What is ML?
(and why it is relevant for INFN)

Daniele Bonacorsi

First ML_INFN hackaton, 7-9 June 2021

ML in context

Textual definition(s) of ML

"The capacity of a computer to learn from experience, i.e. to modify its processing on the basis of newly acquired information"

– The Oxford dictionary of statistics terms (today)

"ML is the field of study that gives computers the ability to learn without being explicitly programmed"

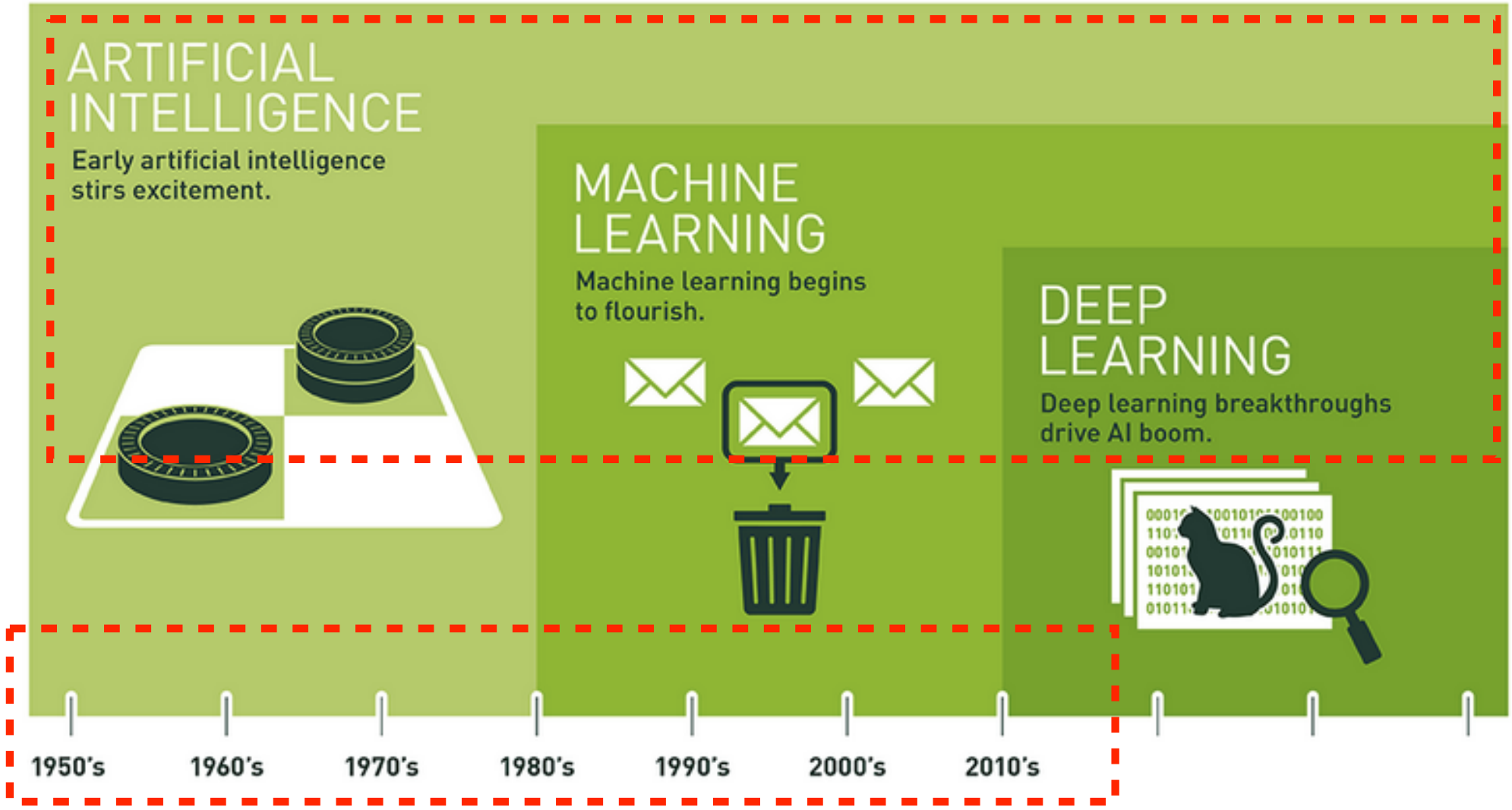
– Arthur Samuel (1959), author of the Samuel Checkers-playing Program (and some TeX..)

"A machine is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P** if its performance at tasks in T, as measured by P, improves with experience E."

– Tom Mitchell (1997)

Visual definition of ML

A pictorial definition (by Nvidia)



"AI", really?

"AI" terminology perhaps misleading in most practical discussions

Most of AI research today is actually not trying to recreate intelligence in any shape or form

Automation of task execution and (eventually) **decision making**

Aim at collecting data around how humans make decisions, to perform the same tasks at a scale (LARGE) and latency (SMALL) that are not humanly possible

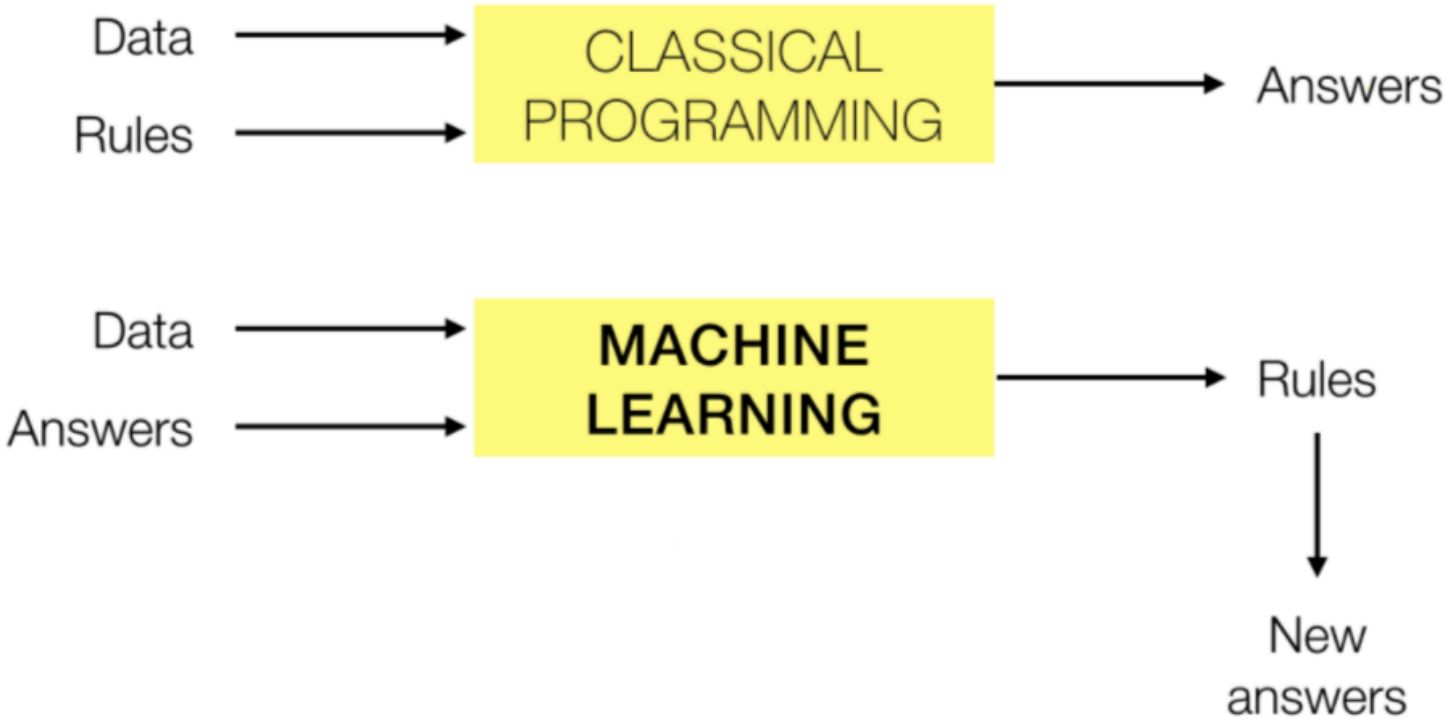
- *Example*: computer vision applications

This is the "artificial intelligence" we are talking about. And a way to implement it is via **Machine Learning** techniques.

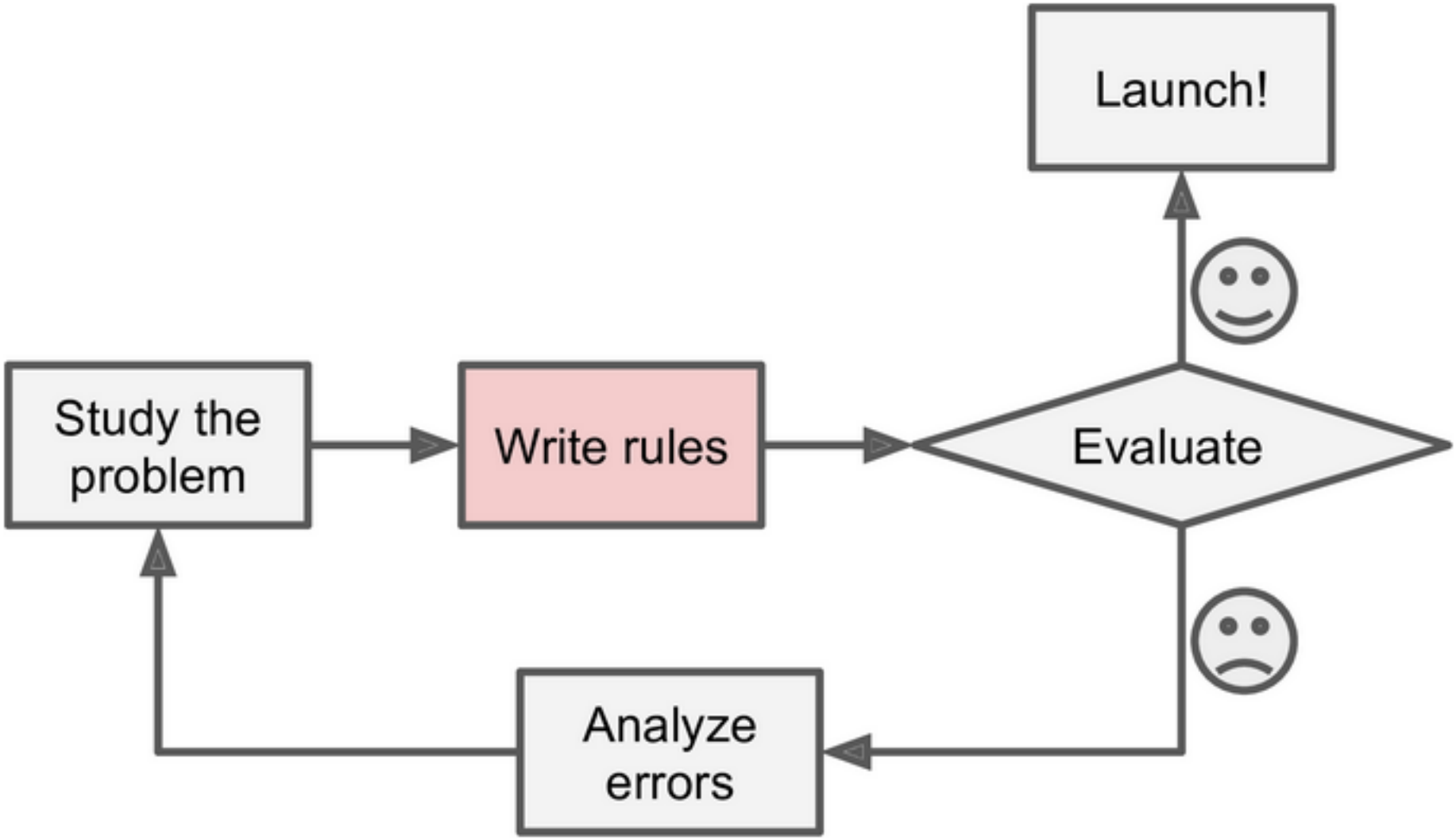
Another (last) definition

"Classical programming uses rules and data to produce answers.
Machine Learning uses data and answers to produce rules."

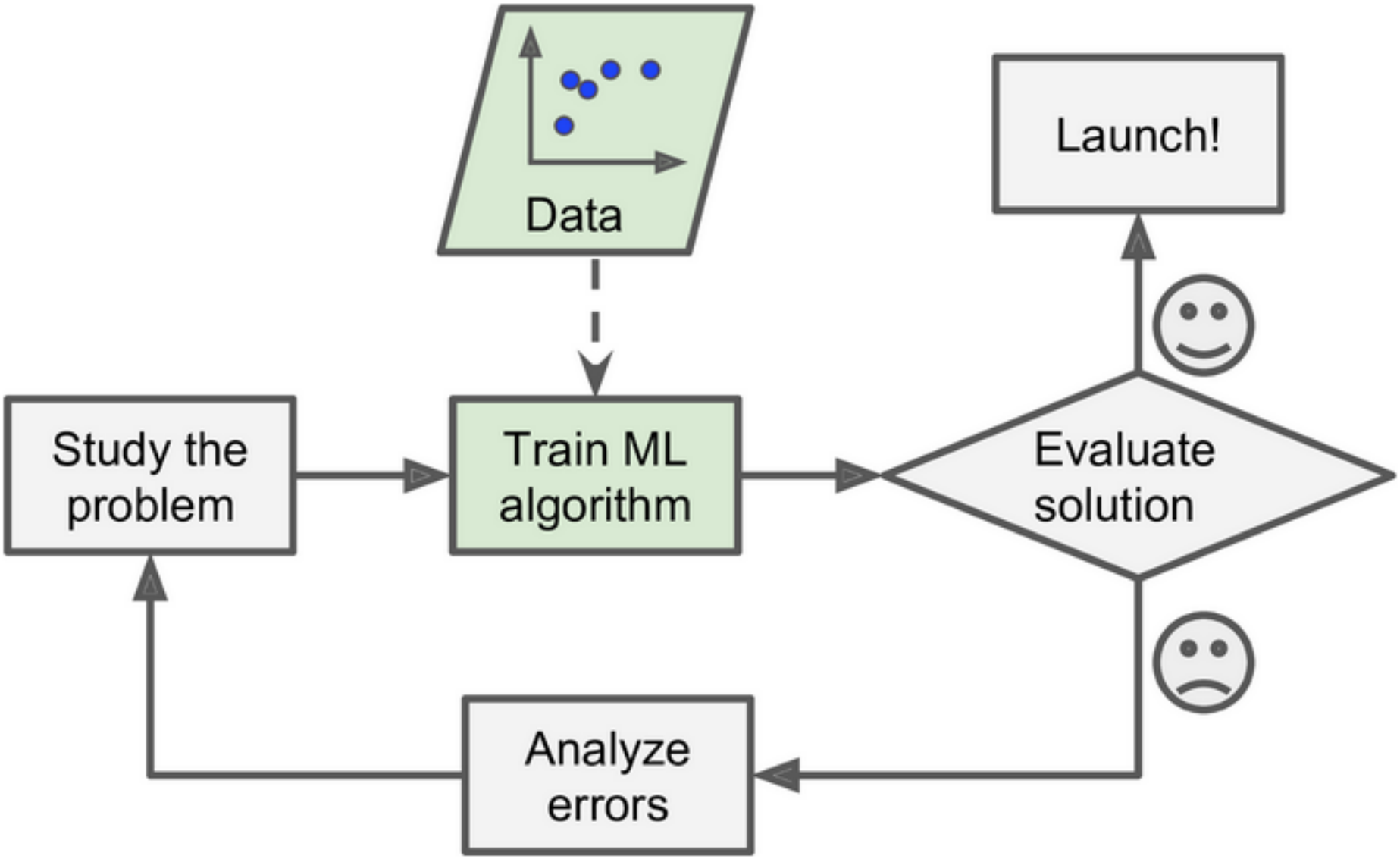
– F. Chollet (author of Keras)



Traditional approach

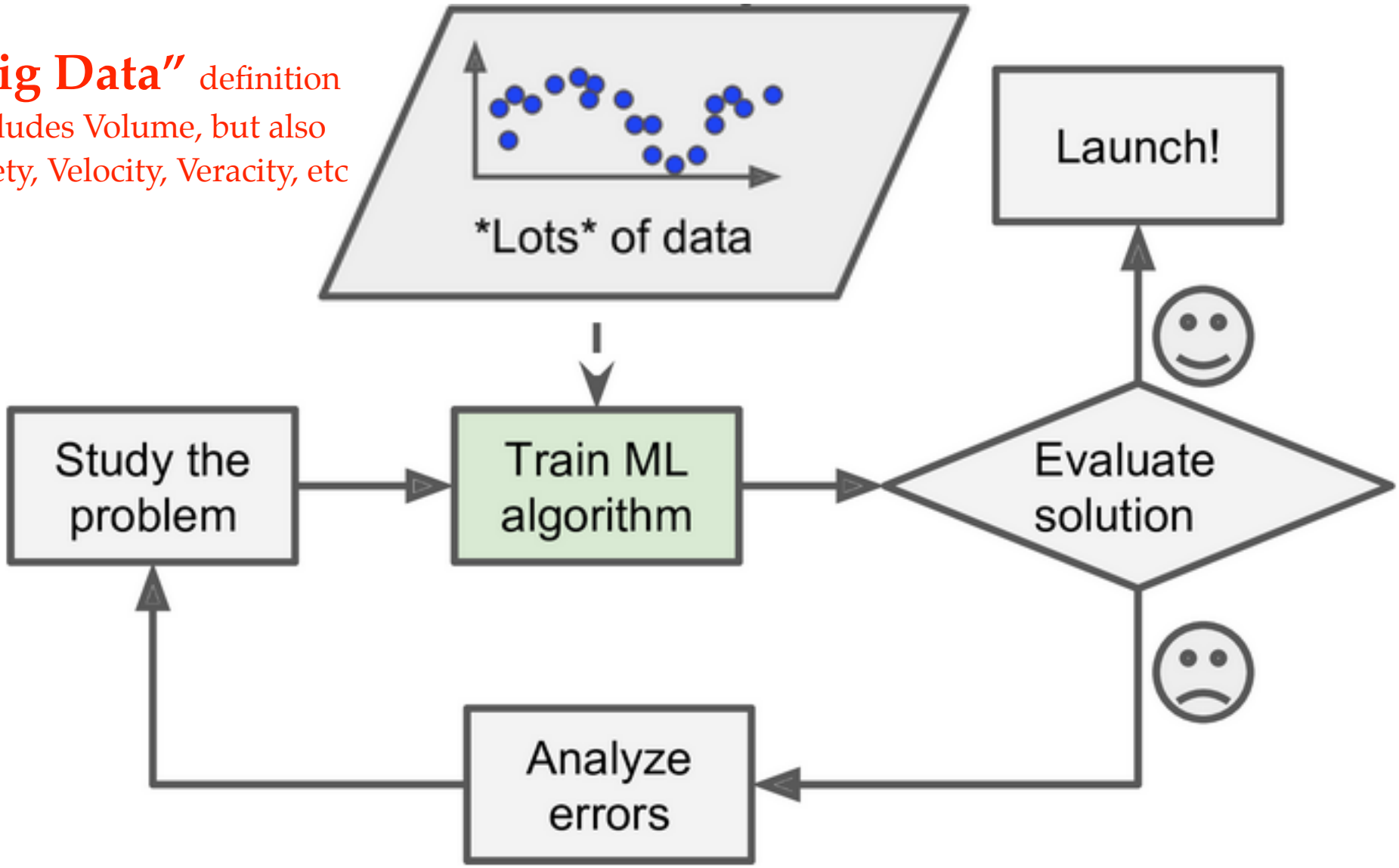


ML approach: data-driven modelling



ML approach: **Big Data**-driven modelling

"Big Data" definition
includes Volume, but also
Variety, Velocity, Veracity, etc



Types of ML

The zoo of ML algorithms

Impossible to go into the details of each.. focus on key concepts

Possible classifications of ML methods

This was a choice. A "flavour" of ML is determined on:

- **the amount and type of supervision** during model creation (aka "**training**"):
  - **Supervised**, **Unsupervised**, .. , **Reinforcement Learning**
- whether or not the machine can **learn incrementally on the fly**
  - **online** learning versus **batch** learning
- which are the **learning criteria** and **how they generalise**
  - **instance-based** versus **model-based** learning
- on the basis of the **algorithm** used
- .. more ..

All these criteria are not exclusive.

Supervised, Unsupervised, Reinforcement Learning

Batch learning versus **Online** learning

Instance-based versus **Model-based** learning

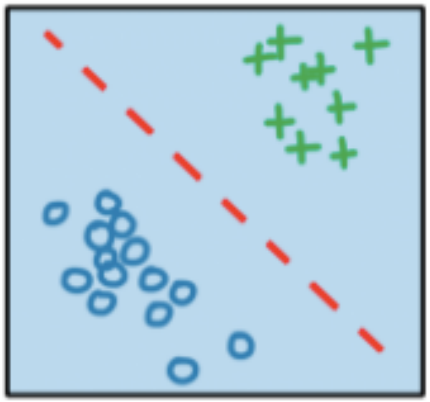
Types of ML

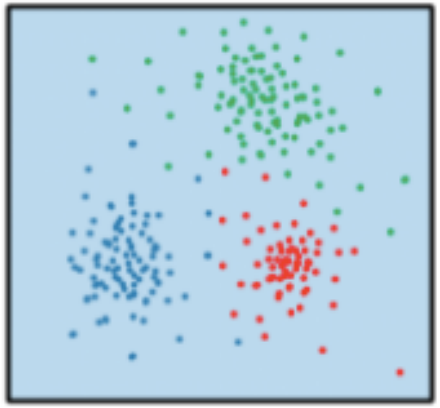
A traditional, lightweight classification

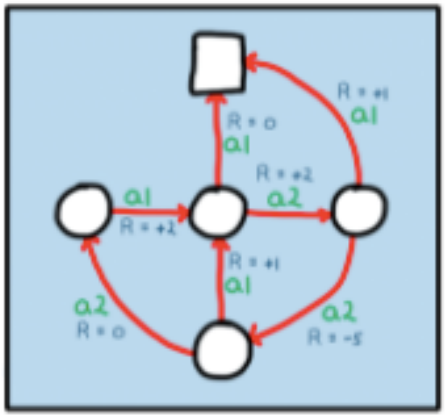
There are different types of ML, in terms of "learning algorithms".

