

Machine Learning workshop

Prof. Daniele Bonacorsi



ISGC 2021



March 22, 2021



Logistics

Plan of the day

The day is dedicated to beginner/intermediate-level ML users.

09:00 - 10:30 **Basic ML concepts and tools** + start with Labs

10:30 - 11:00 — break —

11:00 - 12:30 Lab on **Regression**

12:30 - 13:30 — lunch break —

13:30 - 15:00 Lab on **Classification** + Lab on **CNN**

15:30 - 15:30 — break —

15:30 - 17:00 Lab on **AE** + Lab on **GAN**

All through slides + example application with code

- follow at your pace (i.e. code in real time, or just follow and exercise later)
 - ❖ you only need a browser and a network connection
- all slides and code is available for you (see next)

Material

< a link to the material will appear here (during the workshop)>

LOOK FOR THIS ARROW IN THE SLIDES
(this means there is code for you to play with the concepts!)





Definition of ML

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)*

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..)*

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”

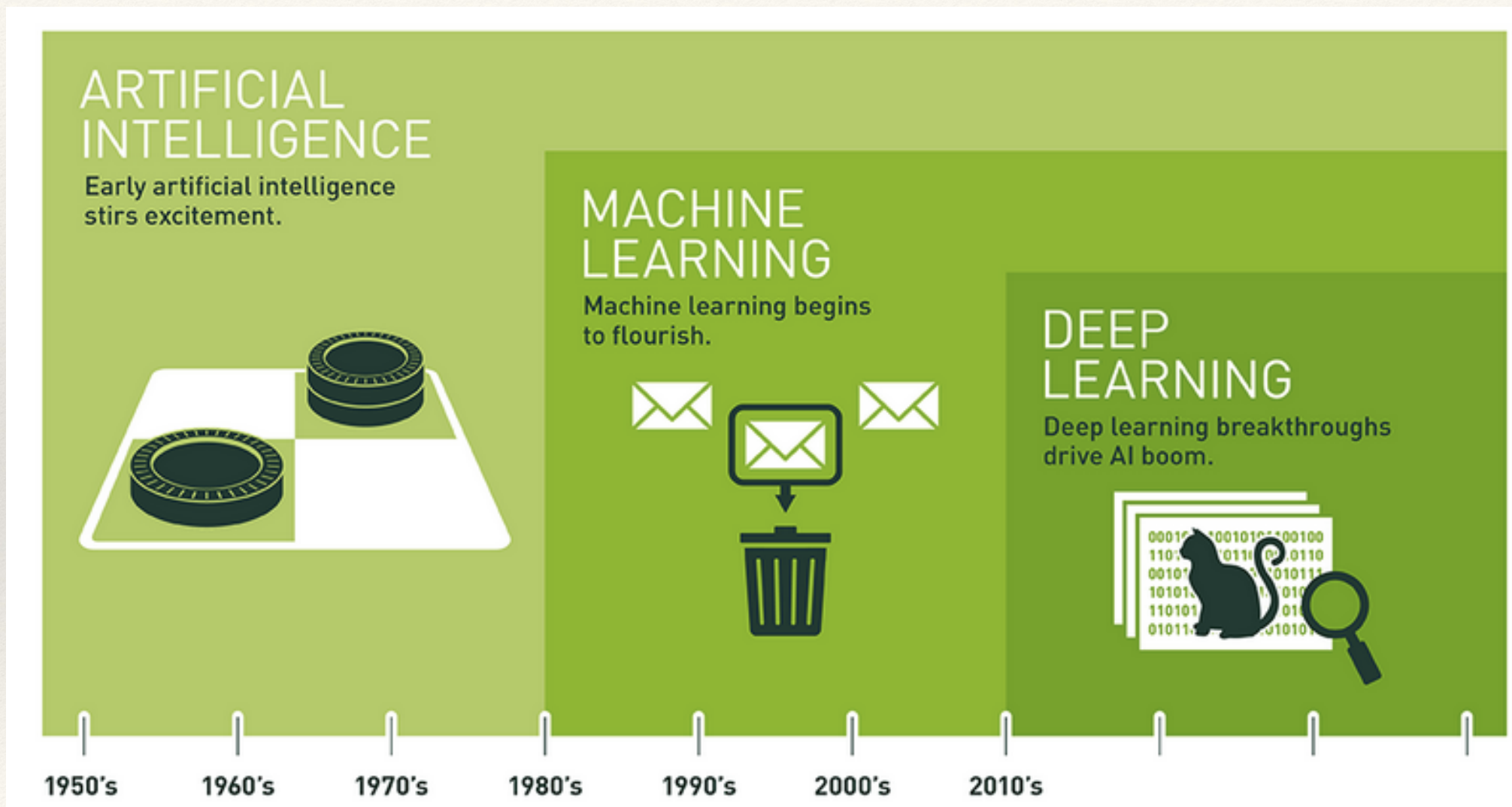
– *Arthur Samuel (1959), author of the Samuel Checkers-playing Program (and some TeX..)*

“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

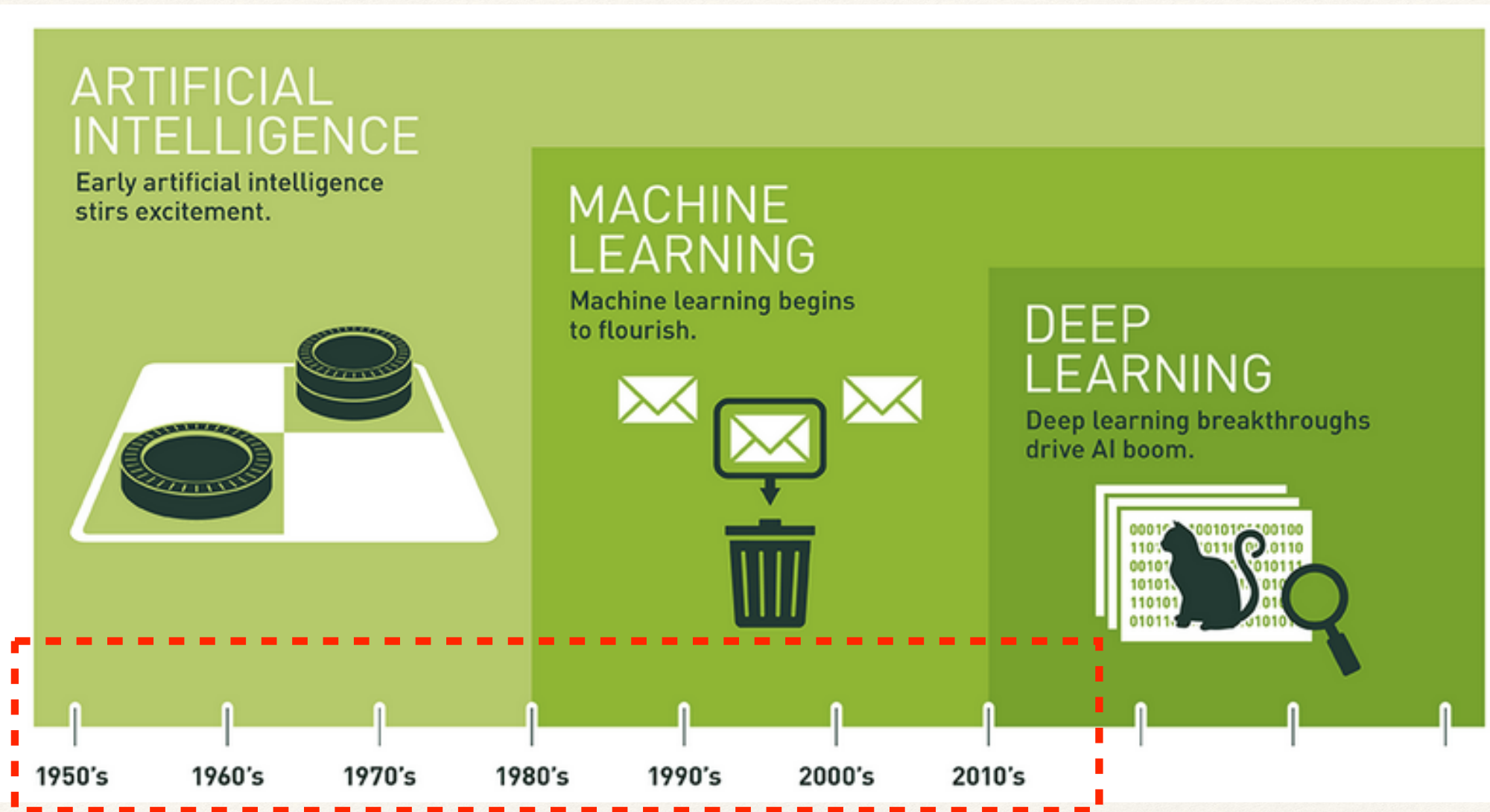
A pictorial definition (by Nvidia)



