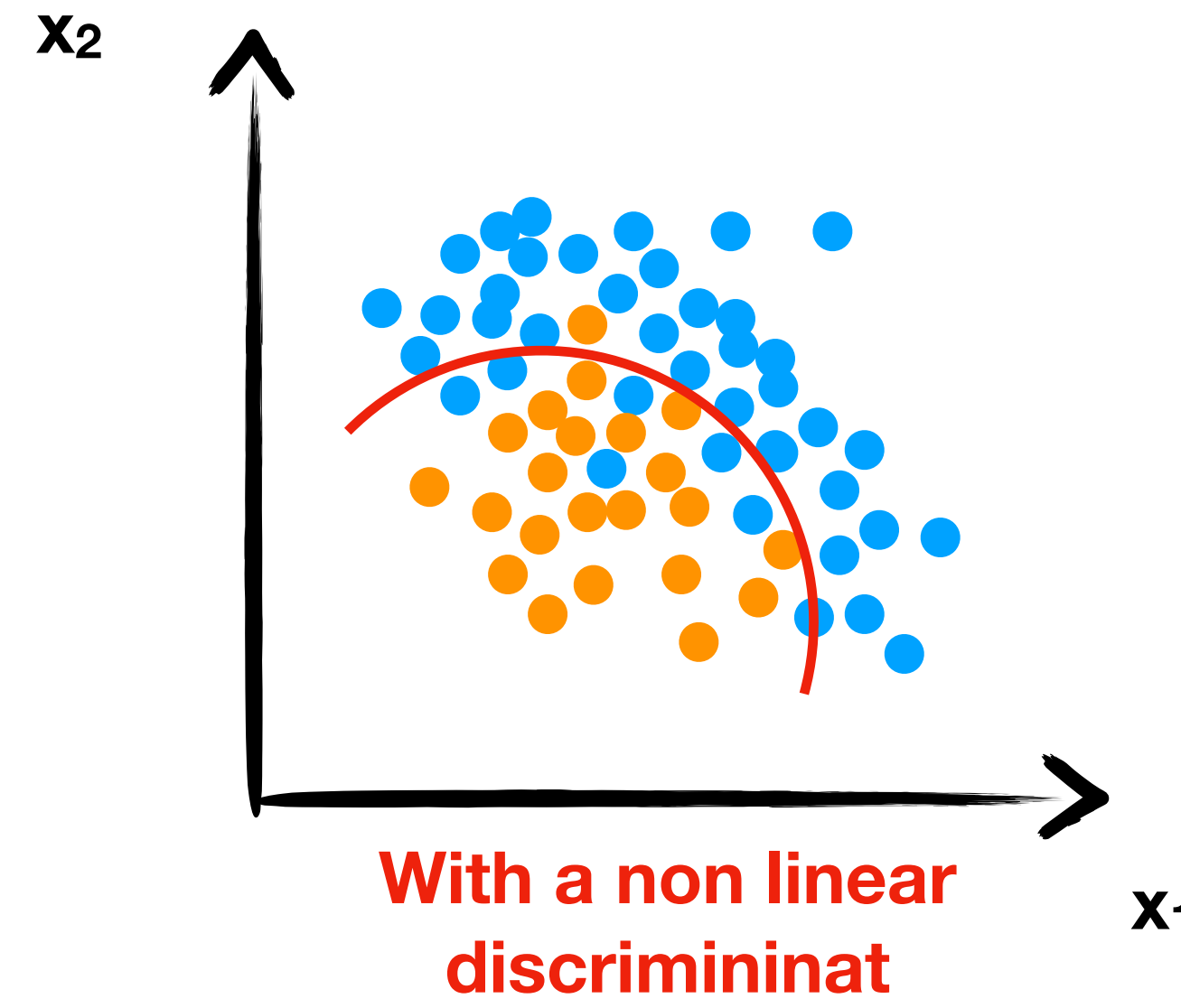
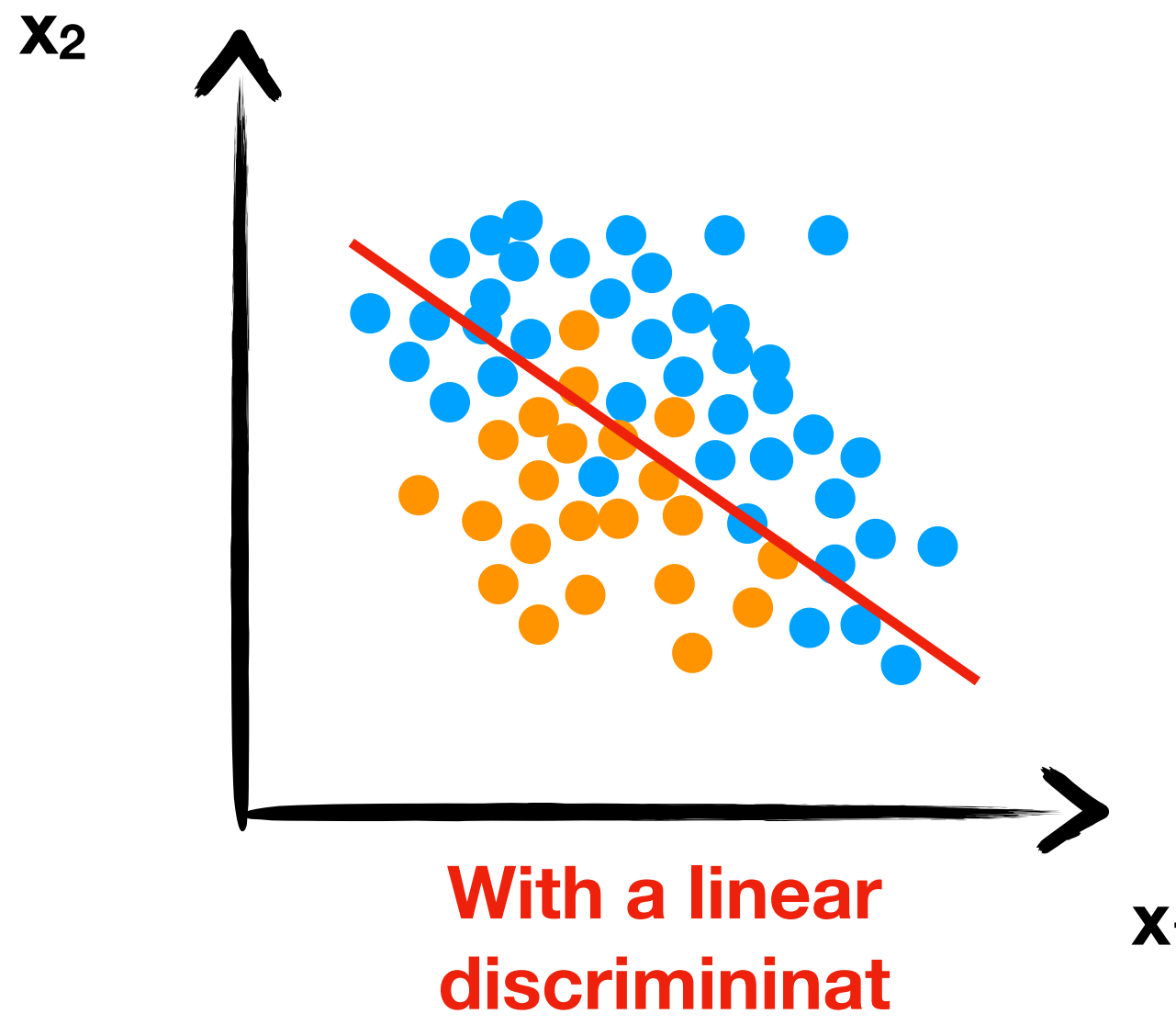
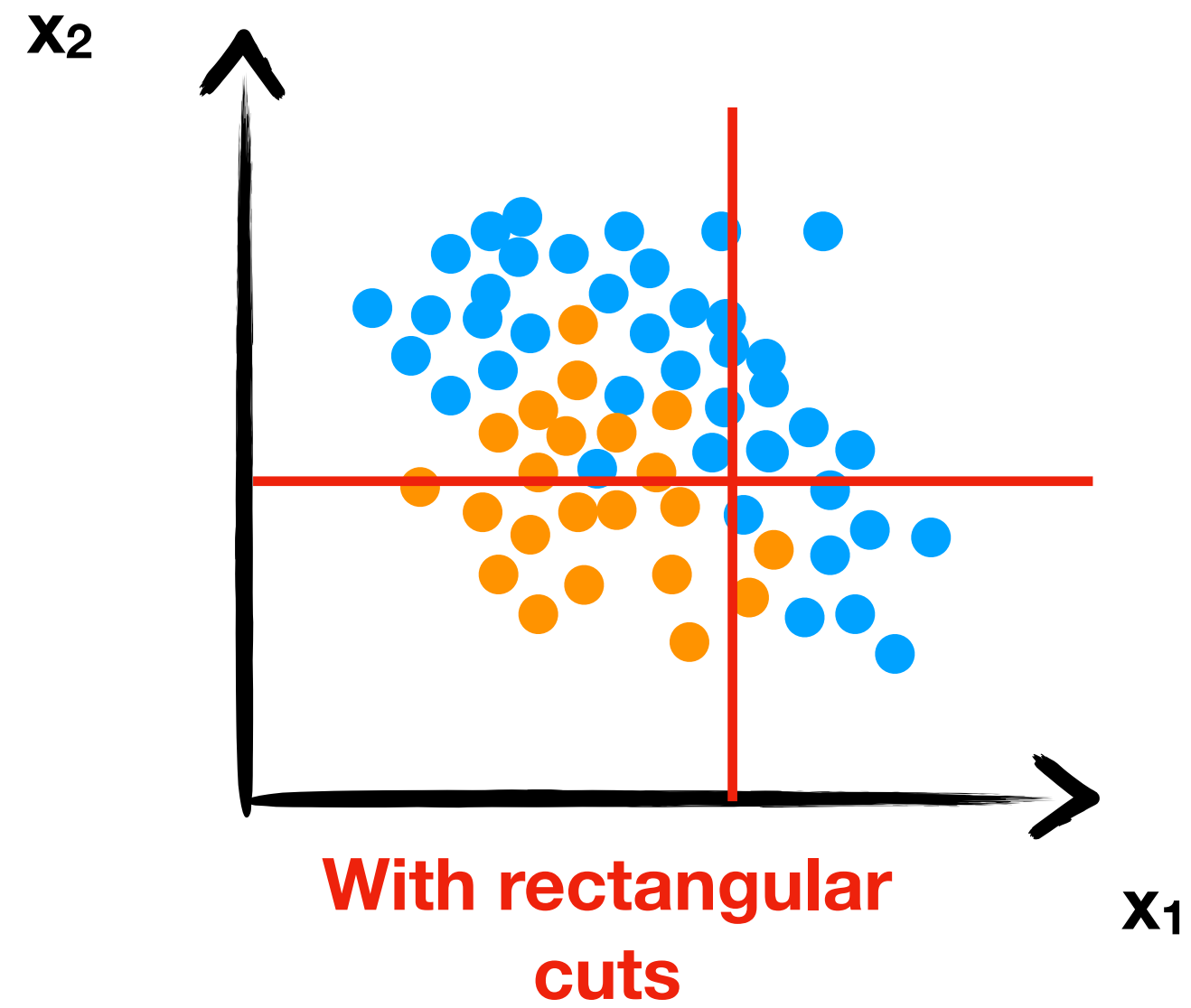
SUPERVISED MACHINE LEARNING



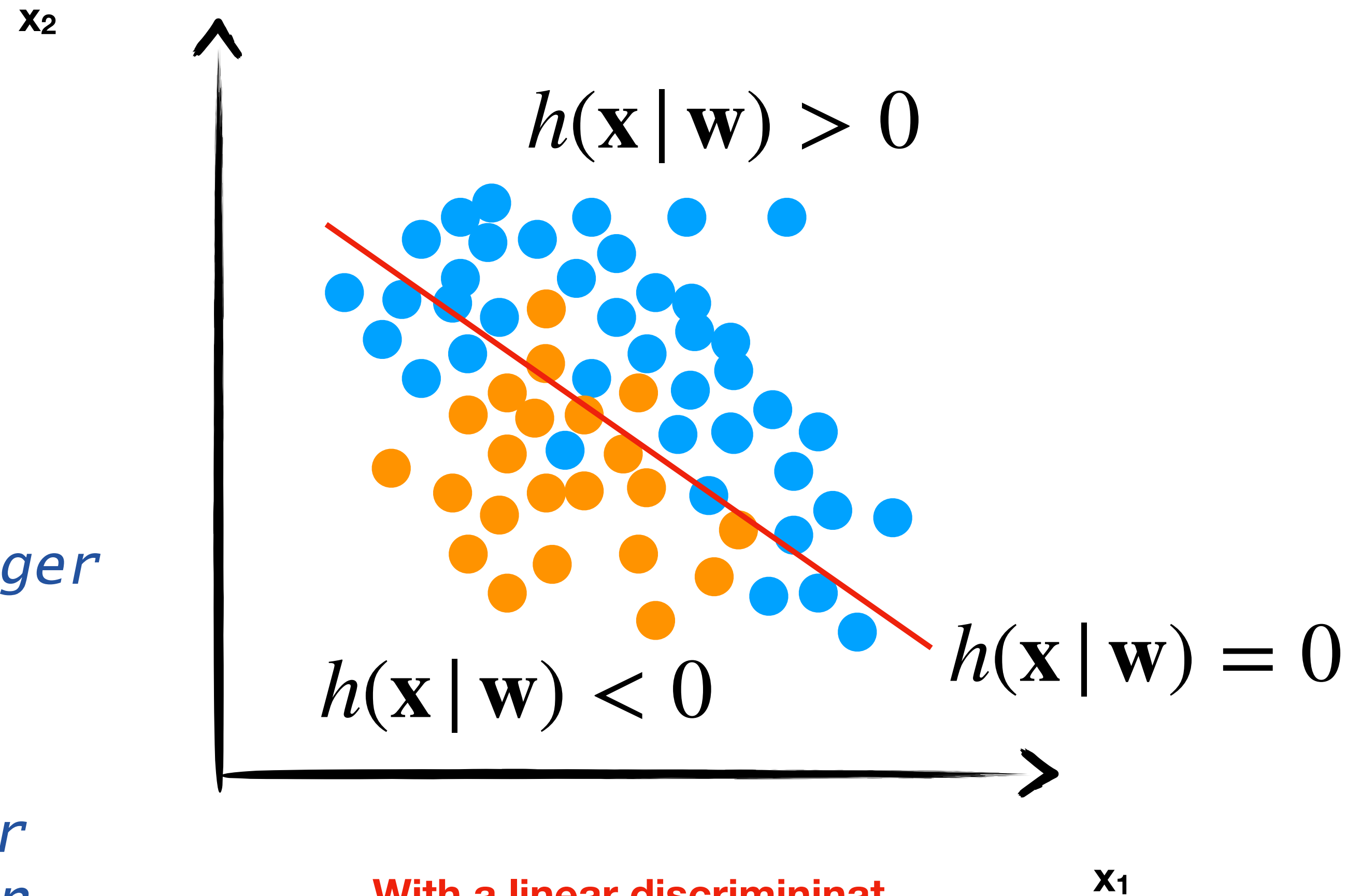
PROFREADERSWIMMY.BLOGSPOT.CA

Supervised Learning

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

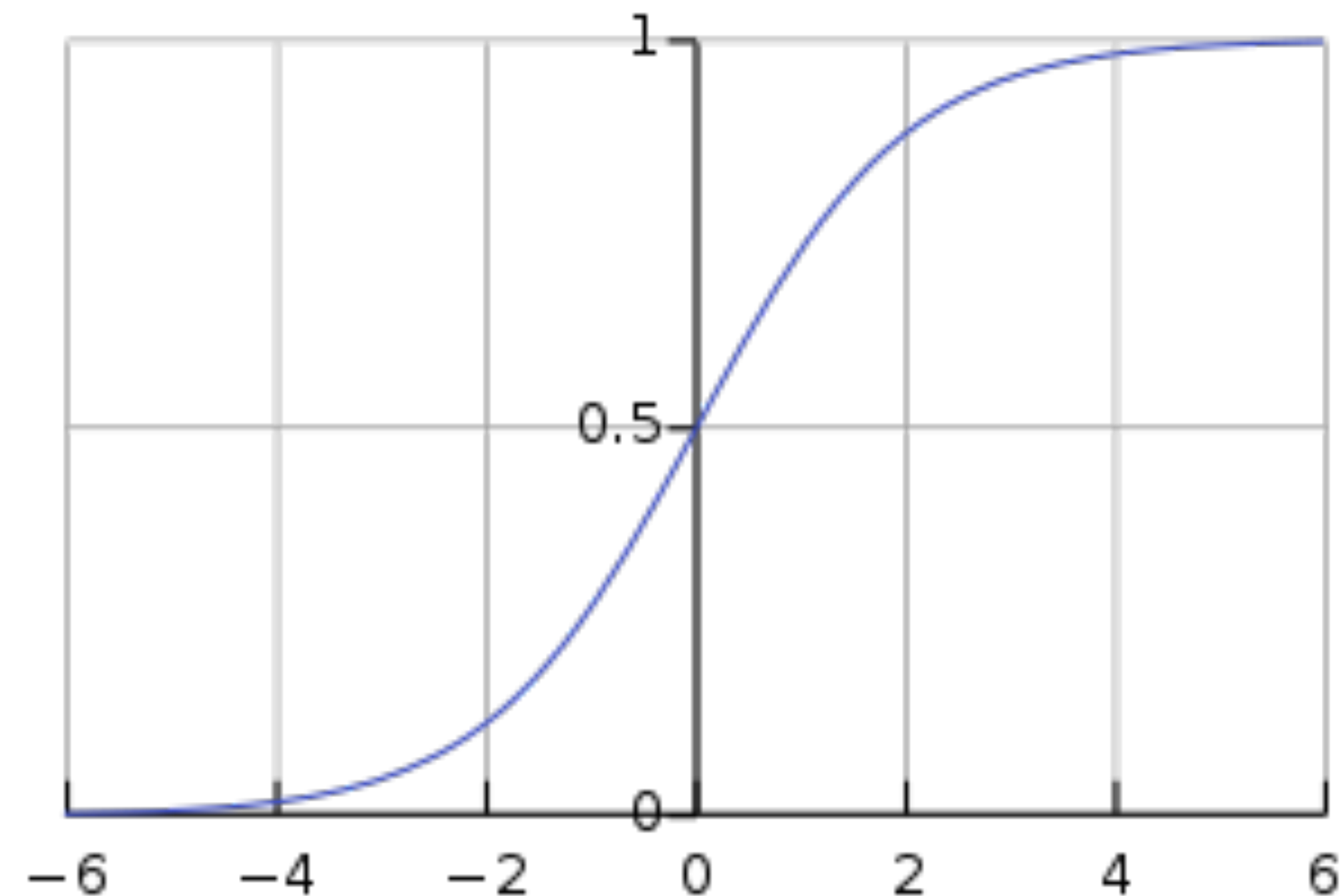


- For any linear boundary, the quantity $h(\mathbf{x} | \mathbf{w}) = w_1x_1 + w_2x_2$ is
 - $=0$ along the boundary
 - >0 above the boundary, the larger the distance, the bigger $h(\mathbf{x} | \mathbf{w})$
 - <0 below the boundary, the larger the distance the bigger $-h(\mathbf{x} | \mathbf{w})$
- In other words, the larger (smaller) is $h(\mathbf{x} | \mathbf{w})$, the larger is the probability for a given point to be blue (signal) or orange (background)



- ⦿ We can model the probability of being a signal with a logistic model
- ⦿ This definition has the desired properties:
 - ⦿ The larger (and positive) $h(\mathbf{x}|\mathbf{w})$, the closer p to 1
 - ⦿ The larger (and negative) $h(\mathbf{x}|\mathbf{w})$, the closer p to 0
- ⦿ The optimal boundary (i.e., the optimal choice of w_1 and w_2) is such that we maximise probability for signal points and minimise that of background points
- ⦿ To do so, we need
 - ⦿ A set of points for which we have a ground truth $x_i \in \mathbb{R}^n$ and $y_i = \{0,1\}$
 - ⦿ A loss function to minimise

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



- Probability: When we introduced distributions, we started from known distributions (e.g., a Poisson with known λ) and we tried to characterize a typical experiment outcome
- Hypothesis Testing: Now we inverted the problem: we know the experiment outcome (e.g., we counted events above threshold during a one-year run) and we ask ourselves which of two λ values (bkg-only or sig+bkg) they come from
- Inference: we could also just ask what is the value of λ more compatible with the observation (trivial question in this case - right? - but not in general). This is a typical application of maximum likelihood fits and a regression problem in Machine Learning (not much to say about this today)

Back to our statistics lecture

- We are given a likelihood model $\mathcal{L}(D|w)$ and some data D
 - D is known, w are unknown
- We want to find the \hat{w} values that would make our data D the most probable outcome of the experiment
- If we knew these \hat{w} values, the probability of observing D is maximal (here D is unknown and \hat{w} is known)
- You can convince yourselves that

$$\hat{w} = \arg \max_w \mathcal{L}(D | w)$$

- Bernoulli's process (Y/N question):
 - probability of a Y is p
 - probability of a N is $(1-p)$
- If we assign labels ($Y \rightarrow 1$ and $N \rightarrow 0$) we can write the probability for a i -th event as $p^{x_i}(1-p)^{1-x_i}$, where x_i is the label for the i -th event
- The likelihood is then written as

$$\mathcal{L}(y|p) = \prod_i p_i^y (1-p)^{(1-y)}$$

⊙ We can make our probability model more complicated

⊙ p could be a function of a set of quantities x that we know about our data

⊙ for instance, this could be our logistic regression problem

⊙ **Minimizing the $-\log L$ corresponds to minimizing the binary cross entropy**

$$p_i = p(y_i = 1 | x_i) = \frac{1}{1 + e^{-h(x_i|w)}}$$

$$\arg \max_w \mathcal{L} = \arg \min_w [-\log \mathcal{L}]$$

$$= \arg \min_w [-\log(\prod_i p_i^{y_i} (1 - p_i)^{1-y_i})]$$

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

⦿ *Given a set of points, find the curve that goes through them*

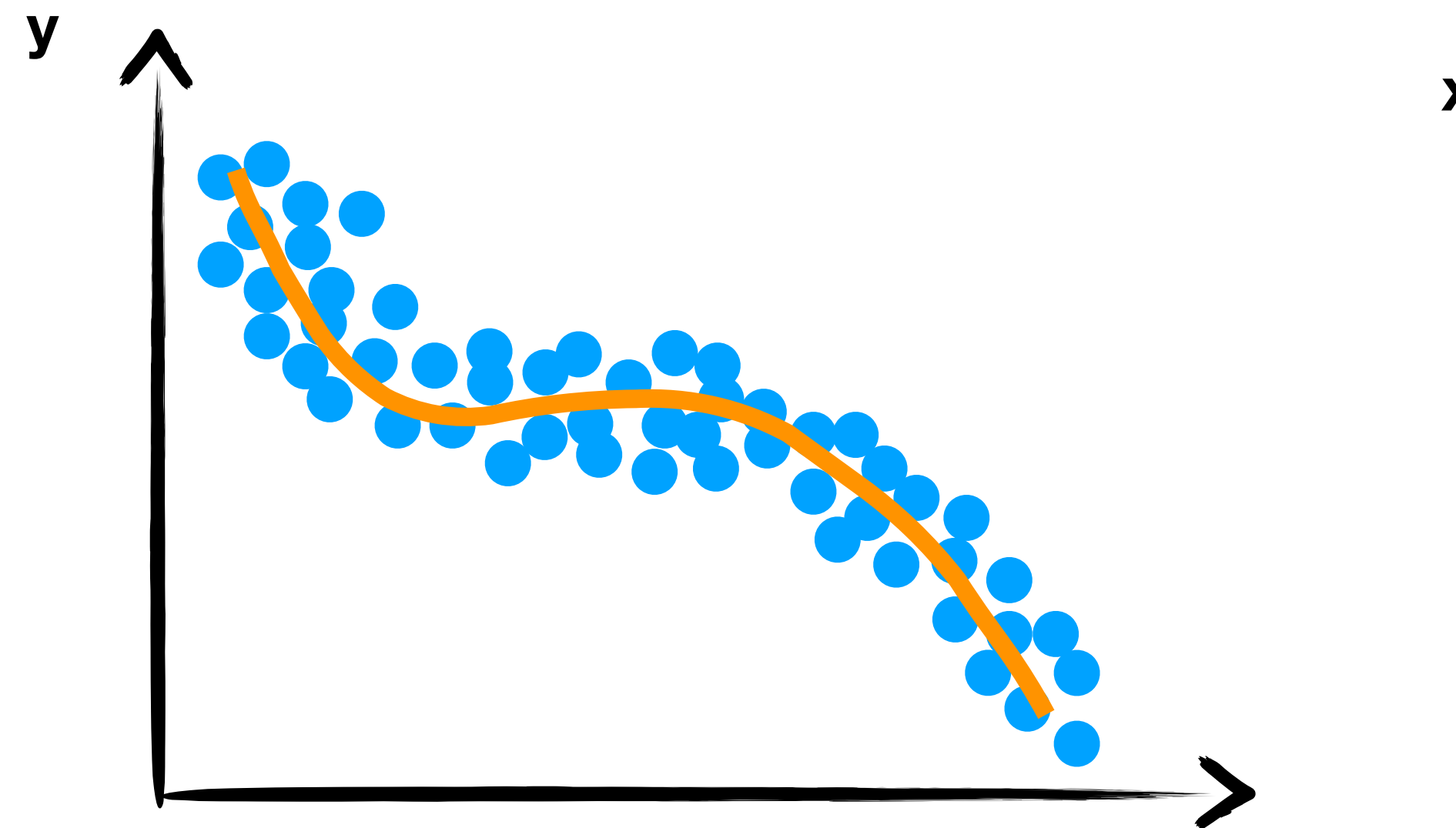
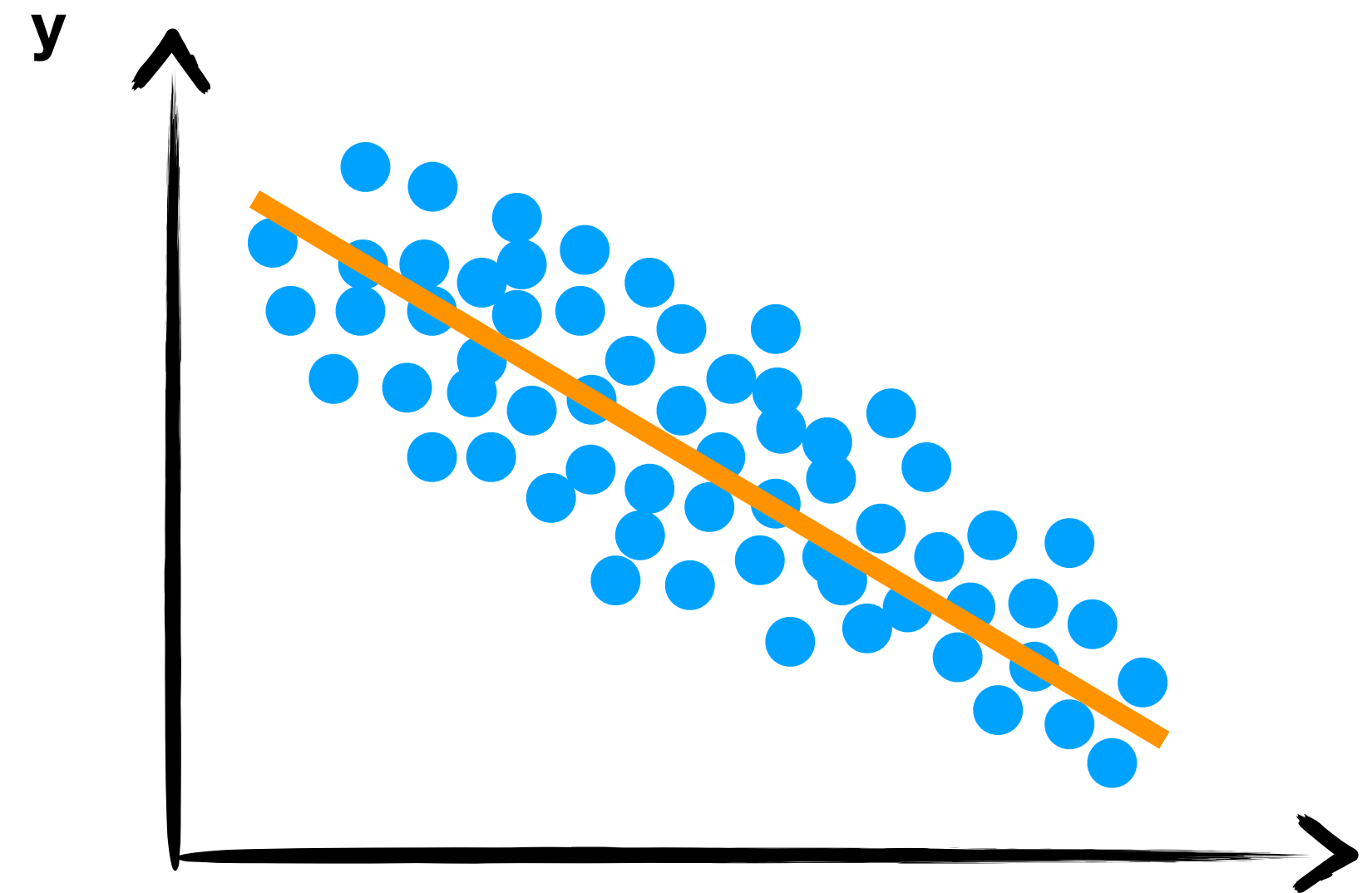
⦿ *Can be a linear model*