The 3 most populated categories are:

- **supervised** learning: teach the machine how to learn something from data
- **unsupervised** learning: let the machine learn by itself how to learn from data
- **reinforcement** learning: make the machine learn by feedback

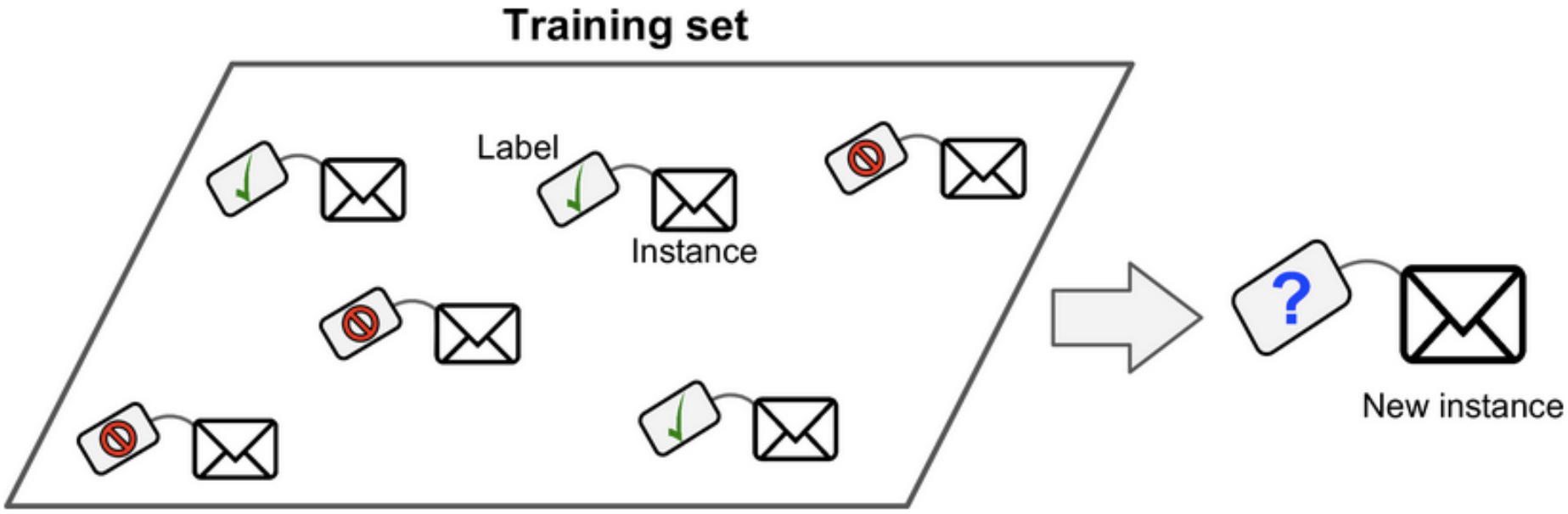






In **supervised learning**, the data entries ("**instances**" or "**examples**") you feed to the algorithm for it to learn (through its "**attributes**" - instantiated in"**features**" - in a process called "**training**") includes the truth info, i.e. the so-called "**labels**"

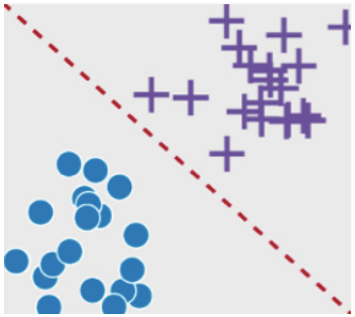
- e.g. for a spam detection problem:



Classification vs Regression

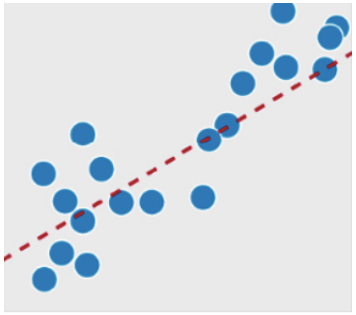
A typical supervised learning task is **classification**

- predict "classes": binary (0/1, yes/no) or multi-class (A/B/C/D)
  - e.g. predict "spam" vs "not-spam" in a spam filter



Another typical supervised learning task is **regression**

- predict "target numeric values" (in a continuum of values)
  - e.g. predict the price of a house knowing a set of attributes



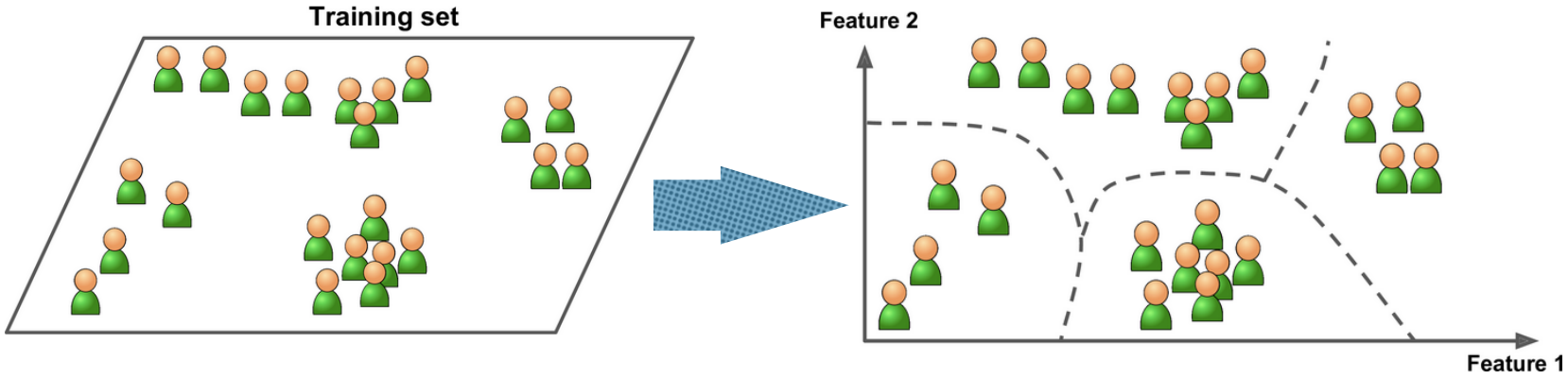
Examples of algorithms

Most commonly used / important:

- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- k-Nearest Neighbours
- Decision Trees and Random Forests
- Neural Networks
  - NOTE: some neural network architectures can be unsupervised (or self-supervised), e.g. Autoencoders, or Restricted Boltzmann Machines. Other can be semi-supervised, such as in Deep Belief Networks..

Example of "unsupervised ML": clustering

In **unsupervised learning** the training data is **unlabeled**, so the system tries to learn without a teacher guiding it



Example: data about blog readers

- run an unsupervised (e.g. hierarchical clustering) algo to detect "groups of similar visitors"
- at no point you tell the algo which group a visitor belongs to, but it finds it out: e.g., it might detect that 30% are females who comment on your posts on topic X, and usually read the blog in the evening, etc.

Examples of applications

Example: used to **organise large computer clusters**, trying to figure out which machines tend to "work together": if set-up takes this into account, the data centre works more efficiently.

Example: **social network clustering**. Given knowledge about which friends you message the most, or given your <pick-your-social> connections, try to automatically identify which are cohesive groups of people who know each other (or may want to connect).

Example: **Market segmentation**. Analyse huge DBs of customers' info and try to automatically group customers into different market segments, to target advertisement, offers, etc.

Example: e.g. **Physics/Astronomy data analysis**. Clustering algos might give insight into possible logical grouping of previously disconnected data.

Examples of algorithms

Clustering → try to detect groups

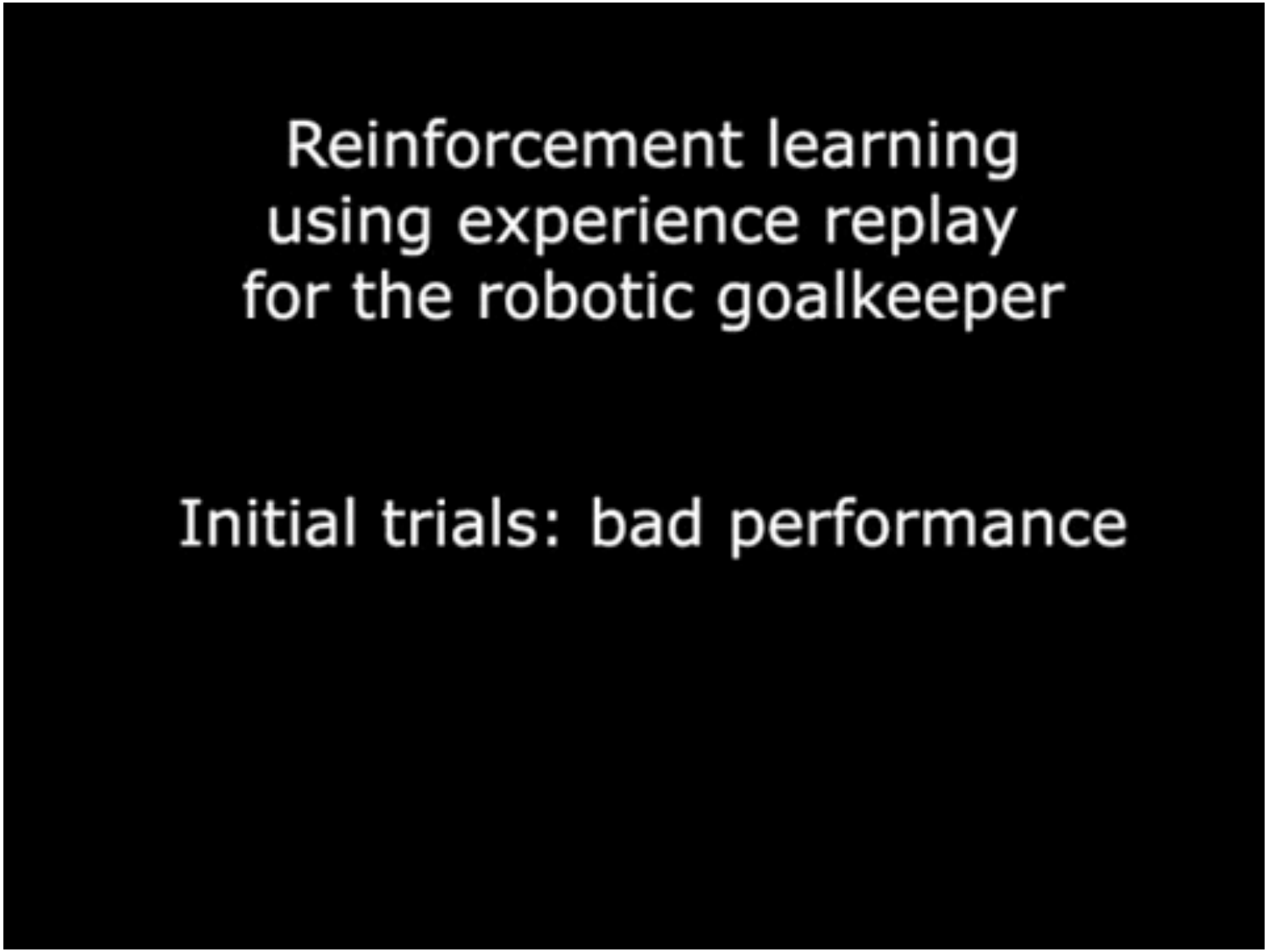
- K-Means, DBSCAN, Hierarchical Cluster Analysis (HCA), Anomaly detection and novelty detection, One-class SVM, Isolation Forest, ..

Dimensionality Reduction (and **Visualisation**) → display/simplify data w/o losing much info

- Principal Component Analysis (PCA), Kernel PCA, Locally-Linear Embedding (LLE), t-distributed Stochastic Neighbour Embedding (t-SNE), ..

And more..

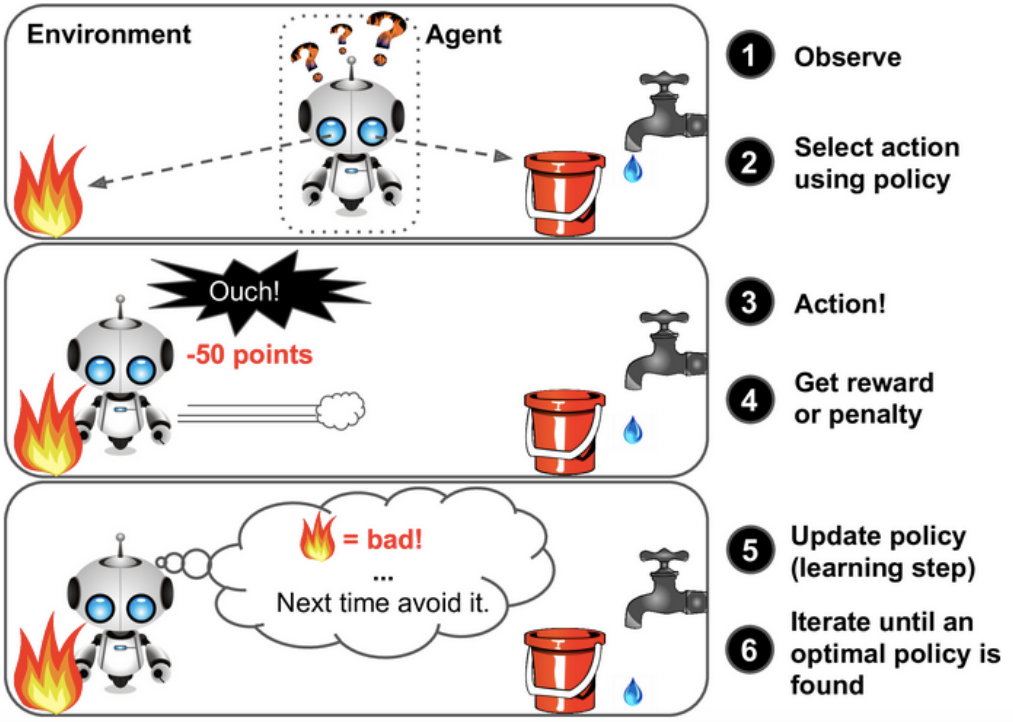
Example of Reinforcement Learning



A different approach

Reinforcement Learning is a completely different beast

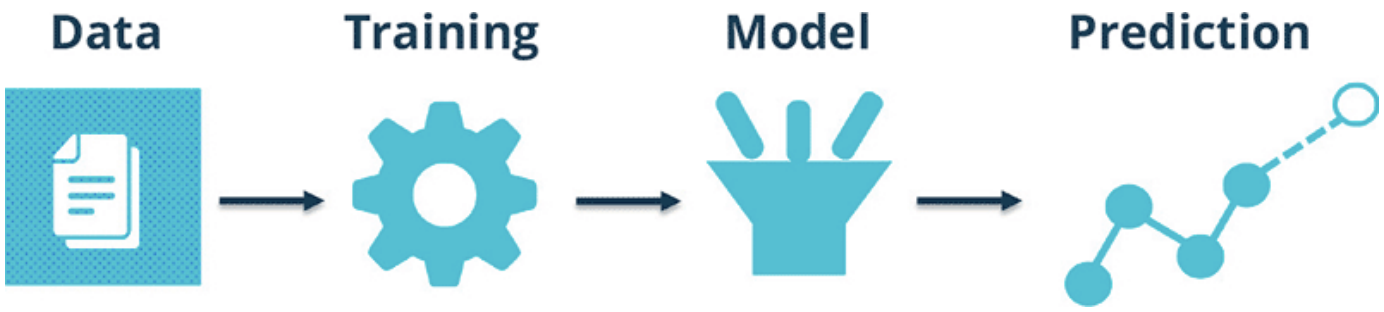
- The learning system is called "**agent**" in this context
- it **observes** the environment
- select and perform **actions**
- get positive **rewards** or negative rewards (i.e. **penalties**) in return
- learning step is define the best **policy** to get the most reward over time



Examples:

- robots implement RL to teach themselves learn how to walk
- DeepMind's AlphaGo program beat Ke Jie at the game of Go (May 2017)

A common denominator: the **training** process



Training = a procedure to calibrate with data the model parameters

- similar to biological systems: the model (e.g. brain structure) is DNA-encoded, and the parameters (e.g. synaptic weights) are tuned through experiences

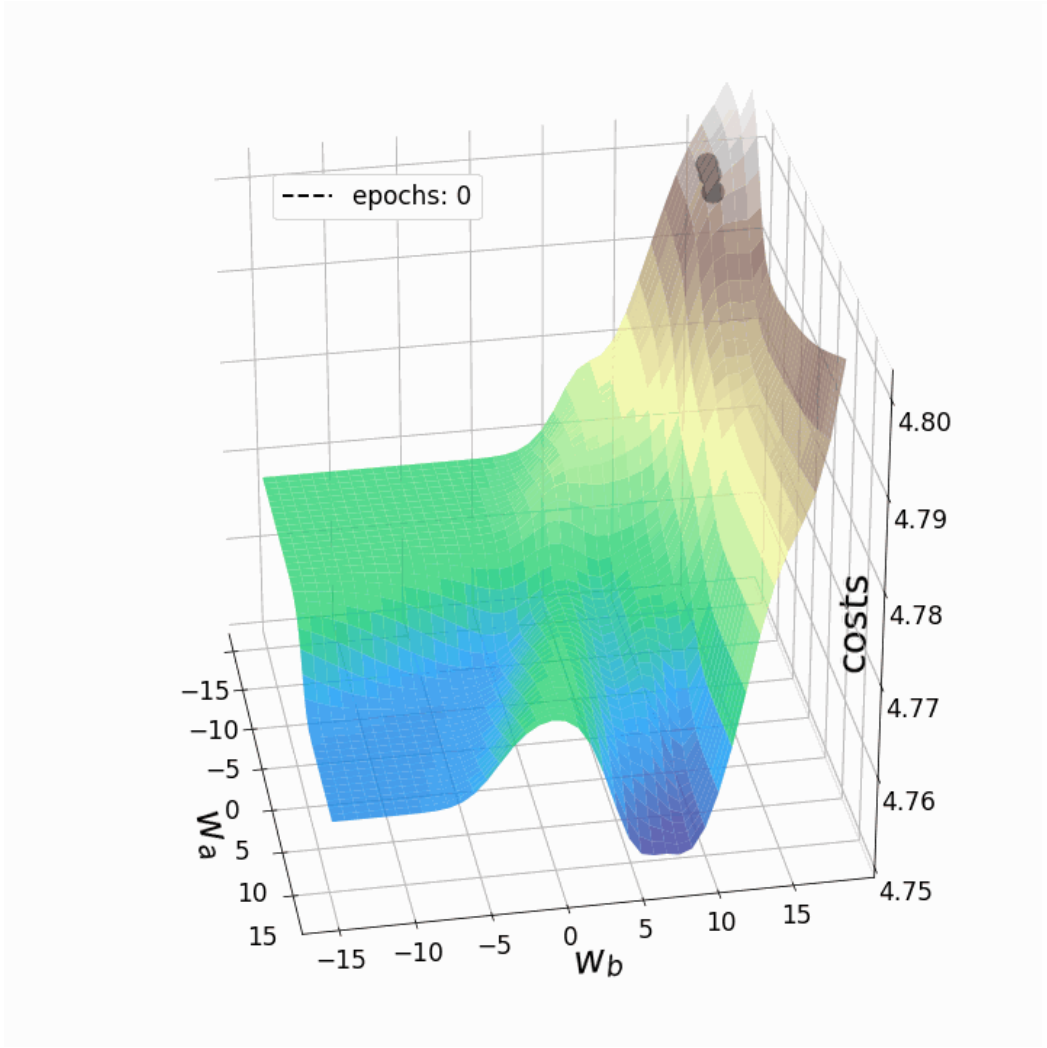
Most common techniques is the **gradient descent** optimisation, i.e. estimation of optimal model **parameters** by an iterative method that proceed in steps to **minimise a cost function**, towards convergence

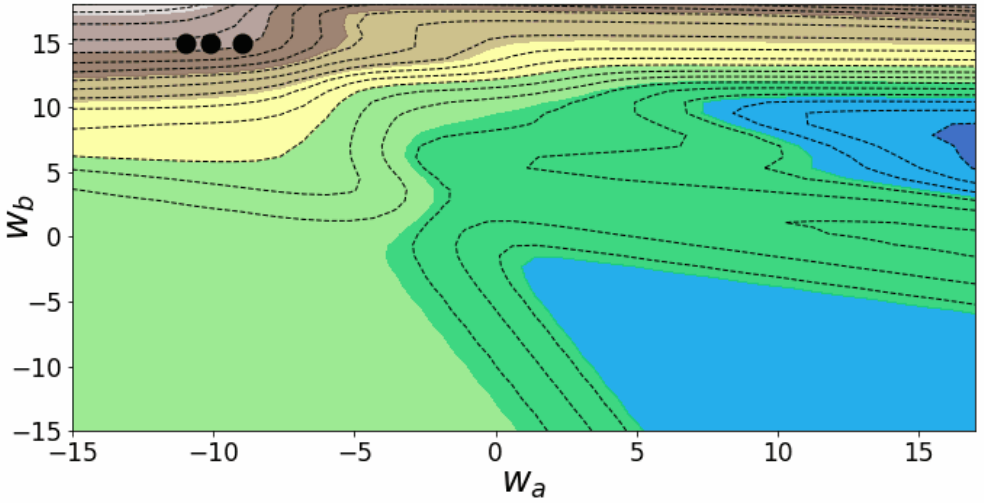
- at each step, reassess the gradient (sensitivity) of the prediction errors (pred vs actual) to changes in model parameters (weights), update such weights, and help converge towards an optimum

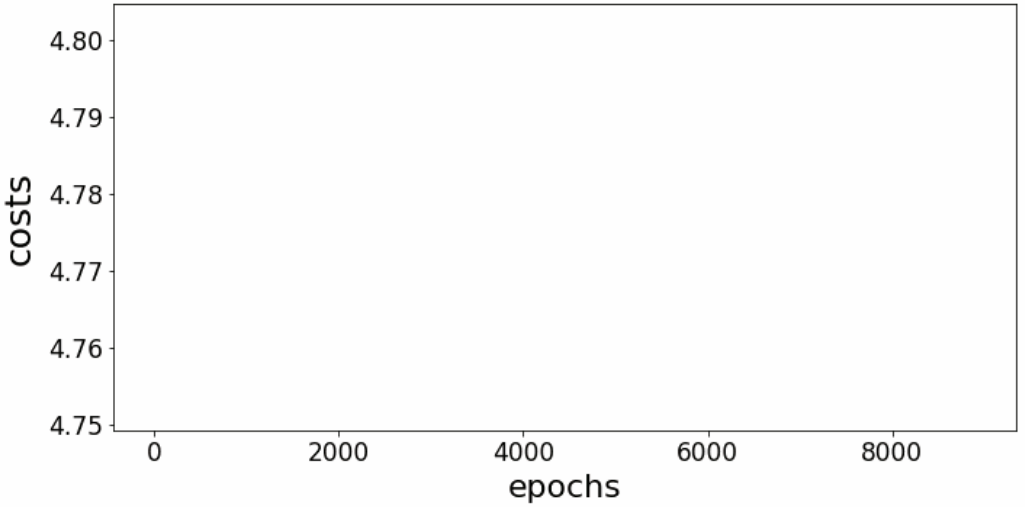
The training process is governed by **hyper-parameters**

- e.g. the size of the steps in gradient descent is determined by a hyper-parameter known as "*learning rate*" (there might be plenty more!)