[NVI1]

Visual definition of ML

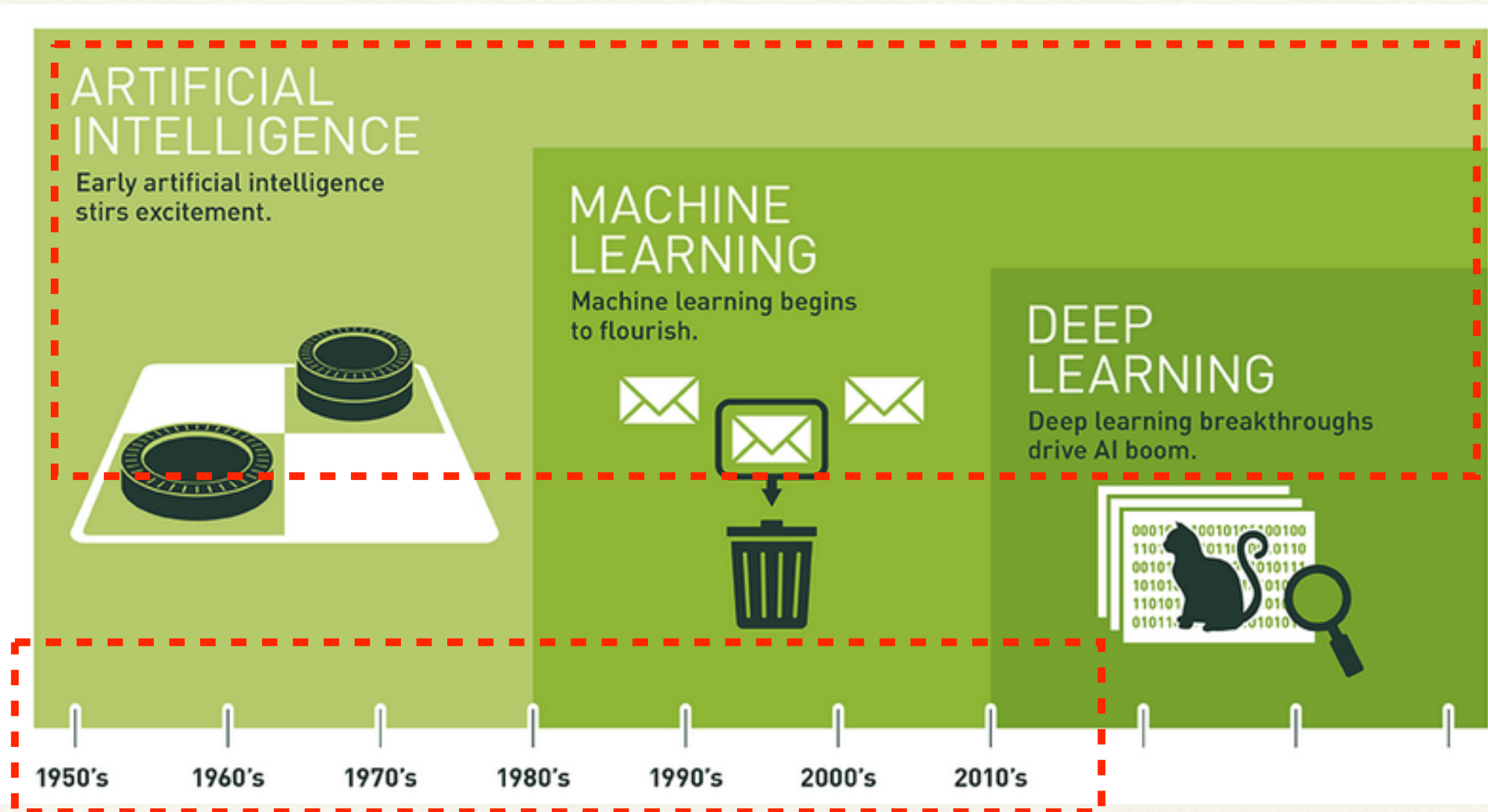
A pictorial definition (by Nvidia)



[NVI1]

Visual definition of ML

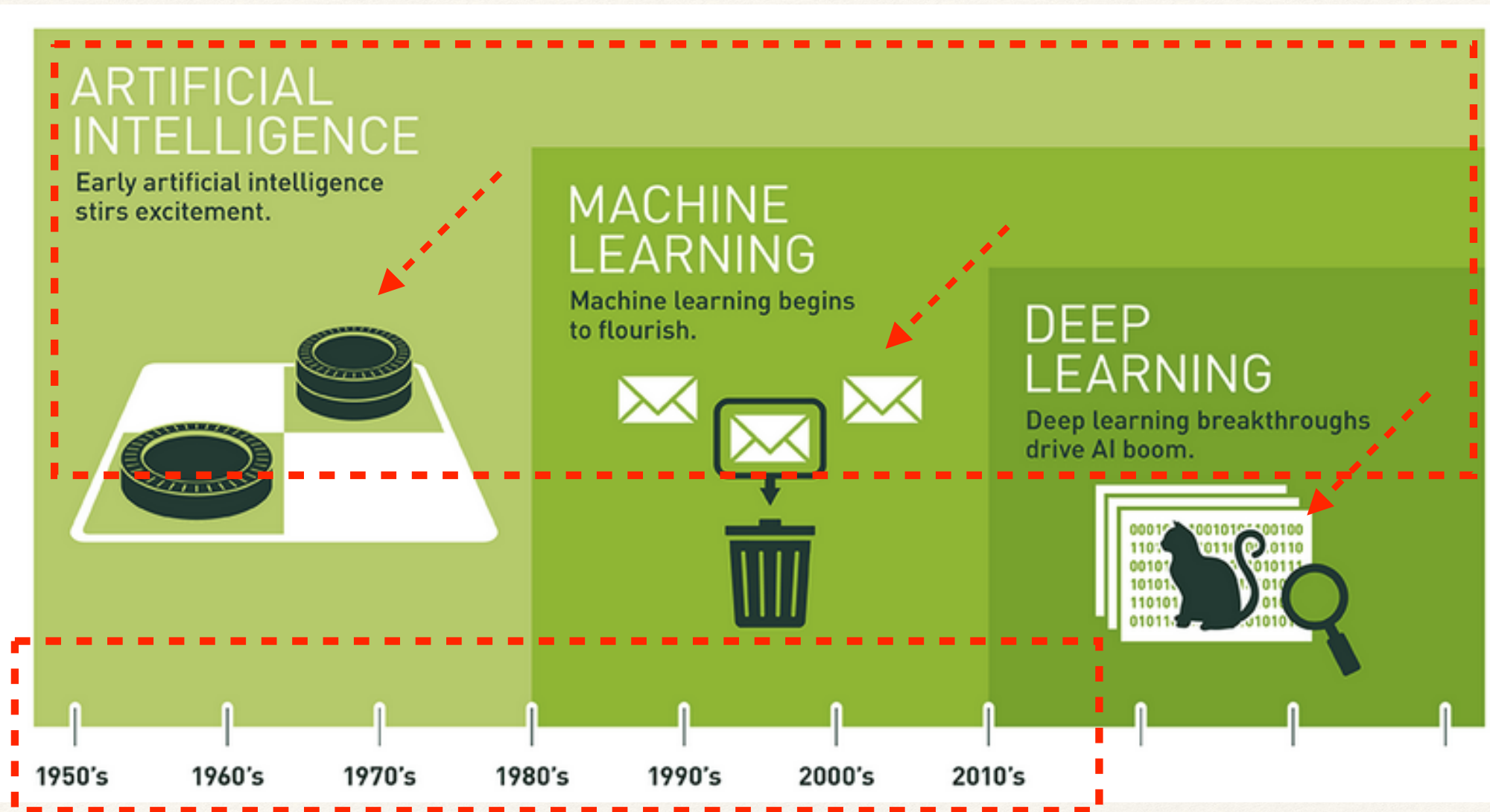
A pictorial definition (by Nvidia)



[NVI1]

Visual definition of ML

A pictorial definition (by Nvidia)



[NVI1]

"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, at all

Artificial Intelligence ?