$$y_i = ax_i + b$$

⦿ *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



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

$$h(x_i | a, b) = ax_i + b$$

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

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

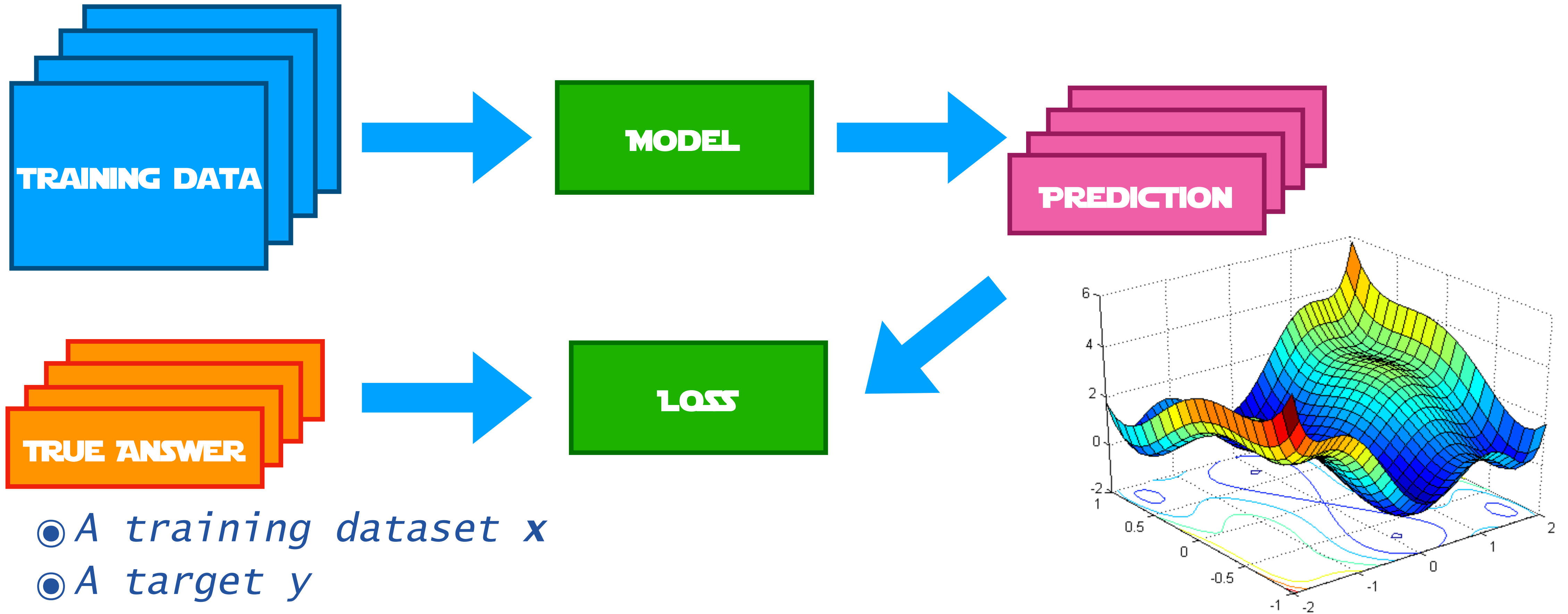
- Assume that the resolution σ is fixed and write down the likelihood

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

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

$$\begin{aligned} \arg \min[-2 \log \mathcal{L}] &= \arg \min \left[-2 \log \left[\prod_i \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(y_i - h(x_i))^2}{2\sigma^2}} \right] \right] \\ &= \arg \min \left[\sum_i \frac{(y_i - h(x_i))^2}{\sigma^2} \right] = \arg \min \left[\sum_i (y_i - h(x_i))^2 \right] = \text{MSE} \end{aligned}$$

- ◎ *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?**



- 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

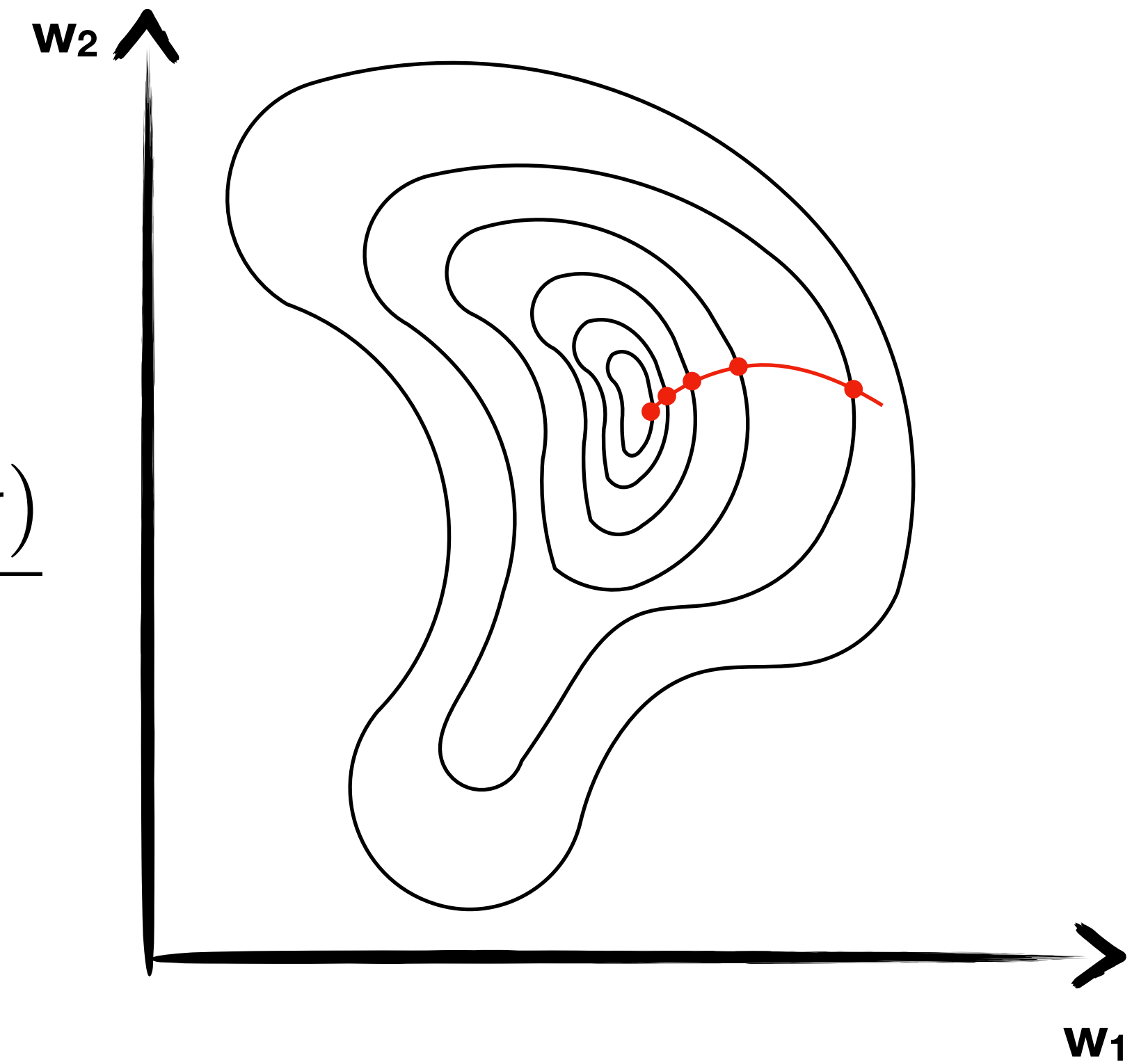
- ◎ *Split your sample in three:*
 - ◎ *Training: the biggest chunk, where you learn from*
 - ◎ *Validation: an auxiliary dataset to verify generalization and prevent overtraining*
 - ◎ *Test: the dataset for the final independent check*

Training

Validation

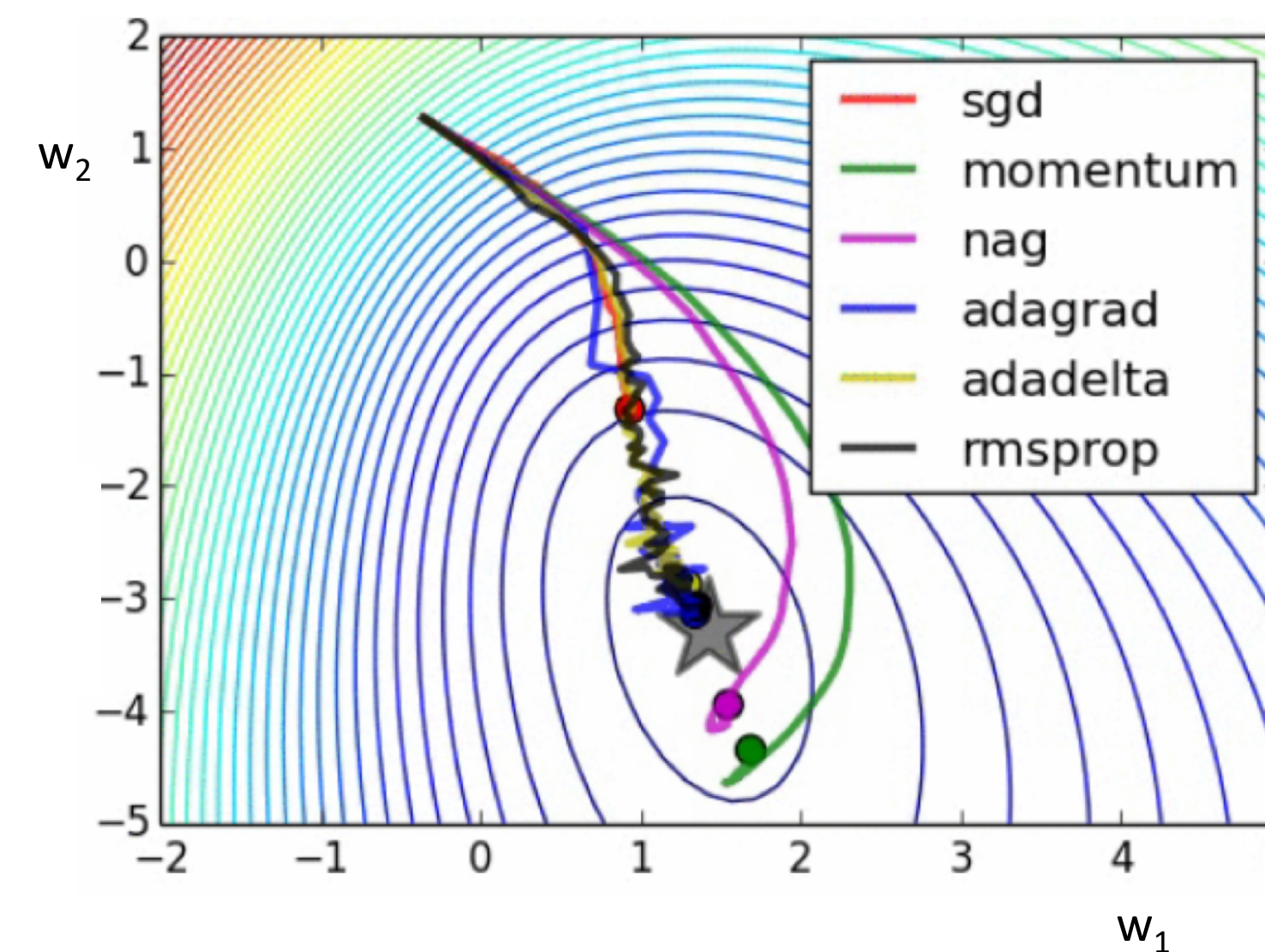
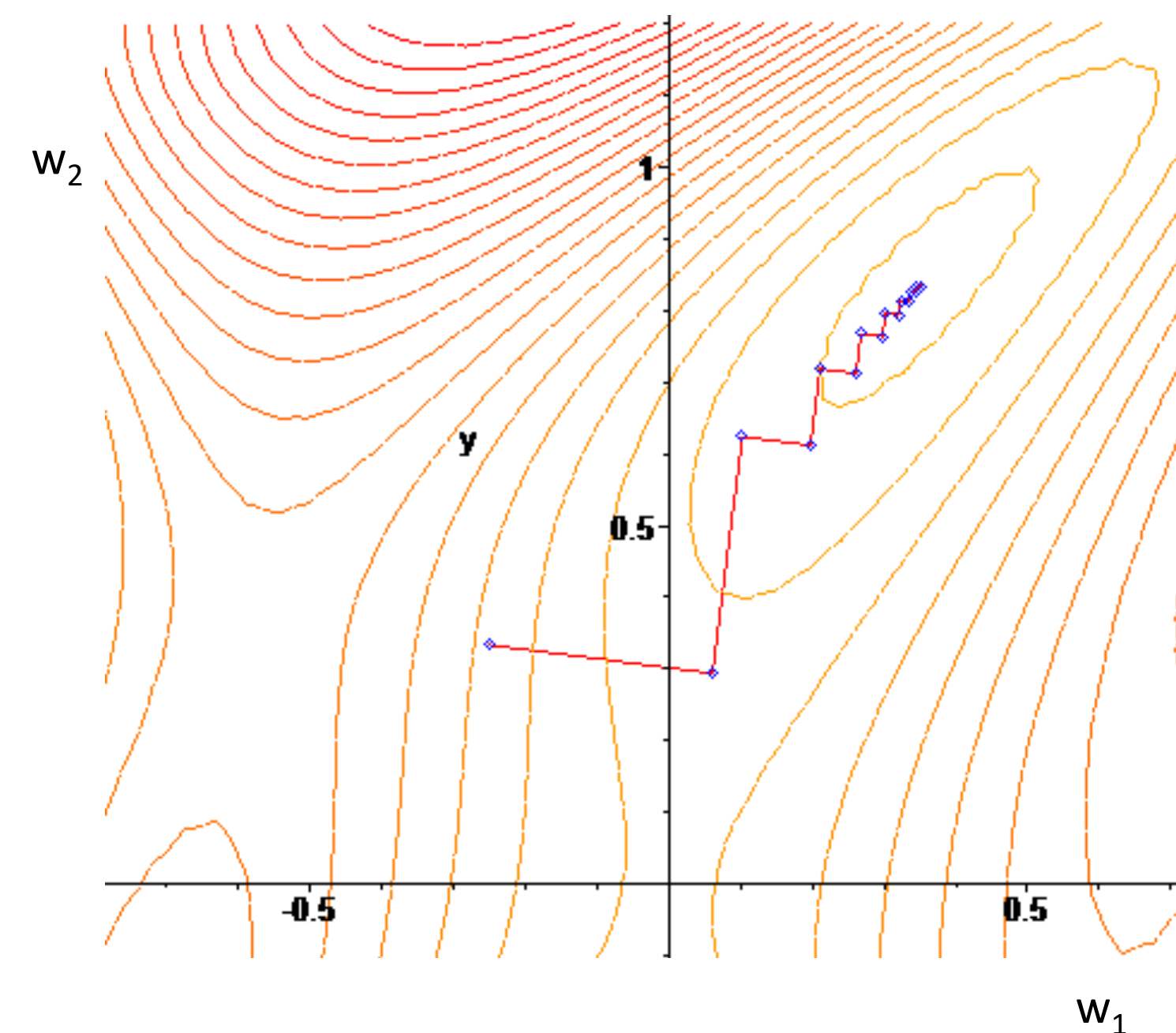
Test

- 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 model accordingly
- Effective, but computationally expensive (gradient over entire dataset)

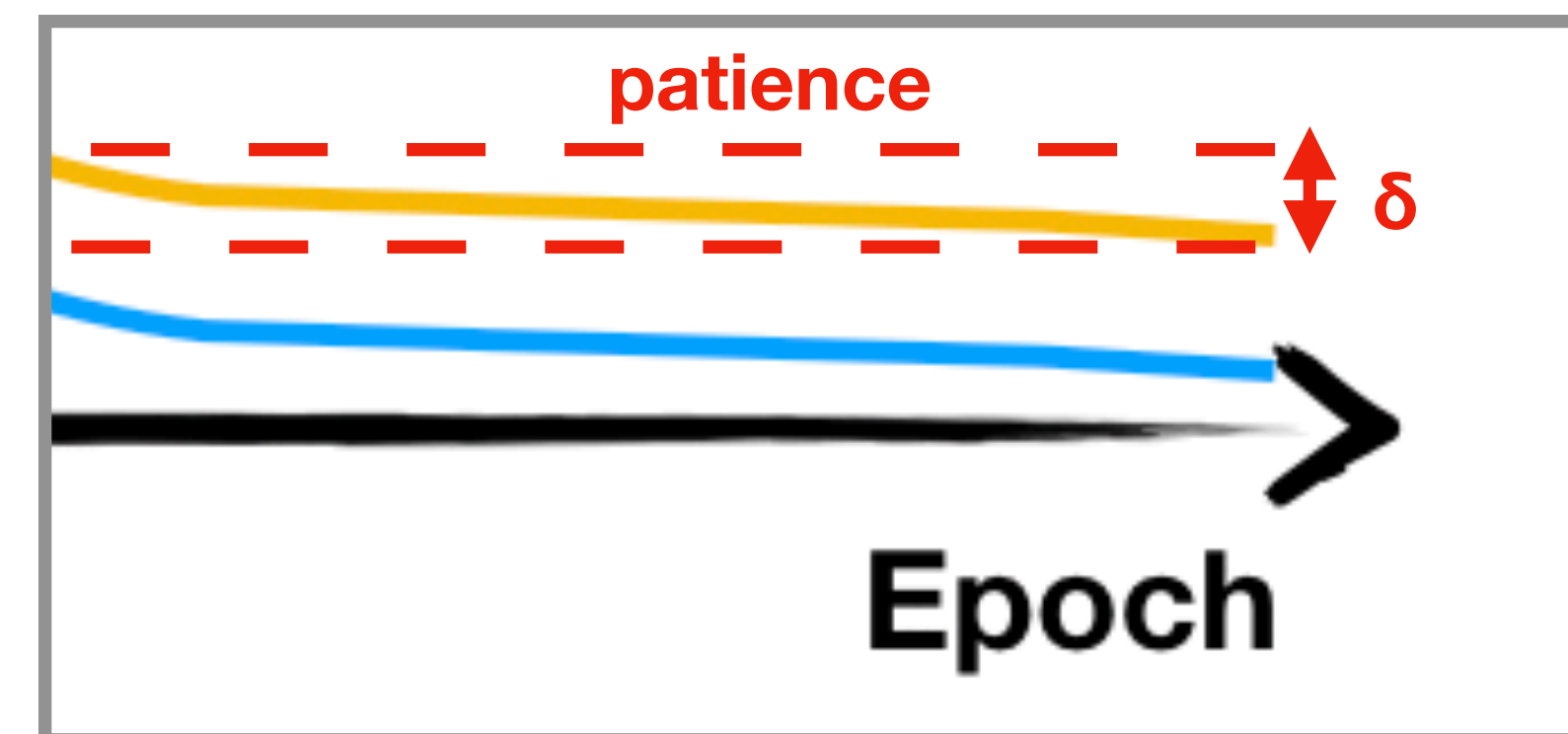
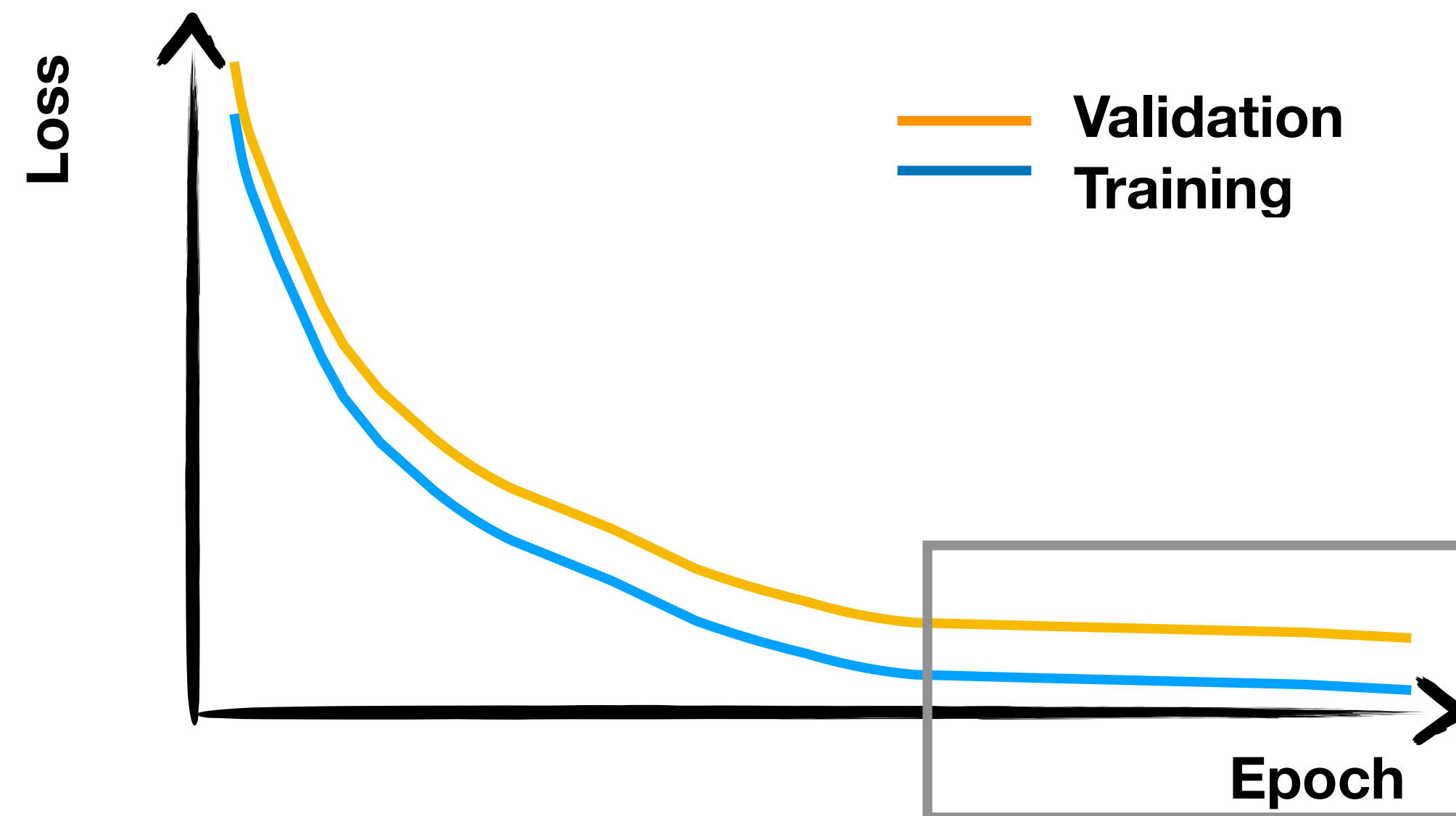


$$\mathbf{w}' \leftarrow \mathbf{w} - \eta \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$$