The search for a minimum in the cost function..

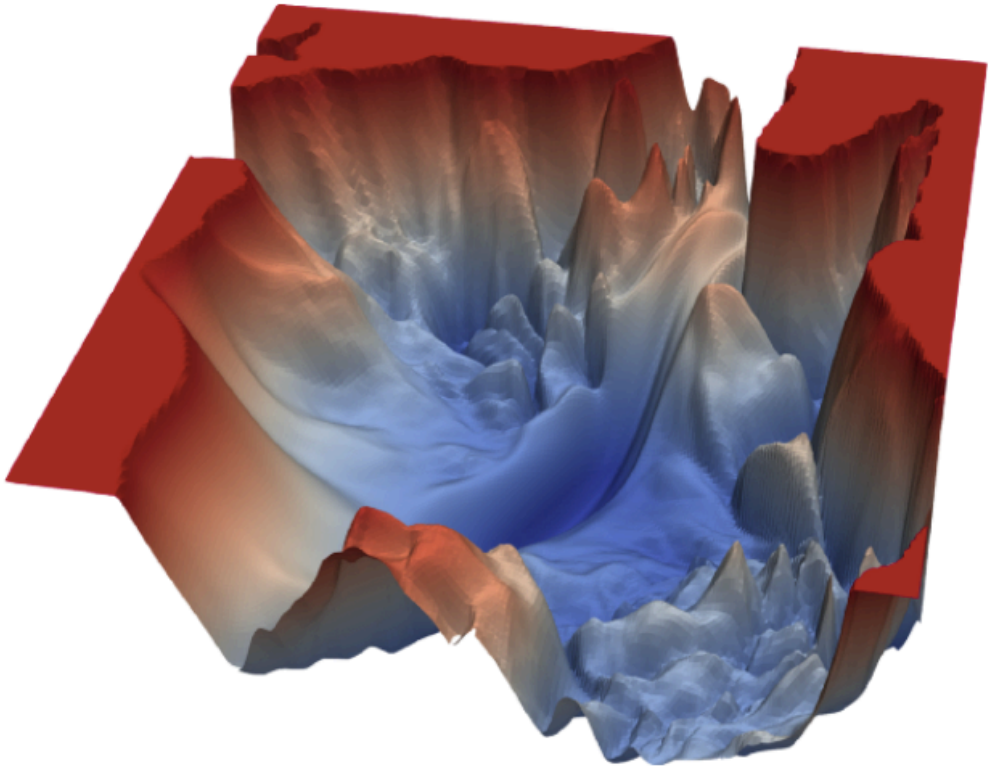




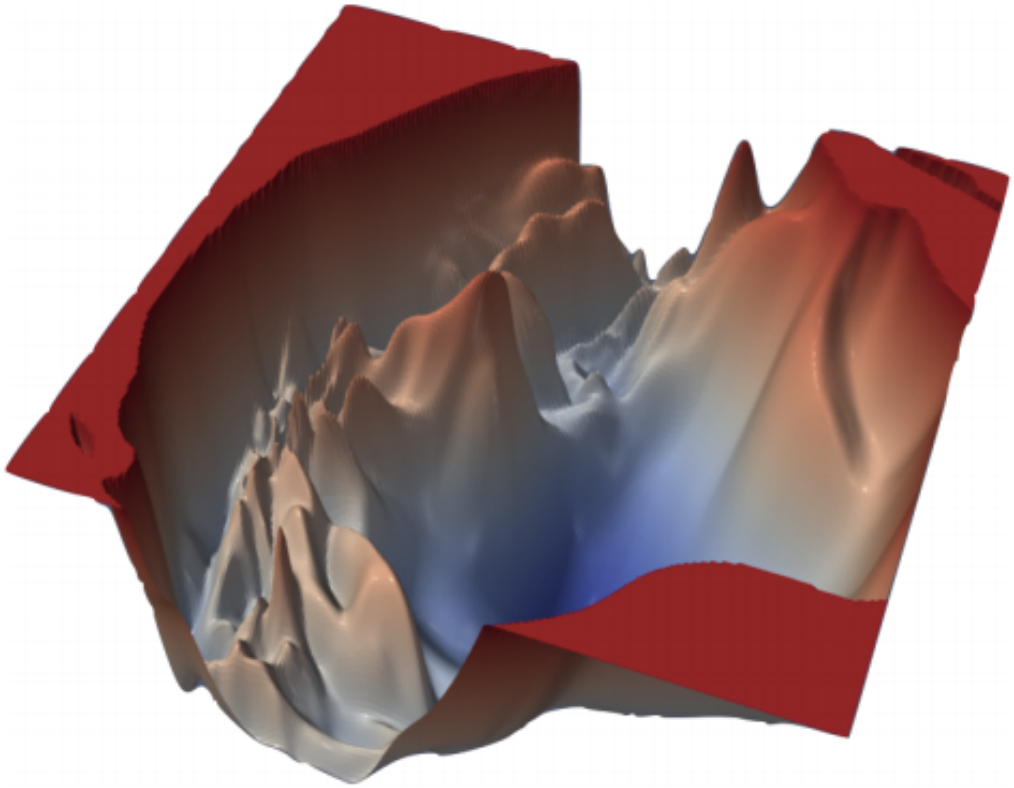


.. is challenging with *real* loss landscapes!

Loss landscape of a **VGG-56** (a CNN with 56 layers)



Loss landscape of a **ResNet-110** (no skip connections) for CIFAR-10



CAVEAT: These are 3D projections of very high-dimensional functions (still, helpful to grow your intuition..) → **see Stefano's talk for more details**

Supervised, Unsupervised, Reinforcement Learning

Batch learning versus **Online** learning

Instance-based versus **Model-based** learning

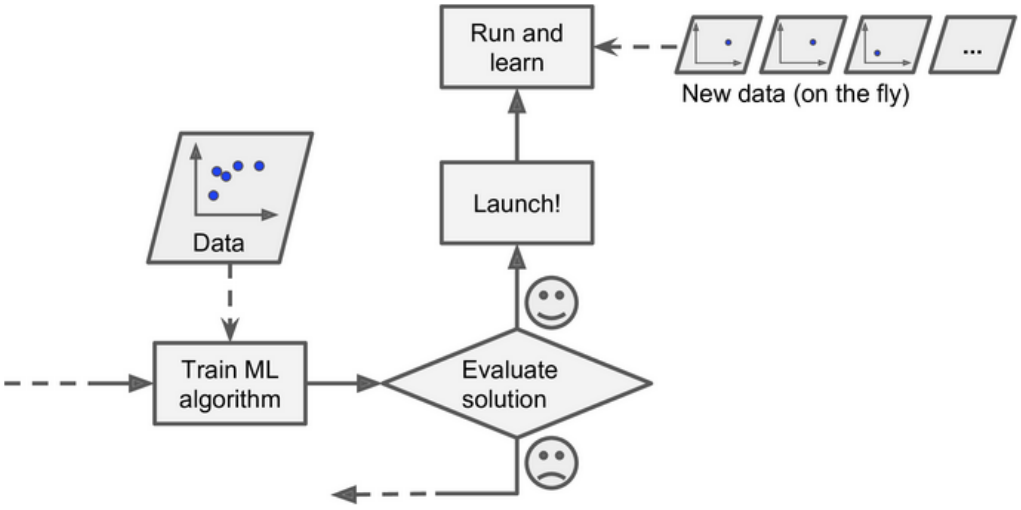
Batch vs **Online** learning

Batch (offline) learning → more learning requires a new training

- training is offline, and uses all the available data
- it can be resource hungry
  - CPU, memory space, disk space, disk I/O, network I/O, etc.

Online learning → the system can be trained incrementally

- training is online, data instances are fed sequentially
- data as a continuous flow, and fits on limited computing resources



Supervised, Unsupervised, Reinforcement Learning

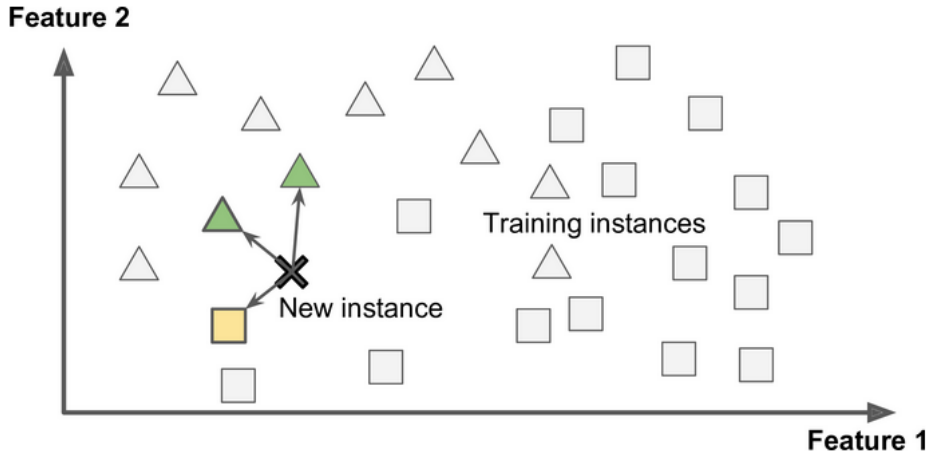
Batch learning versus **Online** learning

Instance-based versus **Model-based** learning

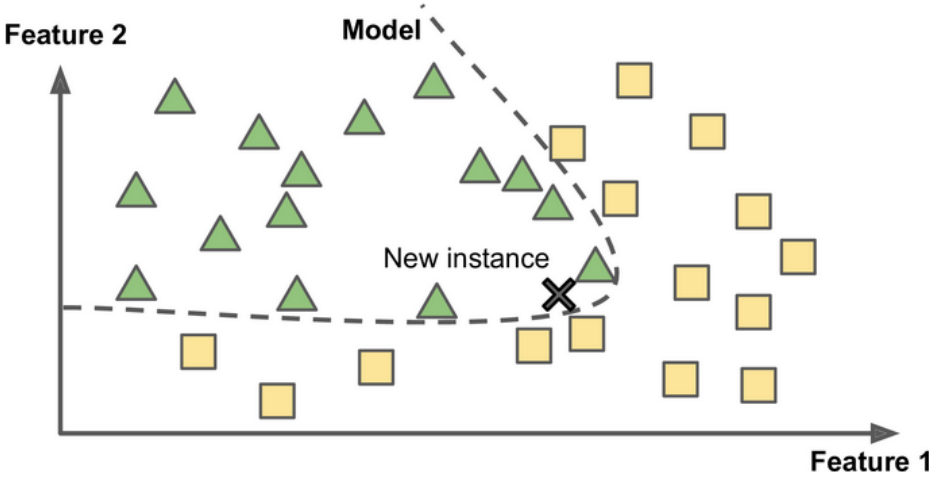
Instance-based vs **Model-based** learning

"**Generalisation**" is key to success of a ML solution

- data → training → measure ability to make predictions on unseen data



Instance-based learning: the system learns, then generalises to new cases by comparing them to the learned examples (or a subset of them), using a similarity measure



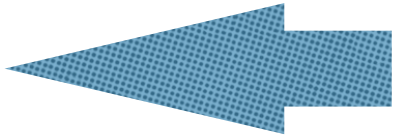
Model-based learning: a "model" from the training examples is built, i.e. based on your data you make an **hypothesis** as of how each data **feature** contributes to its **label**, and apply such model to make predictions

Challenges in ML

Main **challenges** in ML

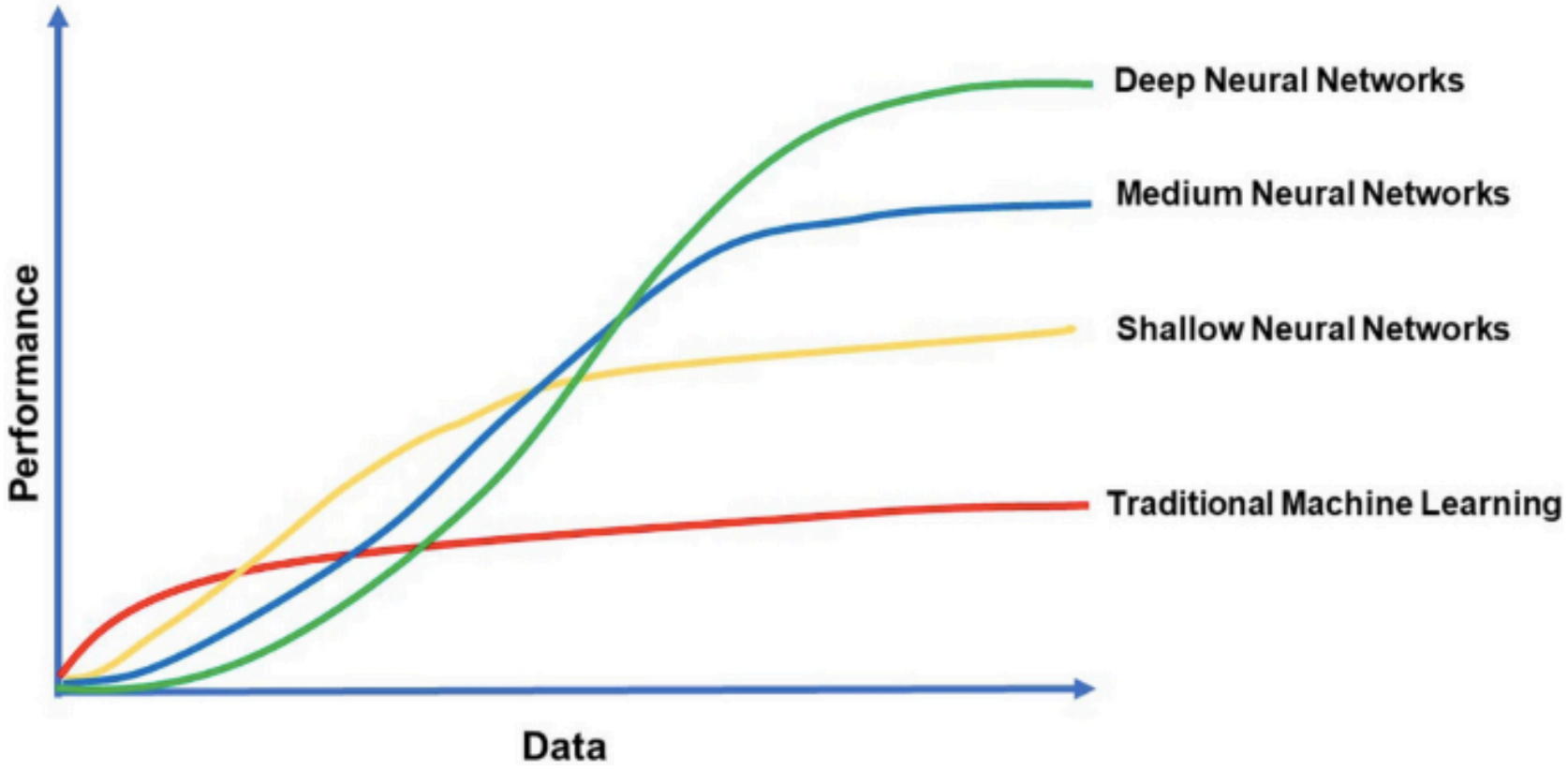
Basically, related to:

- **bad data**
- bad algos
- training technicalities



Main challenges in ML: **Insufficient Quantity of Training Data**

Bad data



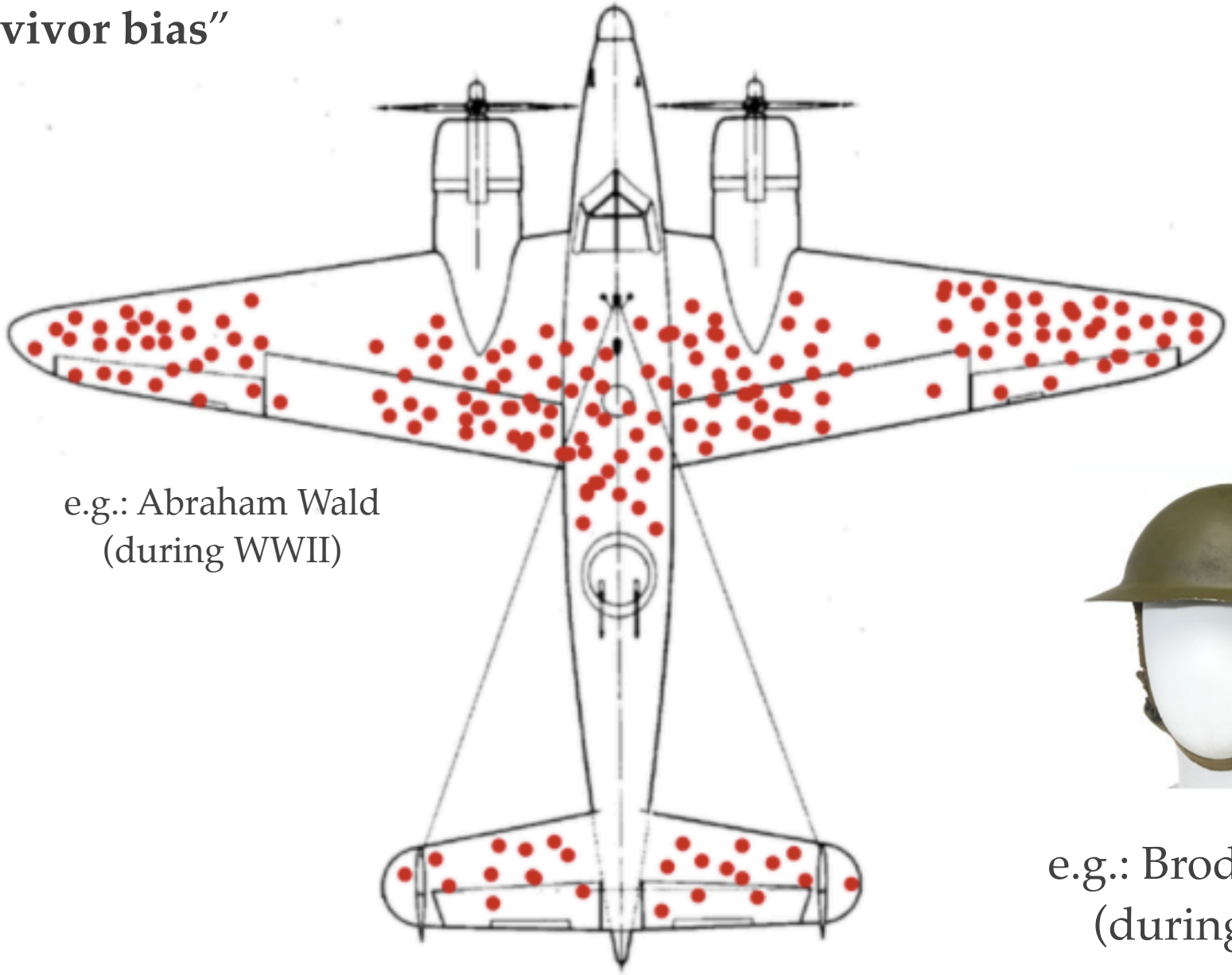
Example: *a baby learning to distinguish car vs bus vs motorbikes*

Large volumes of data (**the Volume "V" of Big Data**..) are needed.