"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, at all

~~Artificial Intelligence~~

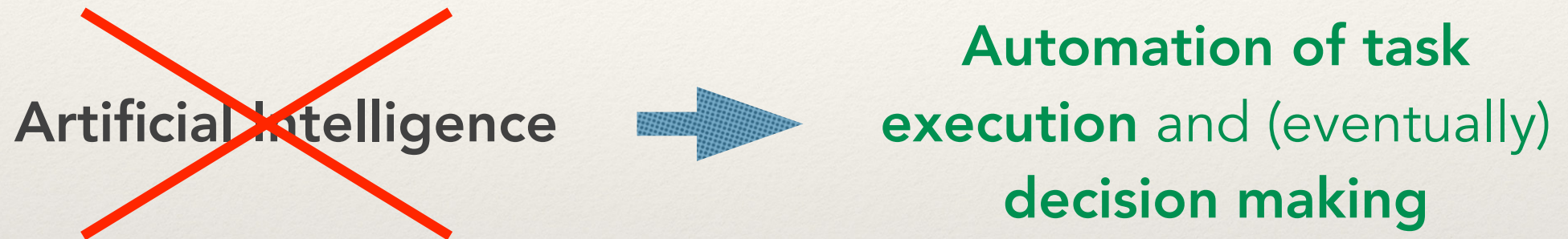


**Automation of task
execution and (eventually)
decision making**

"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, at all



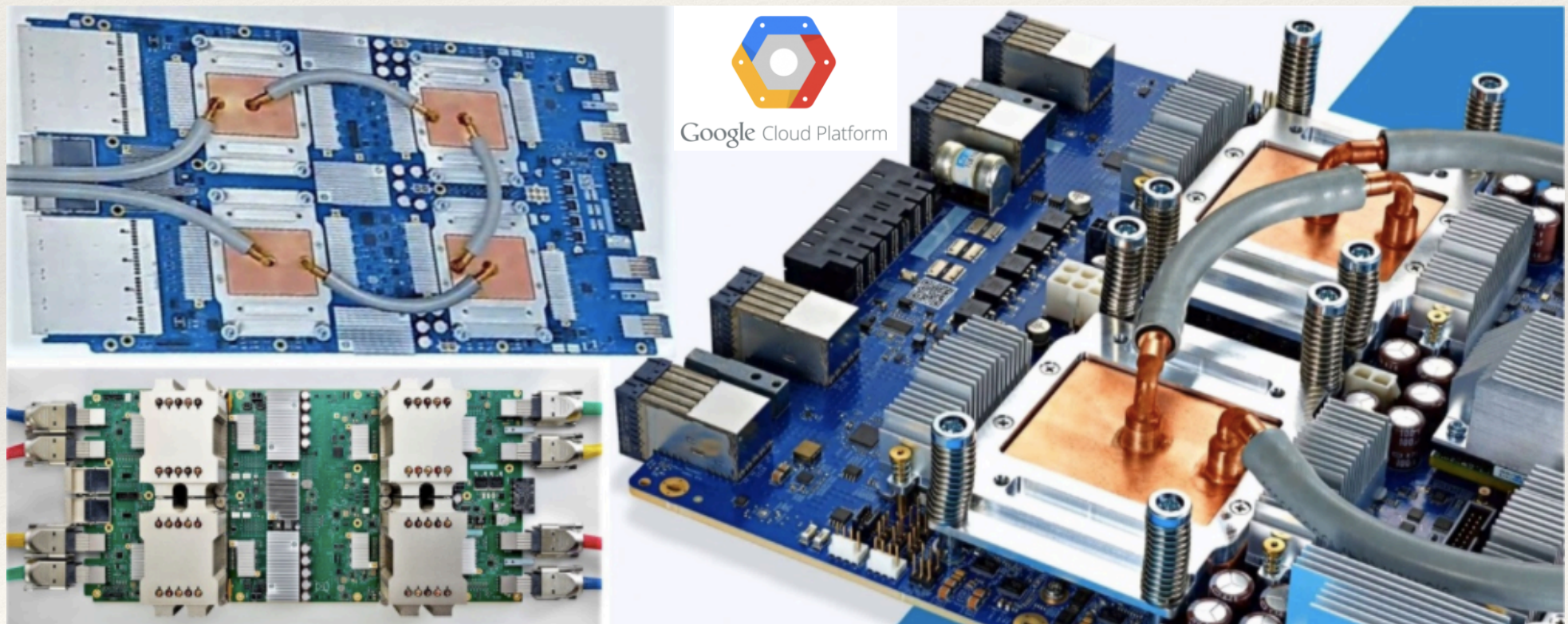
It is aiming at collecting data around how humans make decisions, to perform the same tasks at a scale (LARGE) and latency (SMALL) that are not humanly possible

- Example: facial recognition.

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

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

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



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

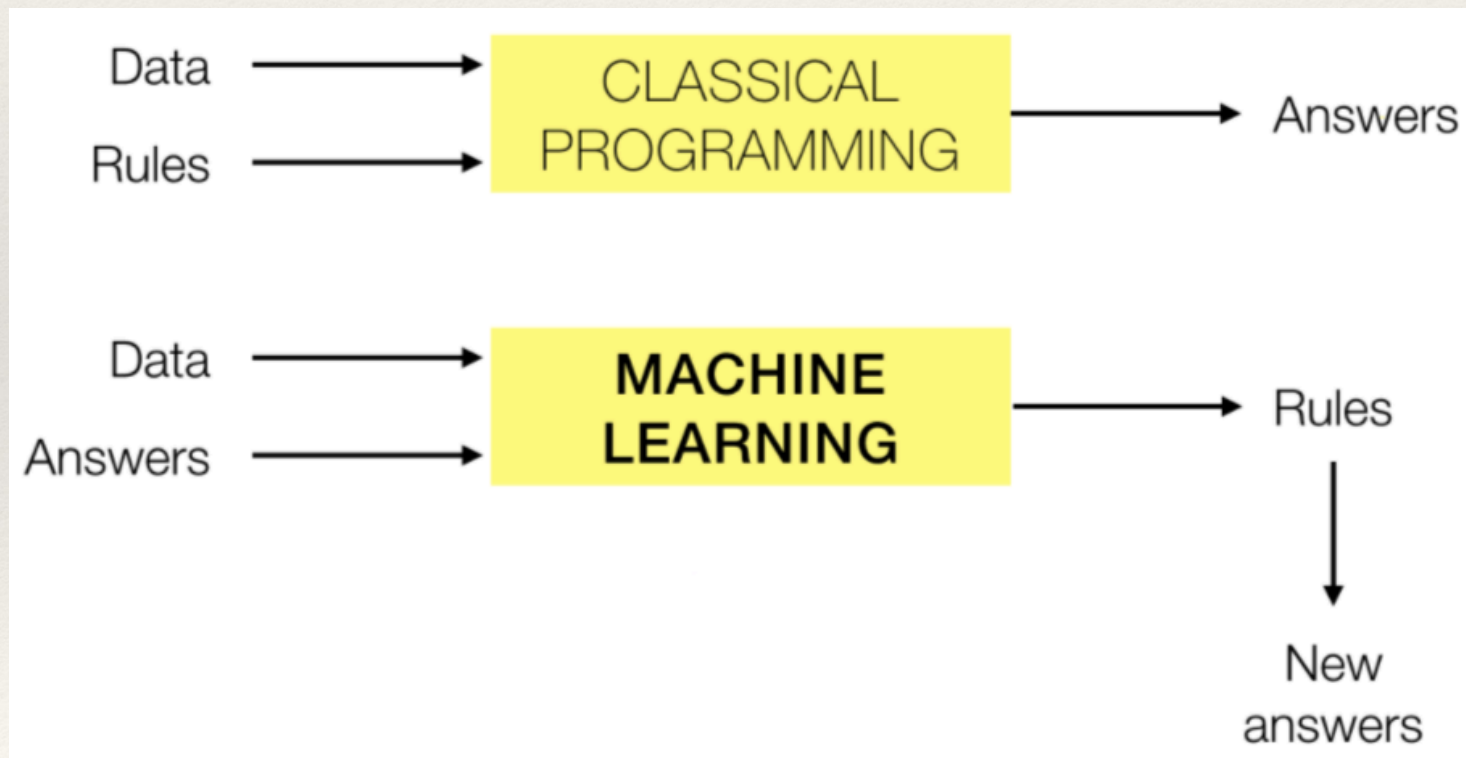
[*] as most of these resources world-wide are far from being free-of-charge, and Big Data companies implement carefully designed business models, this is debatably “democratic” in a social sense. Not discussed here further, though: here we aim at stating that “in principle” you have a “pay-and-access” option on not-on-premise resources, which did not exist before.