- *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)*

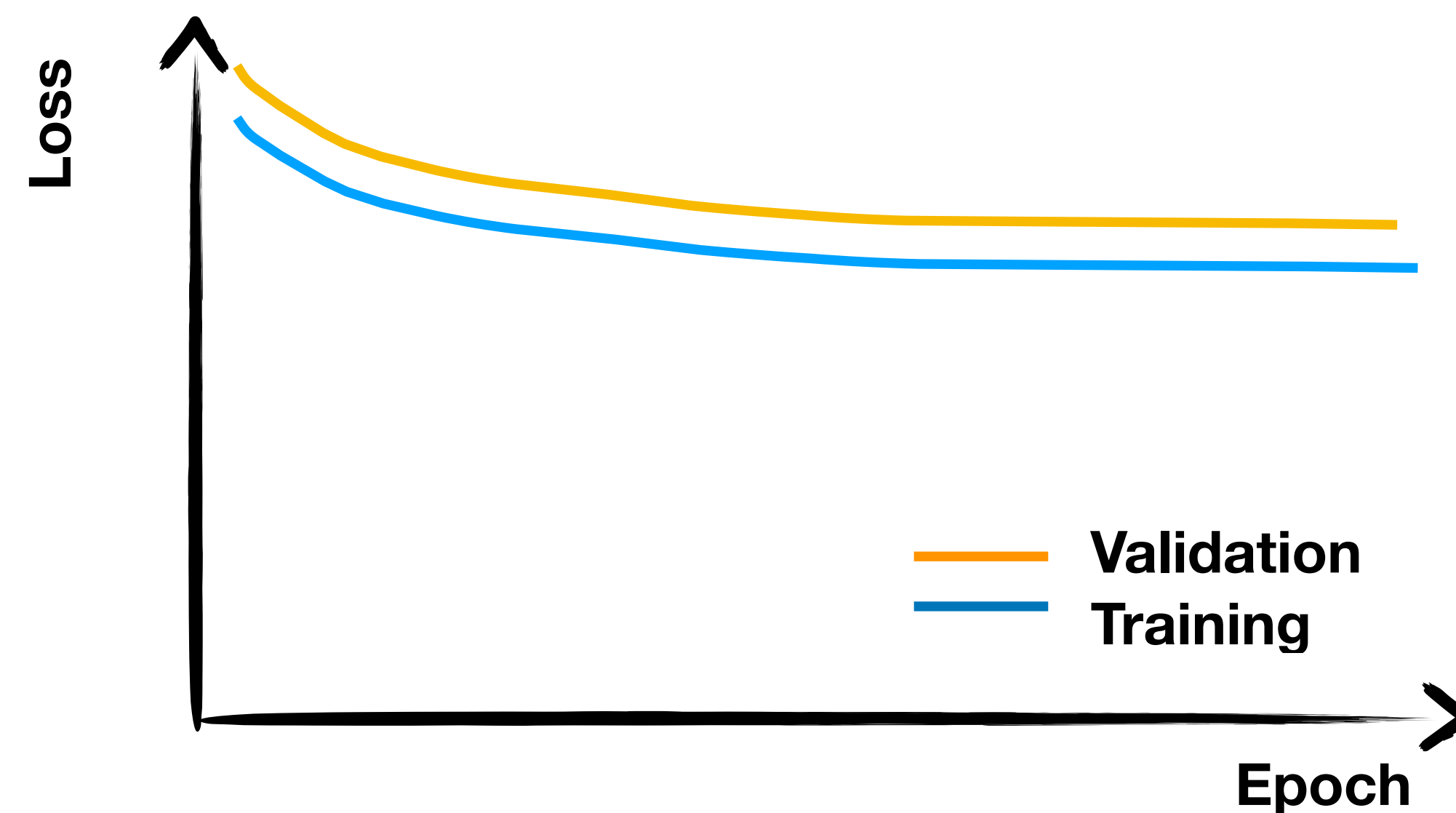
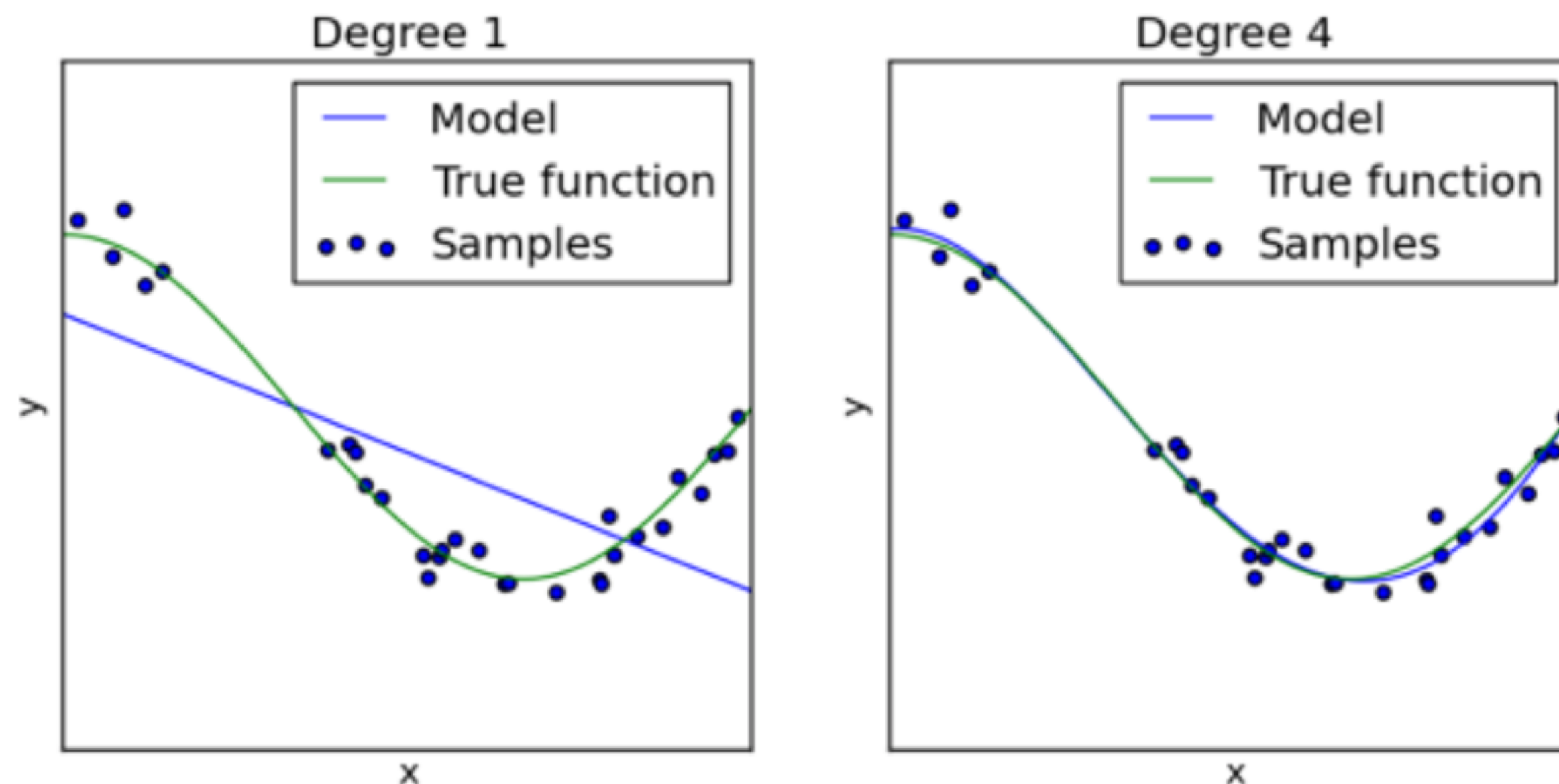


- *Train across multiple epochs*
 - *1 epoch = going once through the full dataset*
- *Use small batches (64, 128, etc)*
- *Check your training history*
 - *on the training data (training loss)*
 - *and the validation ones (validation loss)*
- *Use an objective algorithm to stop (e.g., early stopping)*

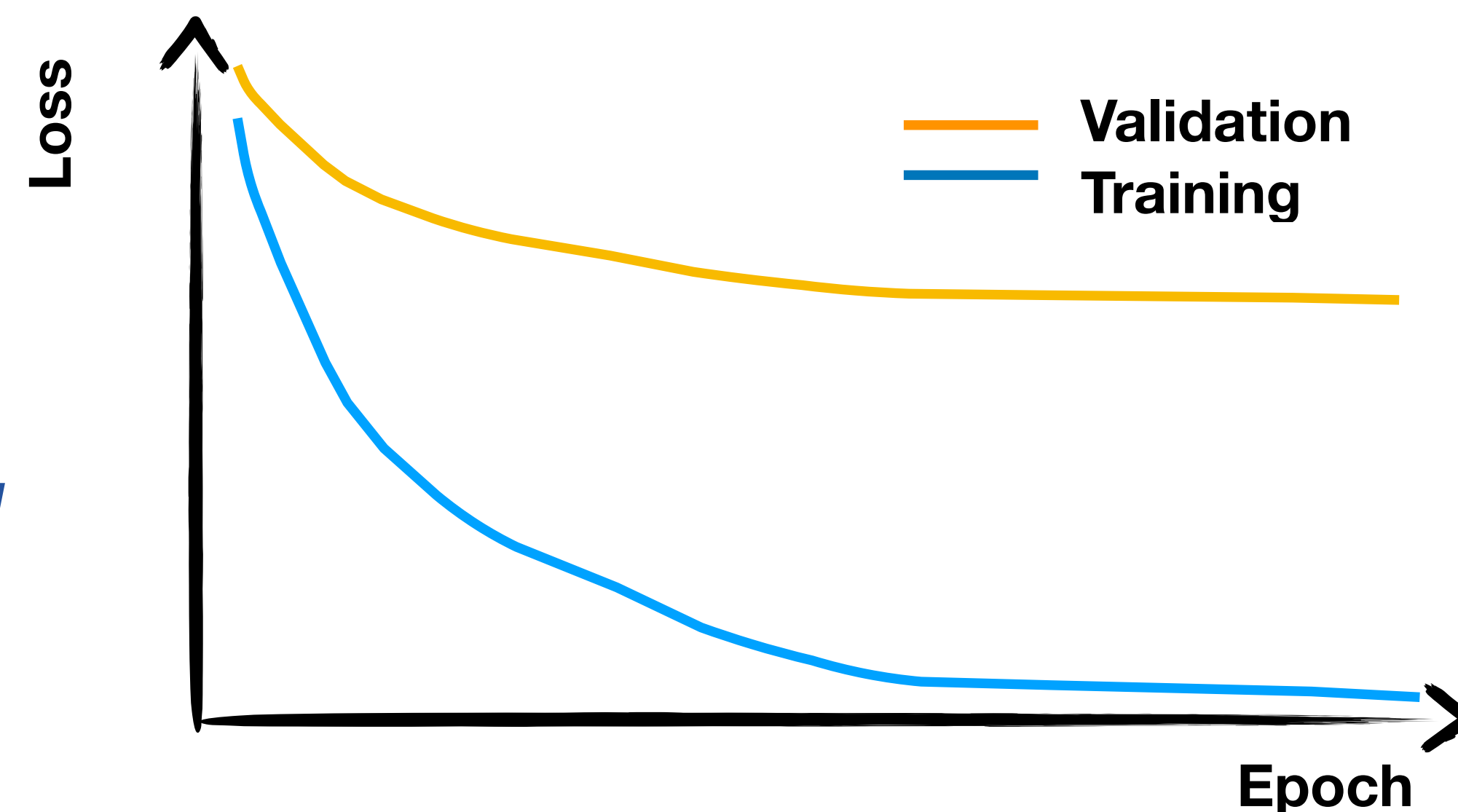
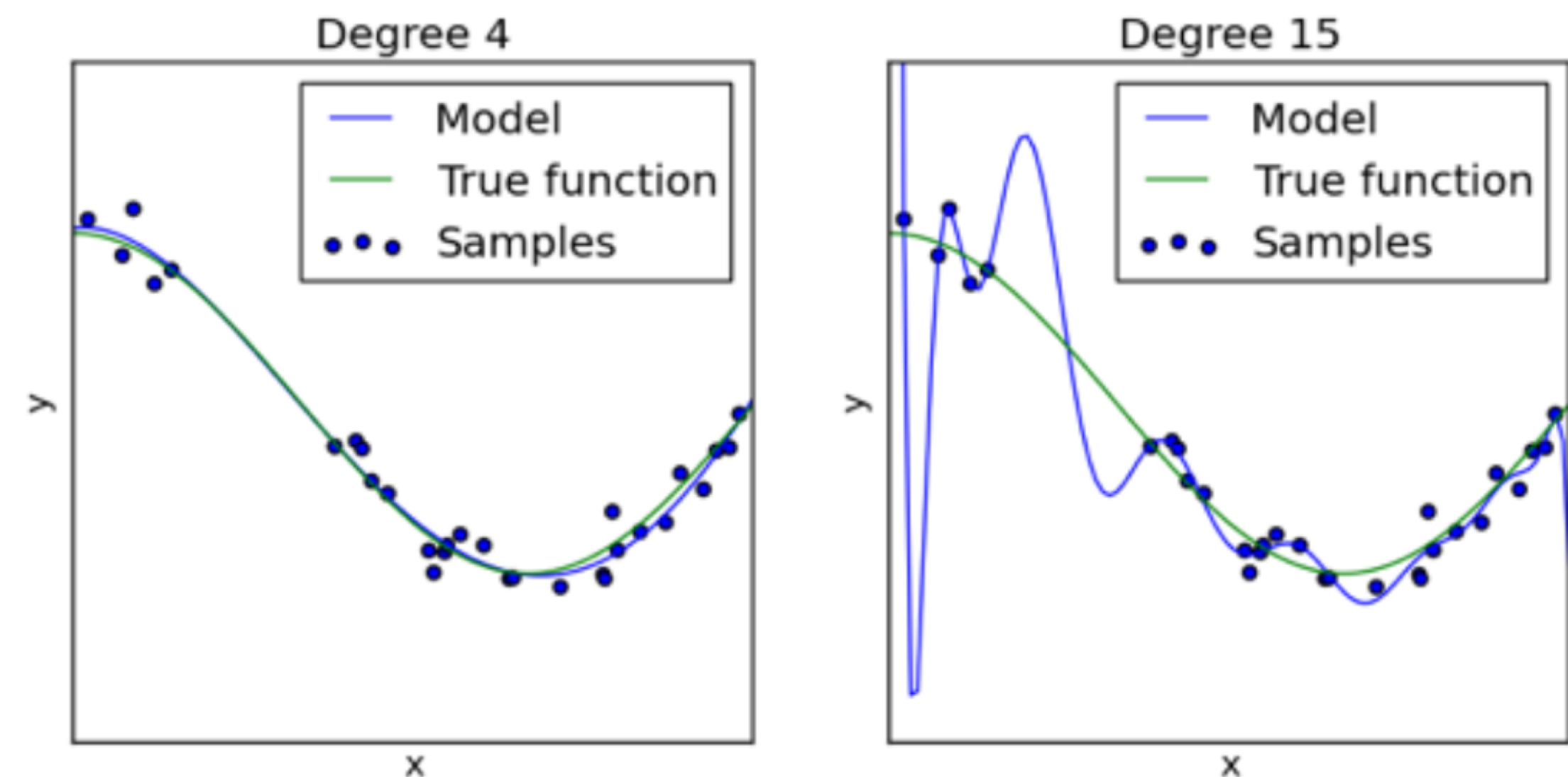


EARLY TOPPING: stop the train if the validation loss didn't change more than δ in the last n epochs (patience)

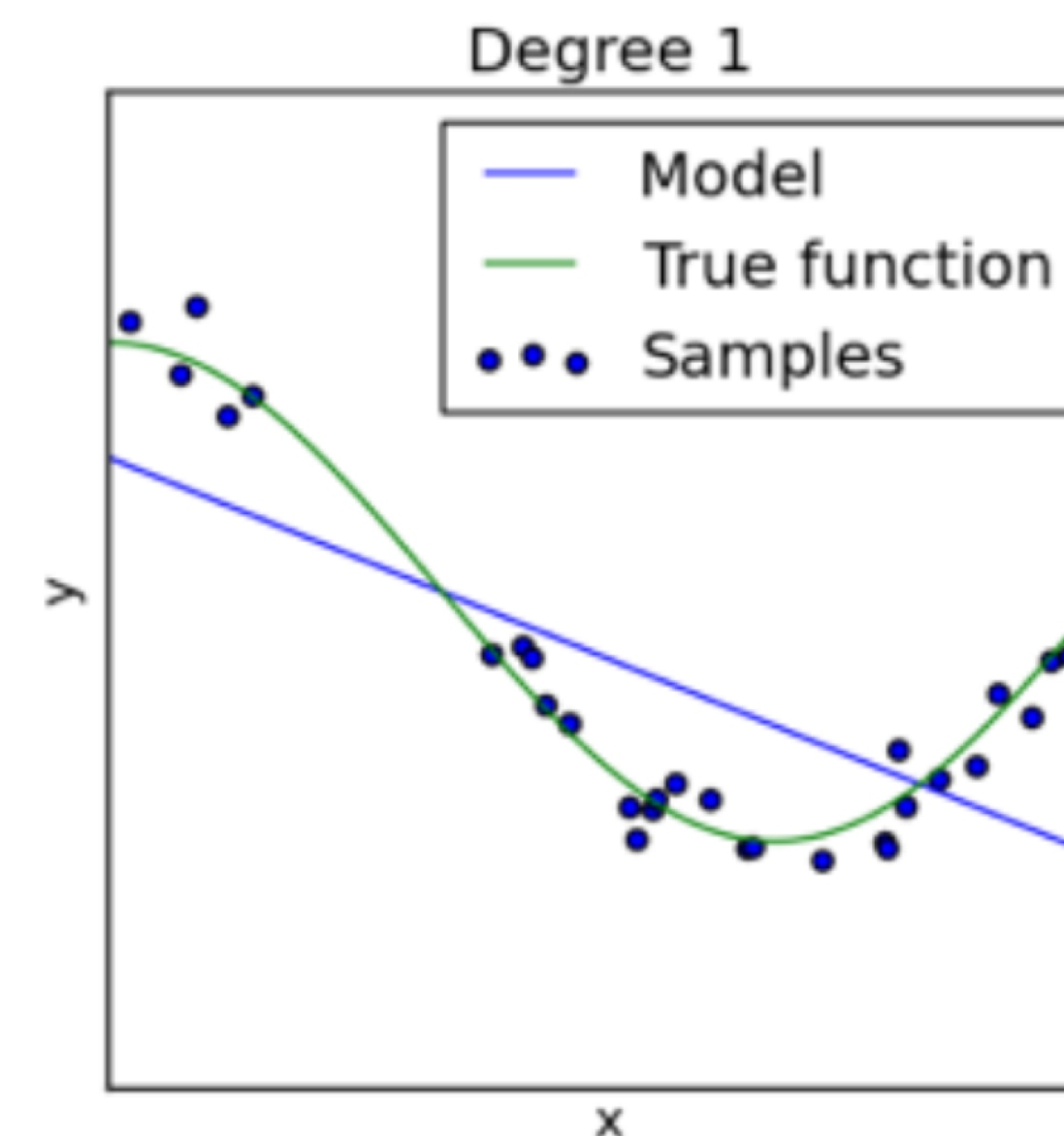
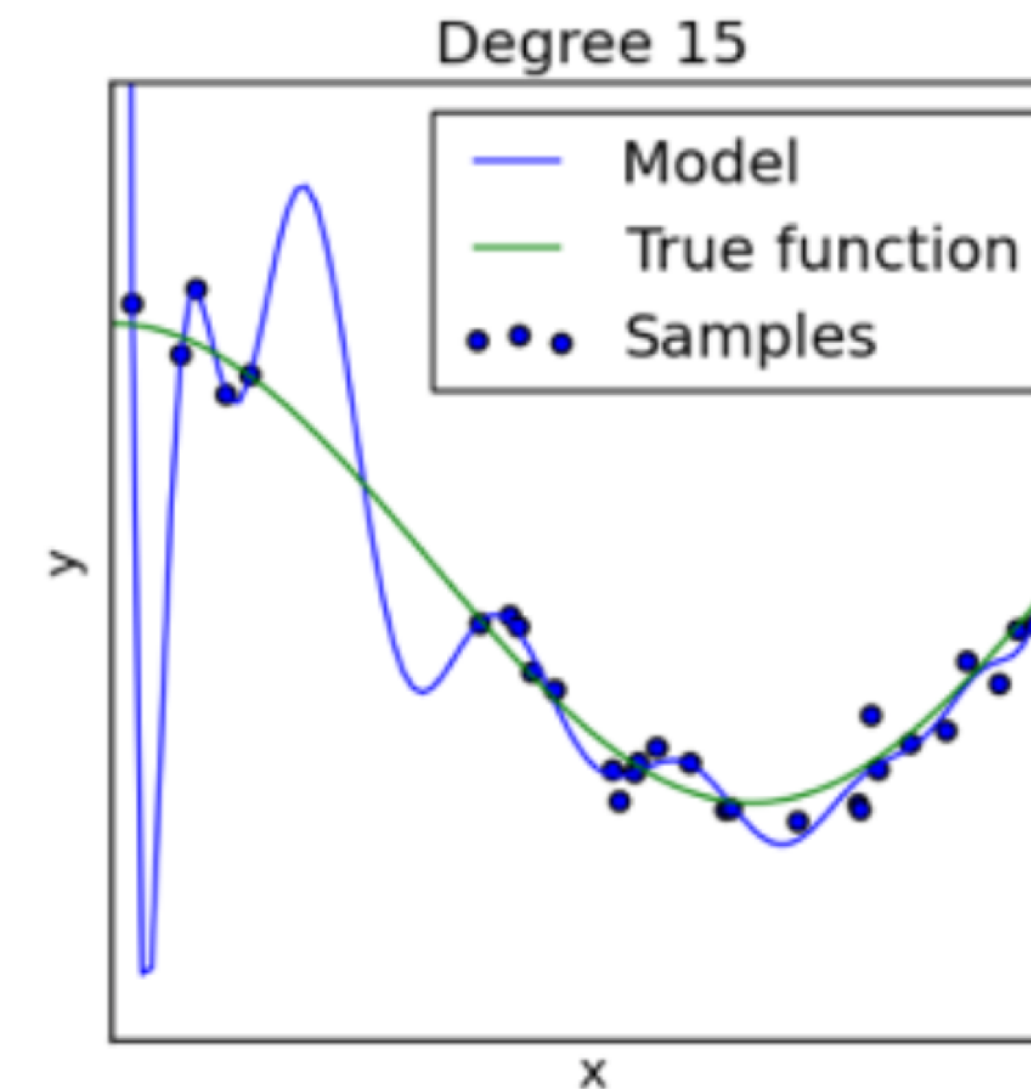
- *If your model has not enough flexibility, it will not be able to describe the data*
- *The training and validation loss will be close, but their value will not decrease*
- *The model is said to be underfitting, or being **biased***



- ◎ 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



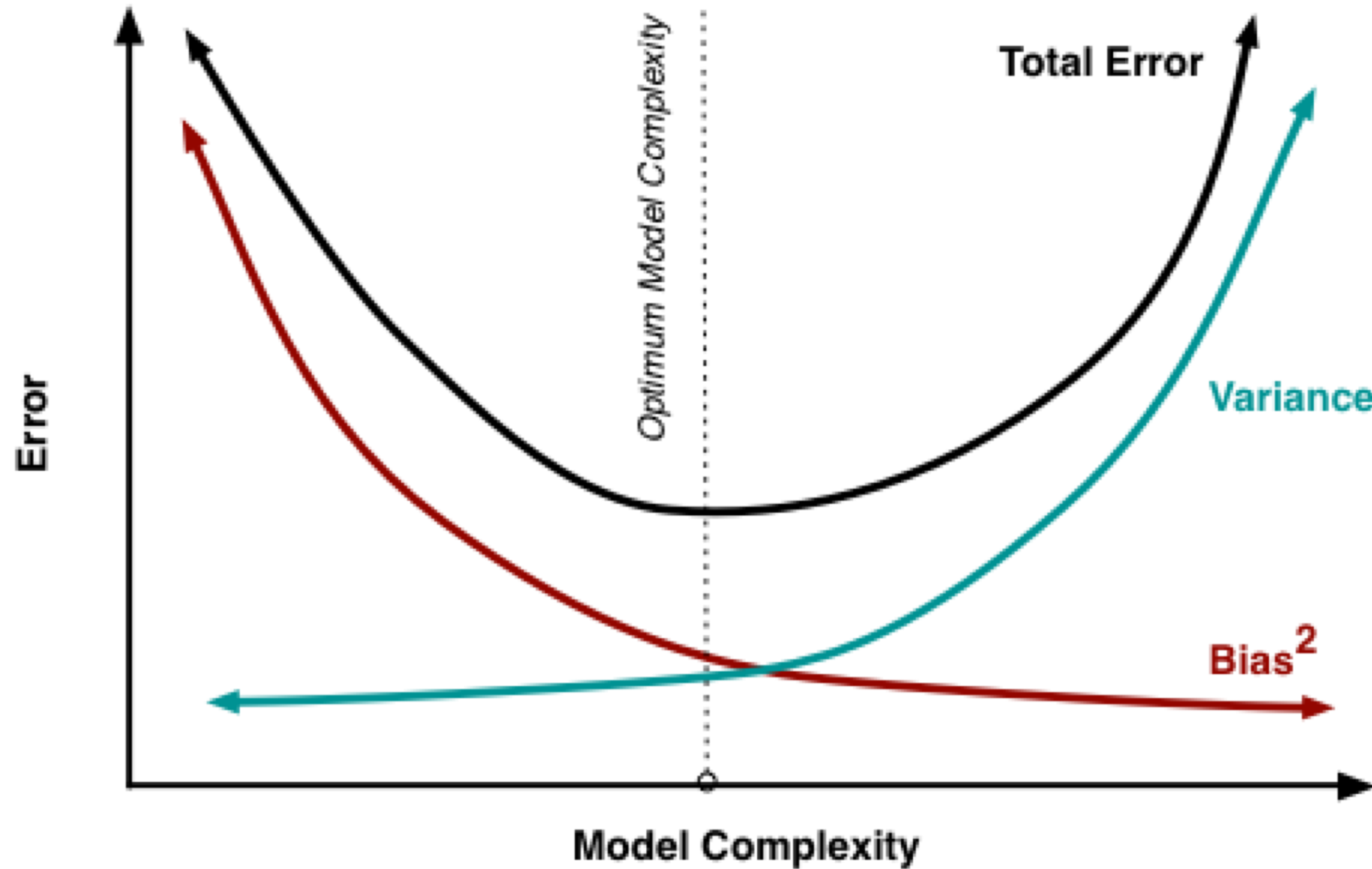
- A model would underfit if too simple: it will not be able to model the mean value
- A model would overfit if too complex: it will reproduce the mean value, but it will underestimate the variance of the data
- The generalization error is the error made going from the training sample to another sample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



- 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**



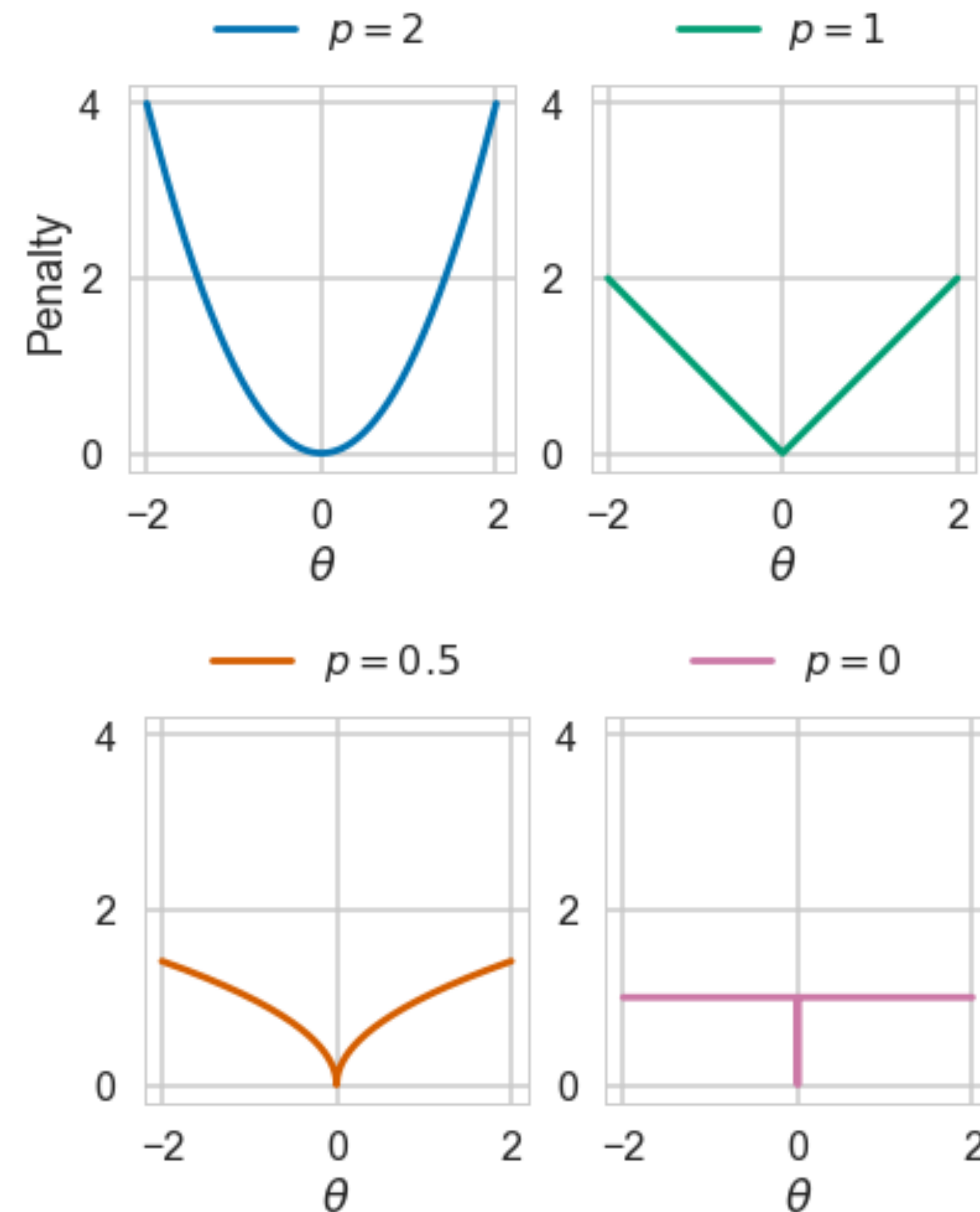
- Model complexity can be “optimized” when minimizing the loss
- A modified loss is introduced, with a penalty term attached to each model parameter