Main challenges in ML: **Non-representative Training Data**

Bad data

E.g. "**Survivor bias**"



e.g.: Abraham Wald
(during WWII)

e.g.: Brodie helmet
(during WWI)

Main challenges in ML: **Poor-Quality Data**

Bad data

ML systems are "**garbage in garbage out**"

- training data might be full of errors, outliers, noise (e.g. poor-quality sensor measurements)
  - → hard to detect underlying patterns

Importance of **data preparation** (including data cleaning)

[credits: xkcd.com]

Main challenges in ML: **Irrelevant Features**

Bad data

Success with traditional ML methods also depends on the ability to feed a training process with data **features** that enable an effective learning

- a.k.a. "help the ML system to help you"

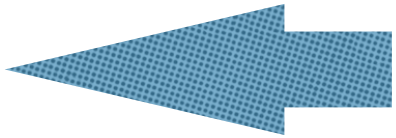
So-called "**Feature engineering**" (at large) can be crucial:

- **feature selection**: among existing and already collected features, select the most useful features to train on
- **feature extraction**: combining existing features to produce a (unreal?) more useful one (dimensionality reduction algos can help here)
- **creating and adding new features** (also by gathering new data)

Main **challenges** in ML

Basically, related to:

- bad data
- **bad algos**
- training technicalities



Main challenges in ML: **Overfitting / Underfitting**

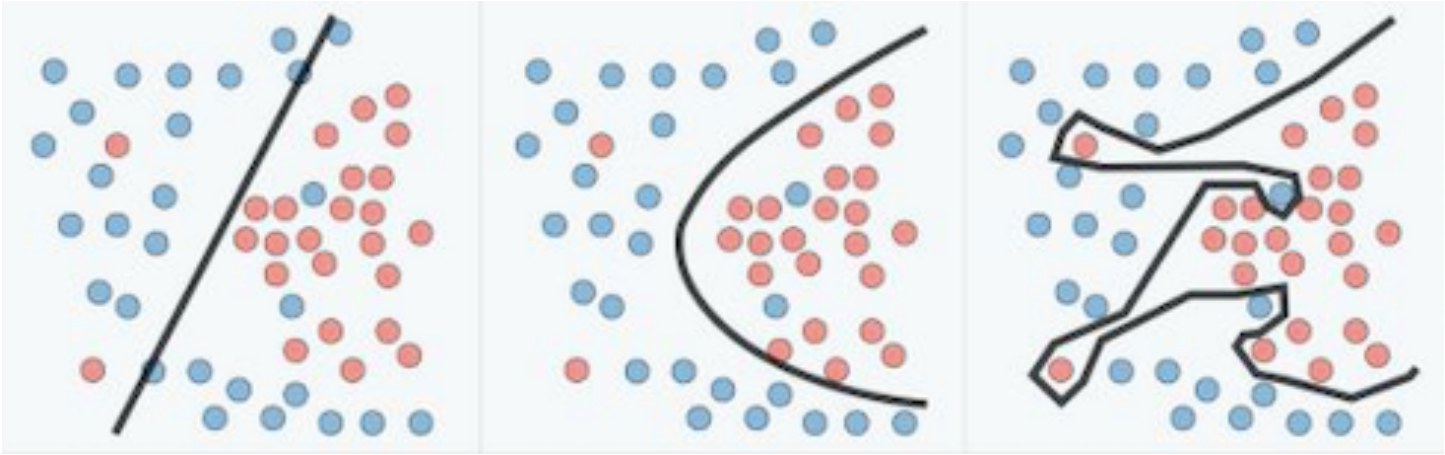
Bad algos

Underfitting

Overfitting

for regression

for classification



Main challenges in ML: **Overfitting / Underfitting**

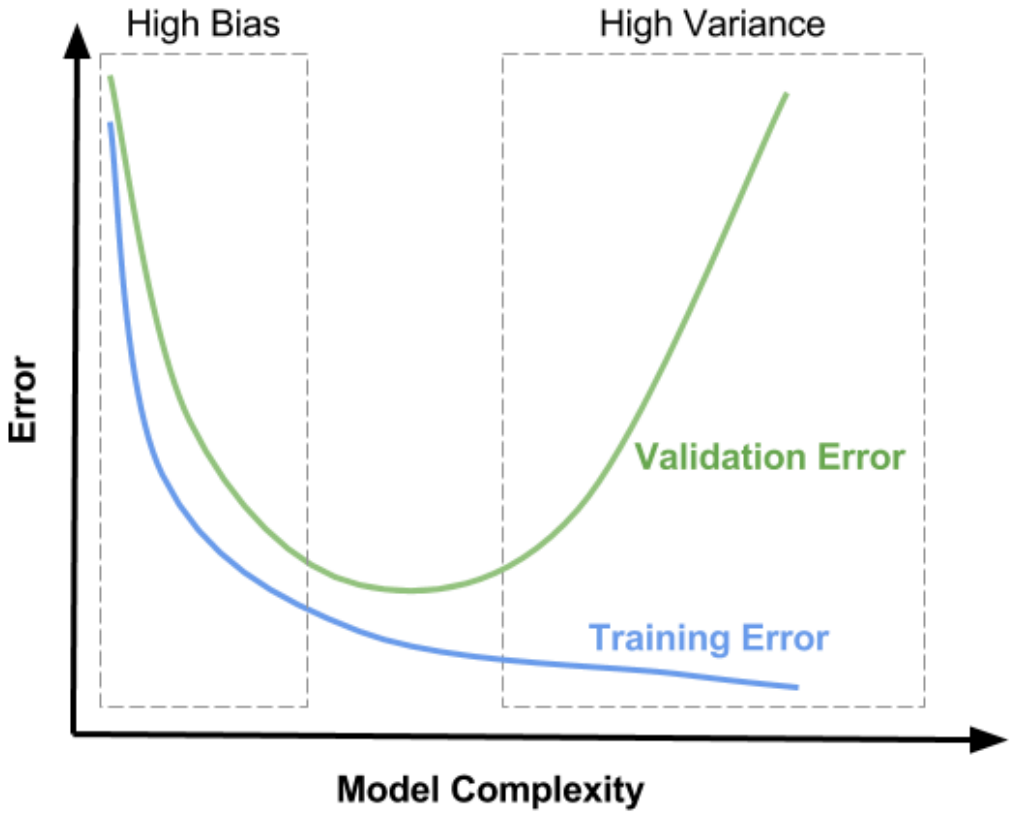
Bad algos

Overfitting (i.e. *high variance*): a model performs well on the training data, but it does not generalise well to new, previously unseen data.

- *Actions* → reduce # features, apply regularisation, sometimes collect more training data, ..

Underfitting (i.e. *high bias*): a model is too simple to learn the underlying structure of your training data

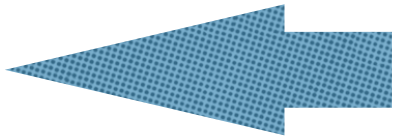
- *Actions* → selecting a more powerful model; add features (feature engineering); train more; reduce constraints on the model, ..



Main **challenges** in ML

Basically, related to:

- bad data
- bad algos
- **training technicalities**



Training + Test

The only way to know how well a model will generalise to new cases is to actually try it out on new data.

Basic choice: to **splitting into training set and test set**

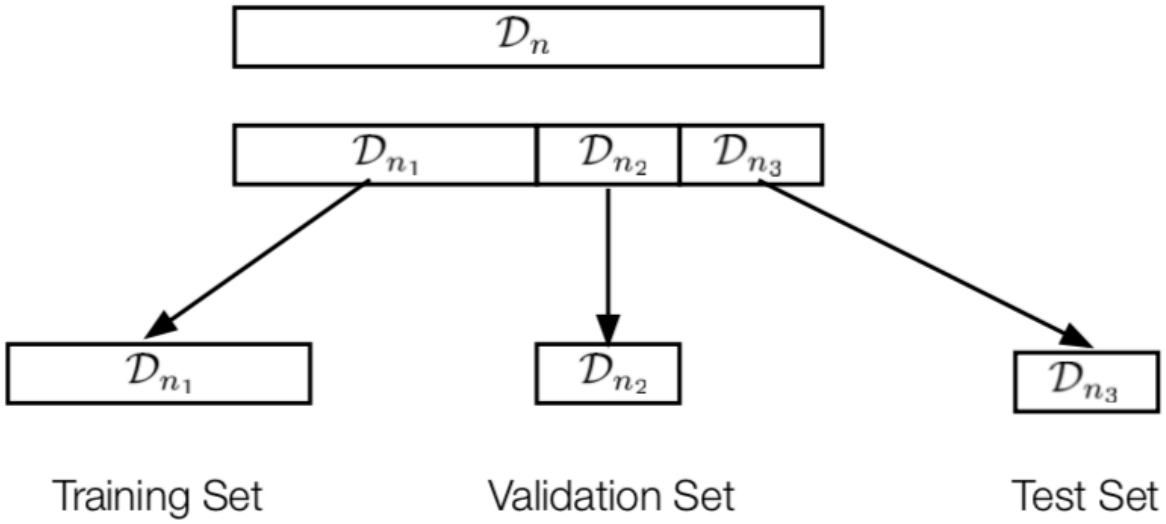
- train the model on the training set, evaluate performance on the test set
  - estimate the "generalisation error": if high, the model is overfitting

More refined options do exist..

Training + Validation + Test

Run a holdout validation process, i.e. use a **validation** set

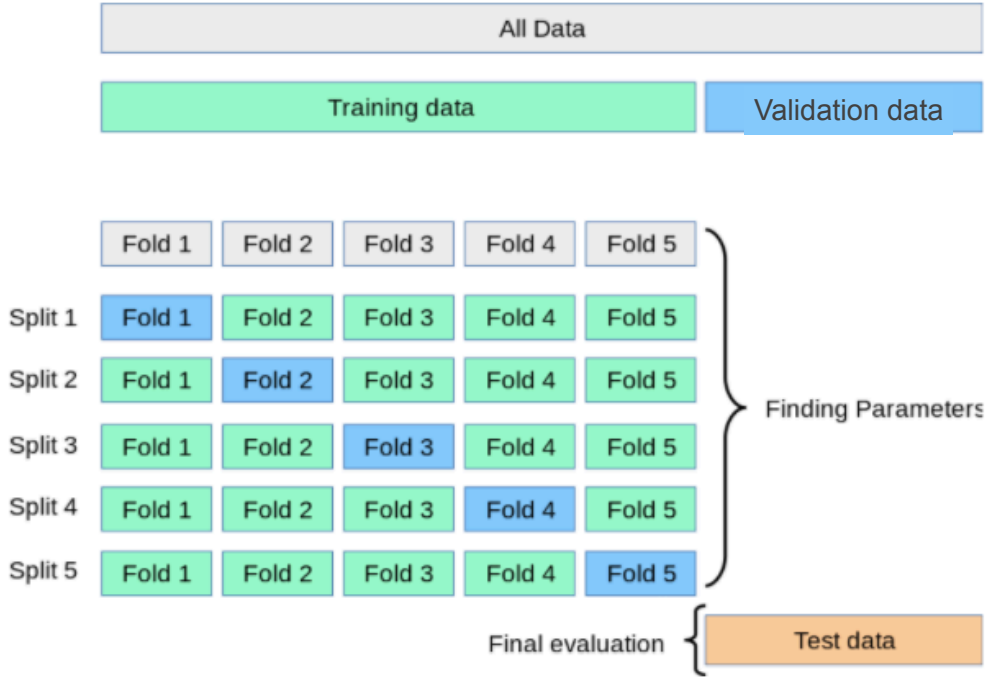
- you train multiple models with various hyperparameters on the "reduced" training set (i.e. full training set minus the validation set), and you select the model that performs best on the validation set.
- Then, re-train the best model on the full training set (including the validation set), and this gives you the final model
- Lastly, you evaluate this final model on the test set to get an estimate of the generalisation error



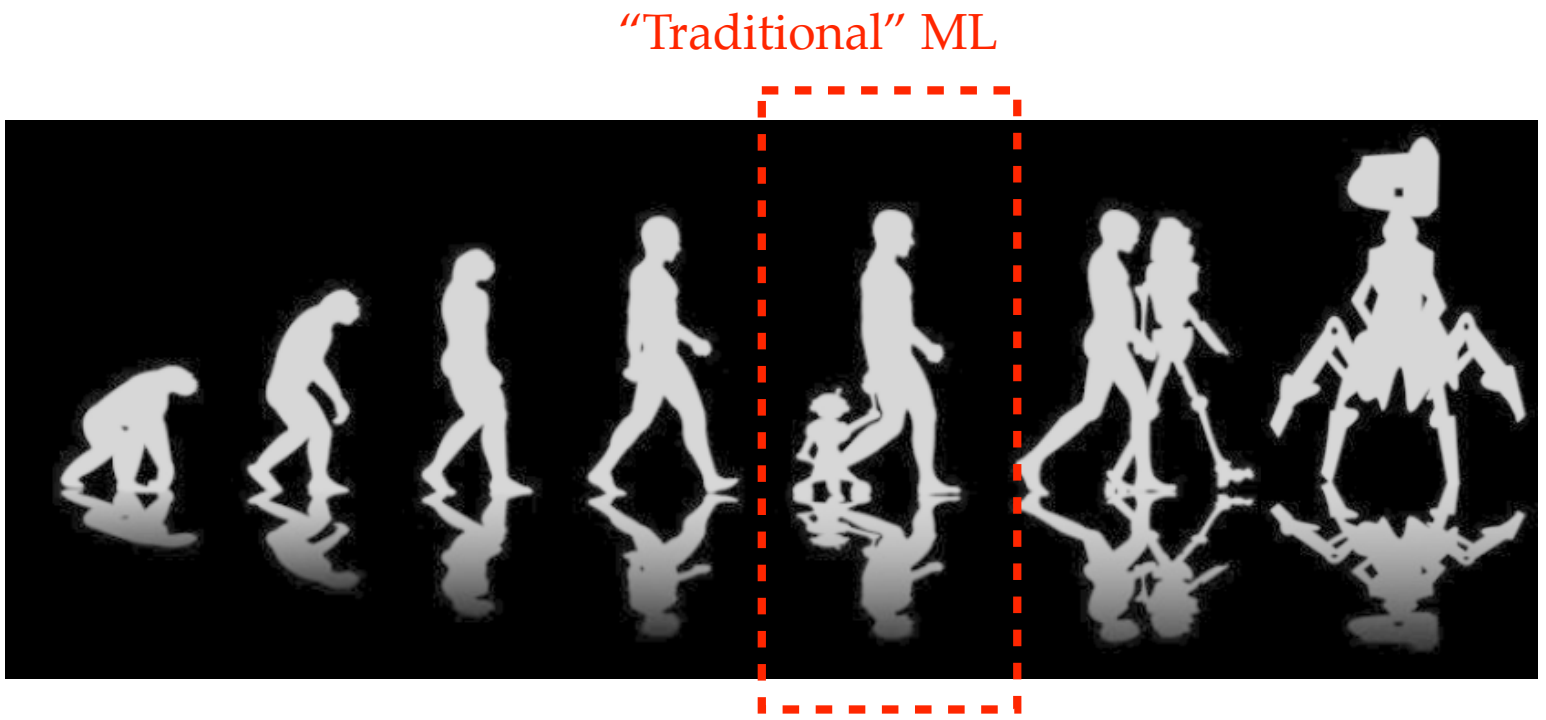
Cross-validation

Cross-validation: split training set in *k* small "folds" and repeatedly hold out one fold, i.e. train on k-1 and test on 1

- by averaging out all the evaluations of the model, a much more accurate measure of its performance can be obtained
  - drawback: increase in the requirements on computing resources



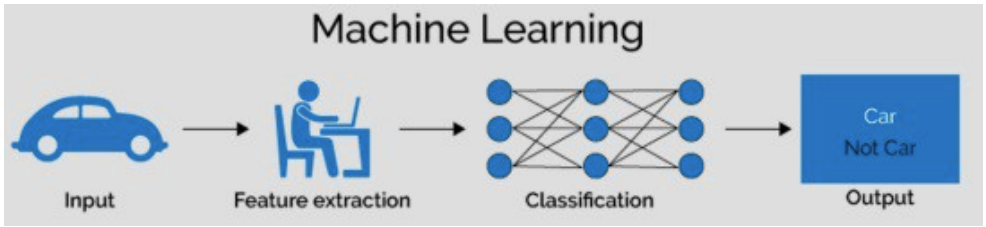
The evolution of ML adoption in HEP (also: at INFN)



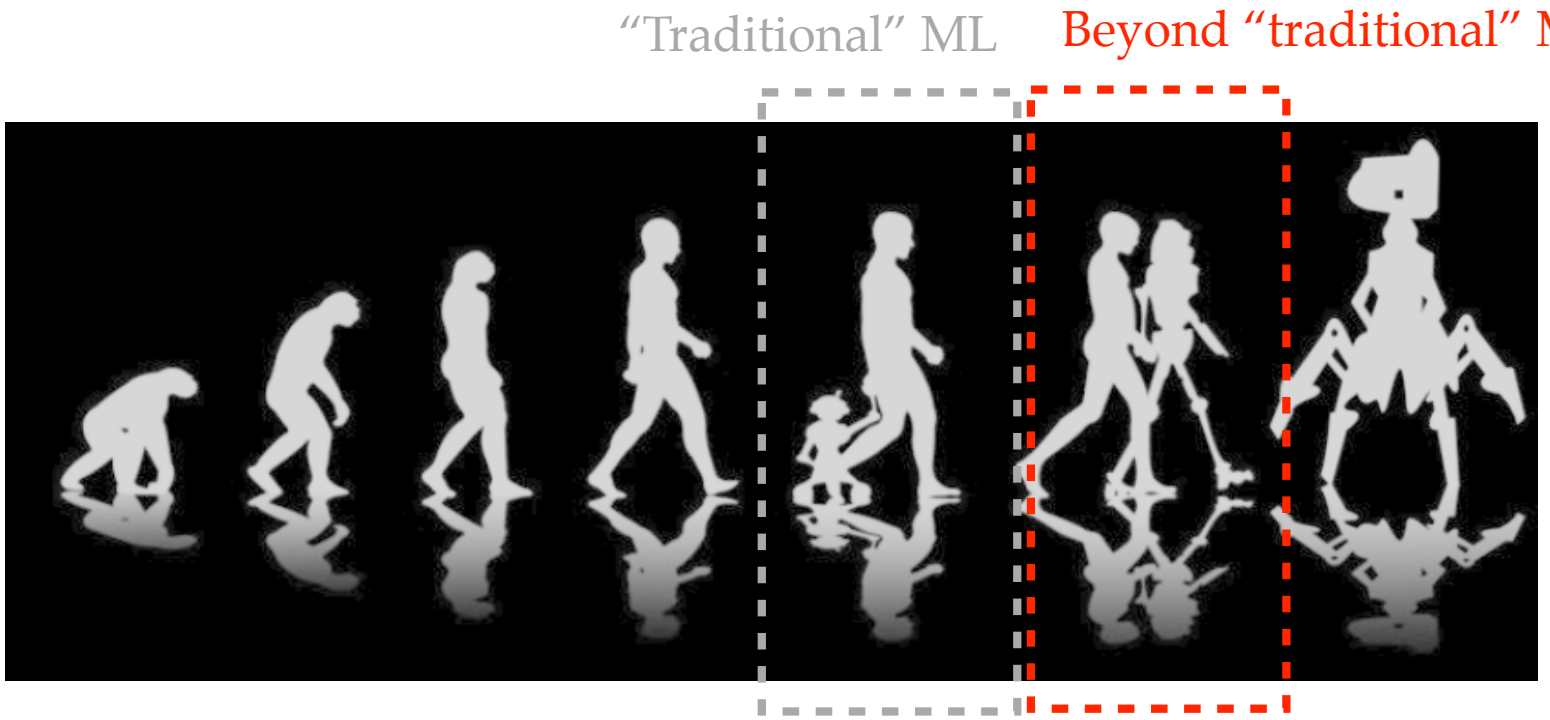
Until few years ago, the overall ML@HEP scenario was based on exploiting field-specific knowledge for **feature engineering**

The approach:

- **use physicist-designed high-level features as input to traditional shallow ML algorithms**



The evolution of ML adoption in HEP (also: at INFN)

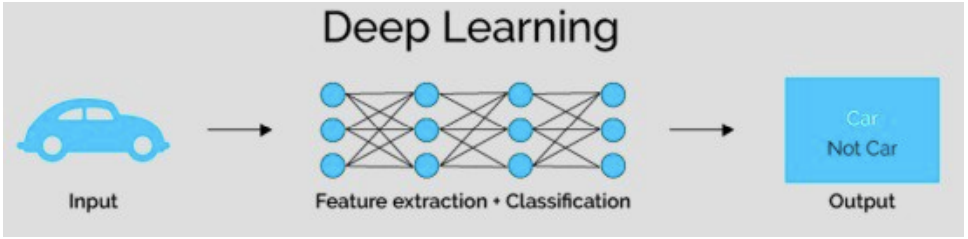


Since few years, ML@HEP exploits cutting-edge ML algorithms

- **multiple architectures of Neural Networks** (NNs), depending on specific use-cases

The approach:

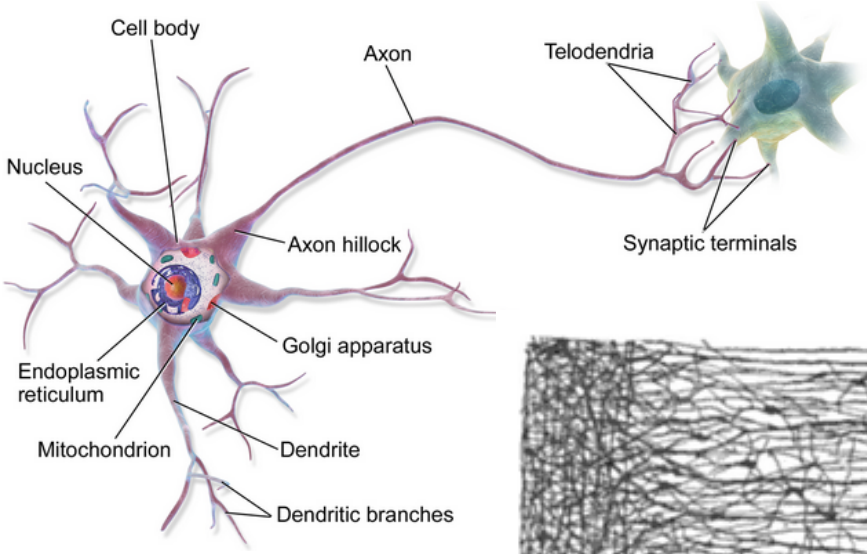
- **use of full high-dimensional feature space to train Deep NNs; growing effort in HEP to skip the feature-engineering step**
  - analogy with progresses in Computer Vision and Natural Language Processing