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

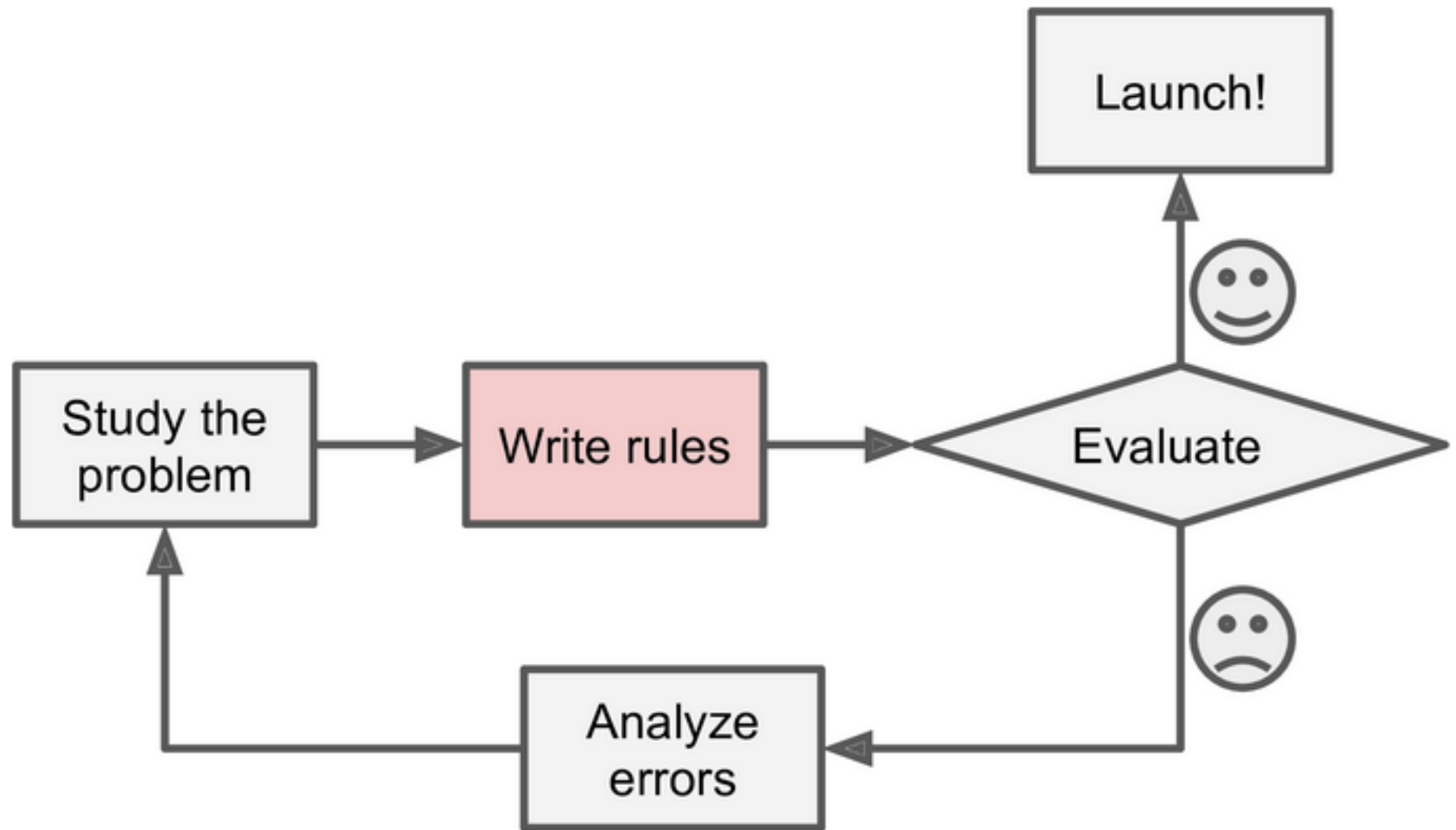
Another (last definition)

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

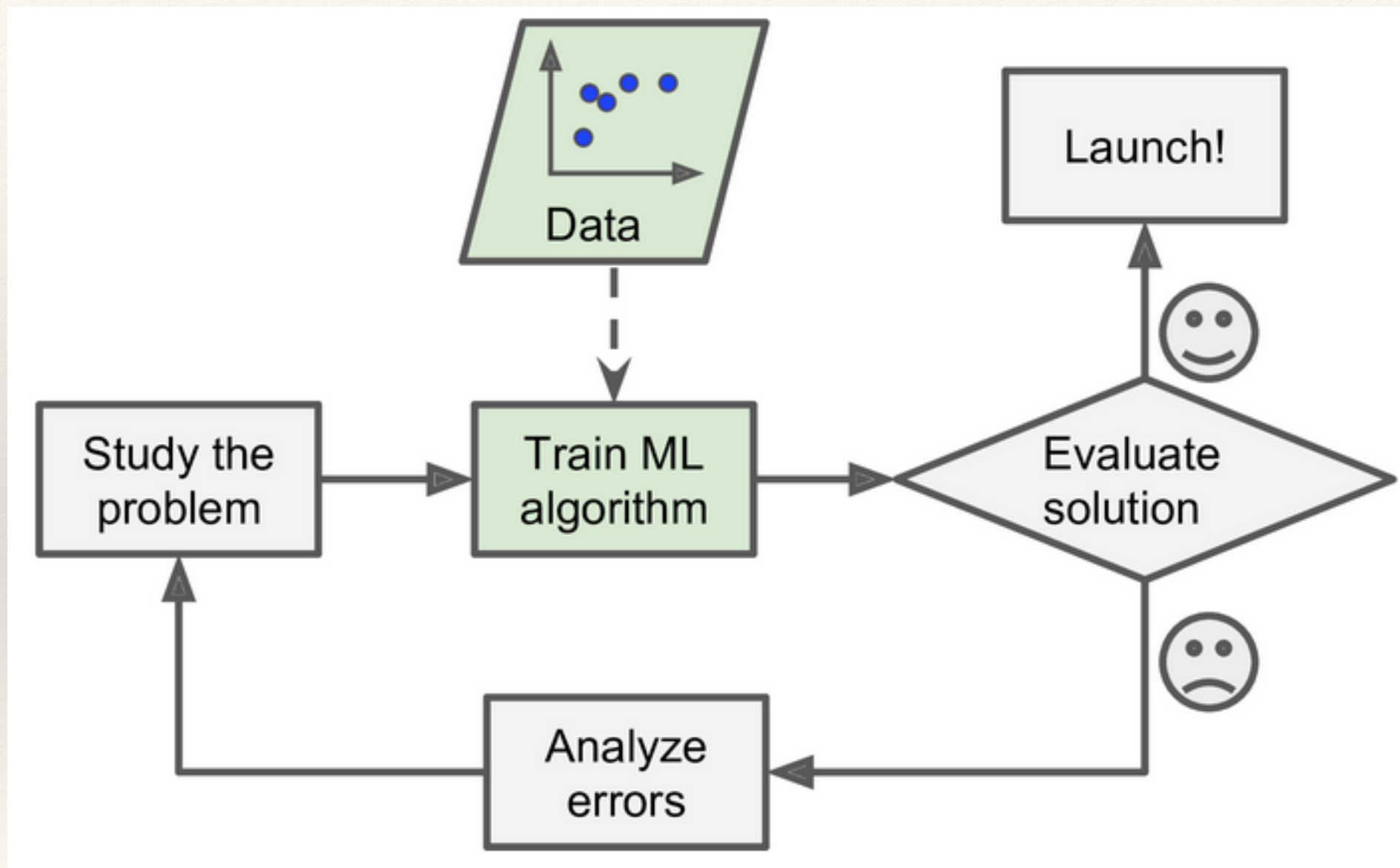
– F. Chollet (author of Keras)