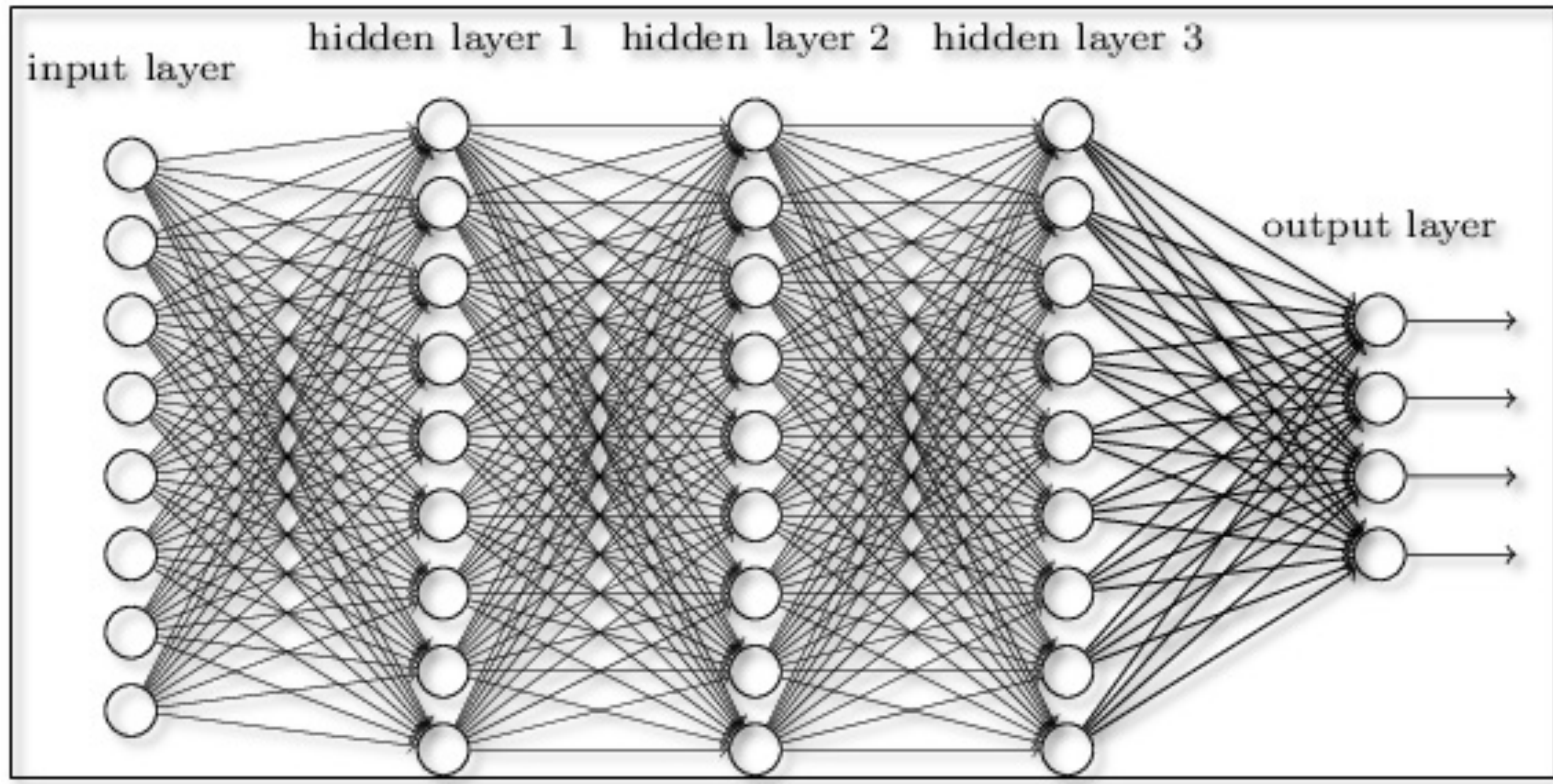
$$L_{reg} = L + \Omega(w)$$

- For instance, L_p regularisation

$$L_p = \|w\|^p = \sum_i |w_i|^p$$

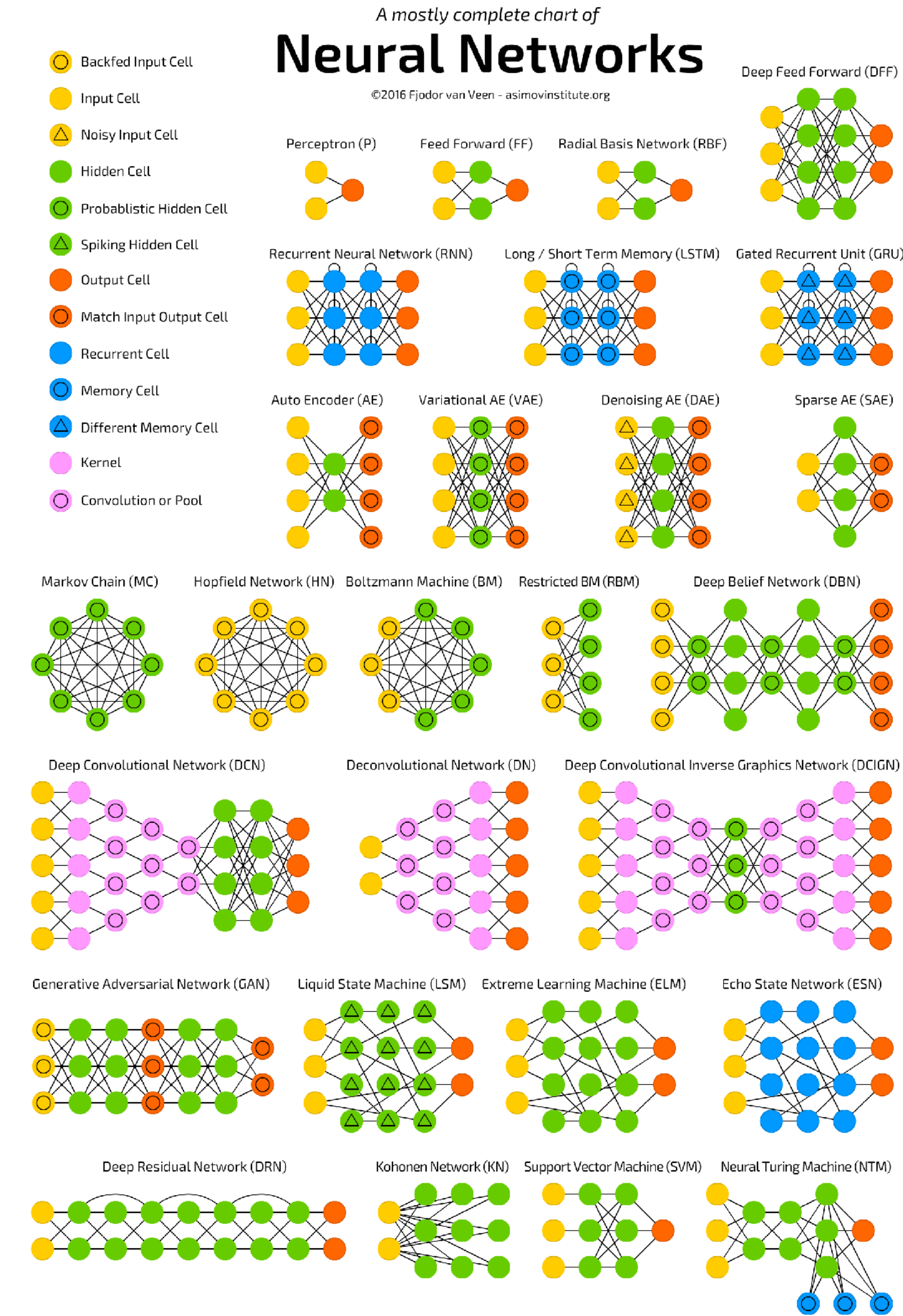
- The minimisation is a tradeoff between:
 - pushing down the 1st term by taking advantage of the parameters
 - pushing down the 2nd term by switching off the parameters



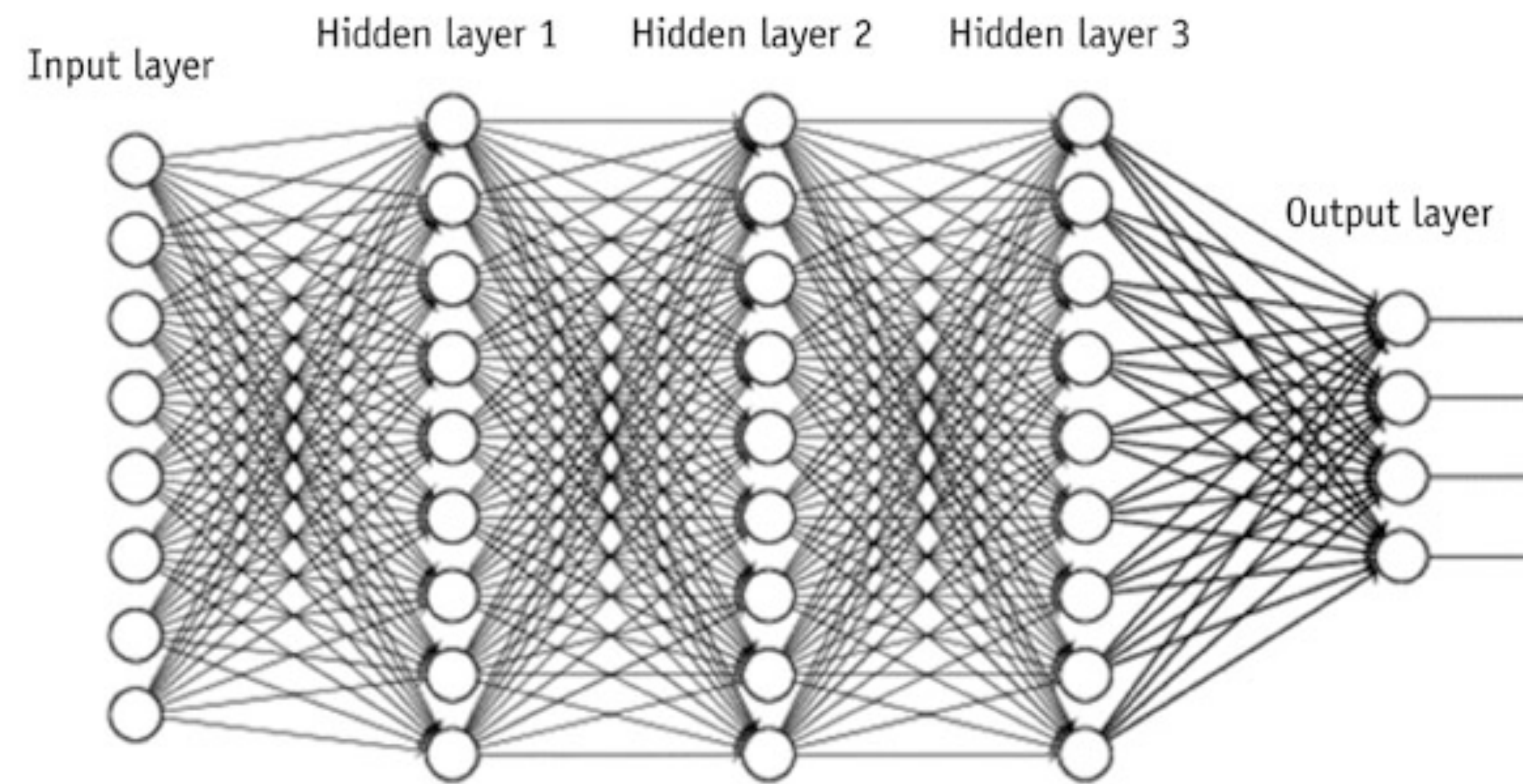


Deep Learning

- 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 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
 Anand Madhavan
 Andrew Y. Ng
 Computer Science Department, Stanford University, Stanford CA 94305 USA

RAJATR@CS.STANFORD.EDU
 MANAND@STANFORD.EDU
 ANG@CS.STANFORD.EDU

What is DL used for

Image processing



text/sound processing



Reinforcement Learning

Everything is a Recommendation

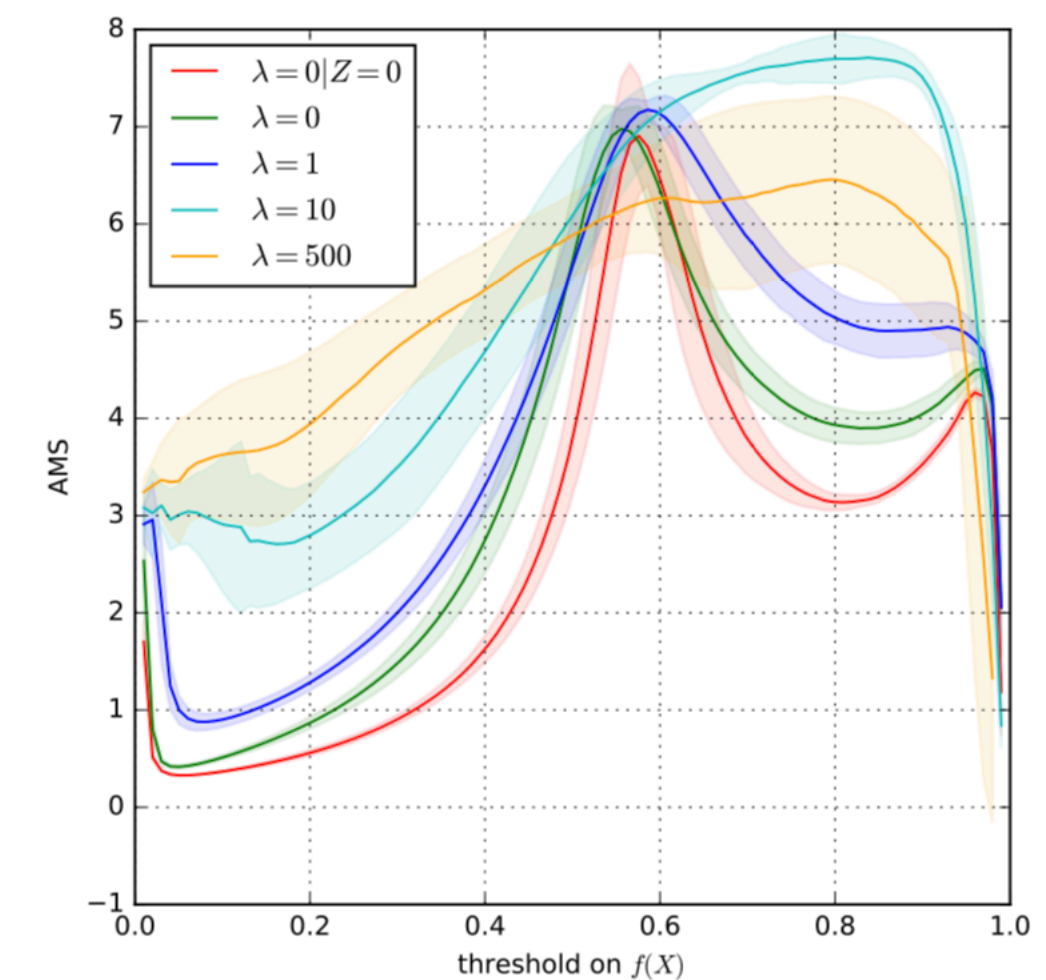
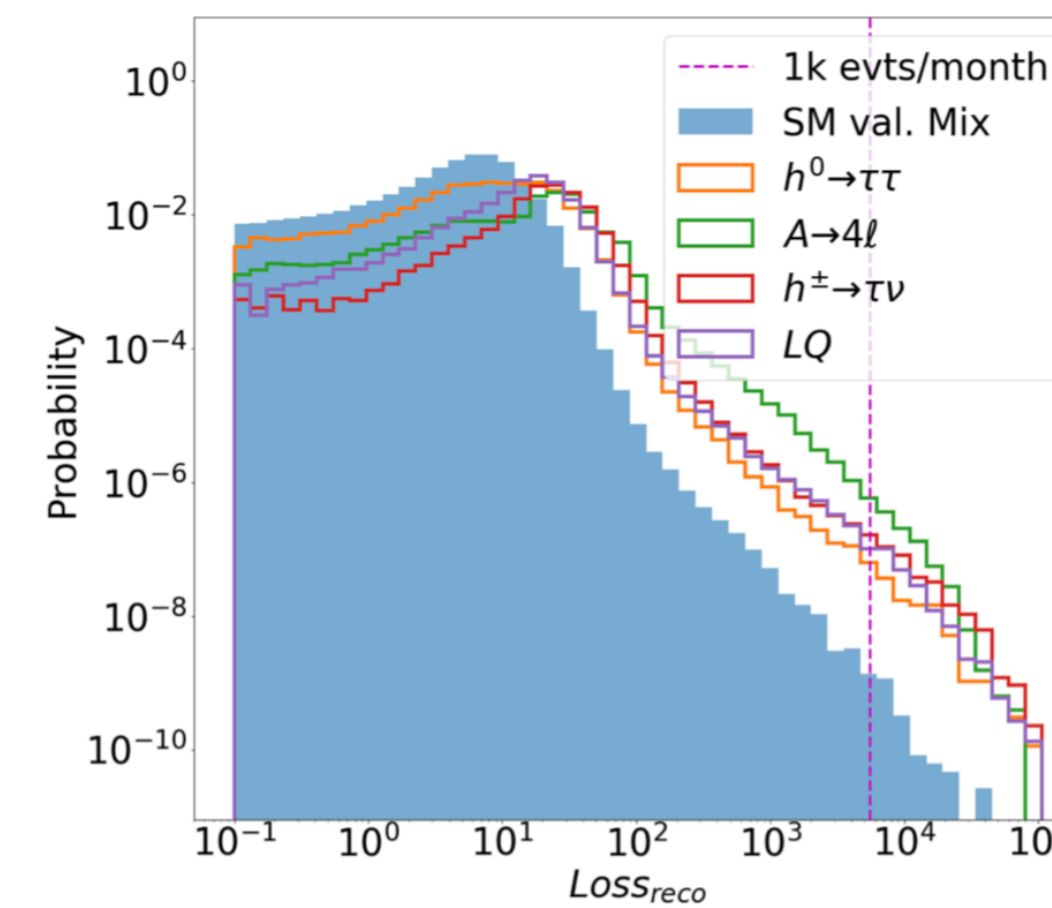
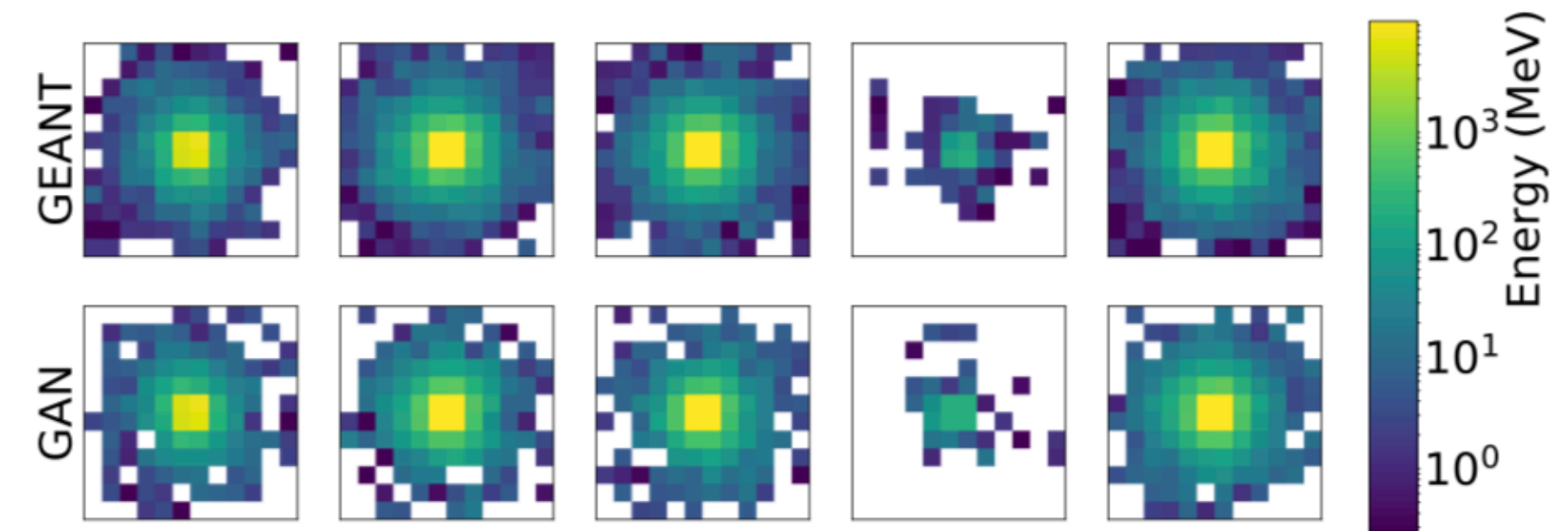


Over 75% of what people watch comes from our recommendations

Recommendations are driven by Machine Learning

Clustering

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



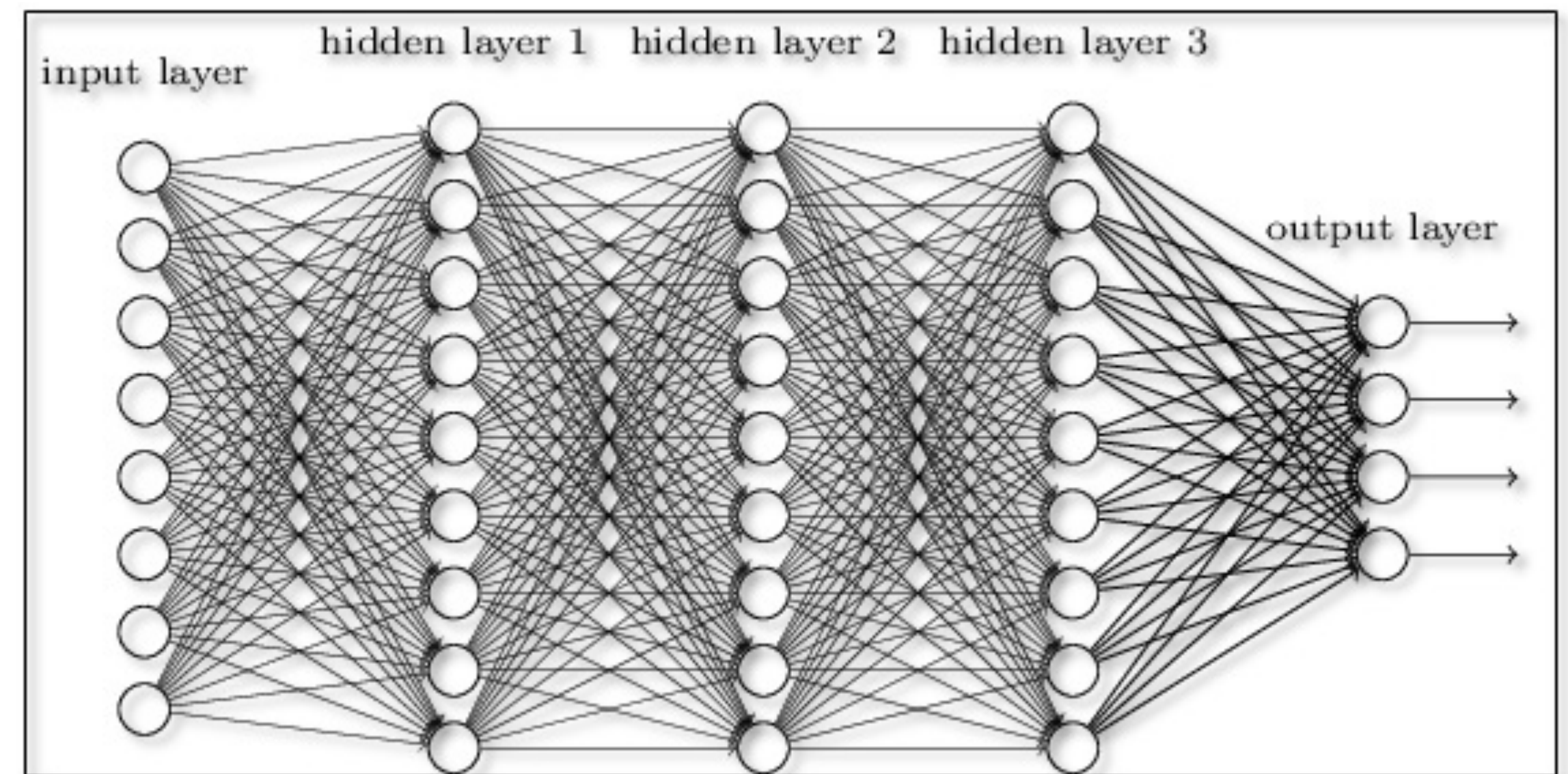
Jet grooming with reinforcement learning

We use a Deep Q-Network as a RL algorithm which uses a table of $Q(s, a)$, determining the next action as the one that maximizes Q .