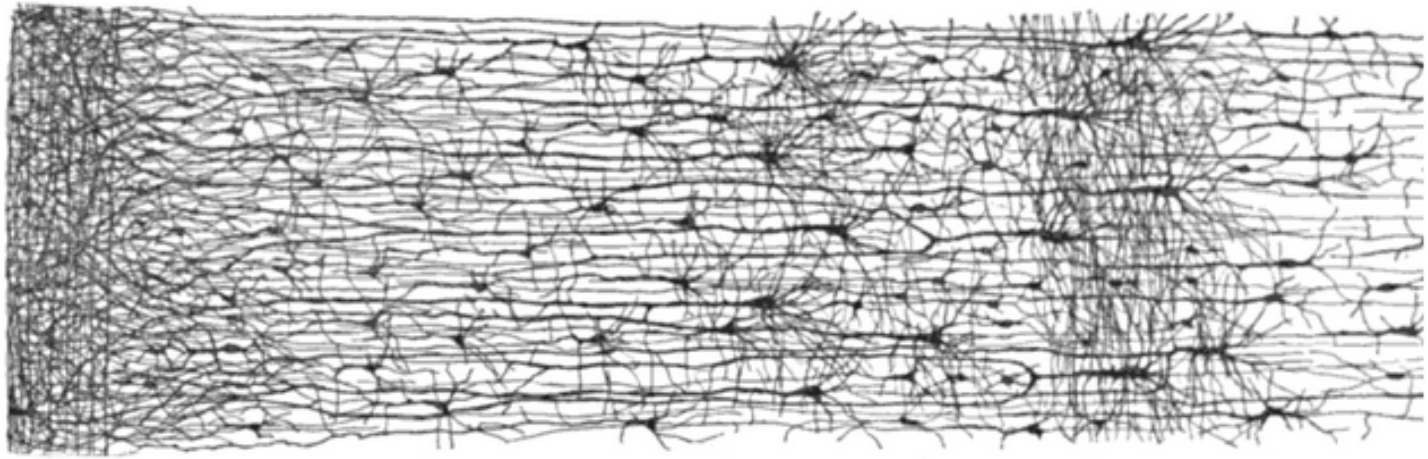


Neural Networks

Inspiration from biology and neurosciences

The architecture of biological neural networks (BNN) is still the subject of active research, but some parts of the brain have been mapped, and we know that **neurons are relatively simple** and they are **connected in large networks**, that are **often organized in consecutive layers**…

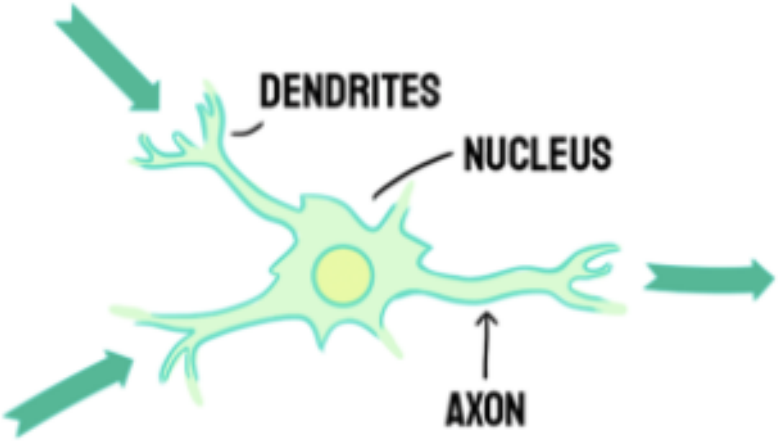




From BNN to ANN

ANN as simulation of Biological NN

- NN are developed by simulating networks of human neurons

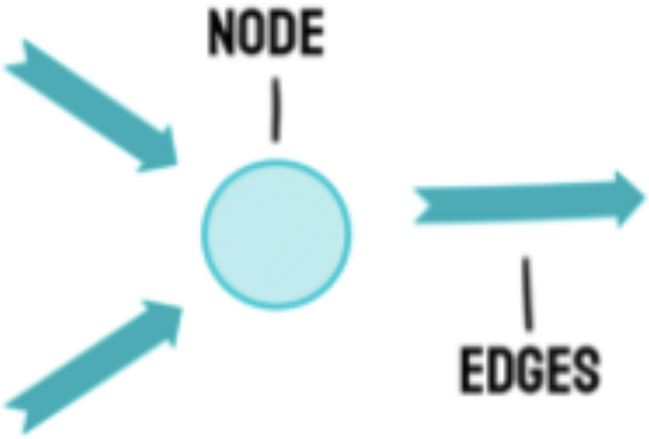


The human neuron has a cell body, a number of input wires (*dendrites*) and an output wire (*axon*)

- we can think of it as a **computational unit** that gets inputs, computes, and spits output to other neurons (which we will call **nodes** or **units**)
- we can think of it simply as a **logistic unit** that computes some h as a sigmoid

Communication in BNNs is done via pulses of electricity

- we can think of **I/O** passing in/out **numerical values**
- a **wide** network..



Vastly oversimplified model..

Why NNs as ML algorithms?

Important to model **non-linear relationships among attributes**, can **avoid the feature engineering step**, and can come in **many architectures** (like NN "flavours") and hence are extremely **flexible**

NNs at the very core of **Deep Learning**

- versatile, powerful, scalable

Ideal to tackle large and highly complex ML tasks, such as:

- classifying billions of **images** (e.g. Google Images)
- powering **speech recognition** services (e.g. Siri, Alexa, ..)
- **recommending** the best videos to watch to *O*(100M) users every day (e.g. YouTube)
- **learning to beat humans** in very specific tasks (e.g. from medical applications - e.g. radiology - to gaming - e.g. Go and DeepMind's AlphaZero)

Brief history of NNs

1943 (!): first introduction by McCulloch and Pitts

- the first ANN architecture in human history

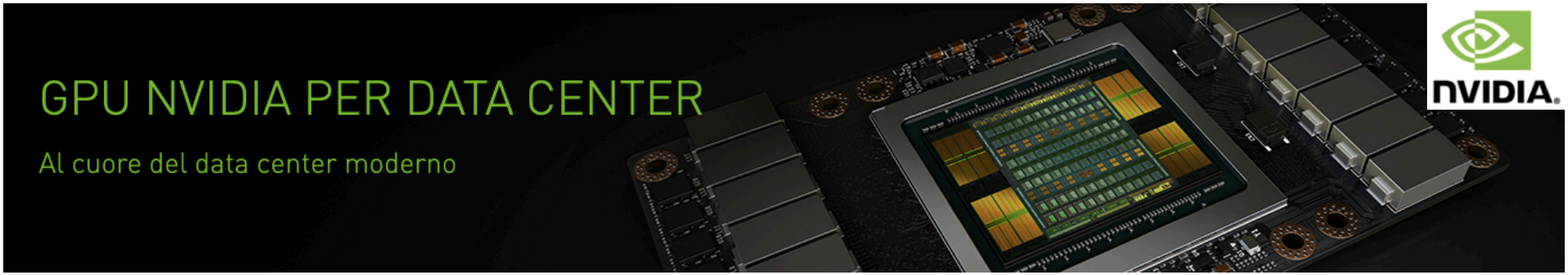
1960s: the successes of ANNs stopped - the "**first AI winter**"

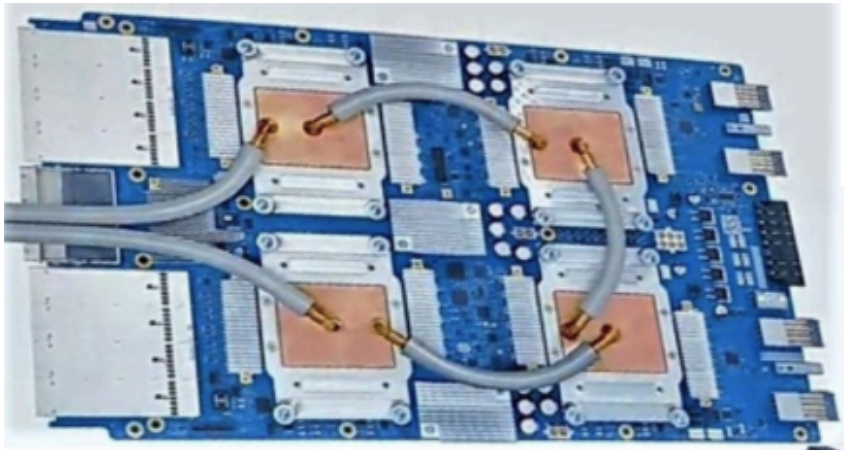
Early 1980s: revival of interest in connectionism, new architectures, better training techniques. Slow progress, though.

1990s: other (not NN) powerful ML techniques were invented, e.g. SVMs. Seemed to offer better results and stronger theoretical foundations than ANNs - the "**second AI winter**"

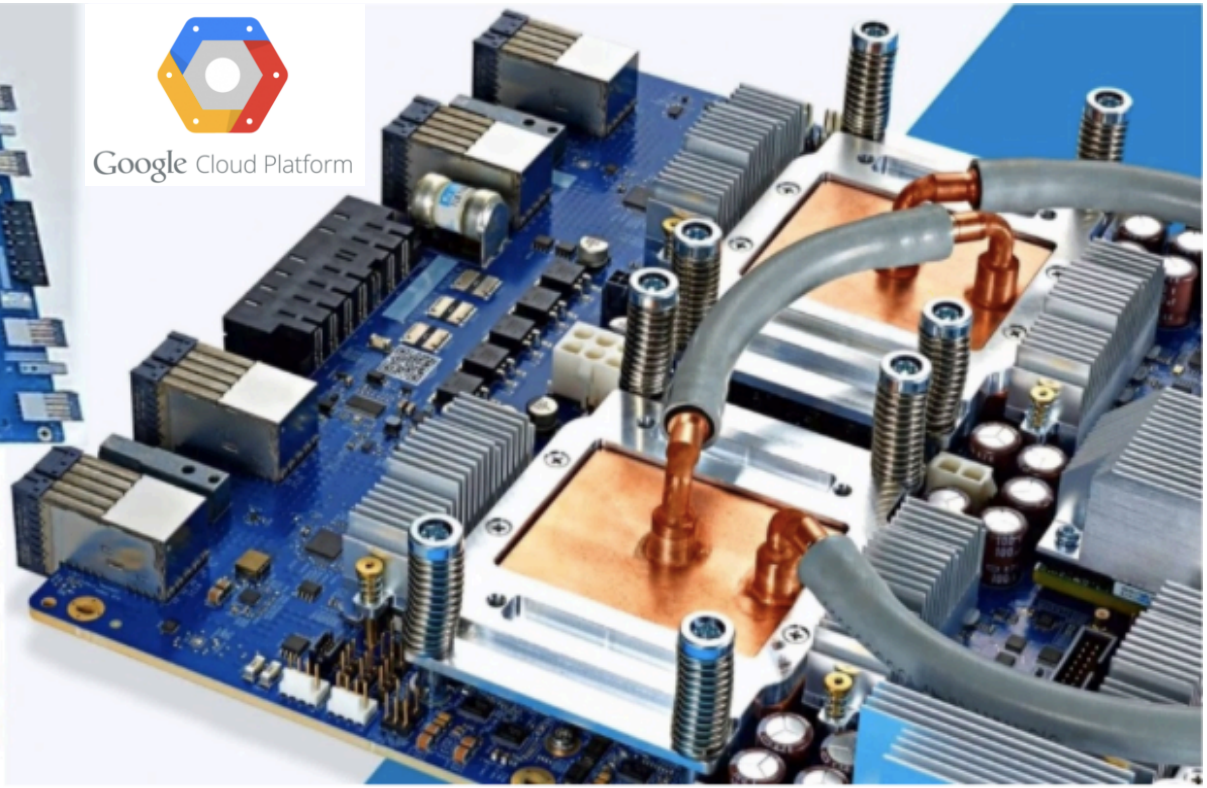
2000s-2010s: a new "AI spring": why now?

Why AI **now**, and not decades ago?





Google Cloud Platform



Why AI **now**, and not decades ago?

A revive and acceleration happened recently, mainly because of factors that I would list as:

- the raise of **Big Data**
- the **technology** progresses (e.g. GPUs)
- "Democratisation" [*] of massive computing resources via **Cloud** approaches

[*] to be discussed…

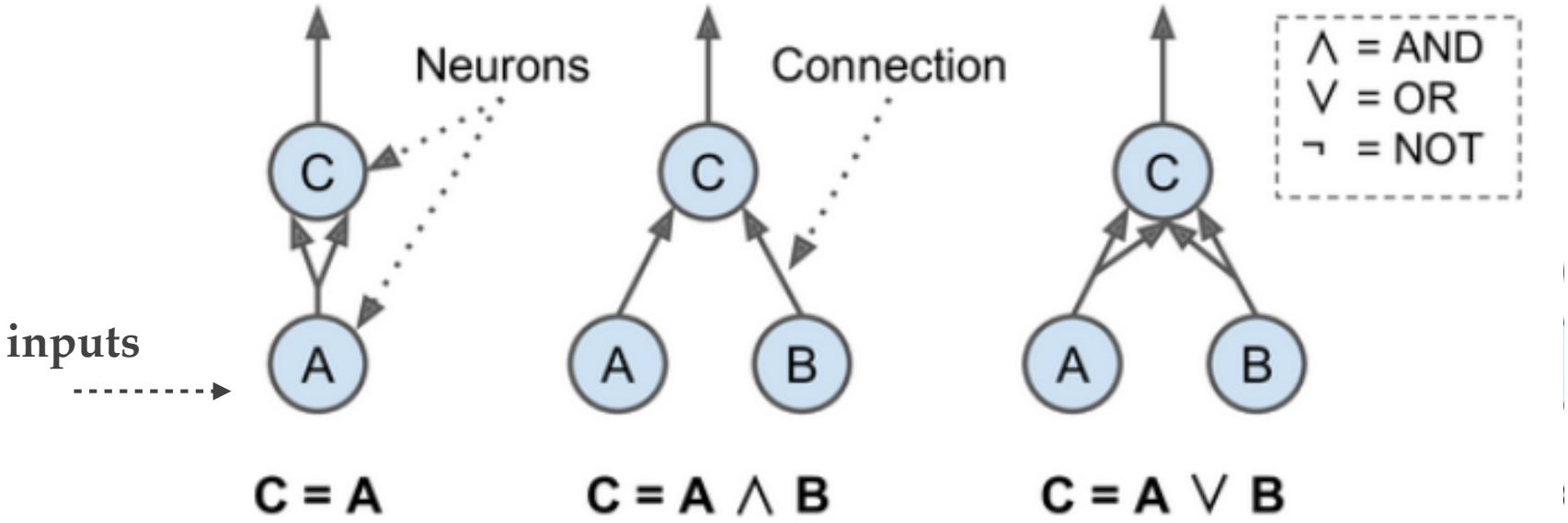
Today, it is a fact that DL are among the core transformative technologies at the basis of most world-wide activities aiming at extracting **actionable insight from (big) data**.

Logical computation with neurons

The McCulloch/Pitts model for a biological neuron was later called "**artificial neuron**"

- it has 1 (or more) **binary (on/off) inputs** and 1 **binary** output (*NOTE: all binaries!*)
- it activates its output when more than a certain number of its inputs are active (e.g. 2 in the examples below)

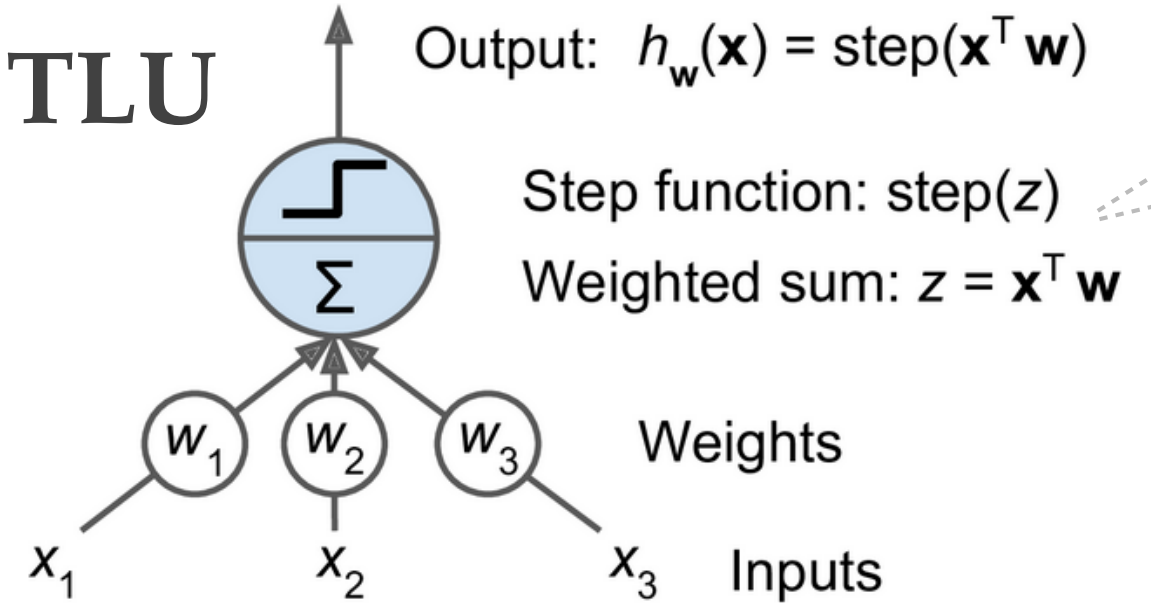
They showed that even with such a simplified model it is possible to build a network of artificial neurons that computes **any basic logical proposition** you want



The **TLU** and the Perceptron

1957, Rosenblatt: "Perceptron" as one of the simplest possible ANN architectures, based on a slightly modified artificial neuron, called **threshold logic unit** (**TLU**) or linear threshold unit (LTU)

- the **inputs/output** are now **numbers** (not binary on/off values)
- each input connection is associated with a **weight**
- the TLU computes a **weighted sum of its inputs**, applies a **step function** to that sum, and **outputs** the result



$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

1 single TLU can act as a simple **linear binary classifier**

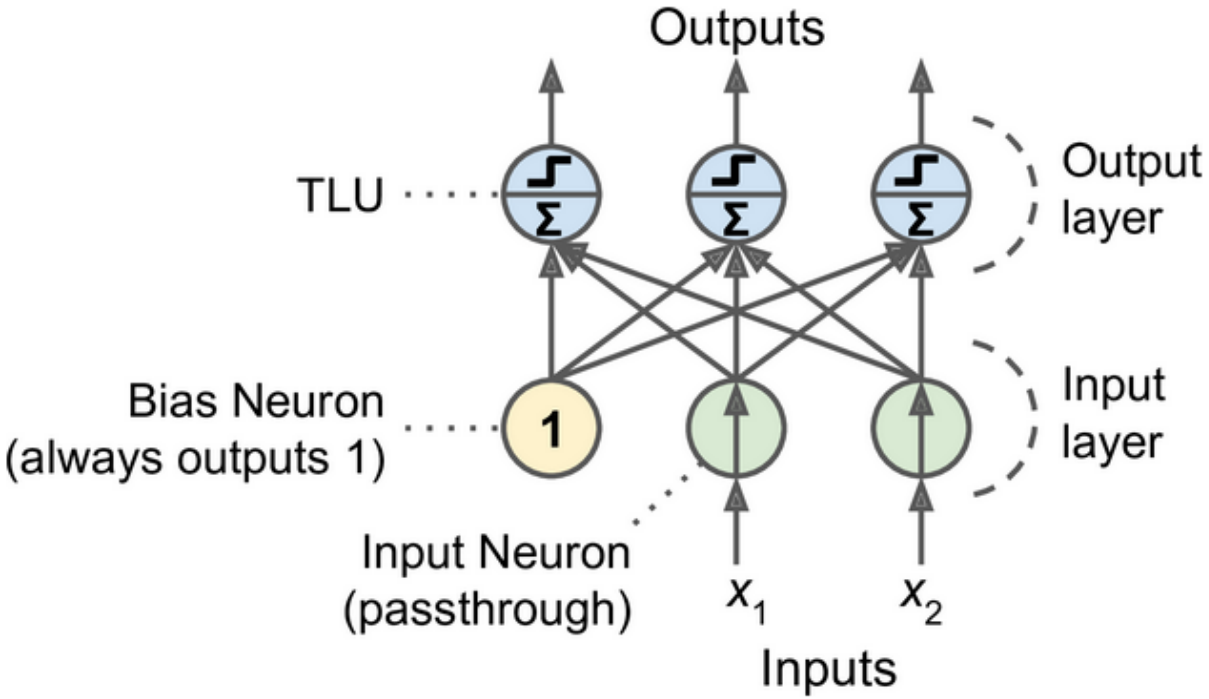
- linear combination of the inputs, and depending of the thresholds → positive/ negative class (just like Logistic Regression classifier or a linear SVM)

The TLU and the **Perceptron**

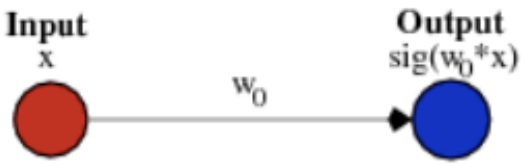
A **Perceptron** is then composed of **a single layer of TLUs** with each TLU connected to all the inputs