Traditional approach

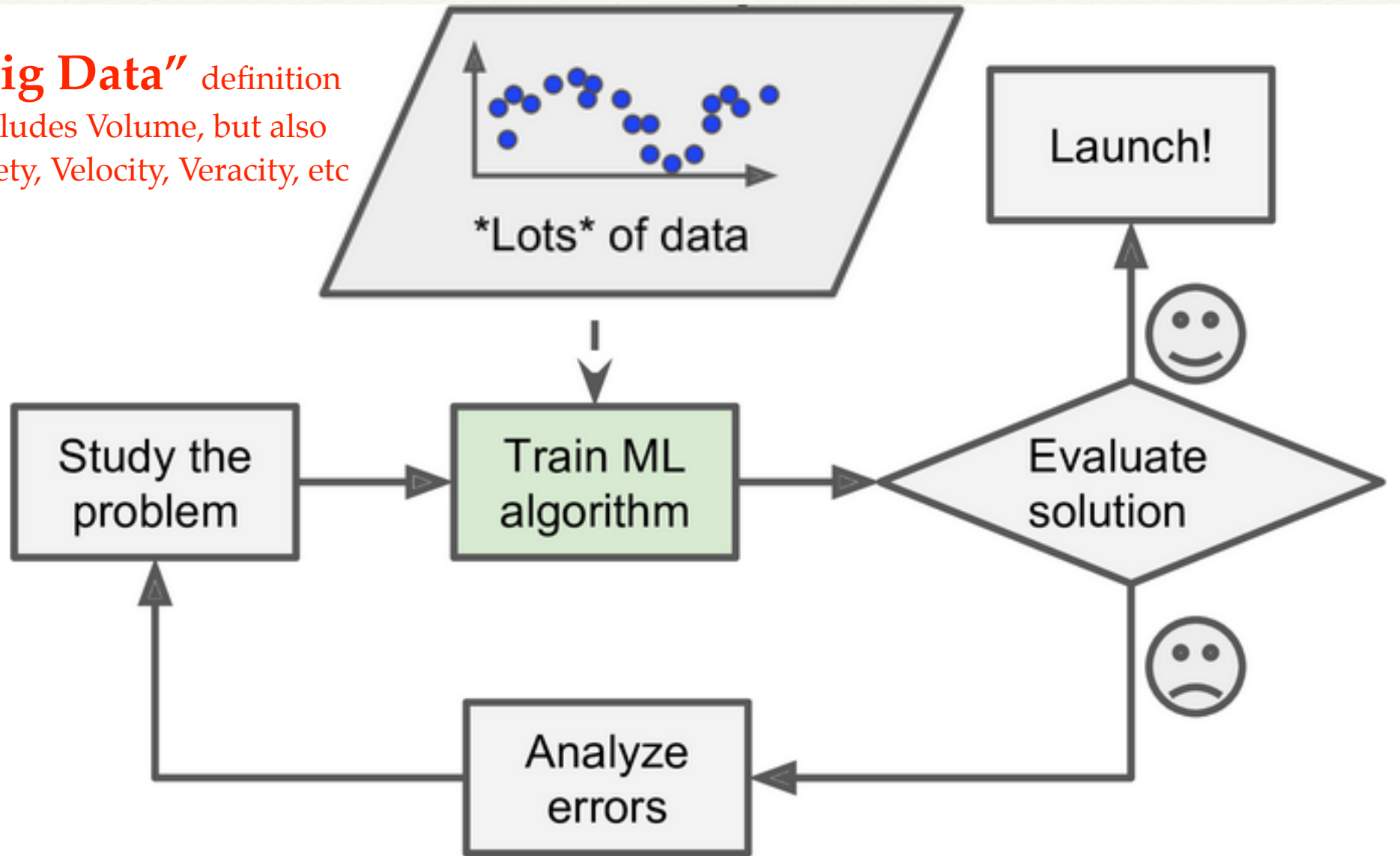


ML approach: data-driven modelling



ML approach: Big Data-driven modelling

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





Types of ML

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
- **unsupervised** learning
- **reinforcement** 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
- **reinforcement** 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

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

Possible classifications of ML methods

Classify ML in **broad categories** based on:

- **the amount and type of supervision** during model creation (aka “**training**”):
 - ❖ **Supervised, Unsupervised, (Semisupervised), Reinforcement Learning**
- whether or not they can **learn incrementally on the fly**
 - ❖ **online** learning versus **batch** learning
- whether they work by simply **comparing new data points to known data points**, or instead **detect patterns** in the training data and build a predictive model (much like scientists do)
 - ❖ **instance-based** versus **model-based** learning

These criteria are not exclusive; you can combine them in any way

- e.g. a state-of-the-art spam filter may learn on the fly using a deep neural network model trained using examples of spam and good mails, which makes it an *online, model-based, supervised learning system*

Let's look at key concepts of each.

The algorithms zoo

We need to classify problems and methods, select, and choose..

- Alternative: get into each ML algorithm! (impossible in one day!)



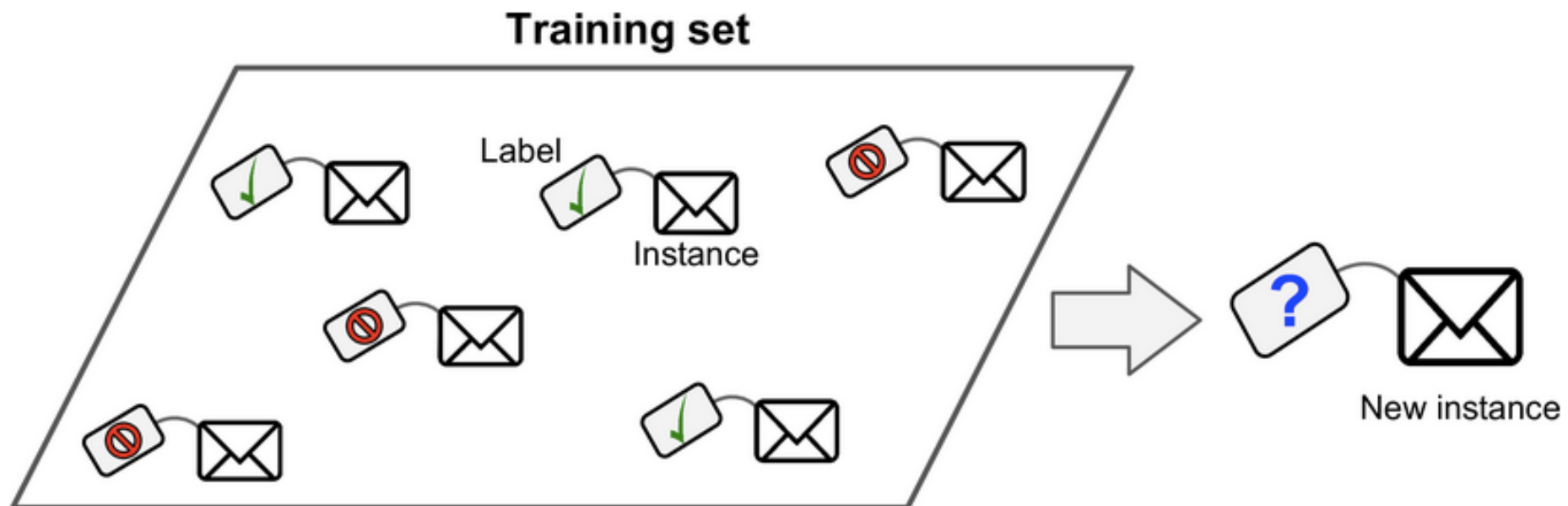
Supervised, Unsupervised, Reinforcement Learning