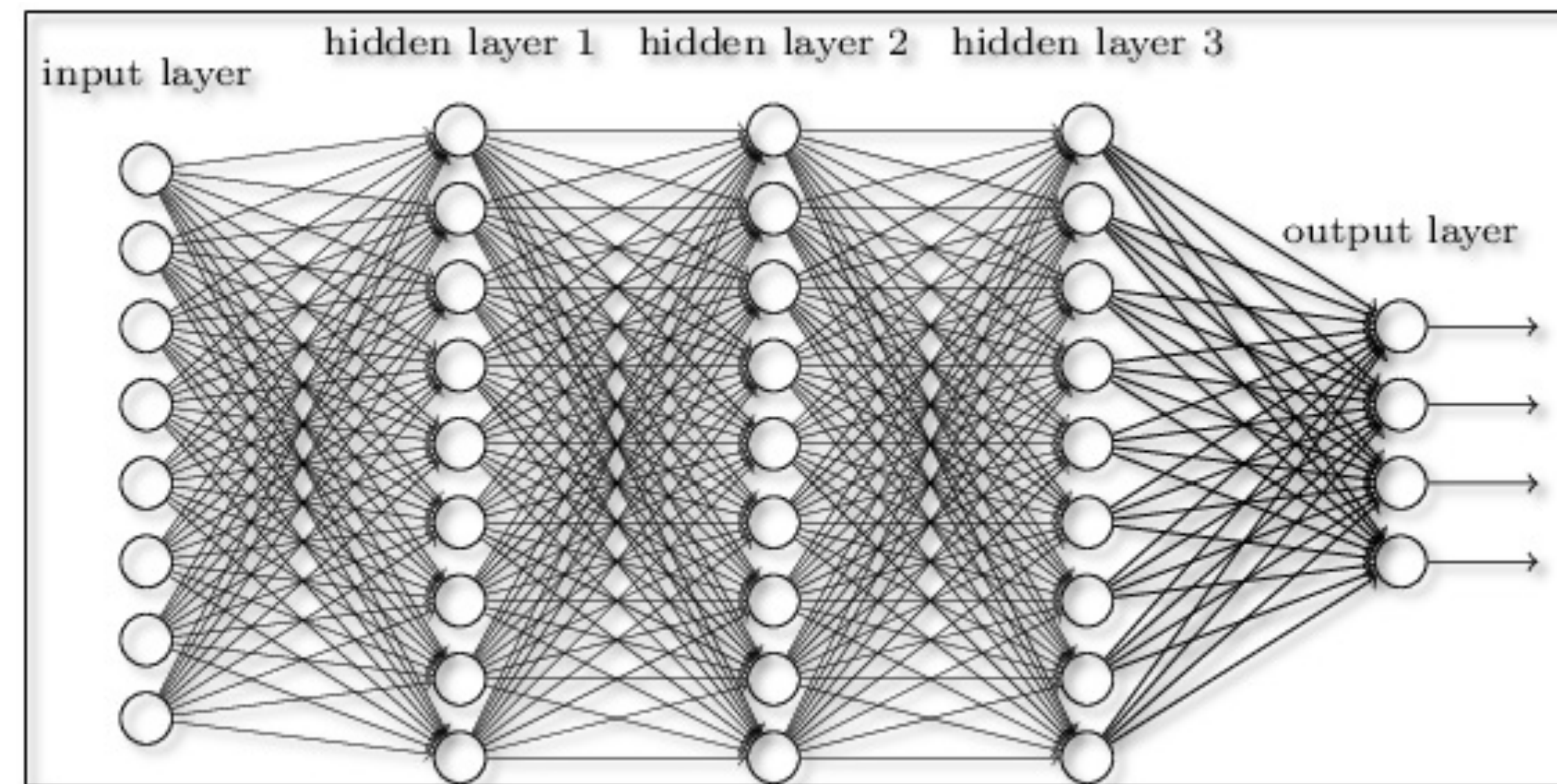
A NN is used to approximate the optimal action-value function:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \dots | s_t = s, a_t = a, \pi]$$

- 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

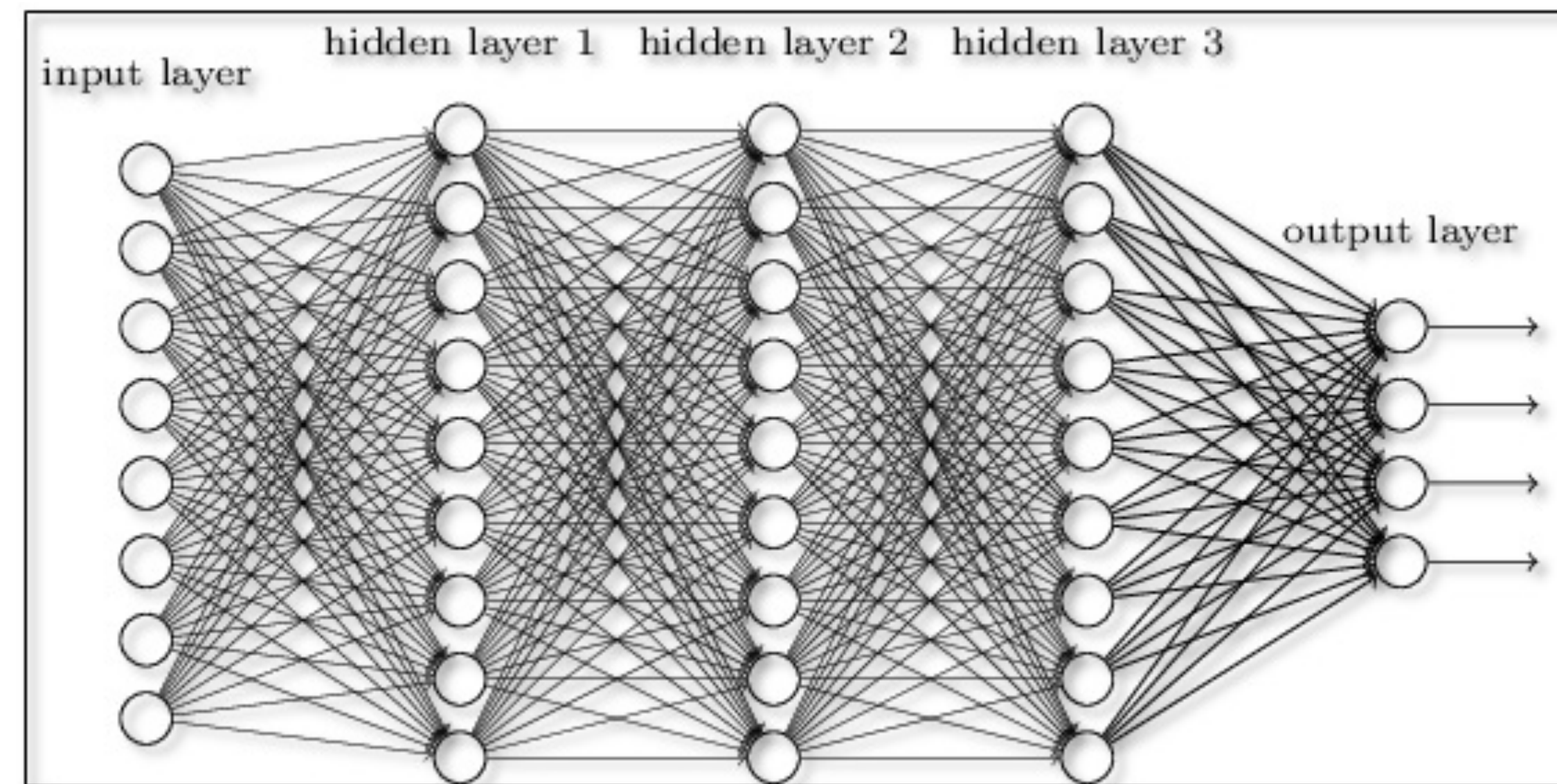


- Each input is multiplied by a weight
- The weighted values are summed
- A bias is added
- The result is passed to an activation function



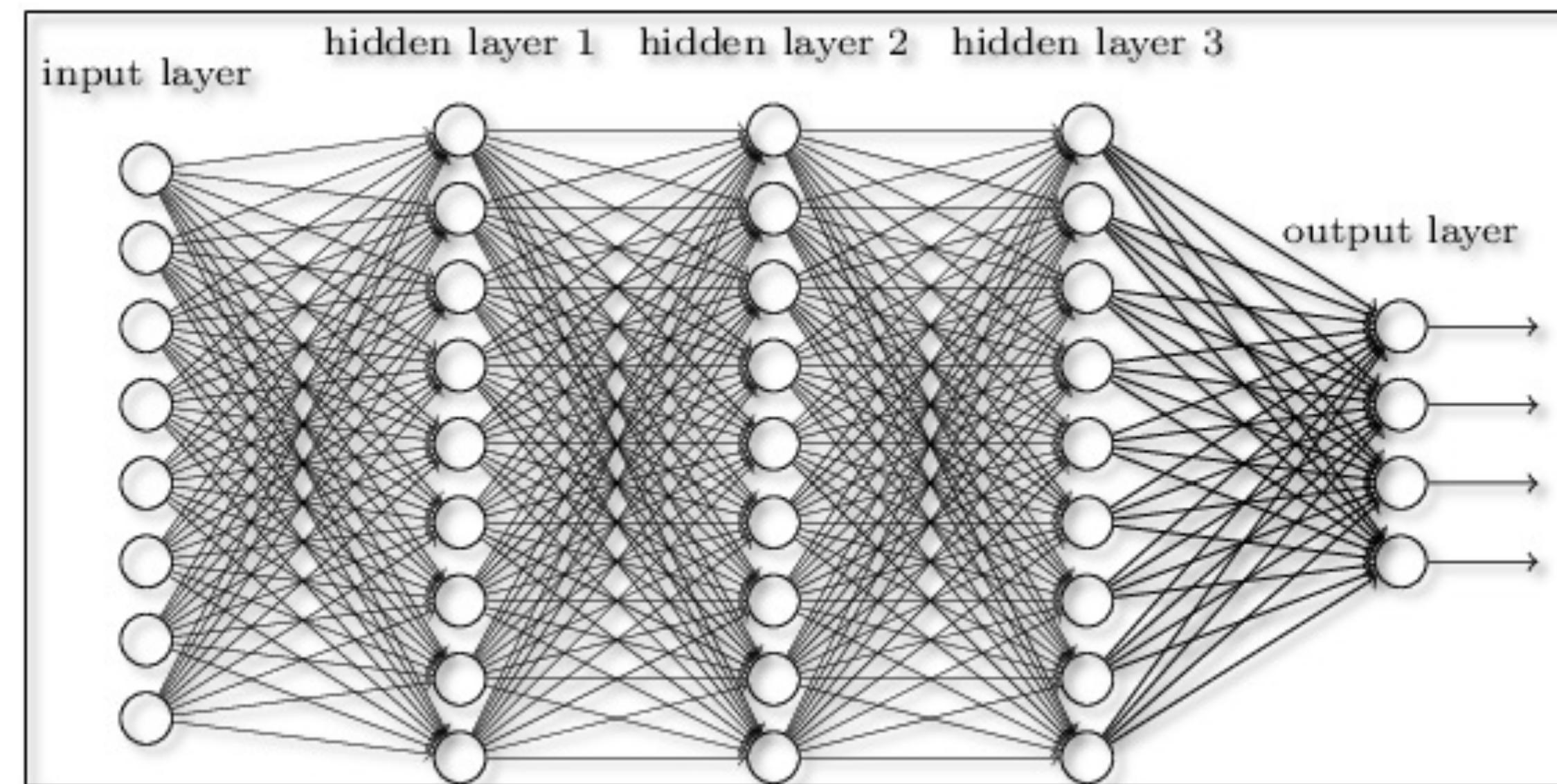
$$W_{ij}x_j$$

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



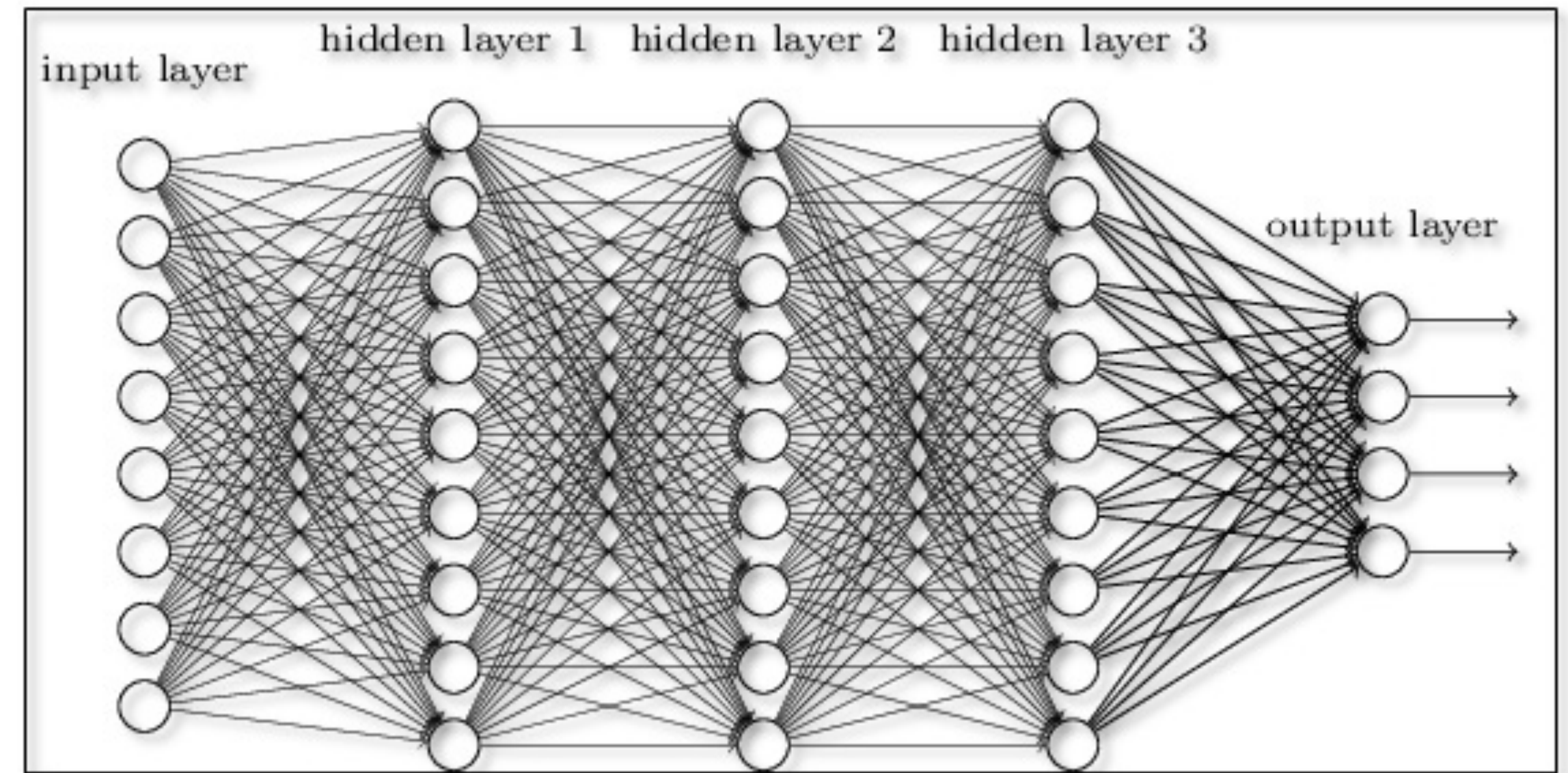
$$\sum_j w_{ij} x_j$$

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



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

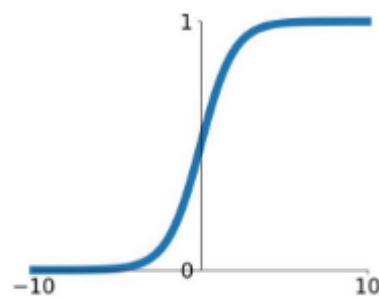
- Each input is multiplied by a weight
- The weighted values are summed
- A bias is added
- **The result is passed to an activation function**



Activation Functions

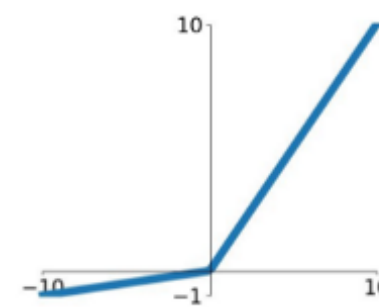
Sigmoid

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



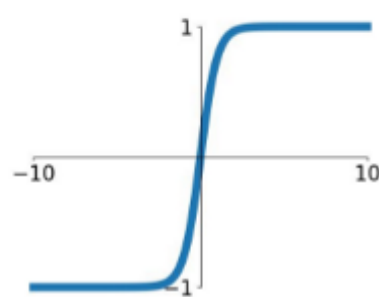
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

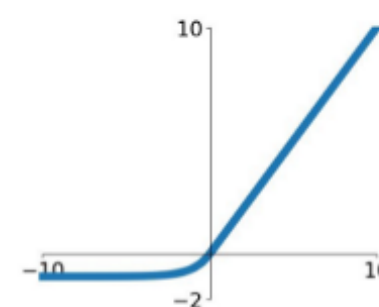


Maxout

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

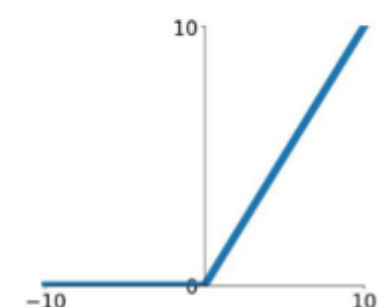
ELU

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



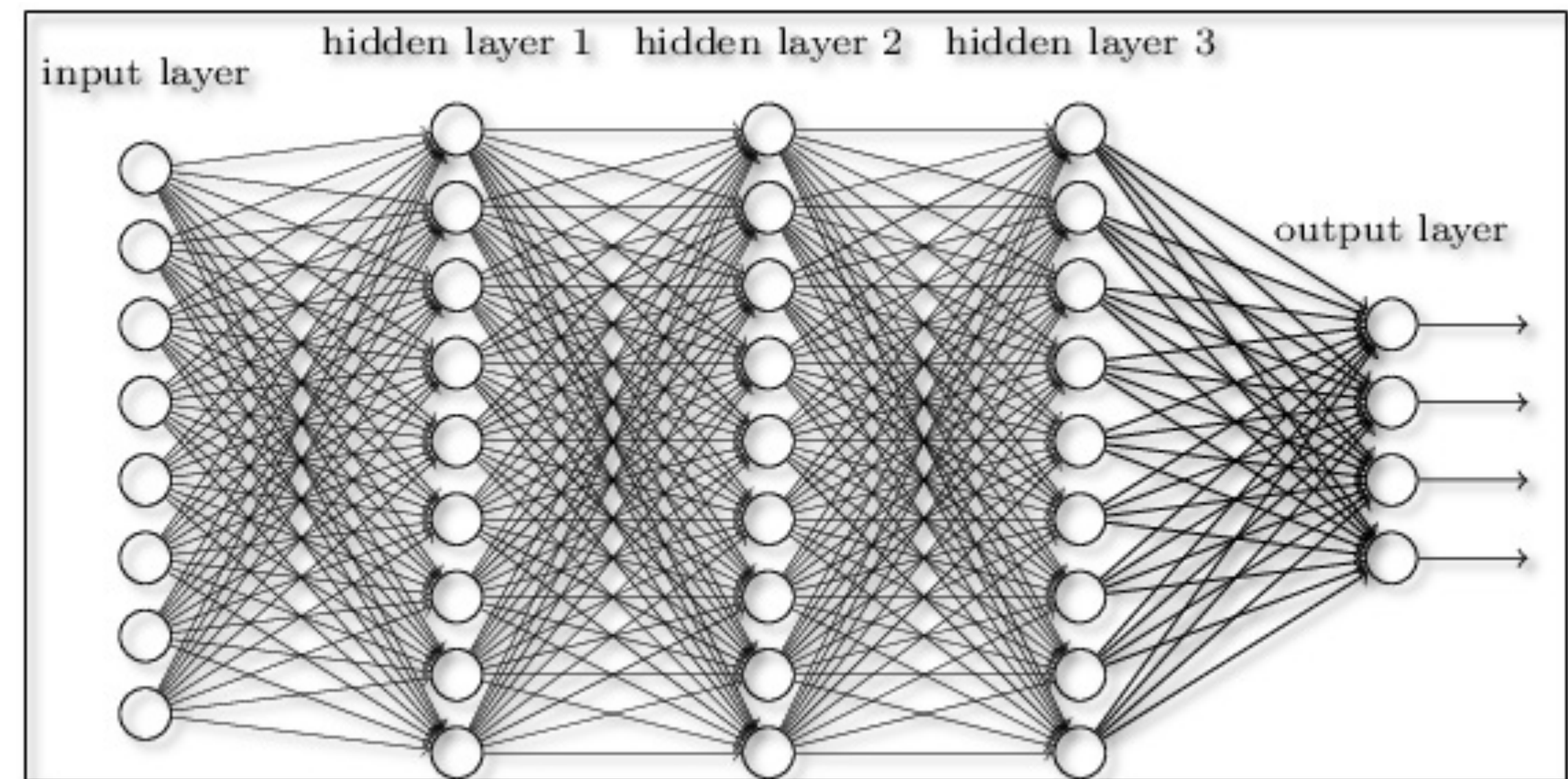
ReLU

$$\max(0, x)$$



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

- *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*



$$y_j = f^{(3)}\left(\sum_l w_{jl}^{(3)} f^{(2)}\left(\sum_k w_{lk}^{(2)} f^{(1)}\left(\sum_i w_{ki}^{(1)} x_i + b_k^{(1)}\right) + b_l^{(2)}\right) + b_j^{(3)}\right)$$

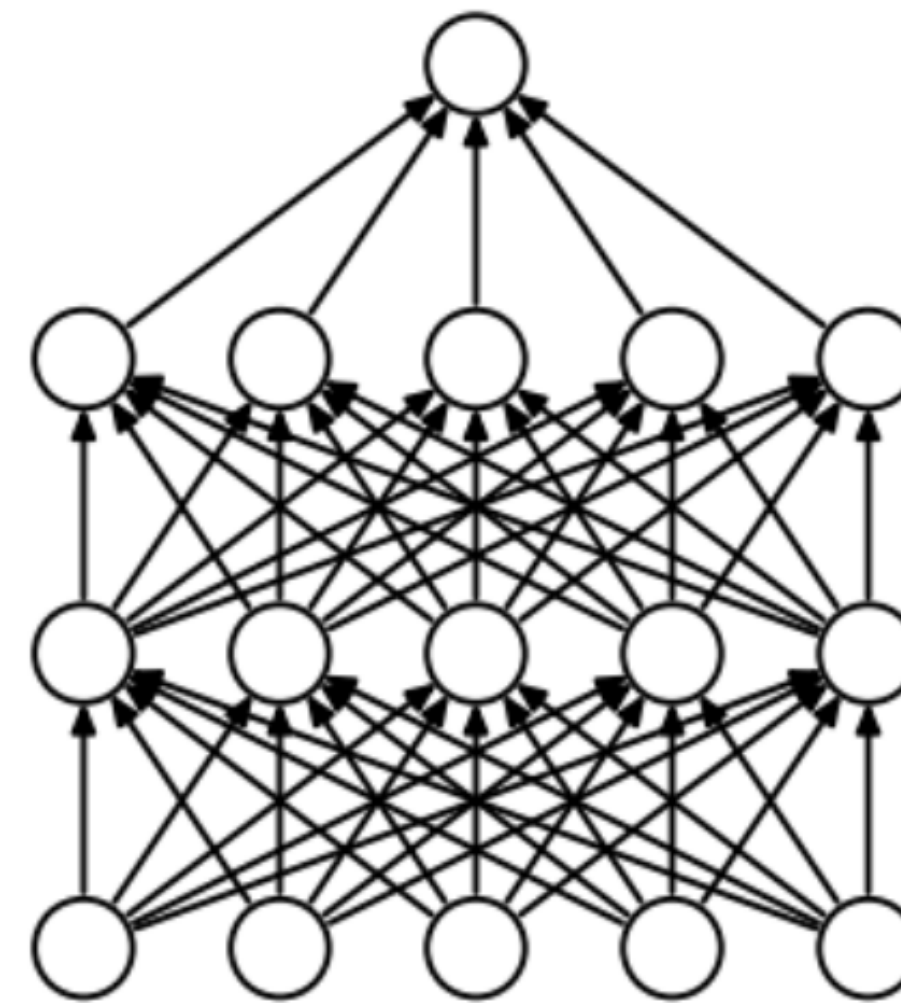
- *One can show that such a mechanism allows to learn generic $\mathbb{R}^N \rightarrow \mathbb{R}^K$ functions*

- 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.)