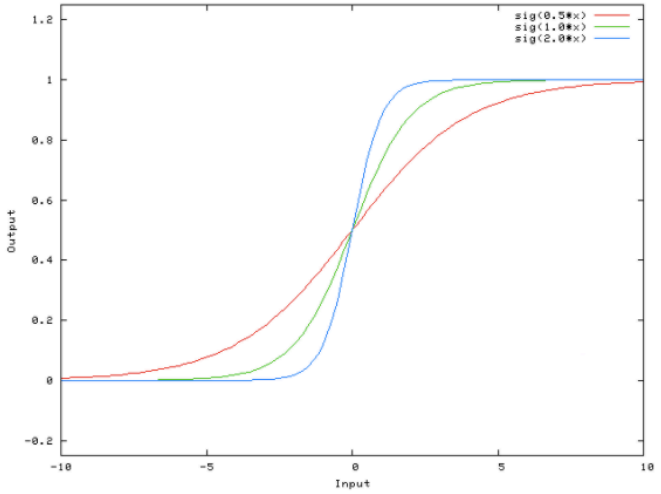
- in the **input layer**, an extra bias feature is generally added ($x_0 = 1$), via a "**bias neuron**" that just outputs 1 all the time
- When all the neurons in a layer are connected to all the neuron in the previous layer (i.e. here the input neurons), it is called a **fully-connected** or **dense** layer

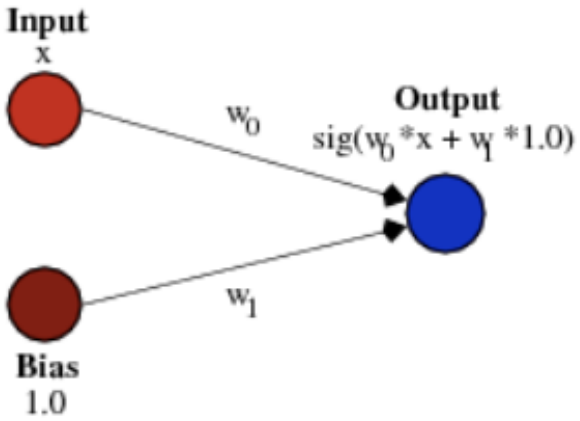
This Perceptron can classify instances simultaneously into 3 different binary classes, which makes it a **linear multi-class classifier**

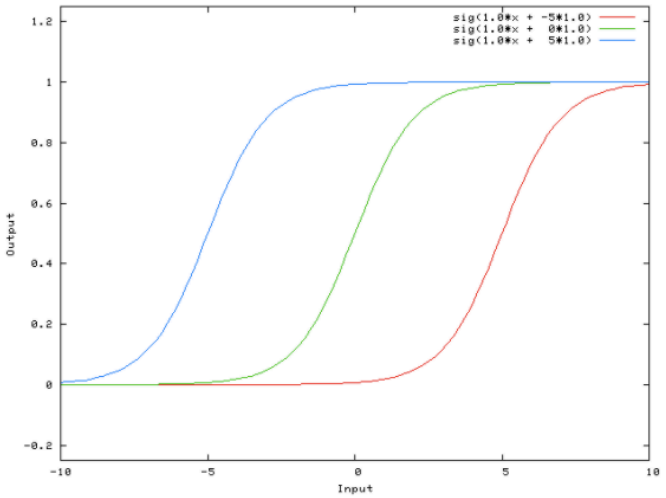


Why a bias neuron









A bias value allows you to shift the activation function to the left or right, which may be critical for successful learning

Training a **Perceptron**

Input from biology and neurosciences

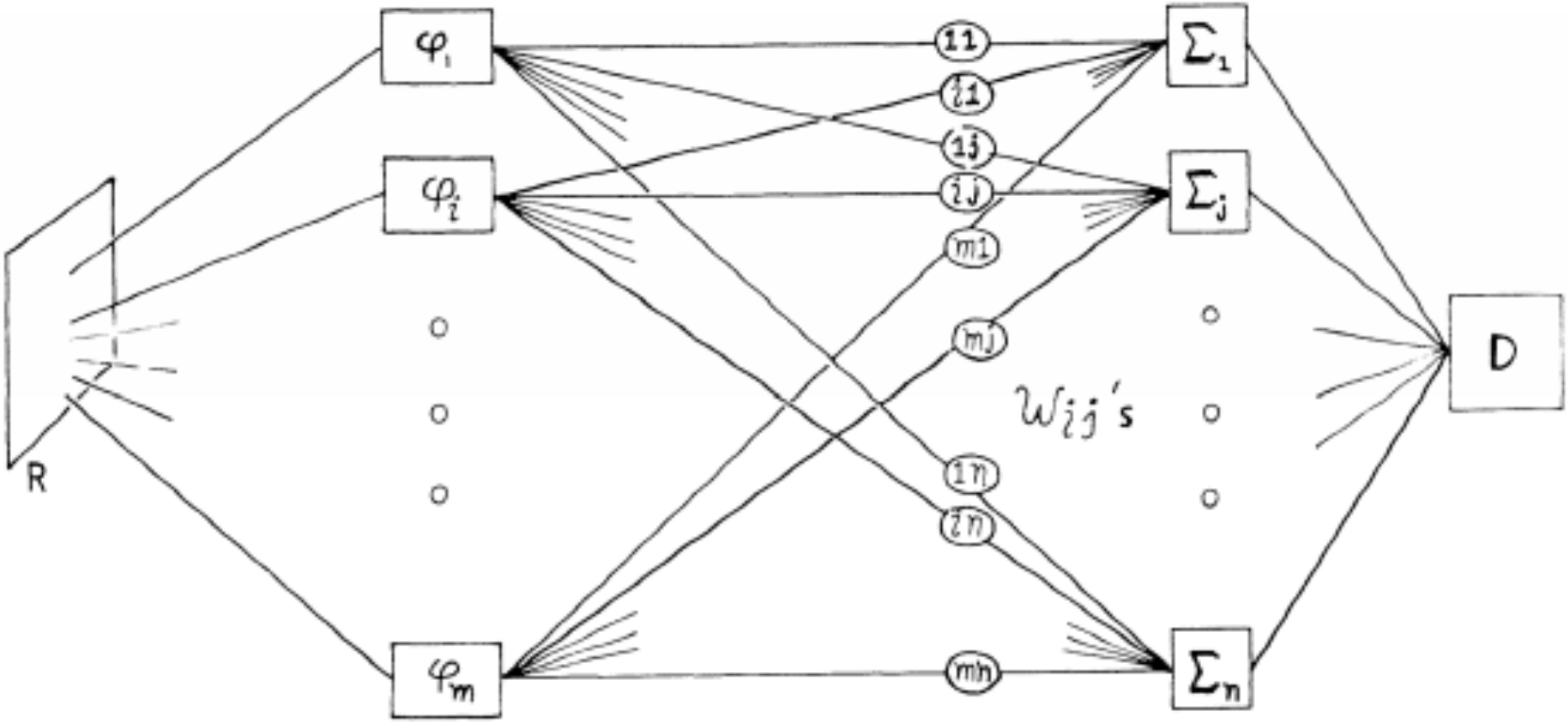
- *1949, Hebb; "Organization of Behavior"*
- **Hebb's rule** (Hebbian learning): when a biological neuron triggers another neuron, the connection between these two neurons **grows stronger**
  - Lowel: "**Cells that fire together, wire together**"

Largely inspired by Hebb's rule, the Perceptron training algorithm proposed by Rosenblatt was a **weight update**

- the Perceptron is fed one training instance at a time. For each instance it makes its prediction(s). For every output neuron that produced a wrong prediction, it reinforces the connection weights from the inputs that would have contributed to the correct prediction

Multi-Layer Perceptron (MLP)

1969, Minsky/Papert, “Perceptrons”, the monograph



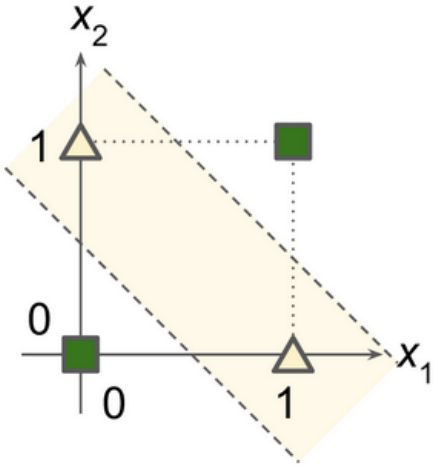
Multi-Layer Perceptron (MLP)

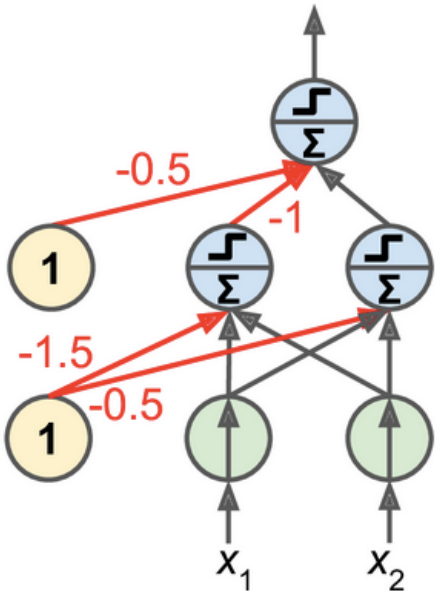
Minsky/Papert pointed to serious weaknesses of Perceptrons

- in particular its incapability of solving relatively trivial problems, e.g. the exclusive-OR (XOR) classification problem
  - Well, true of any other linear classification model (e.g. Logistic Regression classifiers).. but researchers had expected much more from Perceptrons! Great disappointment. People dropping off the field..

But.. it soon turned out most limitations of Perceptrons could be eliminated by **stacking multiple Perceptrons**! → first concept of "**layers**"!

The resulting ANN is called a **Multi-Layer Perceptron (MLP)**

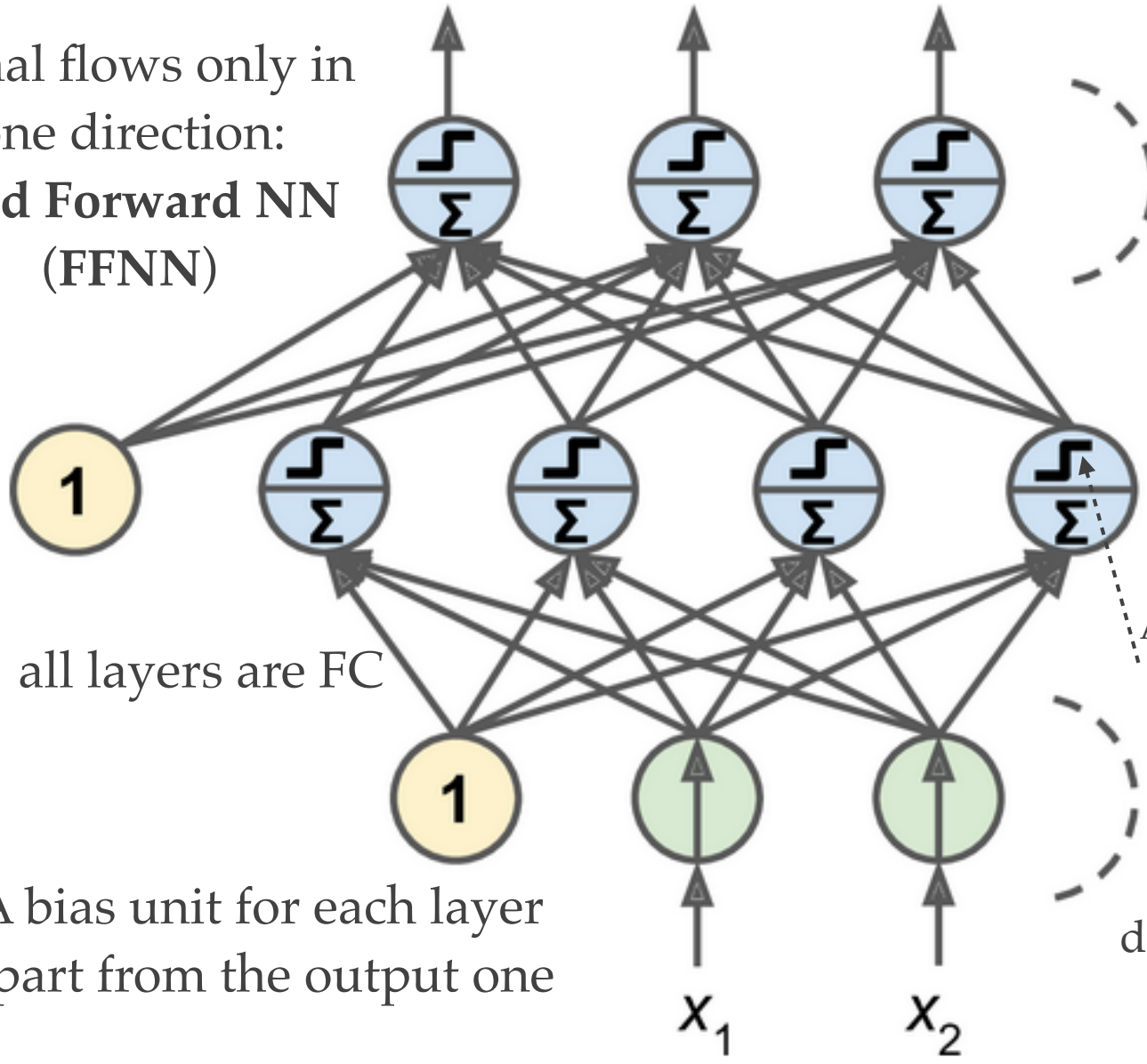




Note that **you added an <u>entire brand new layer</u>** to the original Perceptron architecture

(see next)

MLP - FCNN - FFNN



Signal flows only in one direction: **Feed Forward NN (FFNN)**

layers of TLUs

Output layer

Hidden layer

Input layer

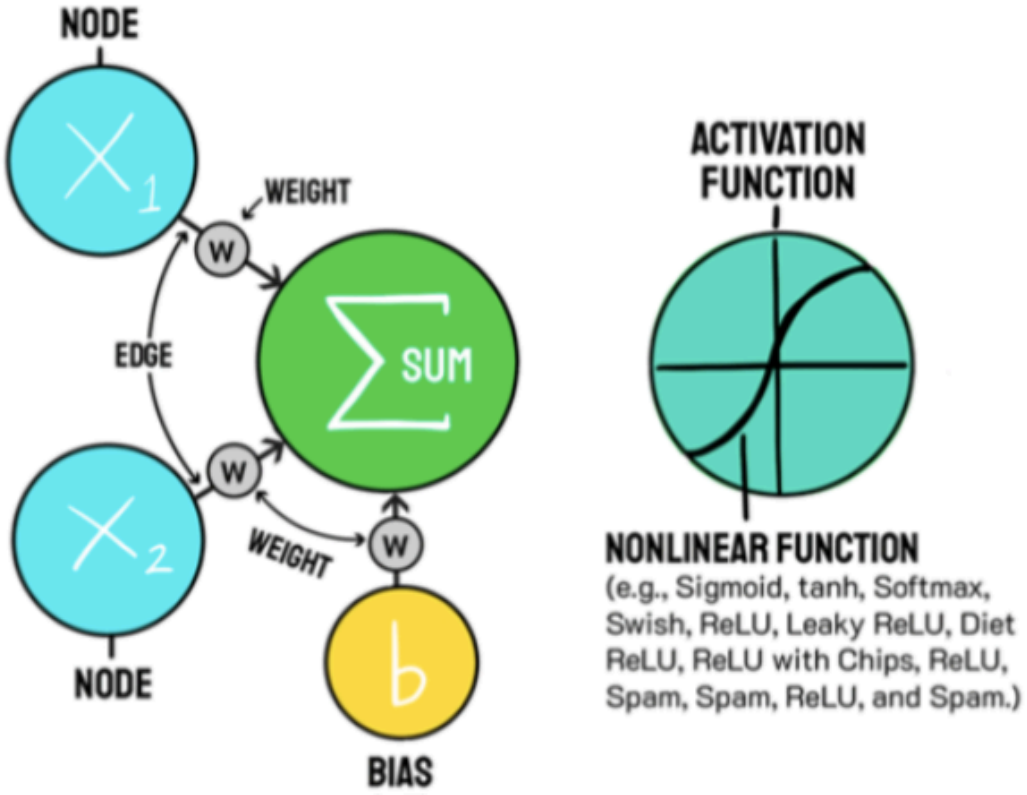
Activation functions important to have some non-linearity between layers

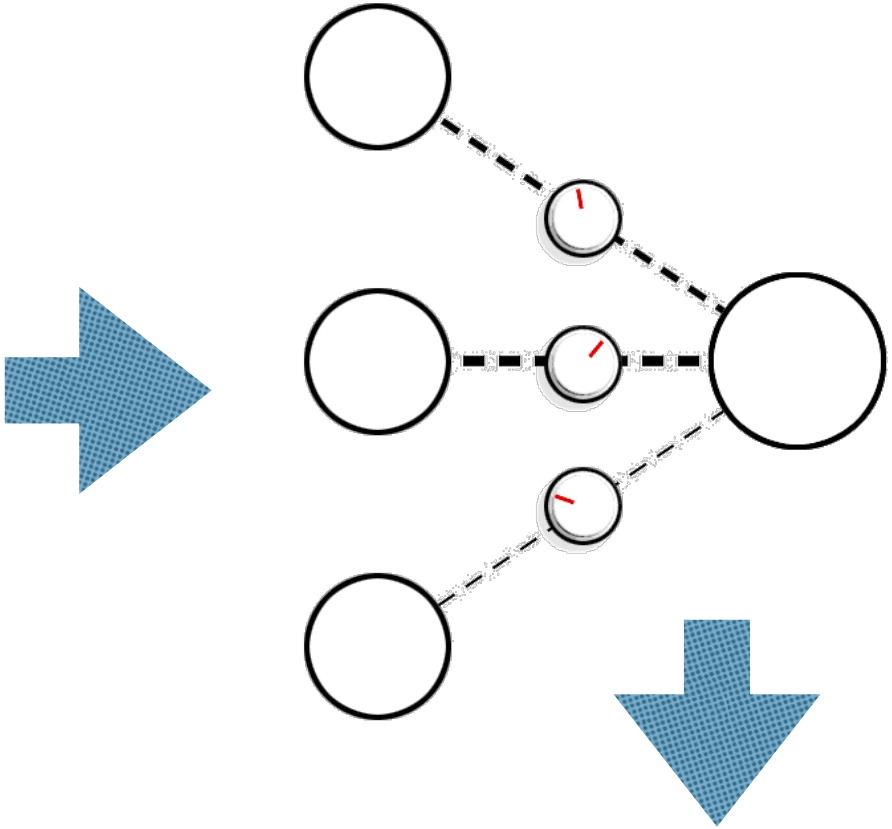
all layers are FC

A bias unit for each layer apart from the output one

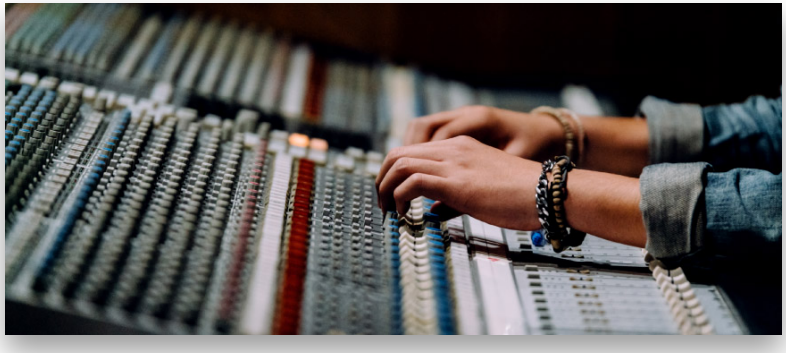
deep stack of hidden layers?
→ **Deep NN (DNN)**
(Deep Learning sometimes used also for shallow, non deep, NN)

Set the weights through training



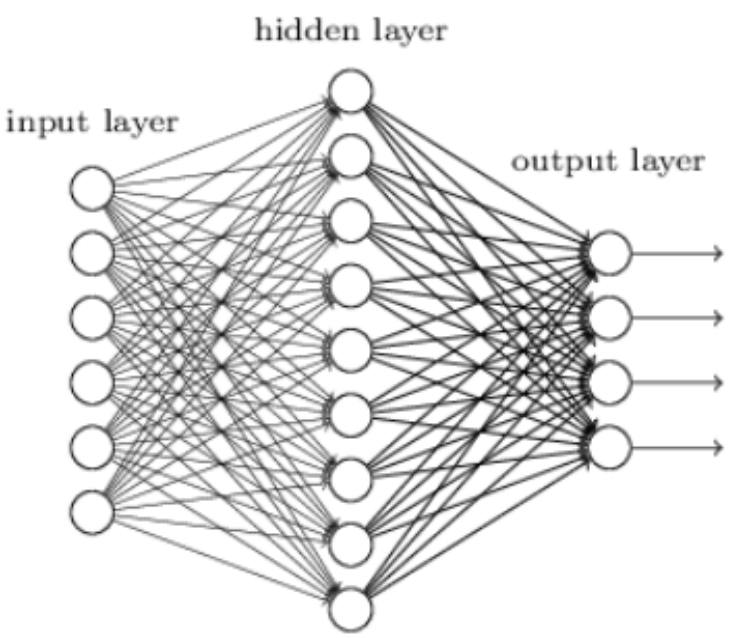


In a simplified analogy, the training process is like a delicate (but automated) sound engineering process that turns all knobs for the best sound !