Batch learning versus **Online** learning

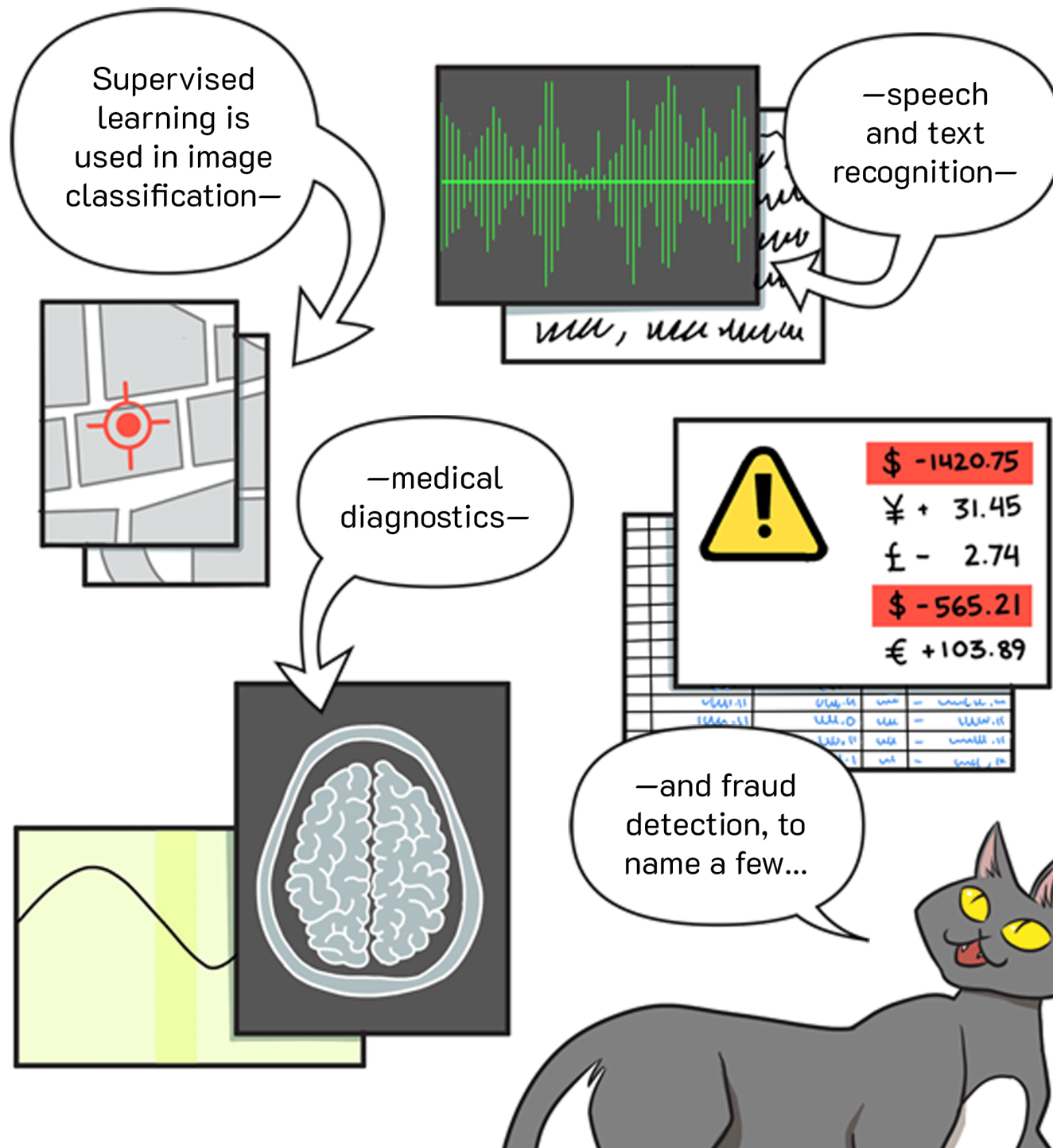
Instance-based versus **Model-based** learning

In **supervised learning**, the data entries ("**instances**" or "**examples**") you feed to the algo 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:



Another (one of my favourite) example is a child learning to recognise car vs bus vs bicycle in traffic with a parent guiding her/him



Supervised ML: 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. spam filter: trained with many example emails along with their class labels ("spam" or "not-spam"), it learns how to classify new coming emails*

Another typical supervised learning task is **regression**

- predict "**target numeric values**" (in a continuum of values)
- *e.g. a price of a house, knowing its attributes, and being given plenty of instances of other houses (both their features and price)*

Note: regression algorithms exist that can be used for classification as well, and vice versa

- *e.g. Logistic Regression is commonly used for classification, as it can output a value that corresponds to the probability of belonging to a given class (e.g., 20% chance of being spam)*

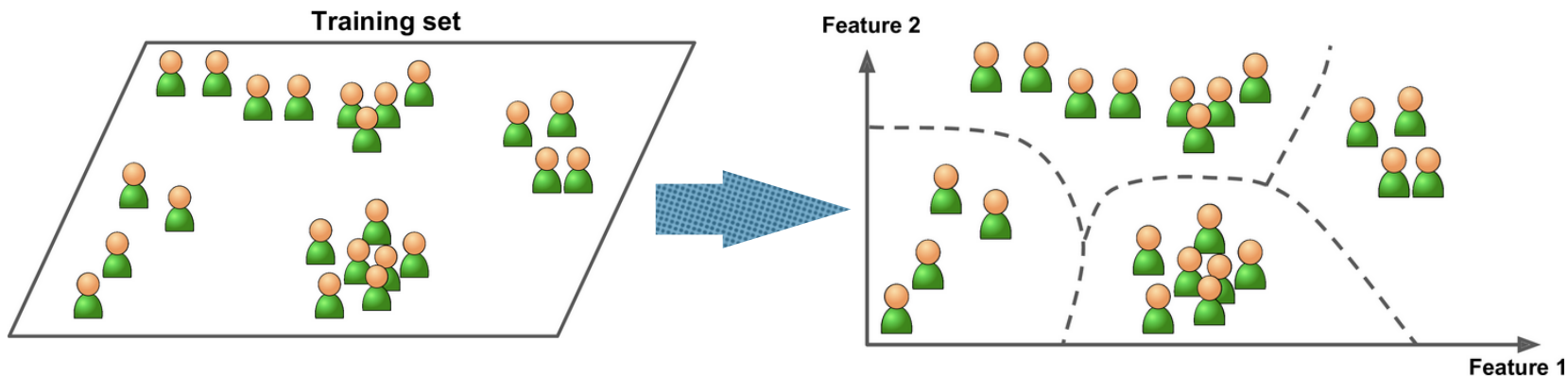
Supervised ML algos

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, such as AutoEncoders and restricted Boltzmann machines. They can also be semisupervised, such as in deep belief networks and unsupervised pretraining... devil is in the details!
- ...

Example of “unsupervised ML”: clustering

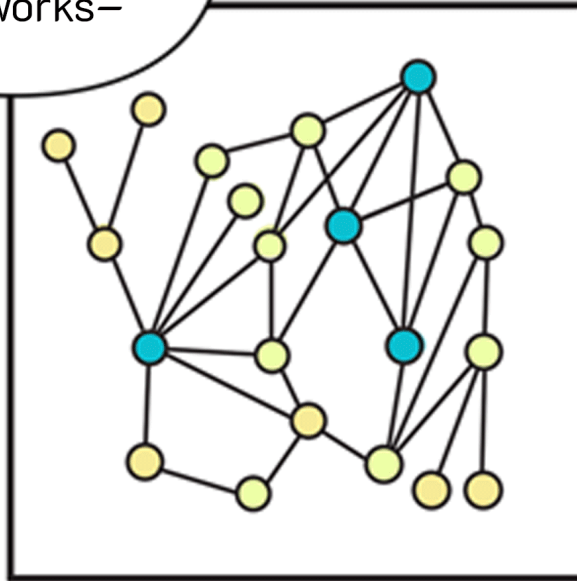
In **unsupervised learning** the training data is **unlabeled**, so the system tries to learn without a teacher guiding it



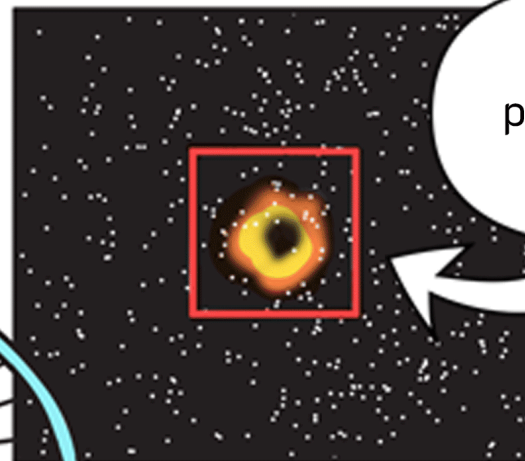
Example: data about blog readers

- run an unsupervised (e.g. 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.
- with unsupervised (hierarchical clustering) algos, you may detect subgroups, etc

Unsupervised
can help us
visualize
networks—



—make personal
connections—



—or find
patterns in
nature...