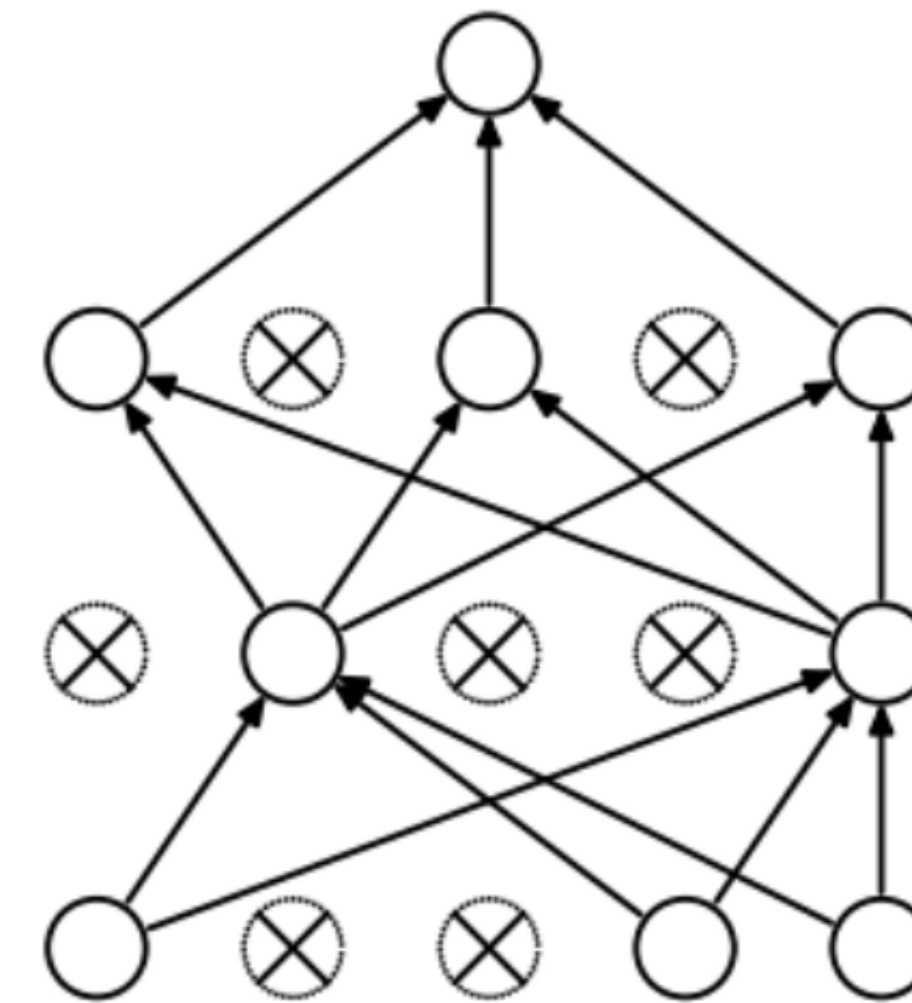
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
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		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Dropout Layer

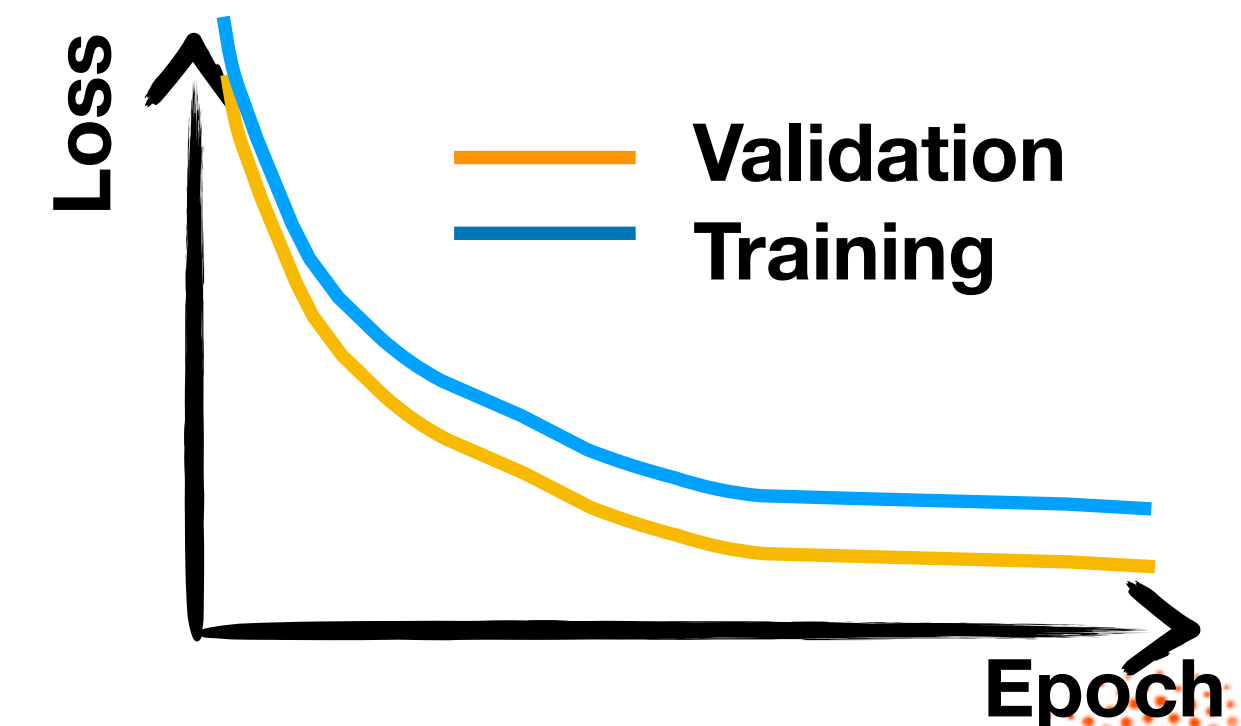
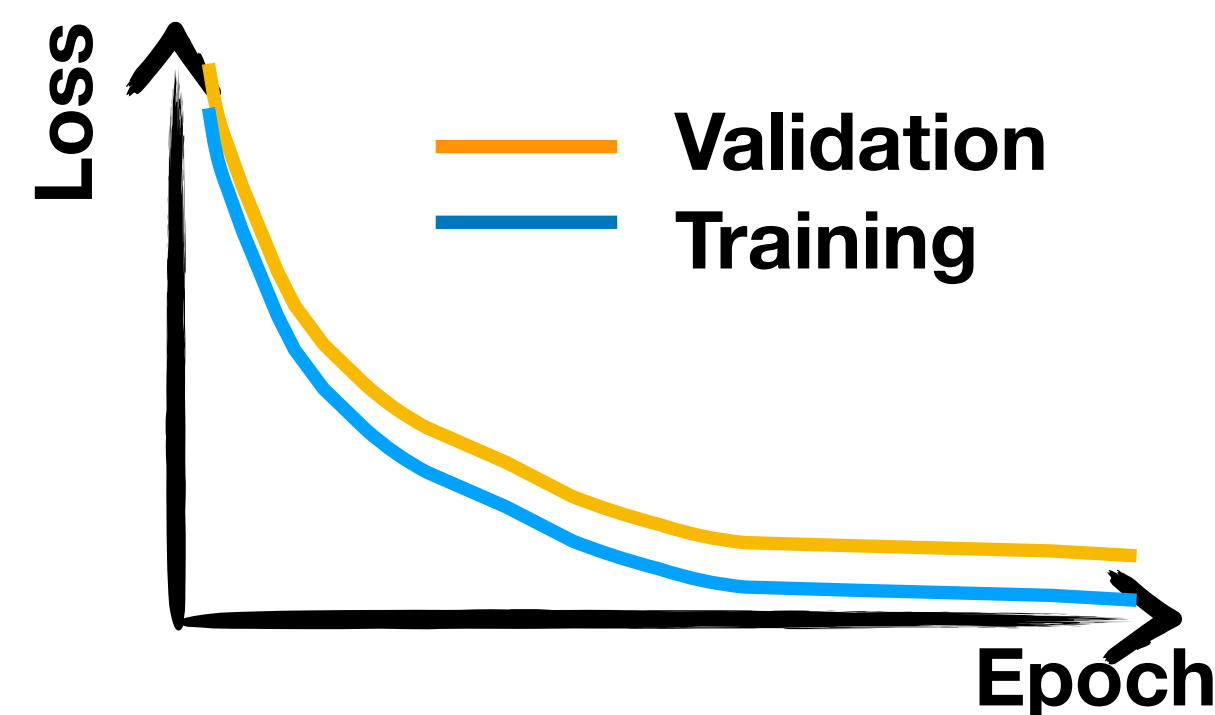
- A special kind of layer, introduced for regularisation purpose
- Randomly drop links between neurons, with probability p
- The connections are re-established during the validation and inference steps
- Typical sign of it: invert hierarchy between training and validation loss



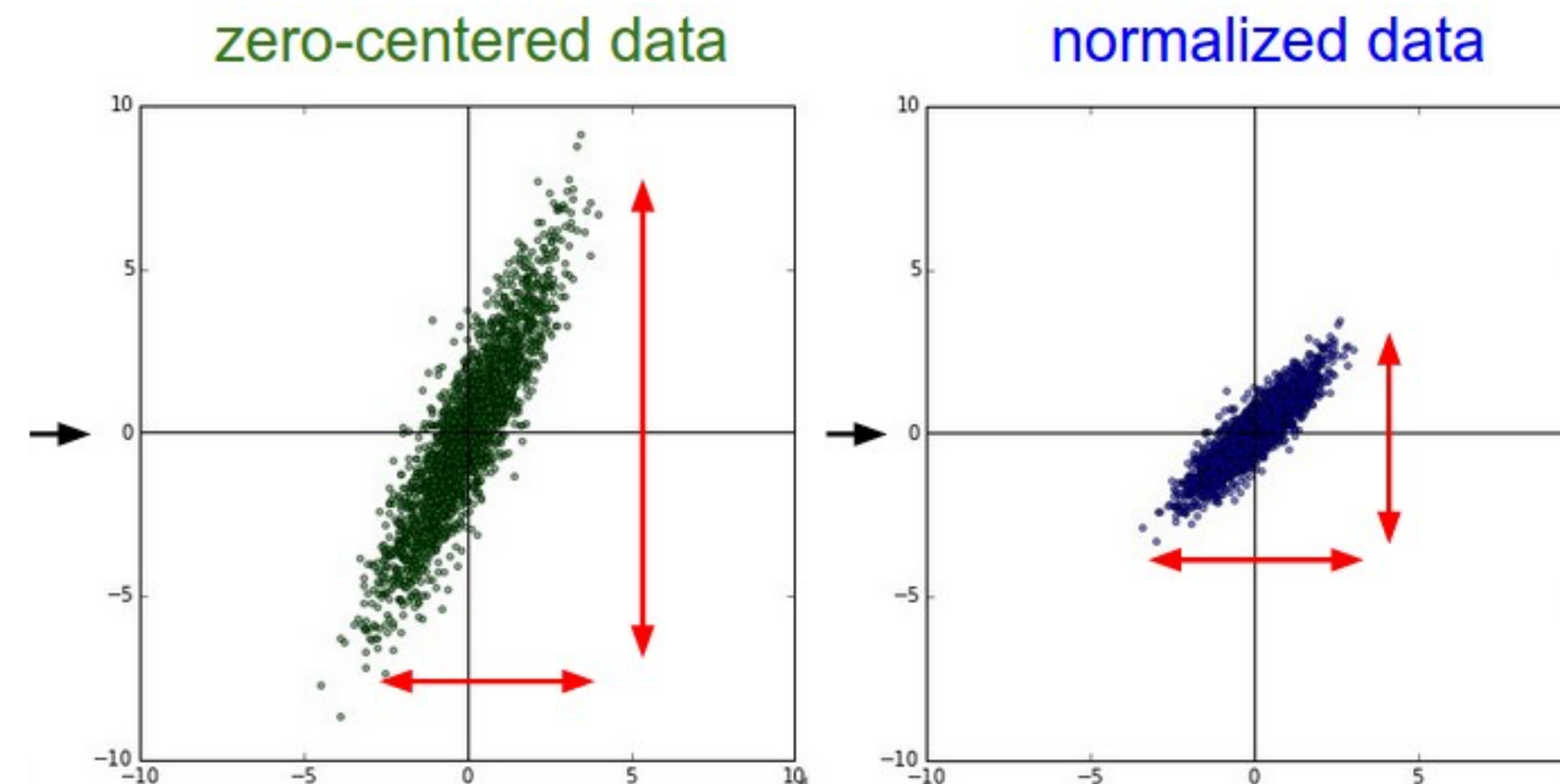
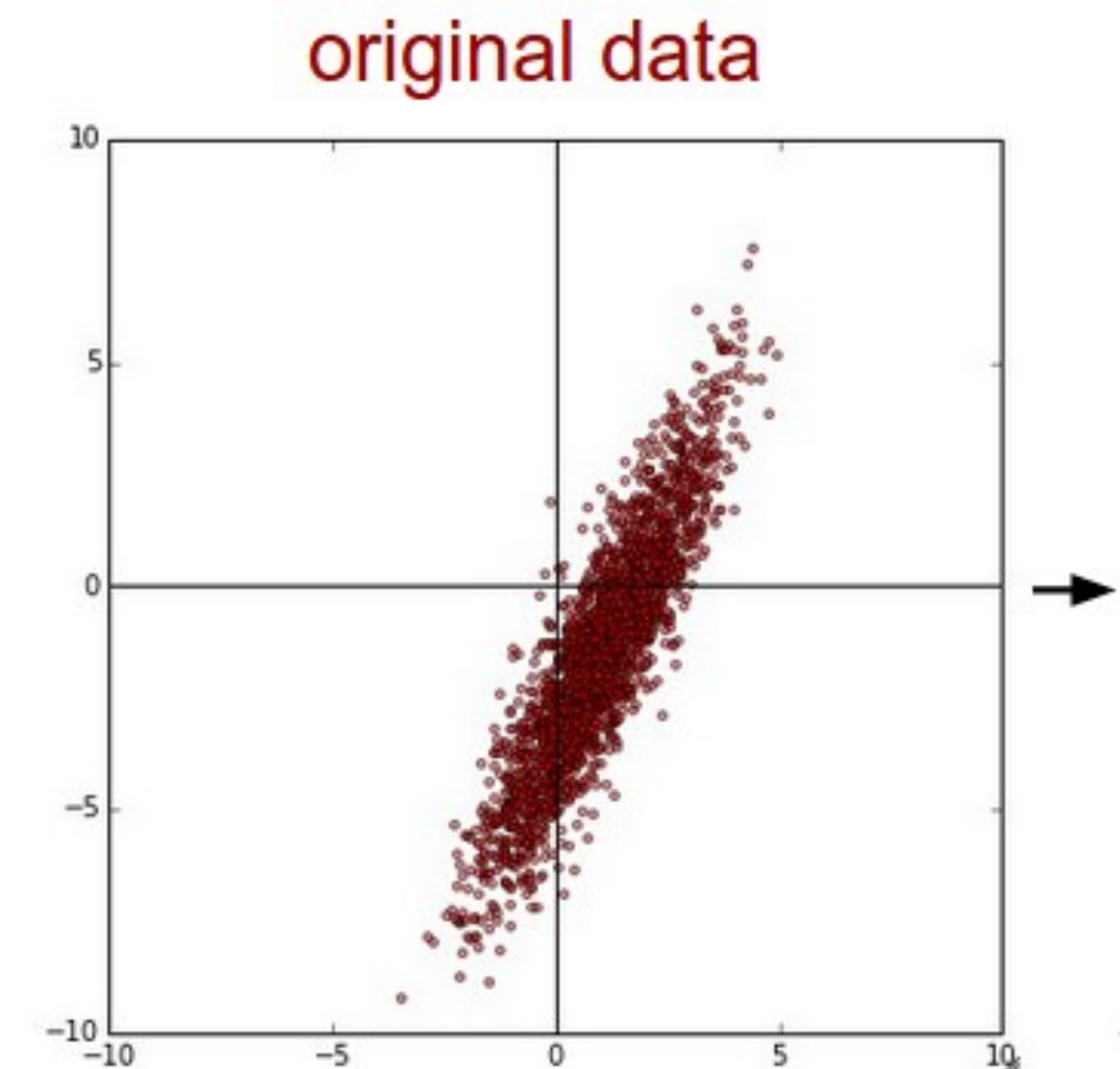
(a) Standard Neural Net



(b) After applying dropout.

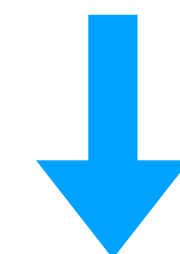


- It is good practice to give normalized inputs to a layer
- With all inputs having the same order of magnitude, all weights are equal important in the gradient
- Prevents explosion of the loss function
- This can be done automatically with BatchNormalization
- non-learnable shift and scale parameters, adjusted batch by batch



- ◎ *Dense NN architectures can be made more complex*
 - ◎ *Multiple inputs*
 - ◎ *Multiple outputs*
 - ◎ *Different networks branches*
- ◎ *This is possible thanks to layer-manipulation layers*
 - ◎ *Add, Subtract, etc.*
 - ◎ *Concatenation*
 - ◎ *Flattening*
- ◎ *All these operations are usually provided with NN training libraries*

8 0 4 7 6 8 0 7 9 4 6 5 2 6 8 3 4 5 5 3 4



Concatenation

8 0 4 7 6 8 0 8 3 4 5 5 3 4 7 9 4 6 5 2 6

8	0	4	7	6	8	0
8	3	4	5	5	3	4
7	9	4	6	5	2	6

Flattening

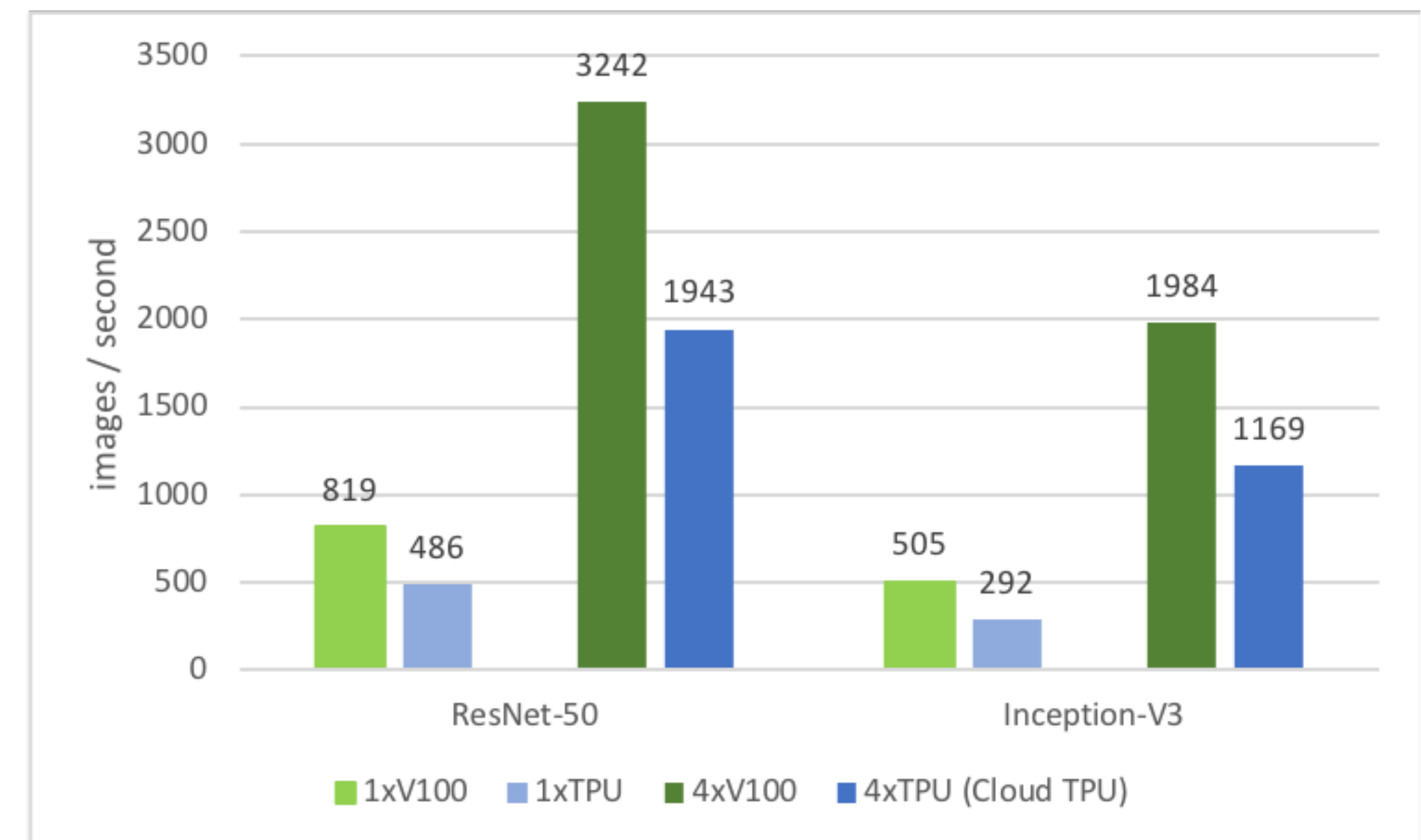


8 0 4 7 6 8 0 8 3 4 5 5 3 4 7 9 4 6 5 2 6

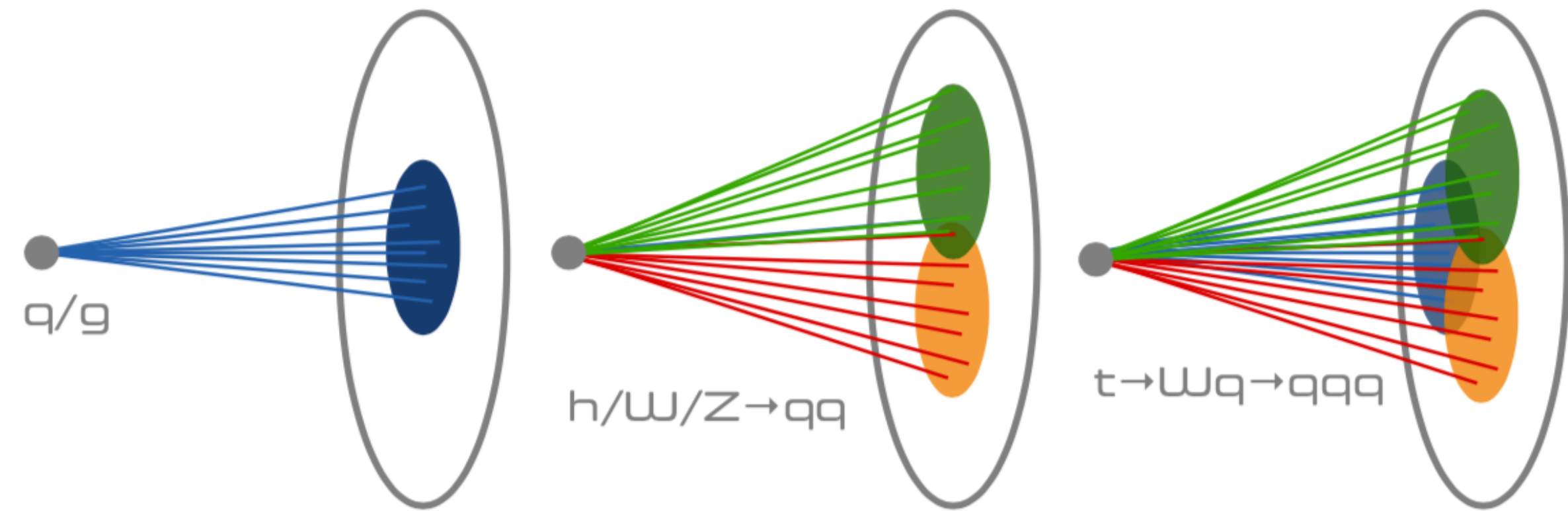
- ◎ *Many solutions exist. Most popular softwares live in a python ecosystem*
 - ◎ *Google's TensorFlow*
 - ◎ *Facebook's Pytorch*
 - ◎ *Apache MXnet*
- ◎ *All of them integrated in a data science ecosystem*
 - ◎ *with numpy, scikit, etc.*
- ◎ *Convenient libraries built on top, with pre-coded ingredients*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



- ⦿ 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 parallelise 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



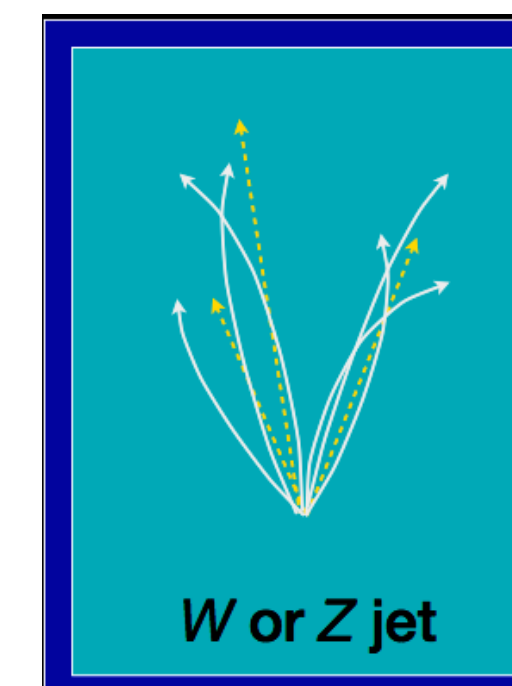
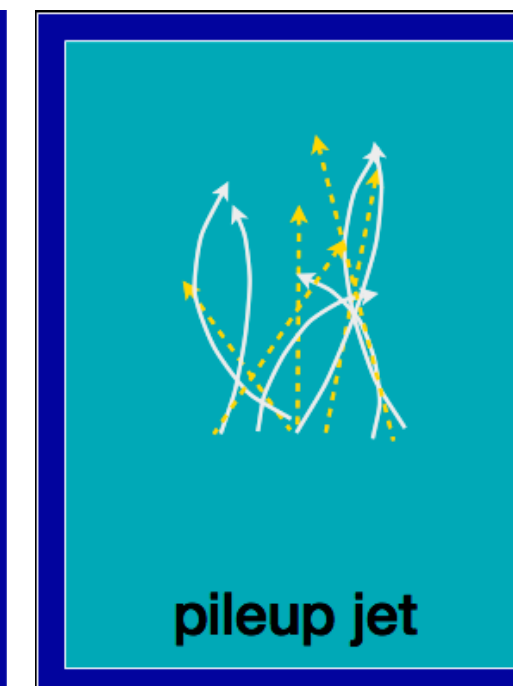
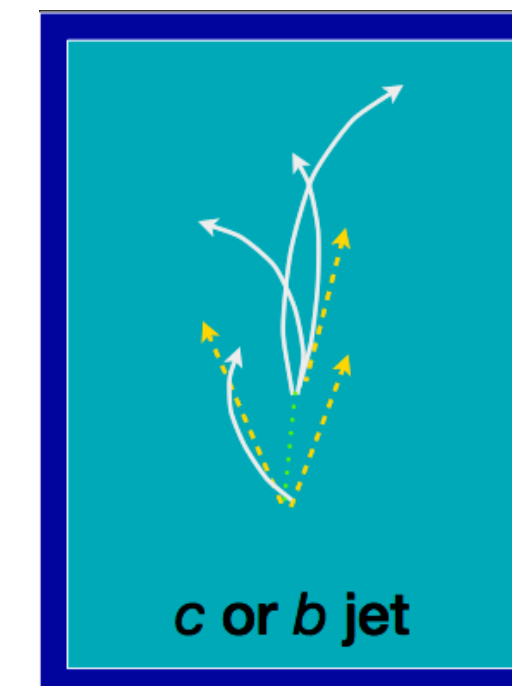
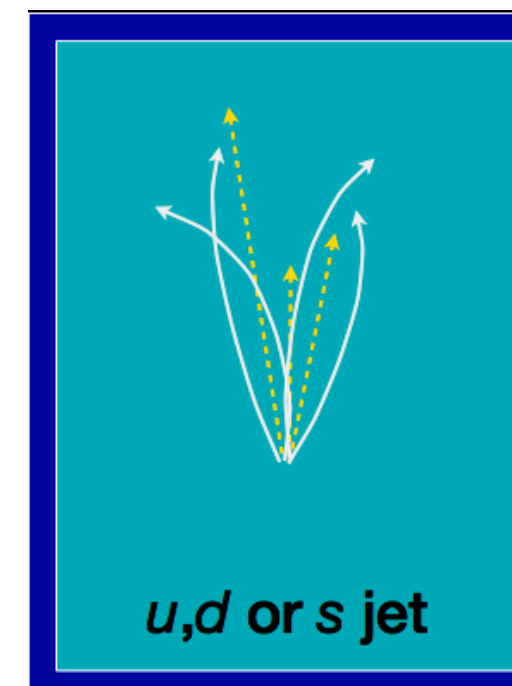
● You have a jet at LHC: spray of hadrons coming from a “shower” initiated by a fundamental particle of some kind (quark, gluon, $W/Z/H$ bosons, top quark)