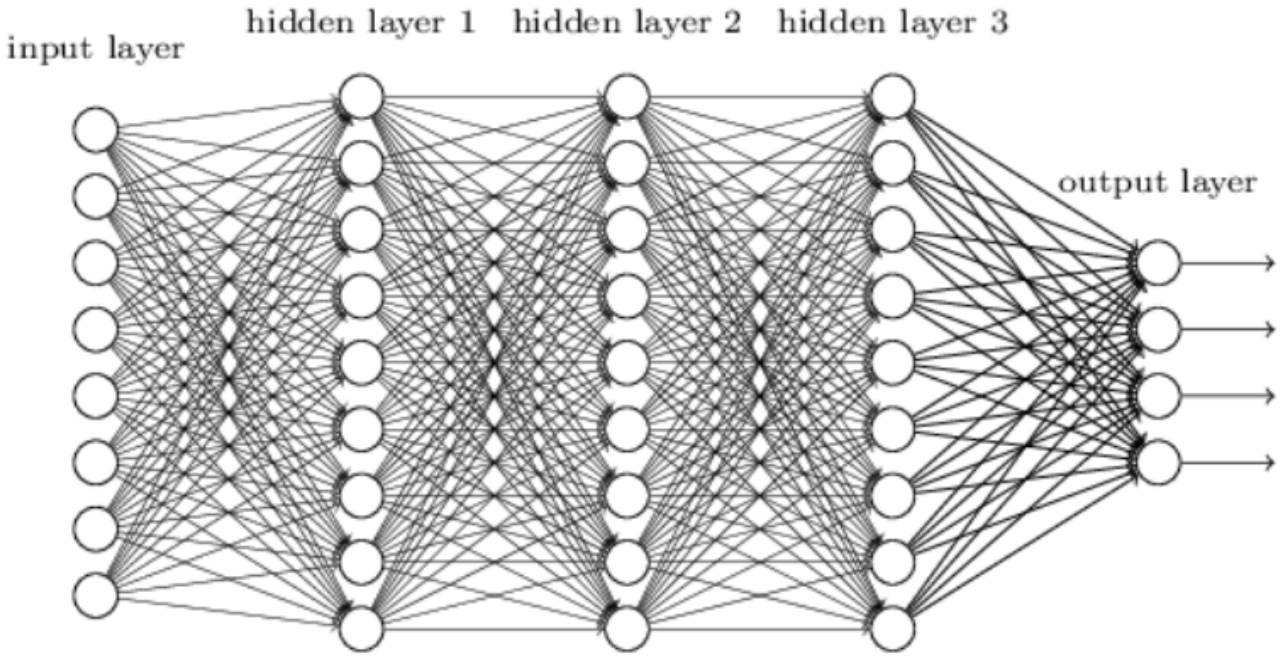


From shallow to **deep** NN

"Non-deep" feedforward neural network

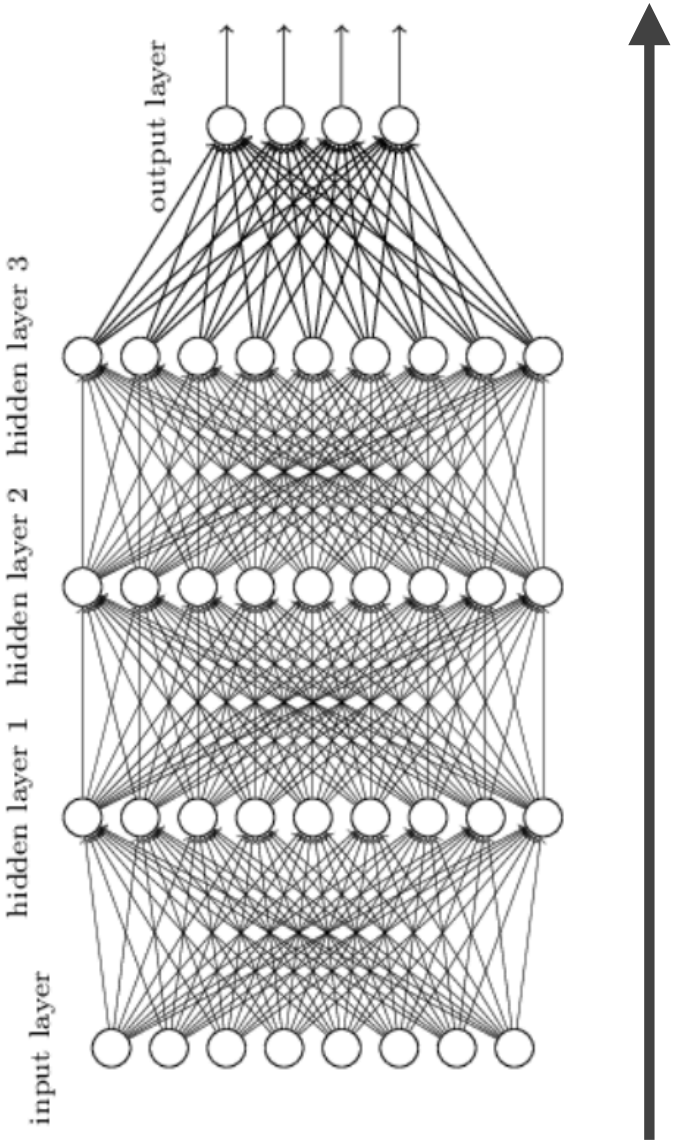


Deep neural network



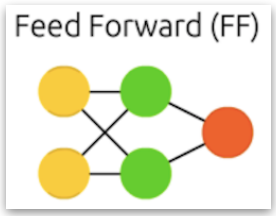
From shallow to **deep** NN

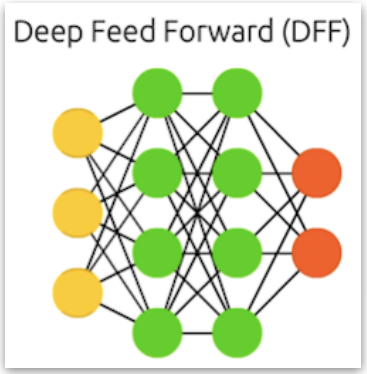
Deep neural network

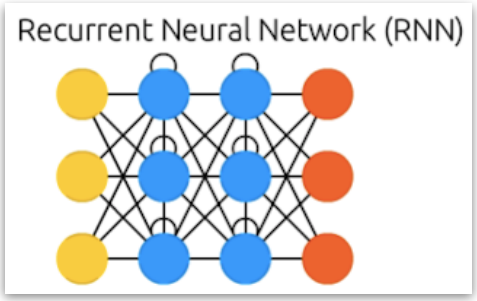


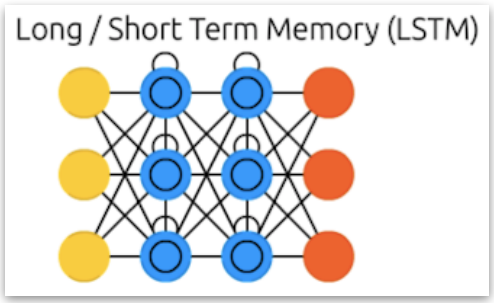
This is the **depth** dimension in Deep Learning

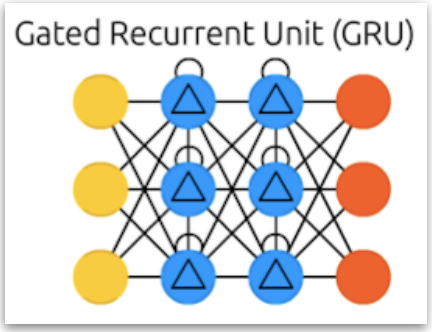
Deep NN.. ok.. but which one?!

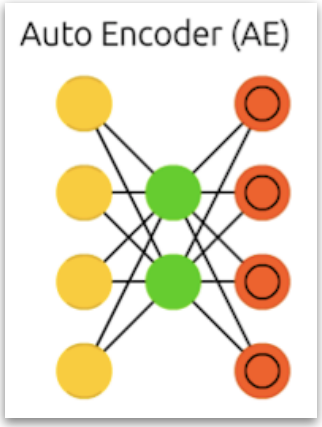


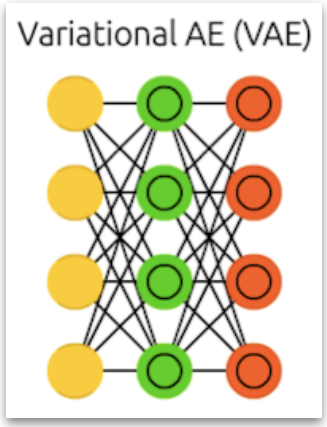


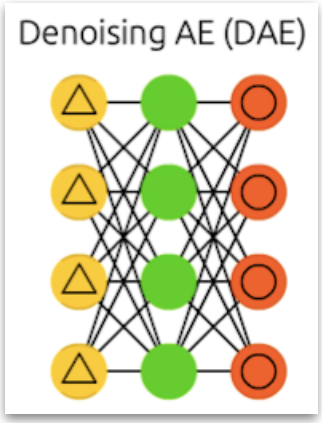


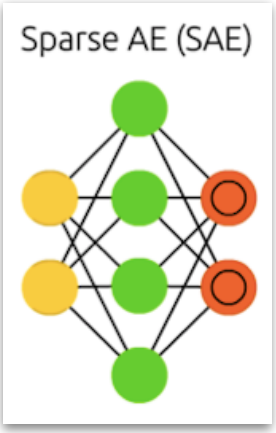


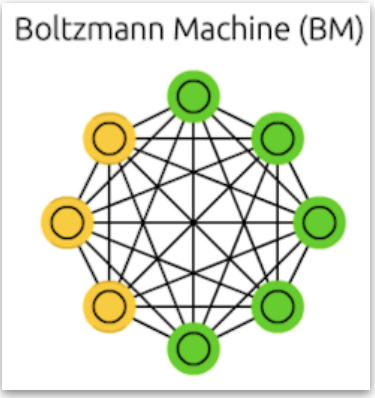


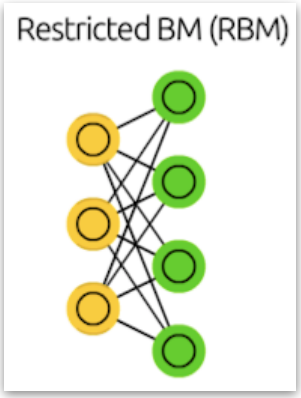


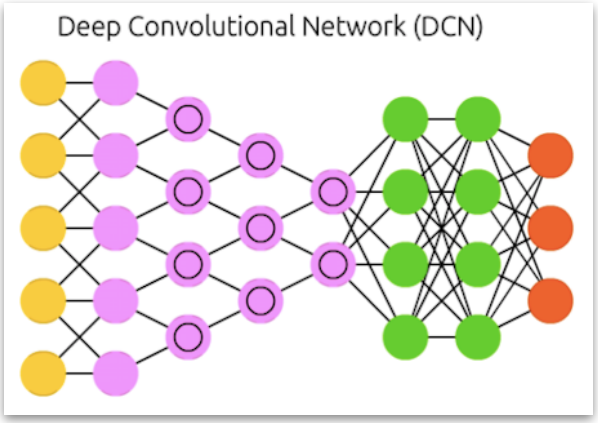


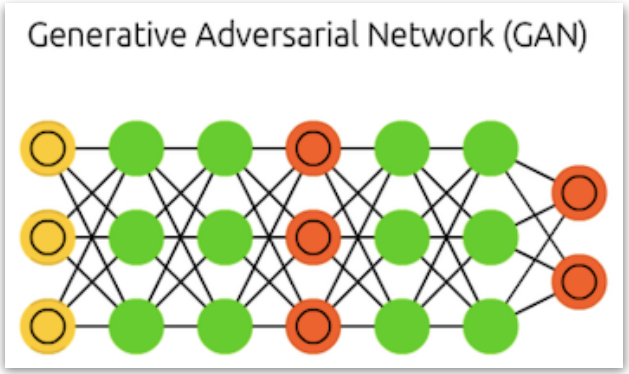










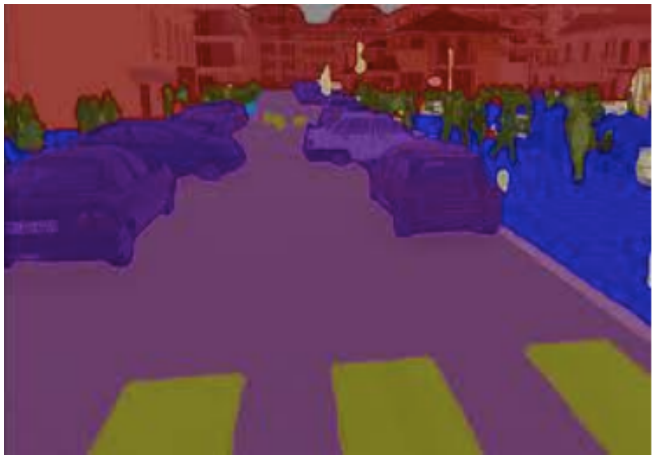


Convolutional Neural Networks (**CNN**)

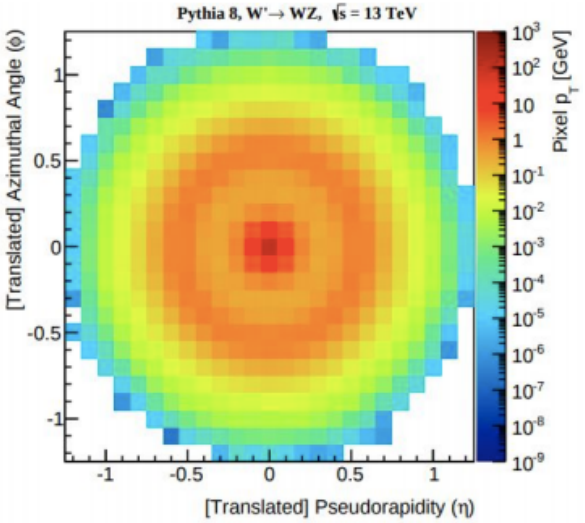
CNNs are based on strategies that decrease their sensitivity to the absolute position of elements in an image, making them more robust to noise

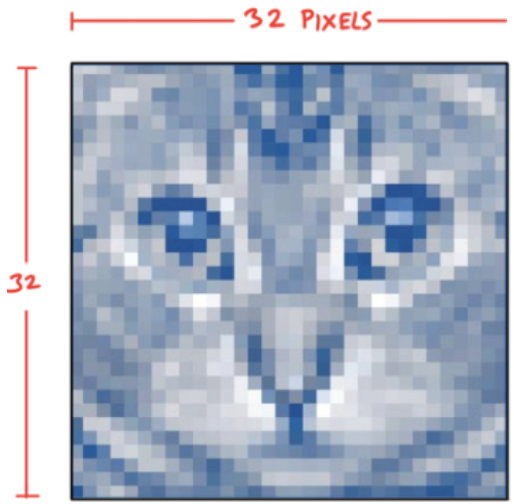
- Deep CNNs capable to extract complex features from images
  - e.g. use in self-driving cars, owing to translation-invariant feature learning
- particularly suited for HEP neutrino experiments
  - but also in simplified settings in collider experiments)

Industry:
large adoption in computer vision tasks

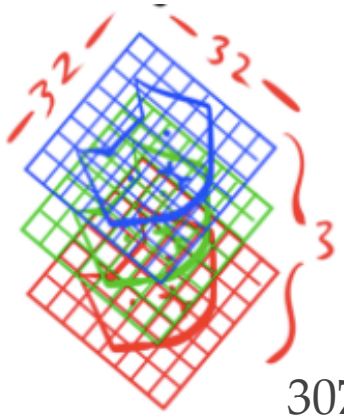


INFN:
3D imaging in detectors, event classification, ..



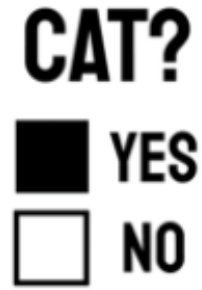


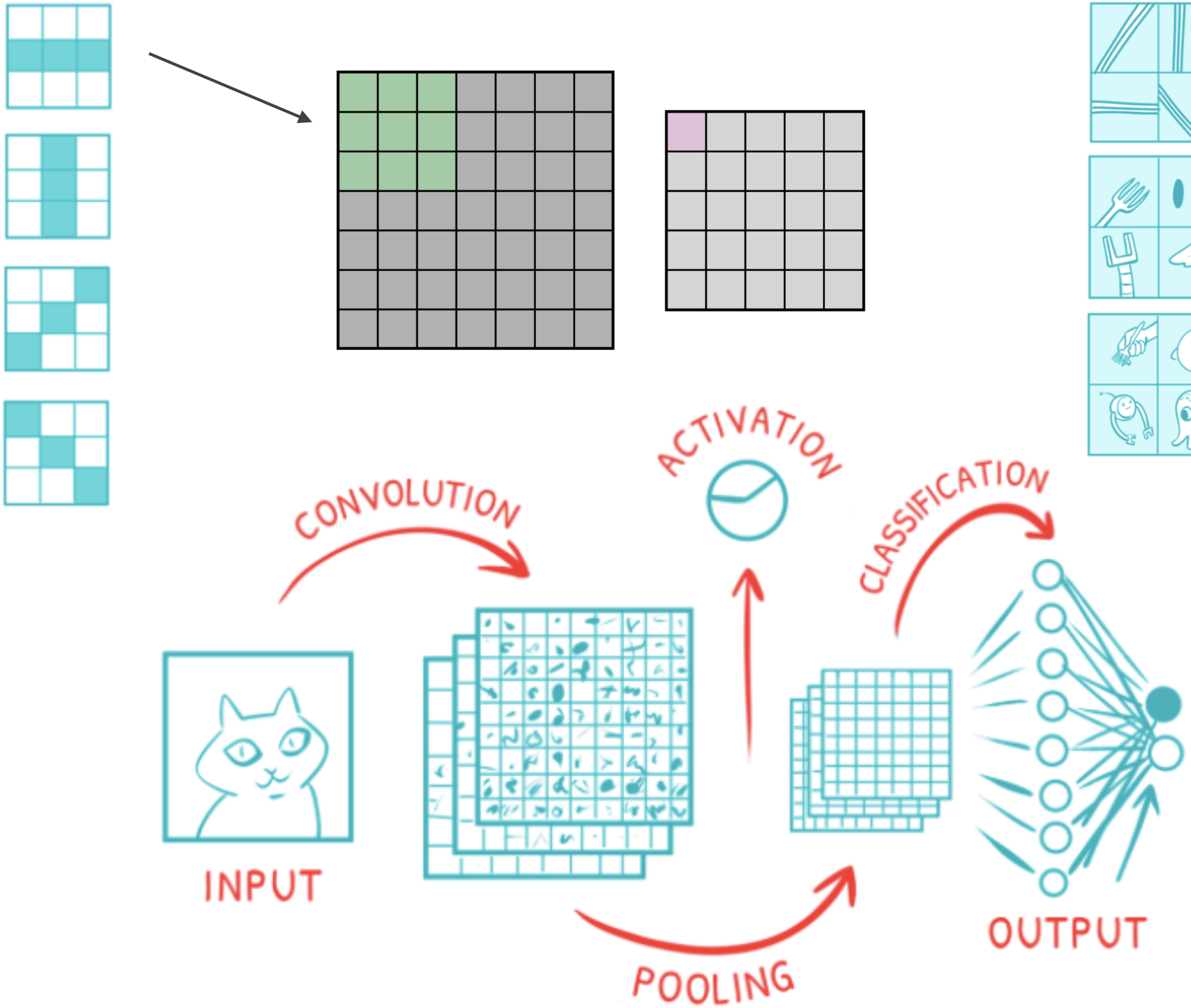
1024 pixels
(b / w)



3072 pixels
(coloured)

every cat, every position, any angle, portion, light condition, ..

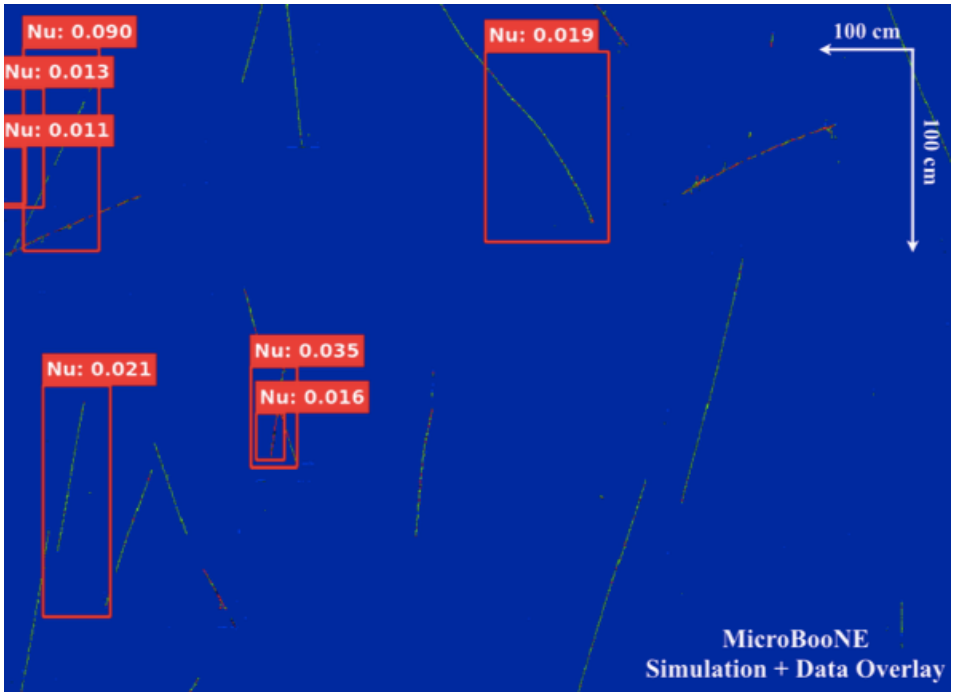



CONVOLUTION
ACTIVATION
CLASSIFICATION
INPUT
POOLING
OUTPUT

".. but HEP is different .."

Bonus feature!

Is it, really?