Other examples of unsupervised ML

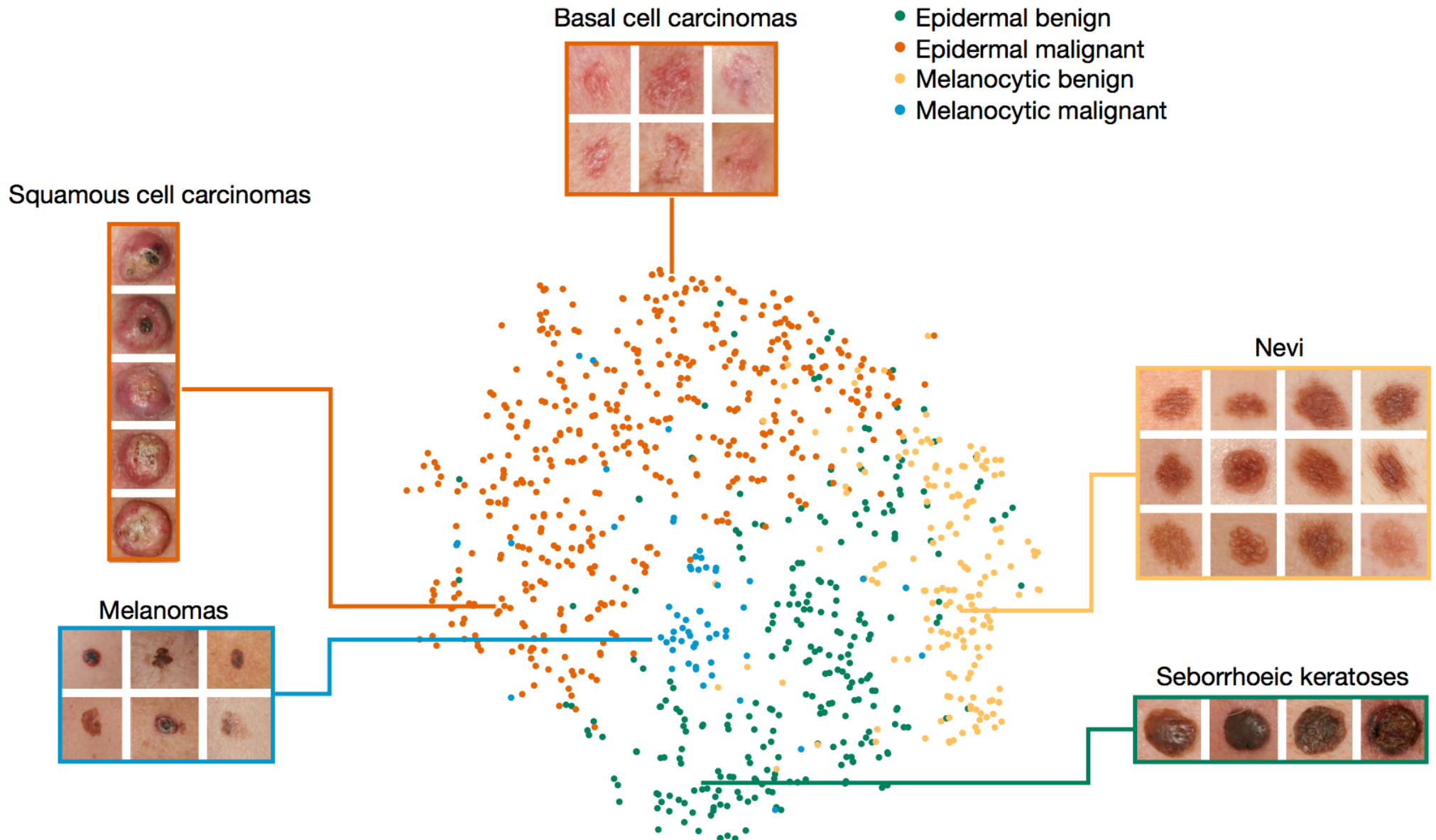
Example: used to **organise large computer clusters**, trying to figure out which machines tend to “work together”: if set-up takes this into account, the data centre works more efficiently.

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.

Unsupervised ML for medical diagnosis

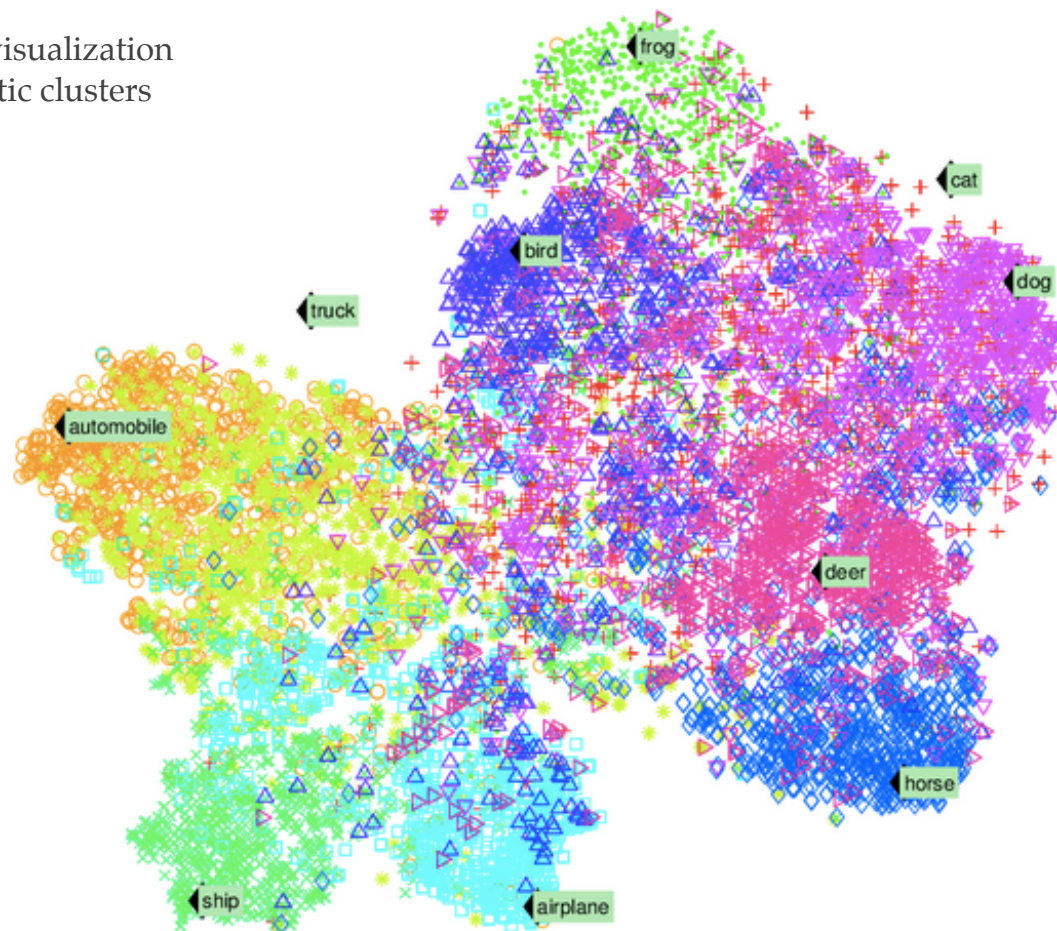
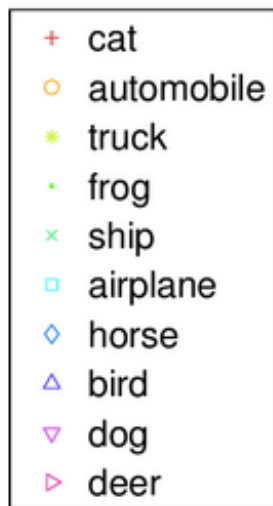


Unsupervised ML: **visualisation**

Data visualisation is also a customer of unsupervised learning algos

- feed visualisation algos with a lot of complex and unlabeled data, and they output a 2D or 3D representation of your data that can easily be plotted