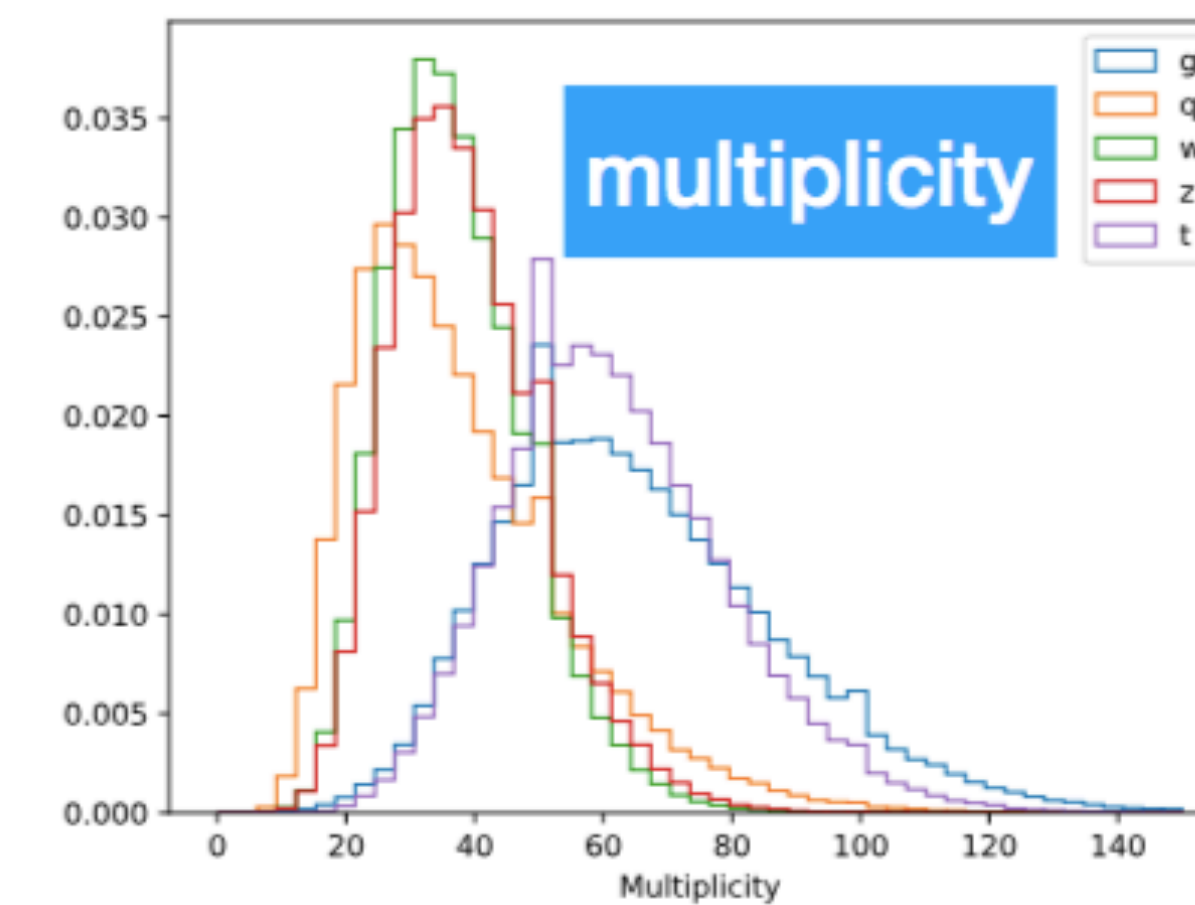
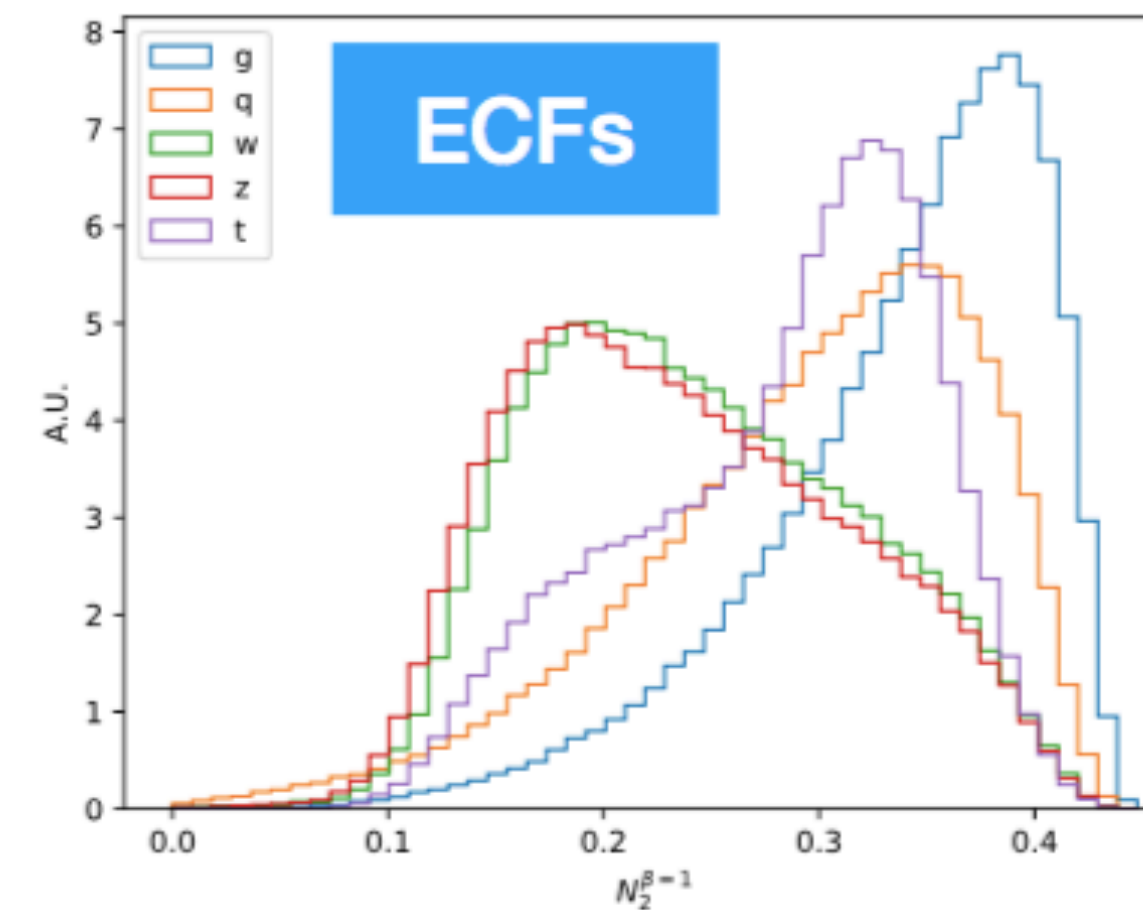
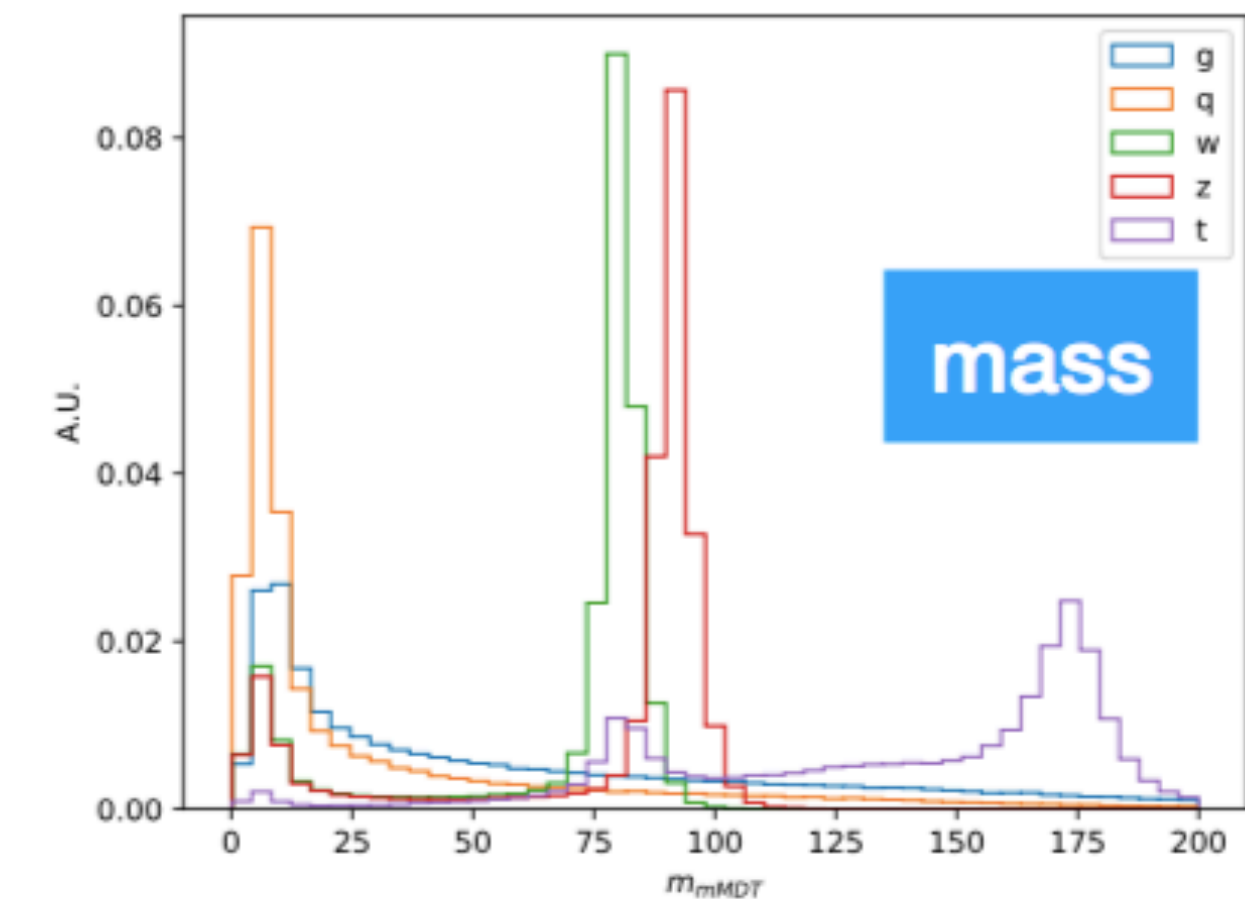
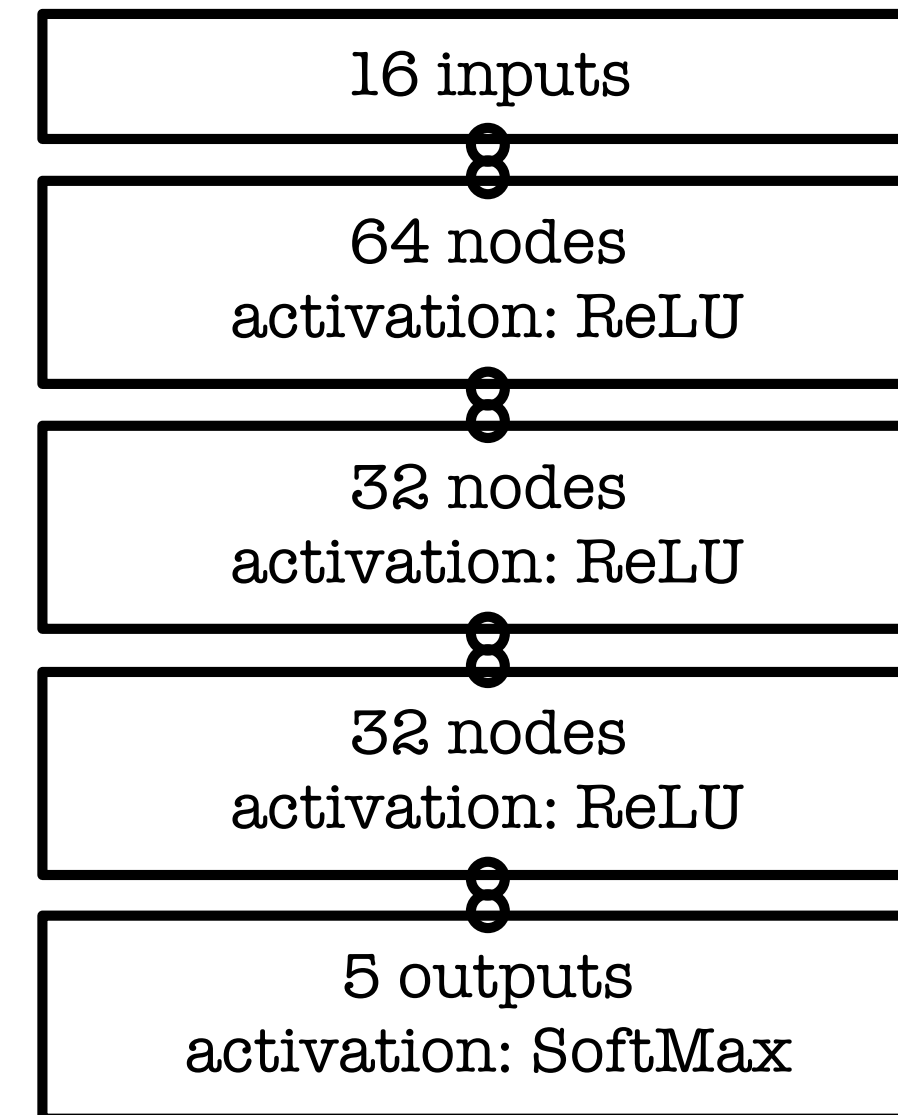
● You have a set of jet features whose distribution depends on the nature of the initial particle

● You can train a network to start from the values of these quantities and guess the nature of your jet

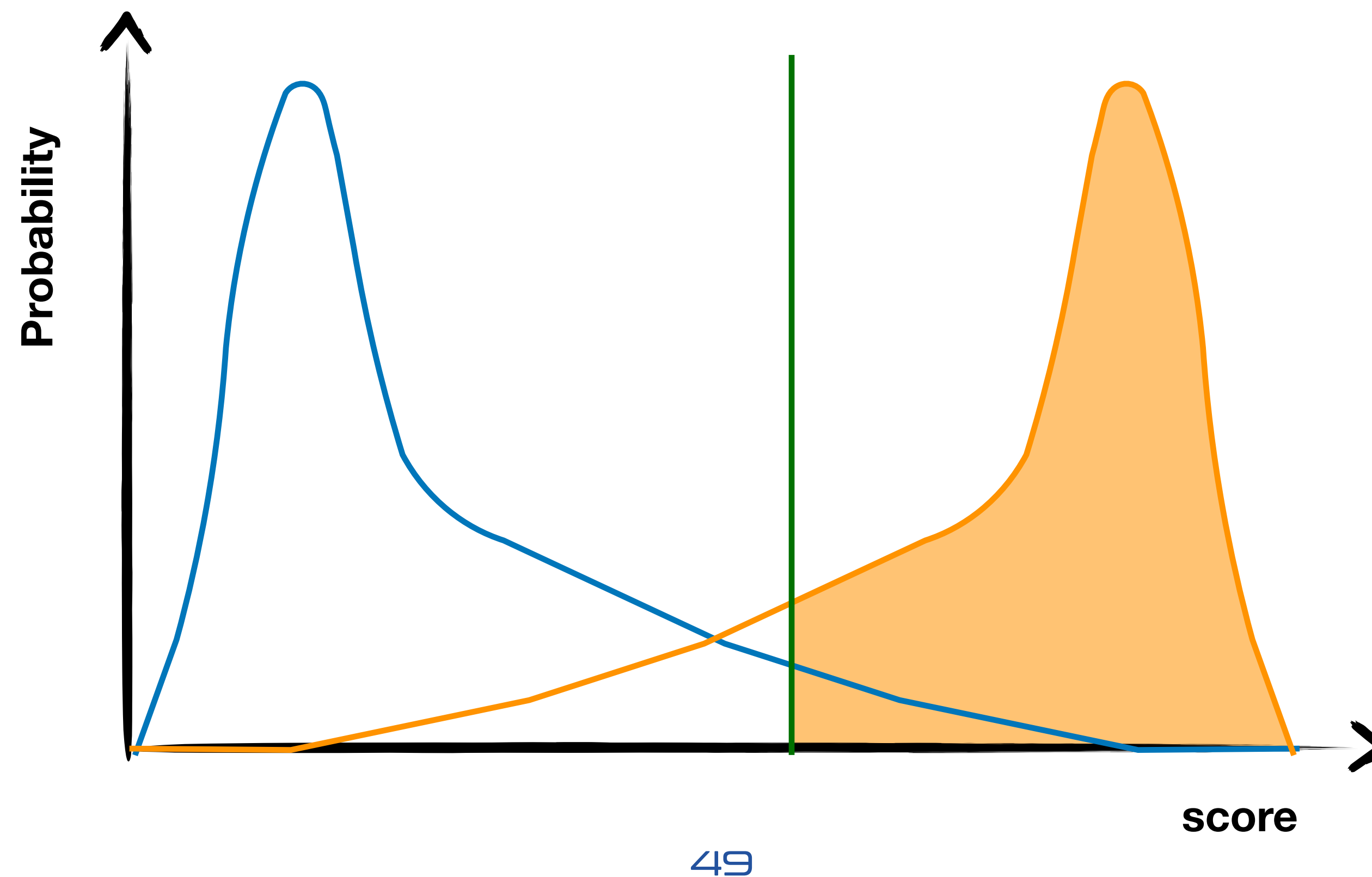
● To do this you need a sample for which you know the answer



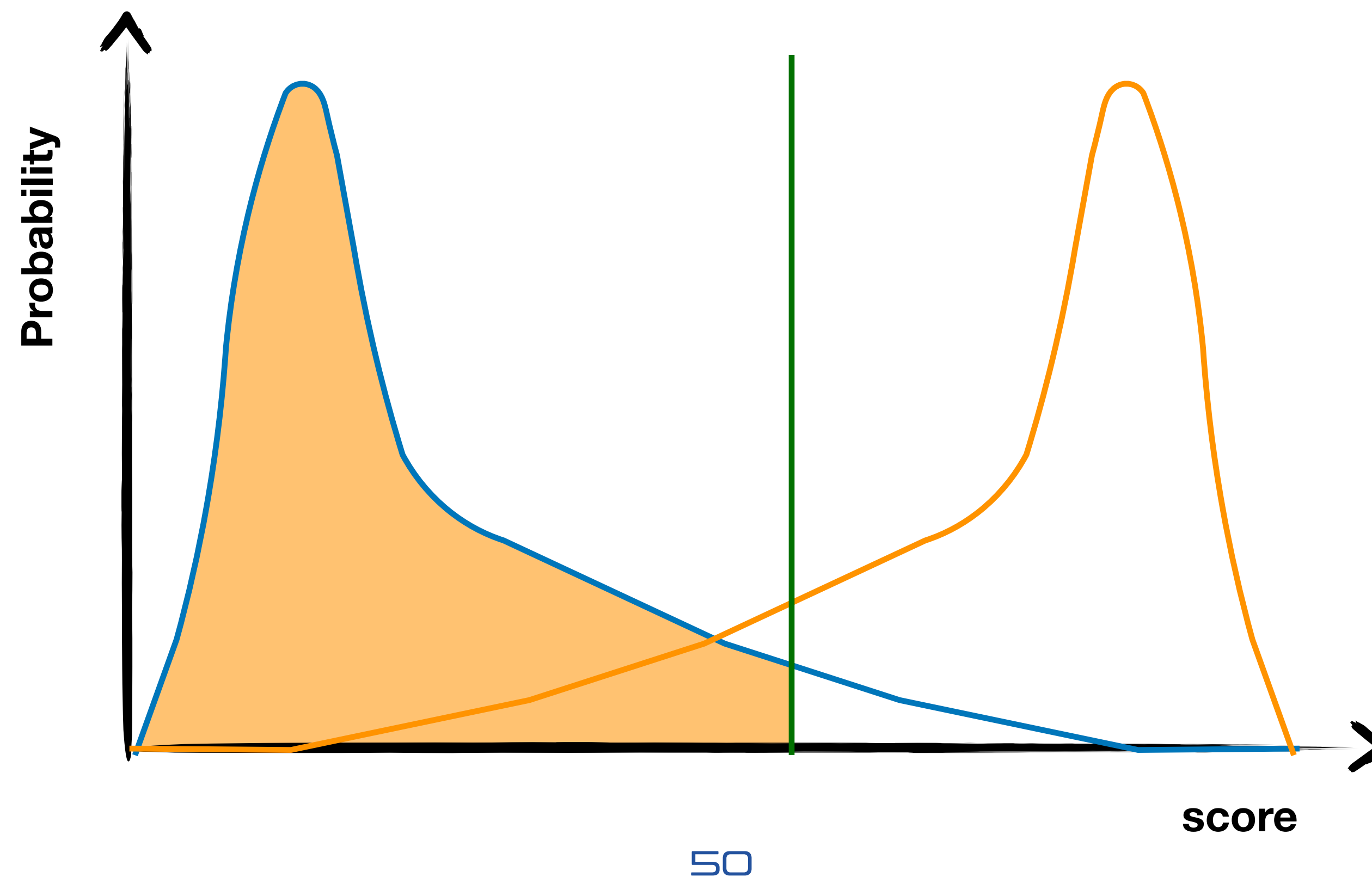
- You have a jet at LHC: spray of hadrons coming from a “shower” initiated by a fundamental particle of some kind (quark, gluon, W/Z/H bosons, top quark)
- You have a set of jet features whose distribution depends on the nature of the initial particle
- You can train a network to start from the values of these quantities and guess the nature of your jet
- To do this you need a sample for which you know the answer



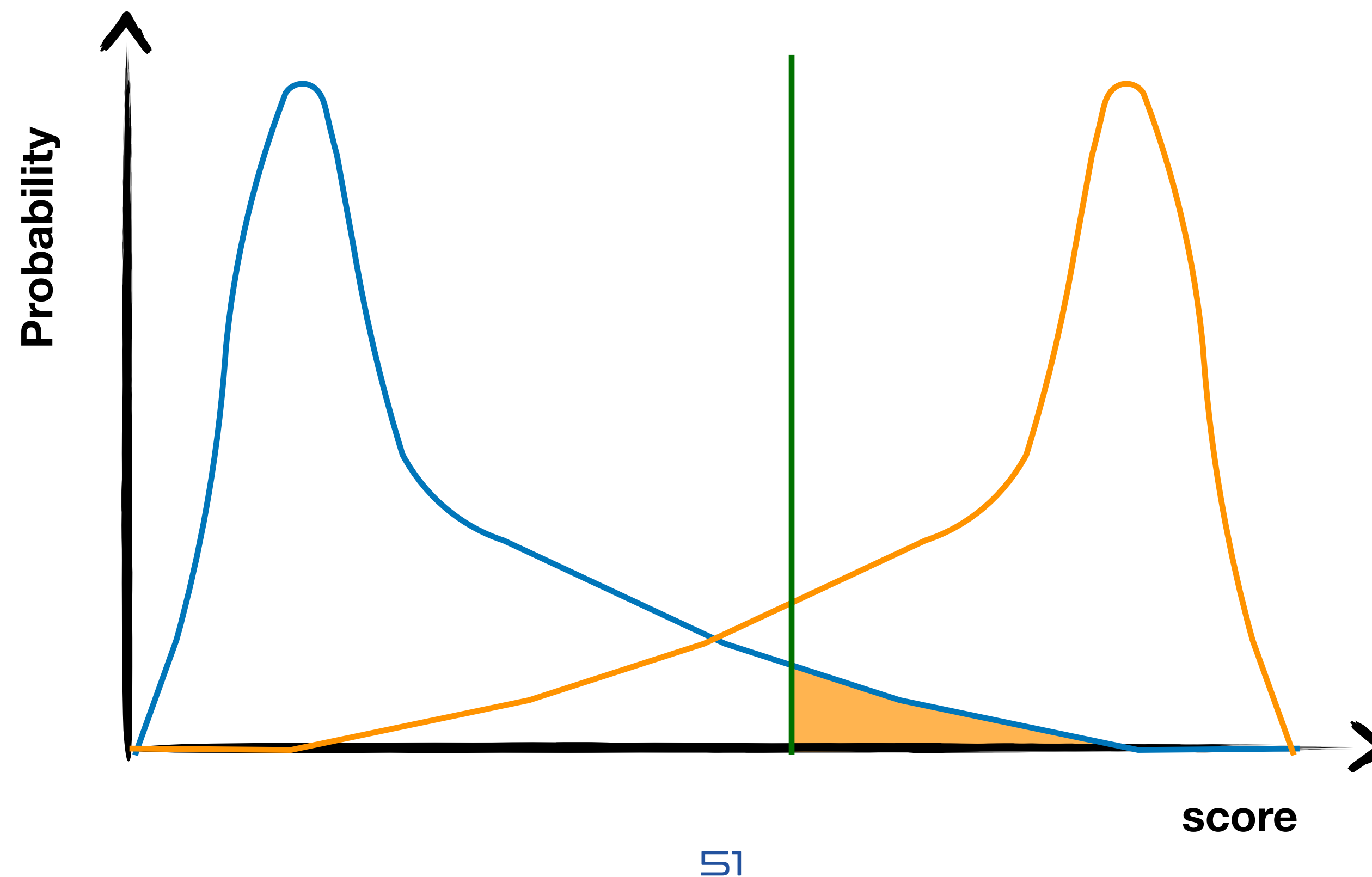
- ⦿ A given threefold defines the following qualities
 - ⦿ **True-positives: Class-1 events above the threshold**
 - ⦿ True-negatives: Class-0 events below the threshold
 - ⦿ False-positives: Class-0 events above the threshold
 - ⦿ False-negatives: Class-1 events below the threshold



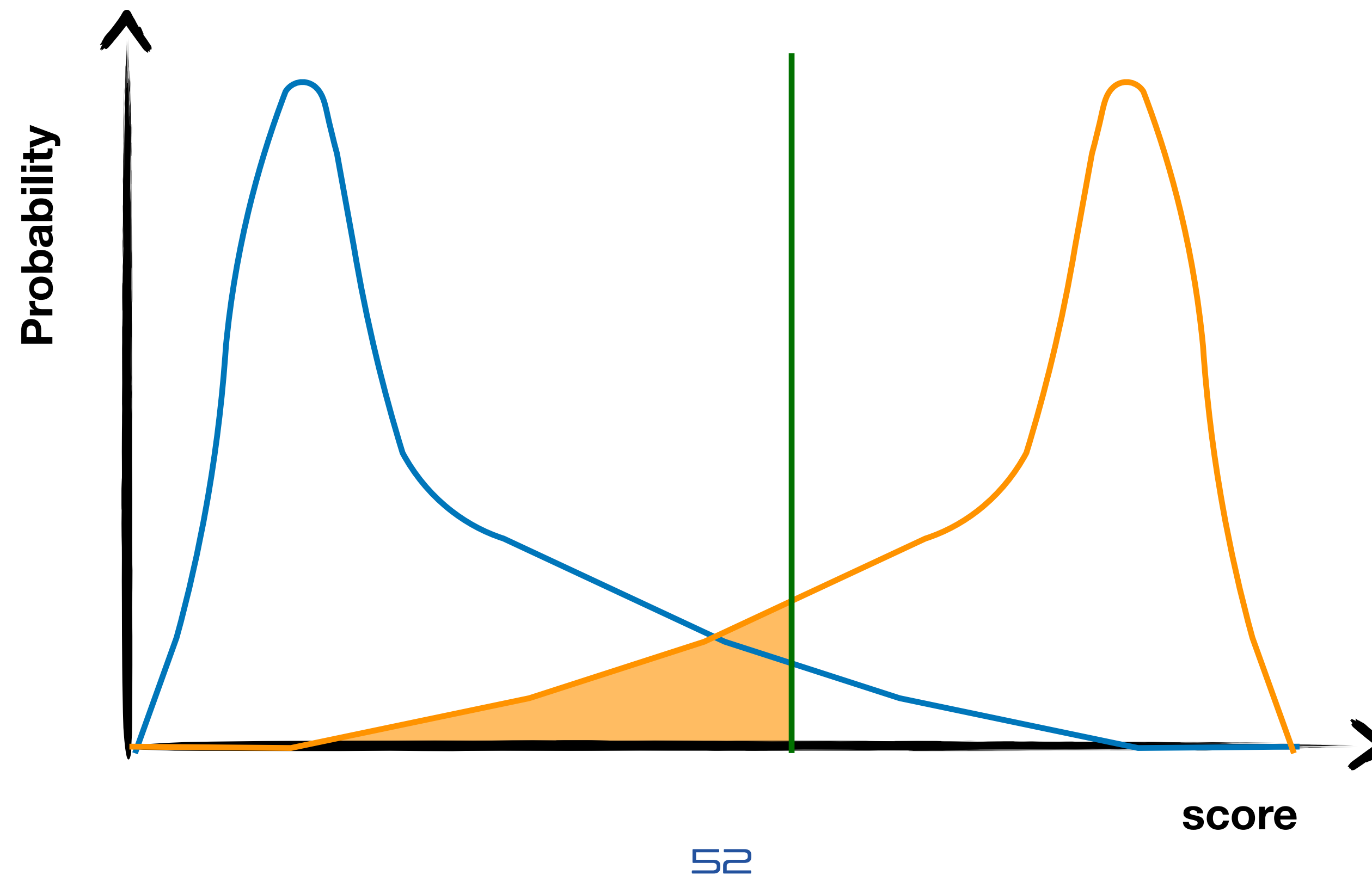
- A given threshold defines the following qualities
 - True-positives: Class-1 events above the threshold
 - True-negatives: Class-0 events below the threshold
 - False-positives: Class-0 events above the threshold
 - False-negatives: Class-1 events below the threshold