Detection of **neutrinos** on cosmic background event (method: **CNN**)

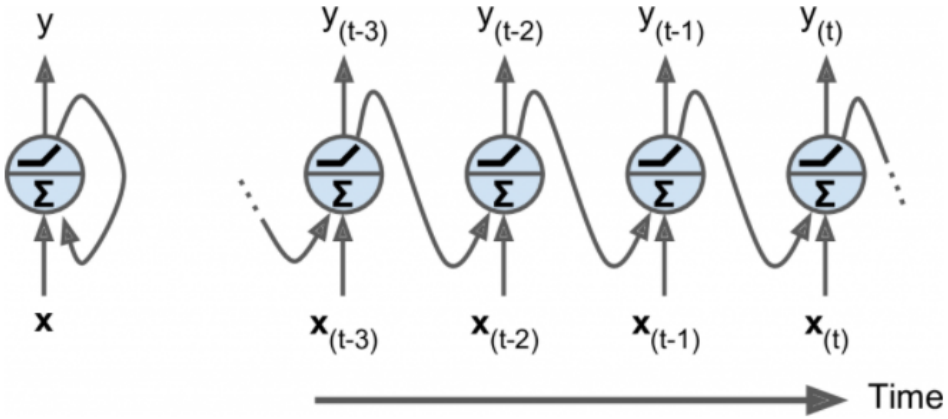
Detection of **airports** from satellite images (method: **CNN**)

Recurrent Neural Networks (**RNN**)

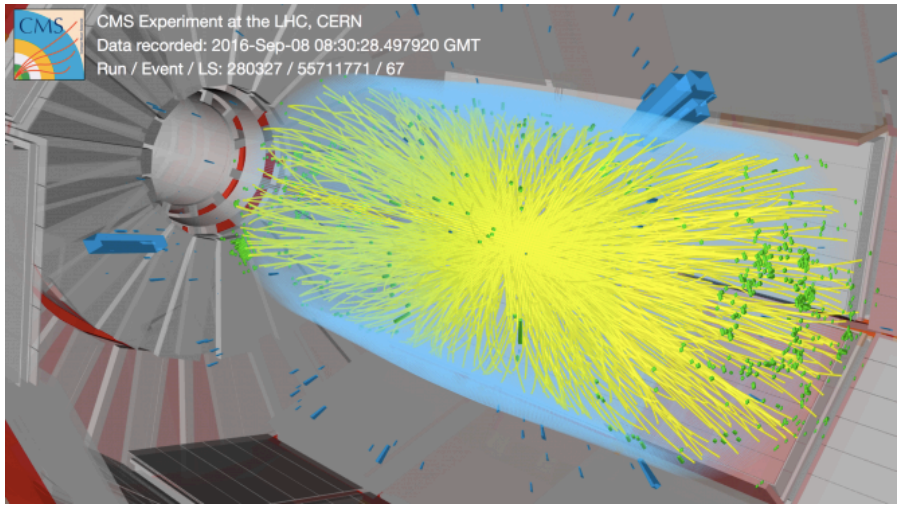
RNNs successful at processing long sequences of data

- based on recurrent neurons (with connections pointing backwards)
- able to treat variable-length input and to process time series by accumulating and using all the info across a sequence
  - e.g. current Google translation service

Industry:
managing "time series"
(audio, video, natural language processing)



INFN:
classifiers capable to process complex signals, or variable-lenght inputs (tracks, particles in jets, etc)

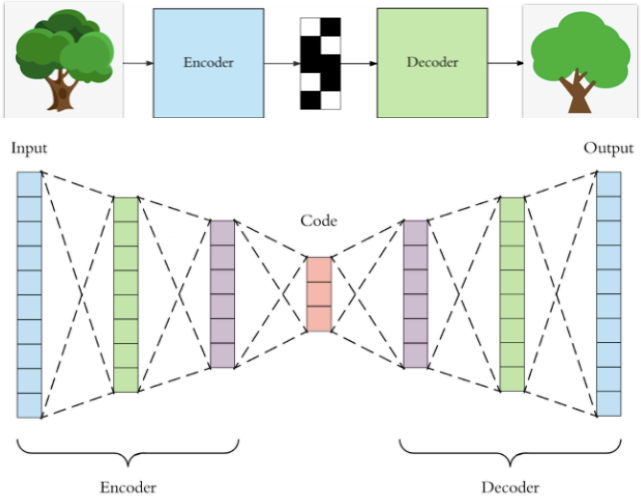


Autoencoders (**AE**) and Variational-AE (**VAE**)

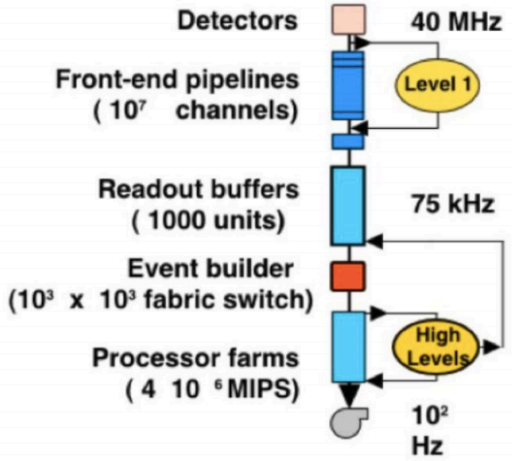
AEs (discriminative models) are feed-forward NN (unsupervised) able to compress input into a lower-dimensional representation ("latent-space", a data-specific compression) and to reconstruct the output

VAEs (generative models) can learn the parameters of a probability distribution representing the data → can generate new input data samples

e.g. **AEs in Industry**:
dimensionality reduction, denoising, …



HEP/INFN:
VAEs could isolate new physics as outliers of known distributions



Generative Adversarial Networks (**GAN**)

GANs as generative ML models

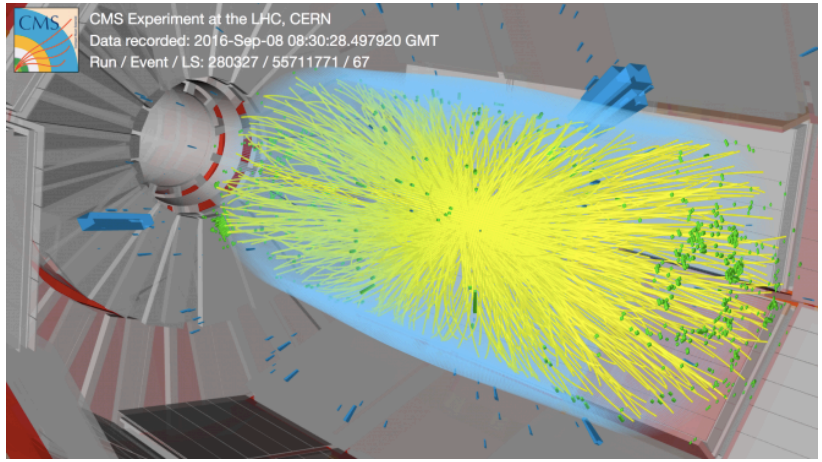
- designed as 2-NN game where one (generator NN) maps noise to images, and the other (discriminator NN) classifies the images as real vs fake (the best generator being the one that maximally confuses its adversary)

Industry:

image editing, data generation, security, ..

INFN:

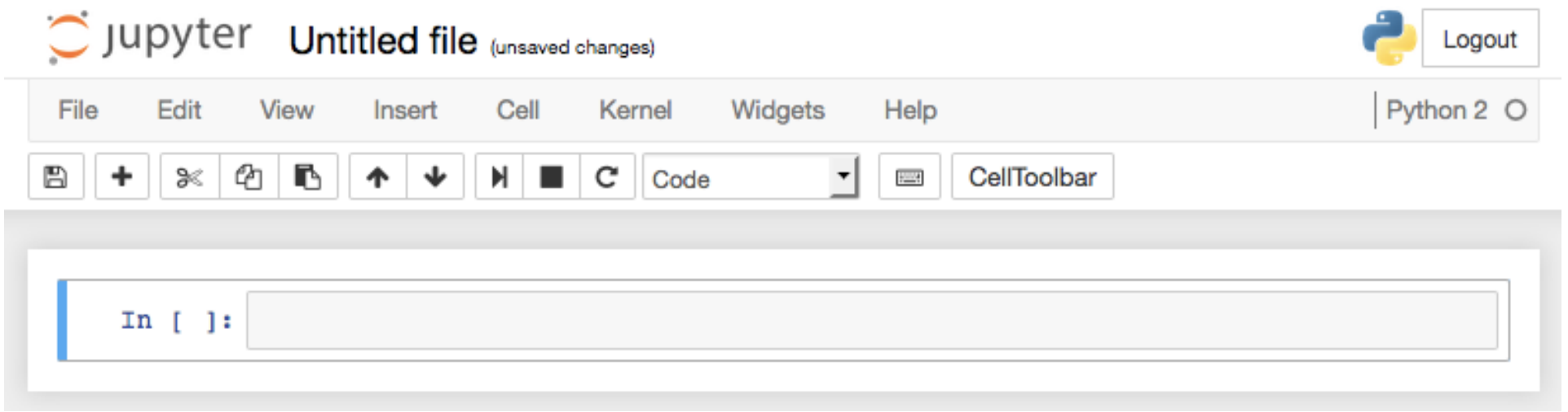
Simulate the detector response (promising alternative to traditional simulation solutions)



Tools and Frameworks

Jupyter notebooks

https://jupyter.org/



An open-source web application that allows to create and share code, plots, documents.

It offers **one single environment** for:

- code, comments on the code, data analysis + data visualisation
- as well as any additional context (e.g. text, formulas, even media files..)

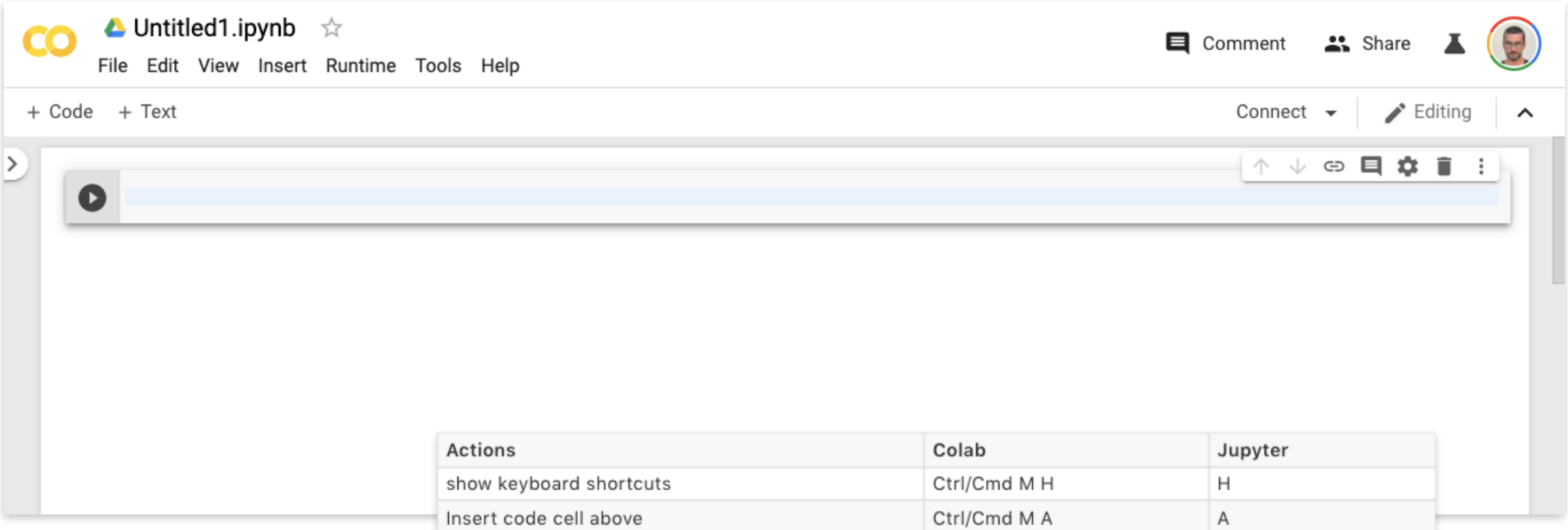
Perfect for **streamlining an entire workflow.**

And excellent **in the prototyping phase**.

Google Colab(oratory)

https://colab.research.google.com/

In a nutshell: **Jupyter notebooks on the cloud**.



Keyboard shortcuts:
Google Colab vs Jupyter

Actions	Colab	Jupyter
show keyboard shortcuts	Ctrl/Cmd M H	H
Insert code cell above	Ctrl/Cmd M A	A
Insert code cell below	Ctrl/Cmd M B	B
Delete cell/selection	Ctrl/Cmd M D	DD
Interrupt execution	Ctrl/Cmd M I	II
Convert to code cell	Ctrl/Cmd M Y	Y
Convert to text cell	Ctrl/Cmd M M	M
Split at cursor	Ctrl/Cmd M -	Ctrl Shift -

NumPy

NumPy

https://numpy.org/

A third-party package added to Python to support scientific computing

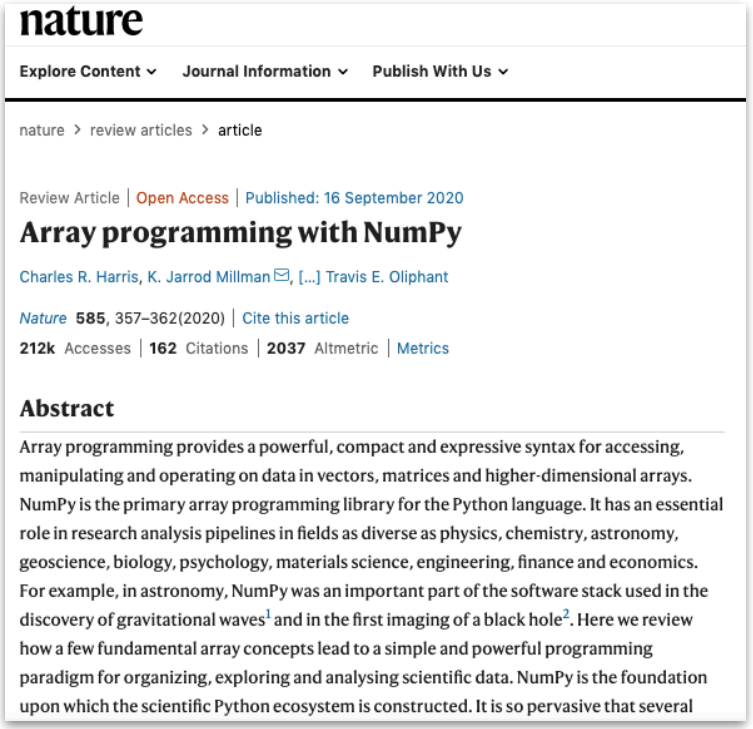
- in particular, it provides you with **multi-dimensional array objects**
  - i.e. support matrix manipulation, linear algebra, all operations you might want to do on large collection of numbers (e.g. plenty in ML!)

The NumPy paper on **Nature**

- https://www.nature.com/articles/s41586-020-2649-2

To be formally cited as:

- Harris, C.R., Millman, K.J., van der Walt, S.J. *et al.* Array programming with NumPy. *Nature* **585,** 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2



nature

Explore Content | Journal Information | Publish With Us

nature > review articles > article

Review Article | Open Access | Published: 16 September 2020

Array programming with NumPy

Charles R. Harris, K. Jarrod Millman, [...] Travis E. Oliphant

Nature **585**, 357–362(2020) | Cite this article

212k Accesses | **162** Citations | **2037** Altmetric | Metrics

Abstract

Array programming provides a powerful, compact and expressive syntax for accessing, manipulating and operating on data in vectors, matrices and higher-dimensional arrays. NumPy is the primary array programming library for the Python language. It has an essential role in research analysis pipelines in fields as diverse as physics, chemistry, astronomy, geoscience, biology, psychology, materials science, engineering, finance and economics. For example, in astronomy, NumPy was an important part of the software stack used in the discovery of gravitational waves[1] and in the first imaging of a black hole[2]. Here we review how a few fundamental array concepts lead to a simple and powerful programming paradigm for organizing, exploring and analysing scientific data. NumPy is the foundation upon which the scientific Python ecosystem is constructed. It is so pervasive that several

Pandas

https://pandas.pydata.org/

Pandas is an open source library providing **high-performance, easy-to-use data structures and data analysis tools** for Python.

Matplotlib

https://matplotlib.org/

A package that offers a huge range of predefined functions to plot and **visualise your data**.

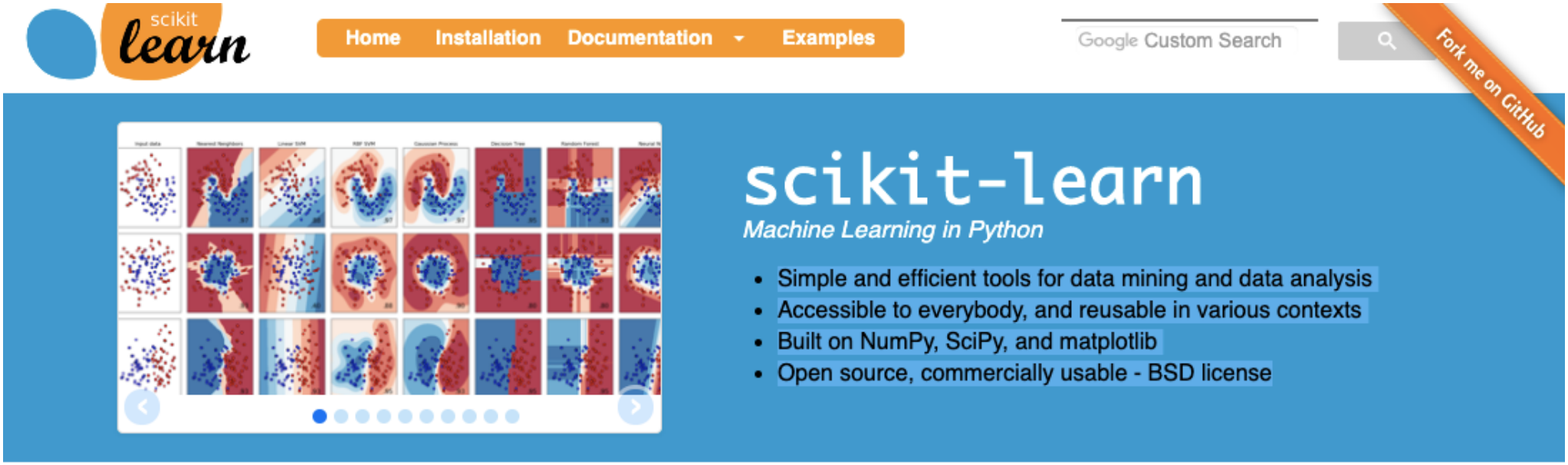
ML tools and frameworks

Most prominent ones:

- Sklearn, Keras/Tensorflow, Pytorch(+fast.ai), …

Scikit-learn

machine learning in Python



The first labs will make a large use of scikit-learn!

Tensorflow and Keras

Adopt something that provides you with a modern description, implementation and application of learning algorithms, including neural networks (of course!)

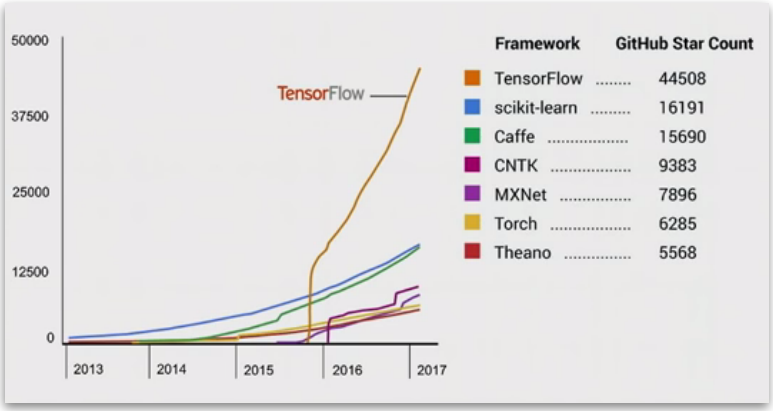
a good backend choice

TensorFlow: Low-level implementation of operations needed to implement (e.g.) neural networks in multi-threaded CPU and multi GPU environments (*basically, all this.. transparently!*)

Keras: High-level convenience wrapper for backend libraries, e.g. TensorFlow, to implement neural network models

a good high-level library, quite handy on top of (e.g.) Tensorflow

Both quite popular, and widely adopted



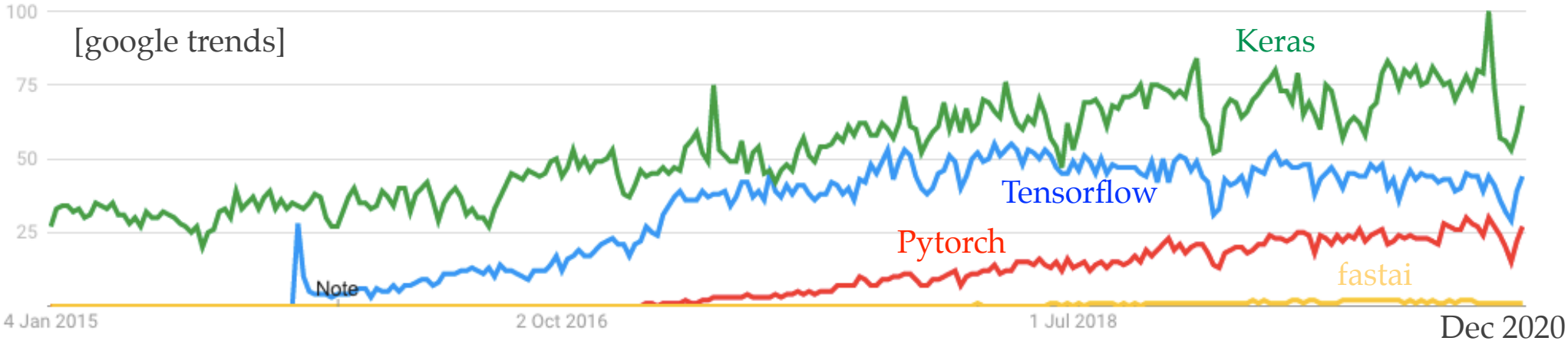
Bump in 2015 as TF became public

Today:

Interest over time

Numbers represent search interest relative to the highest point on the chart for the given region and time. A value of 100 is the peak popularity for the term. A value of 50 means that the term is half as popular. A score of 0 means that there was not enough data for this term.

[Disclaimer: plenty of caveats in such comparisons..]



Contacts

daniele.bonacorsi@unibo.it

daniele.bonacorsi

@DBonacorsi

Thanks for your attention,

please follow Stefano (next) on more details on NNs, and

enjoy the ML_INFN hackaton!