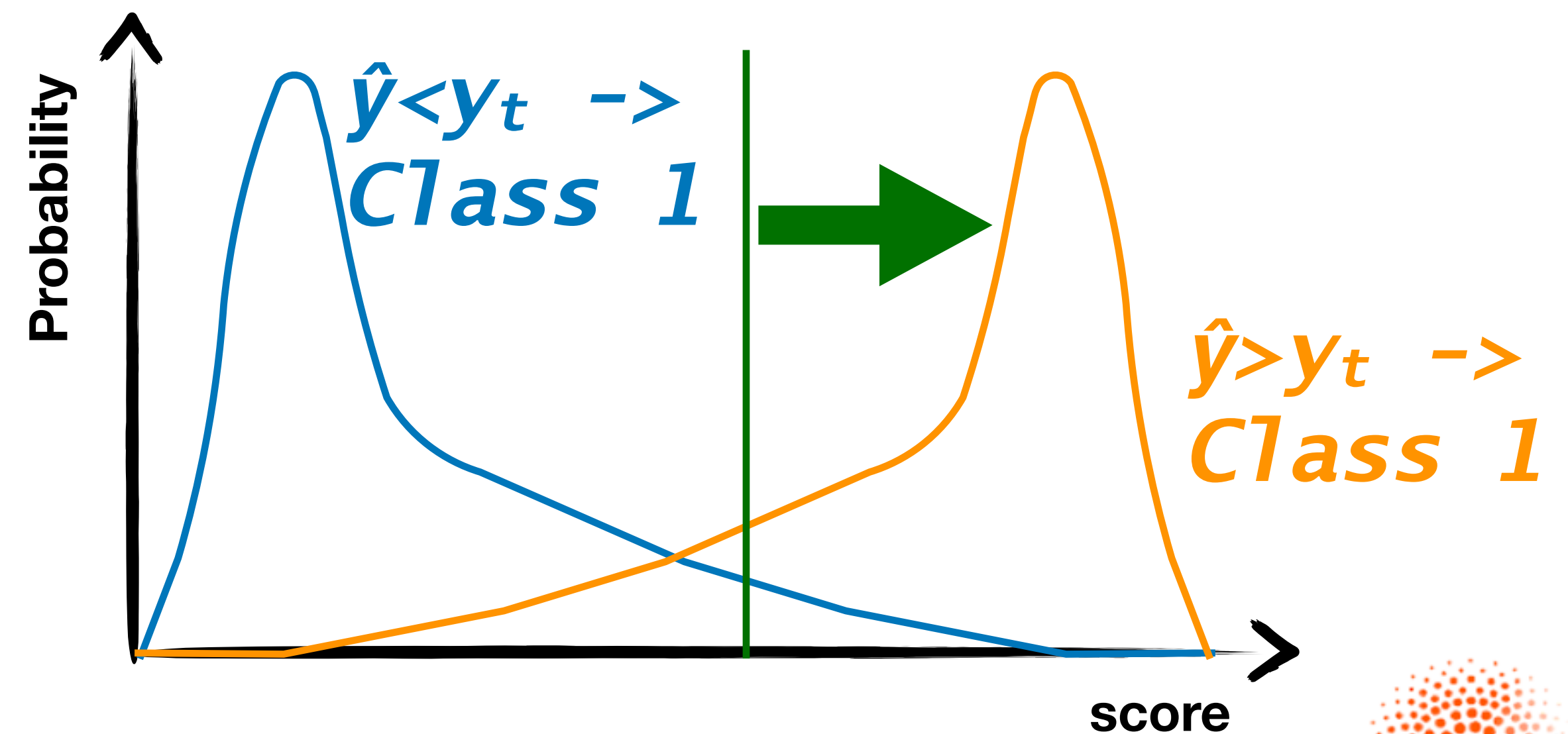
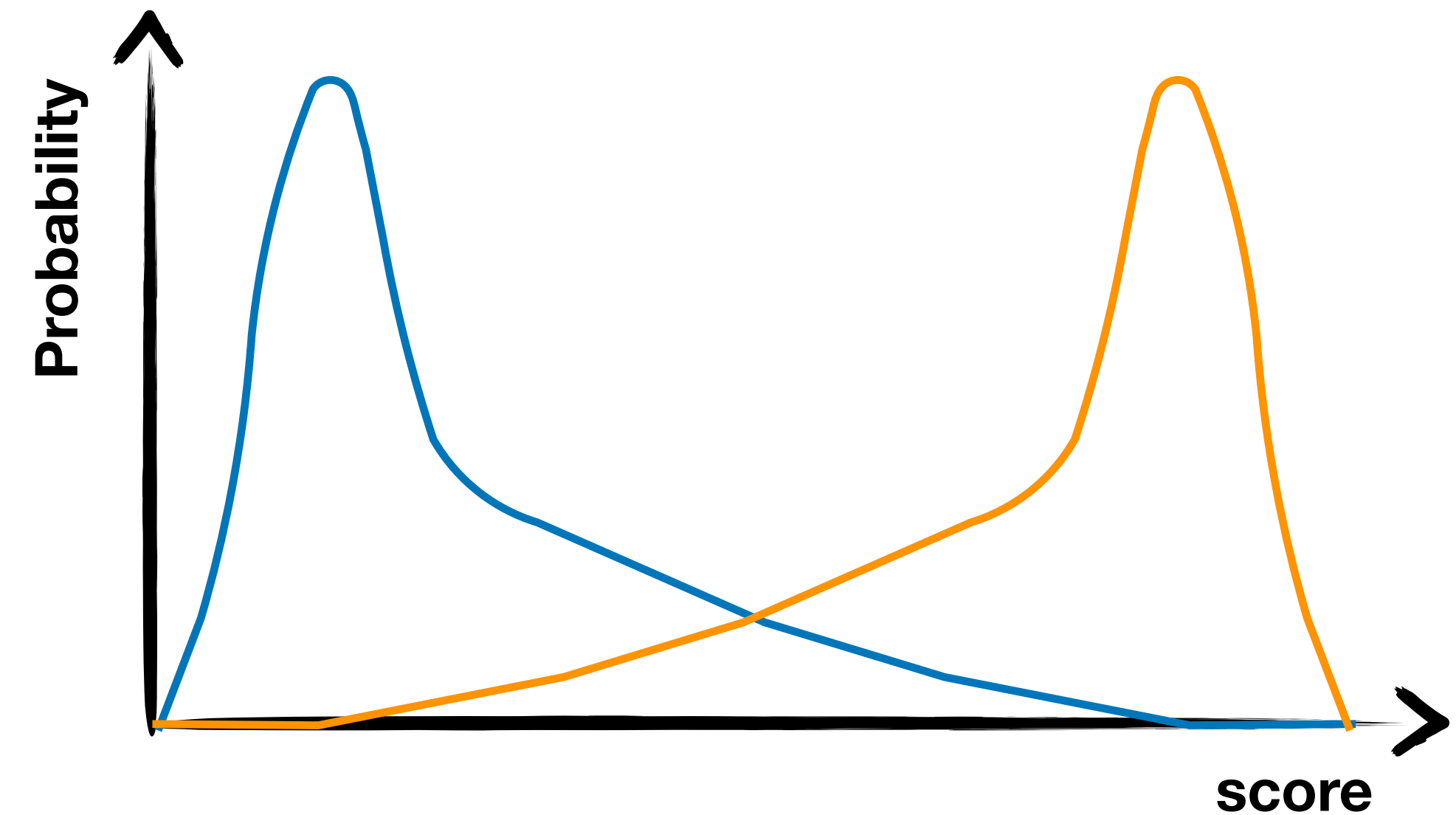
- A given threshold defines the following qualities
 - True-positives: Class-1 events above the threshold
 - True-negatives: Class-0 events below the threshold
 - False-positives: Class-0 events above the threshold
 - False-negatives: Class-1 events below the threshold



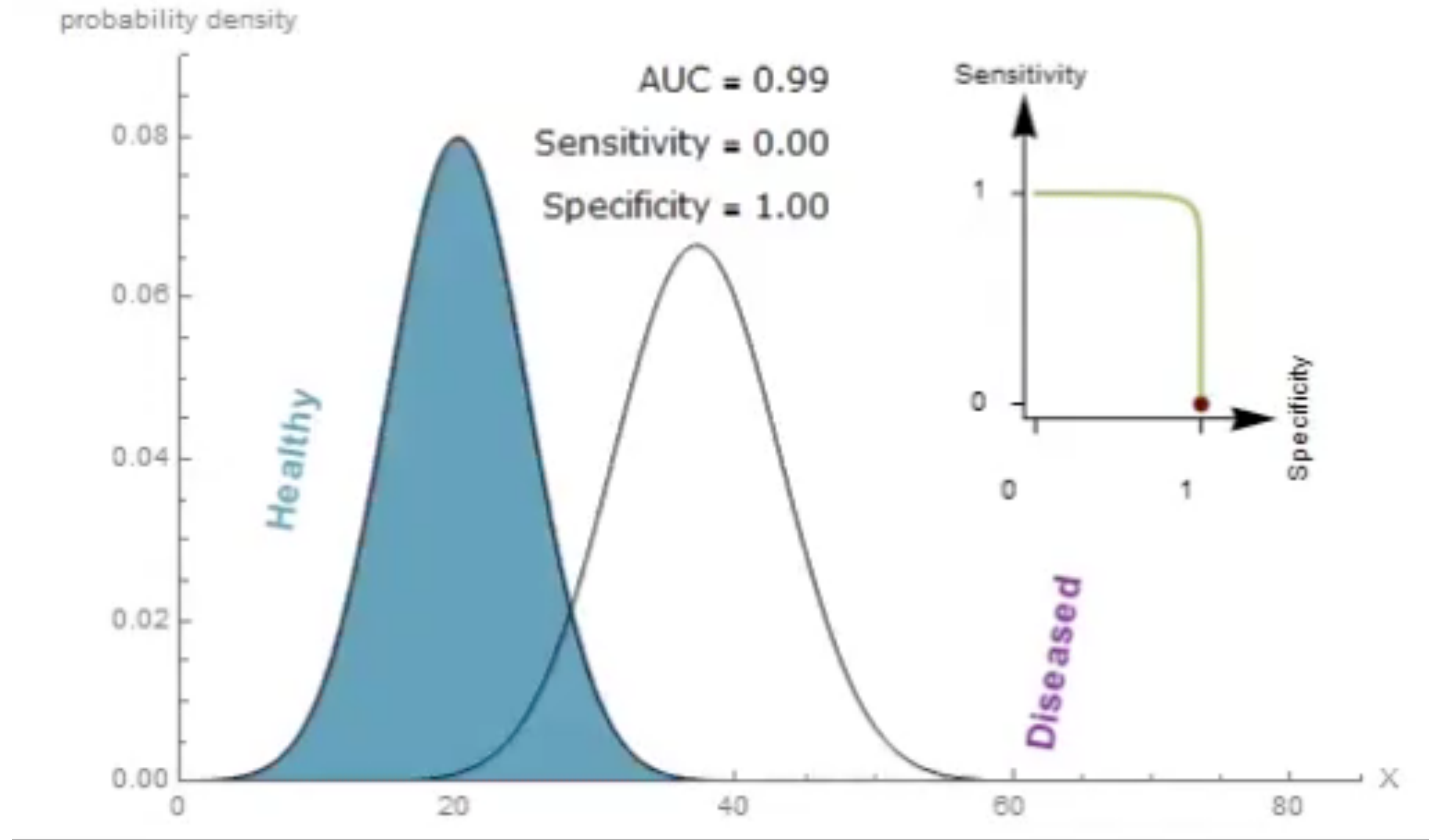
- A given threshold defines the following qualities
 - True-positives: Class-1 events above the threshold
 - True-negatives: Class-0 events below the threshold
 - False-positives: Class-0 events above the threshold
 - False-negatives: Class-1 events below the threshold

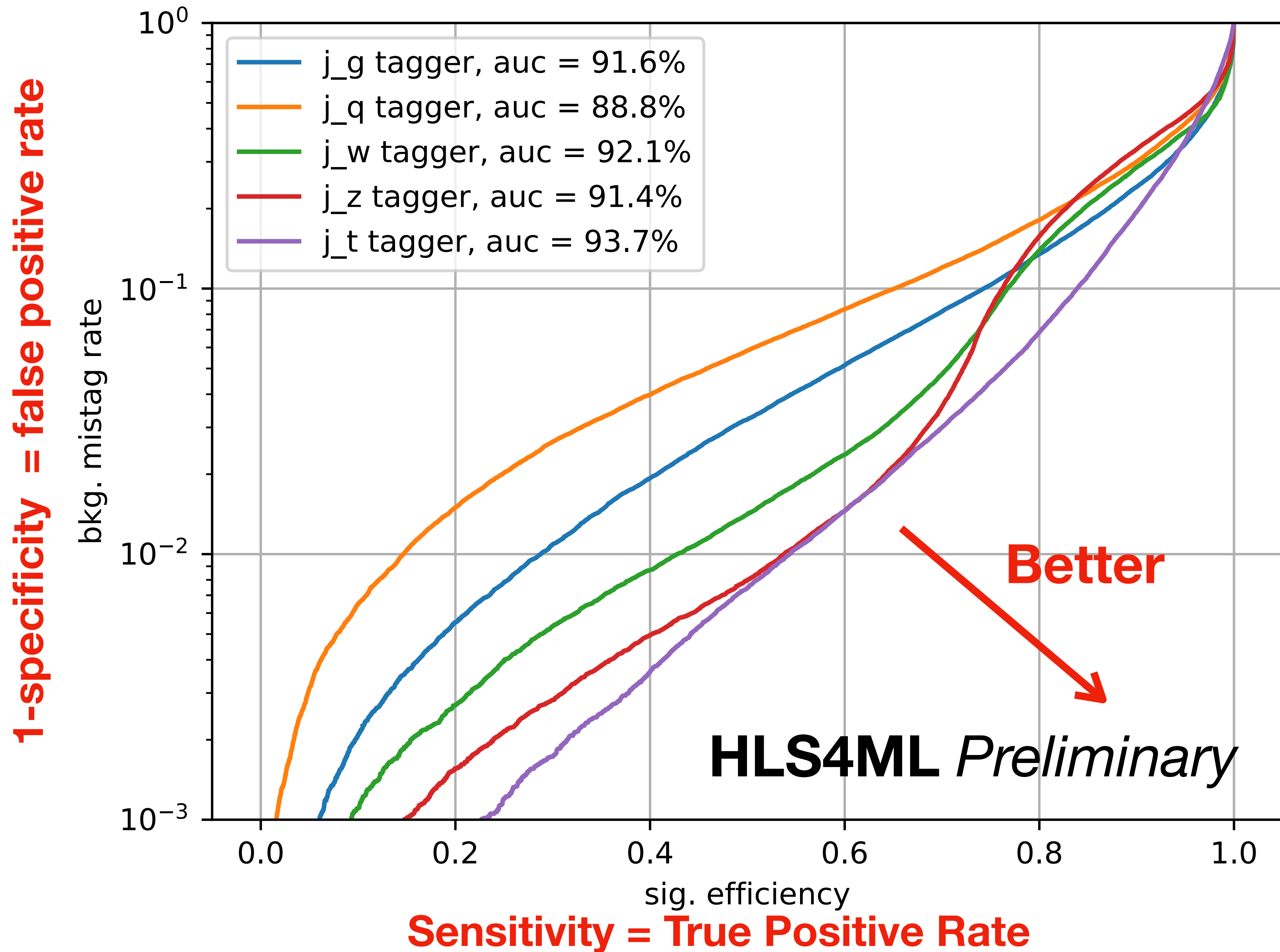


- Consider a binary classifier
- Its output \hat{y} is a number in $[0,1]$
- If well trained, value should be close to 0 (1) for class-0 (class-1) examples
- One usually defines a threshold y_t such that:
 - $\hat{y} > y_t \rightarrow$ Class 1
 - $\hat{y} < y_t \rightarrow$ Class 0



- *Starting ingredients are true positive (TP) and true negative (TN) rates*
- *Accuracy: $(TP+TN)/Total$*
 - *The fraction of events correctly classified*
- *Sensitivity: $TP/(Total\ positive)$*
 - *AKA signal efficiency in HEP*
- *Specificity: $TN/(Total\ negative)$*
 - *AKA mistag rate in HEP*





- *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*
- *It could be relevant to the future of HEP, e.g., to face the big-data challenge of the High-Luminosity LHC*

- ◎ *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/SMARTHEP>*
- ◎ *Full dataset available at: <https://zenodo.org/record/3602260>*

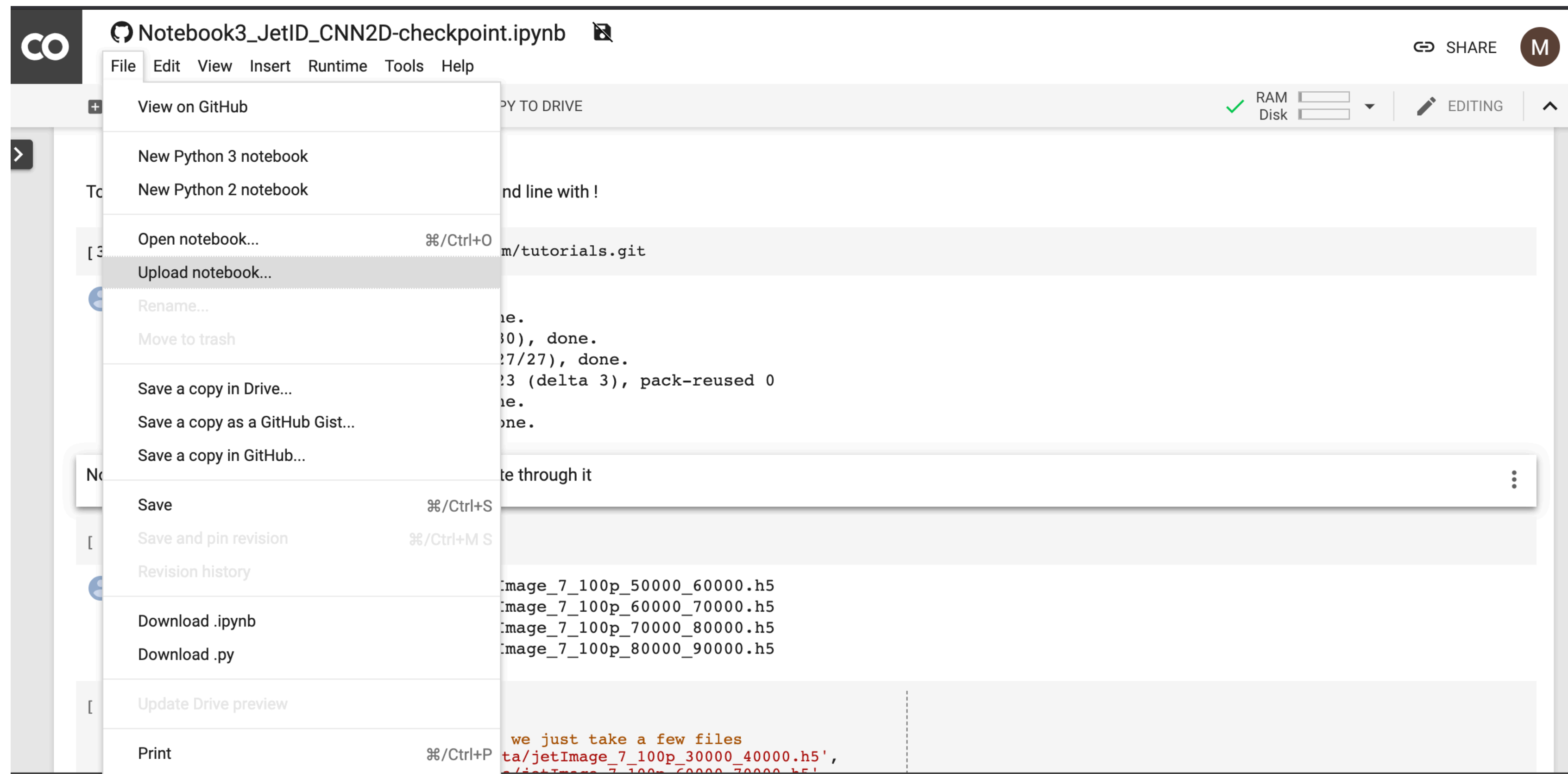
Running Tutorial Notebooks in Colab

Maurizio Pierini



Step1: Open Notebook on Colab

Go to <https://colab.research.google.com>



Step2: import the Tutorial from gitlab

- Click on the GITHUB tab
- Specify the repository pierinim/tutorials/SMARTHEP
- Click on the notebook

The screenshot shows the CO notebook interface with the GITHUB tab selected. The search bar contains the text "pierinim" and a search icon. Below the search bar, there are dropdown menus for "Repository:" (set to "pierinim/tutorials") and "Branch:" (set to "master"). A list of repositories is displayed, including "HiggsSchool/.ipynb_checkpoints/Notebook3_JetID_CNN2D-checkpoint...".

Set your resources to use GPUs

The screenshot shows a Jupyter Notebook titled "Notebook3_JetID_CNN2D-checkpoint.ipynb" in a Colab environment. The "Runtime" menu is open, displaying various options such as "Run all", "Run before", "Run the focused cell", "Run selection", "Run after", "Interrupt execution", "Restart runtime...", "Restart and run all...", "Reset all runtimes...", "Change runtime type", "Manage sessions", and "View runtime logs". The "Change runtime type" option is highlighted. The notebook content includes a git clone command, a list of H5 files, and a code cell for loading data.

```

[3] ! git clone https://github.com/HiggsSchool/data

Cloning into 'tutorials/HiggsSchool/data'...
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (30/30), done.
remote: Compressing objects: 100% (30/30), done.
remote: Total 30 (delta 0), reused 0, pack-reused 0
Unpacking objects: 100% (30/30), done.
Checking out files: 100% (30/30), done.

Now that the gitub repository is checked out, you can run the following command to list the files in the directory:

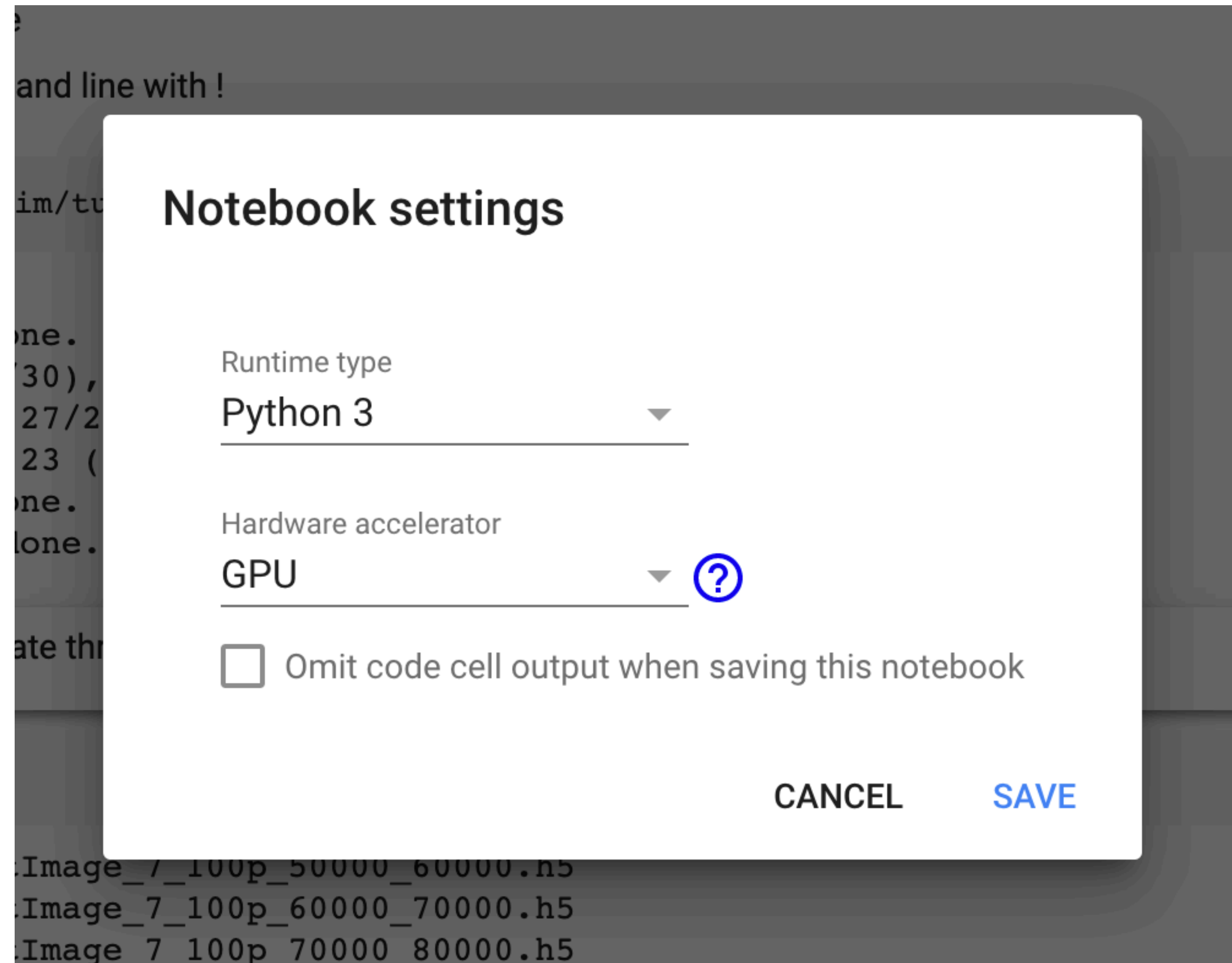
[ ] ! ls tutorials/HiggsSchool/data/

jetImage_7_100p_0_10000.h5      jetImage_7_100p_50000_60000.h5
jetImage_7_100p_10000_20000.h5  jetImage_7_100p_60000_70000.h5
jetImage_7_100p_30000_40000.h5  jetImage_7_100p_70000_80000.h5
jetImage_7_100p_40000_50000.h5  jetImage_7_100p_80000_90000.h5

[ ] target = np.array([])
jetImage = np.array([])
# we cannot load all data on Colab. So we just take a few files
datafiles = ['tutorials/HiggsSchool/data/jetImage_7_100p_30000_40000.h5',

```

Set your resources to use GPUs



The screenshot shows a Jupyter Notebook interface with a 'Notebook settings' dialog box open. The dialog has a title bar and contains the following settings:

- Runtime type:** A dropdown menu currently set to 'Python 3'.
- Hardware accelerator:** A dropdown menu currently set to 'GPU', with a blue question mark icon to its right.
- Omit code cell output when saving this notebook:** An unchecked checkbox.

At the bottom right of the dialog are two buttons: 'CANCEL' and 'SAVE'.

Background text from the notebook is visible, including 'and line with!', 'im/tu', 'ne.', '(30),', '27/2', '23 ('ne.', 'one.', 'ate th', '.Image_7_100p_50000_60000.h5', '.Image_7_100p_60000_70000.h5', and '.Image_7_100p_70000_80000.h5'.