Example of a t-SNE visualization
highlighting semantic clusters



Unsupervised ML algos

Most commonly used / important:

Clustering → try to detect groups

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

Visualisation and Dimensionality Reduction → display / simplify data w/o losing too much info

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

Association rule learning → discover interesting relations between attributes in large datasets

- Apriori
- Eclat

Example of Reinforcement Learning

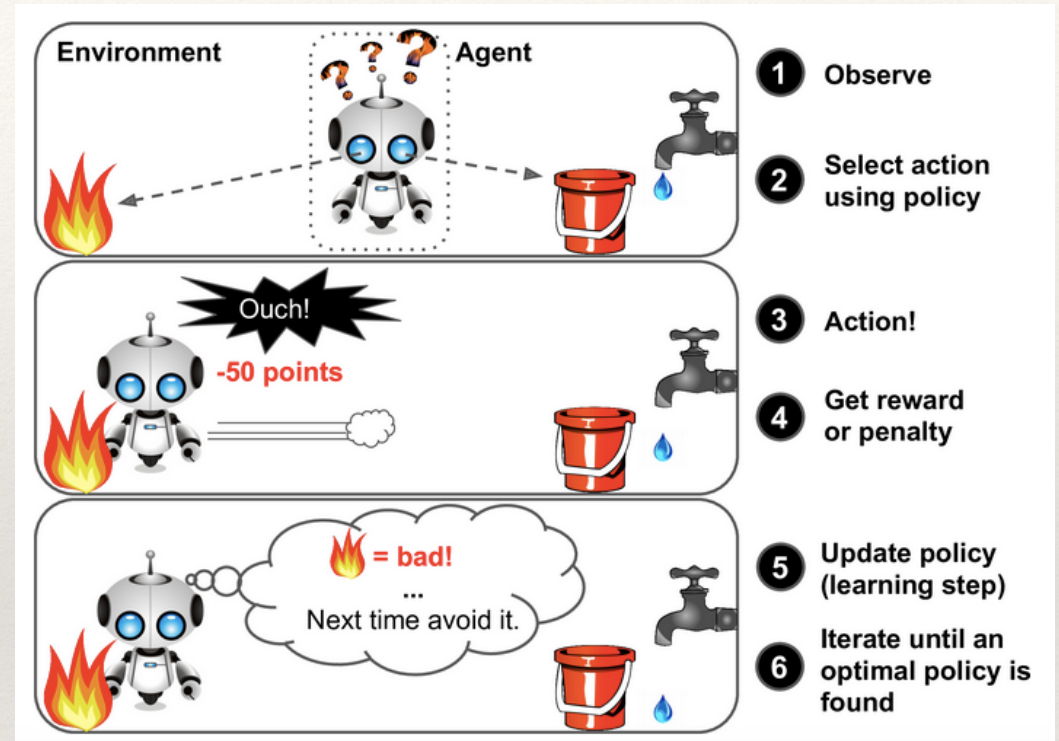
Reinforcement learning
using experience replay
for the robotic goalkeeper

Initial trials: bad performance

Reinforcement Learning (RL)

Reinforcement Learning is a completely different beast

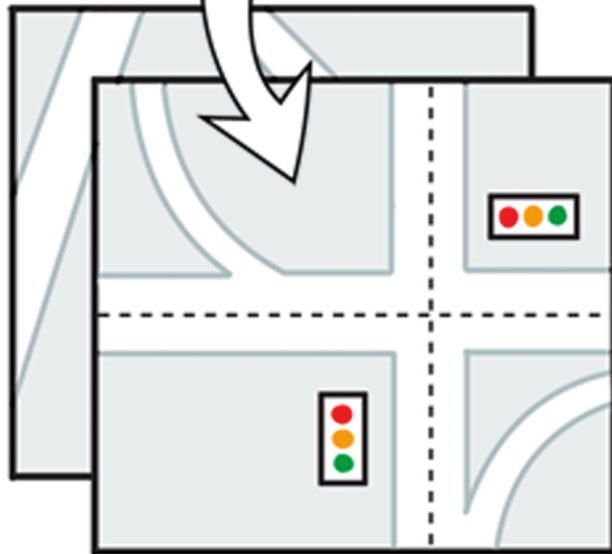
- The learning system is called "**agent**" in this context
- it **observes** the environment
- select and perform **actions**
- get positive **rewards** or negative rewards (i.e. **penalties**) in return
- learning step is define the best **policy** to get the most reward over time



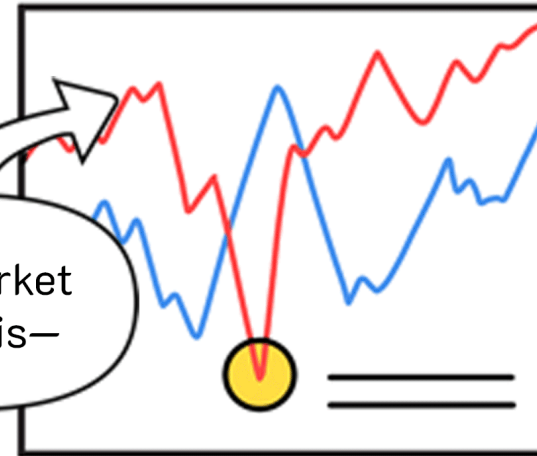
Examples:

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

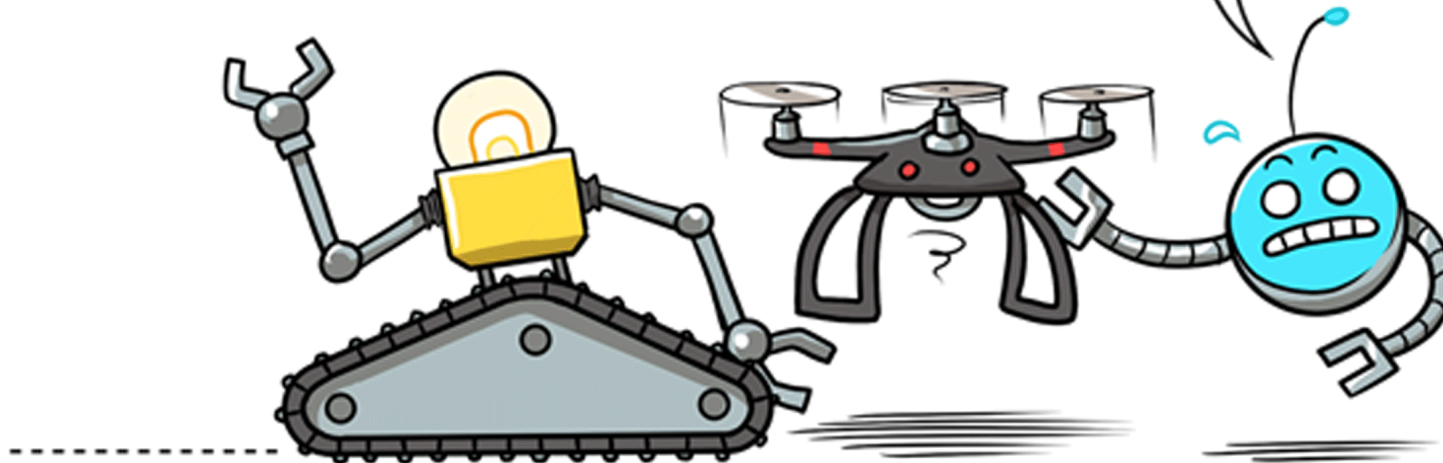
And **reinforcement learning** could be applied to everything from traffic management—



—to market analysis—



—to robot navigation that won't look "OMG so wrong" on YouTube.



Supervised, Unsupervised, Reinforcement Learning

Batch learning versus **Online** learning

Instance-based versus Model-based learning

Batch learning vs Online learning

Classification of ML algos based on **whether or not the system can learn incrementally from a stream of incoming data**

Batch learning

In **batch learning**, the system is incapable of learning incrementally

- it must be trained **using all the available data** (i.e. **offline learning**, in contrast with “online”)
- **more data?** stop, re-train (on old+new data), create new model, refine it, switch to it and abandon the old one, launch the new in production
 - ❖ “new data weekly” mode vs “rapidly changing data” (e.g. aiming to predict stock markets?)
- it can be resource hungry (i.e. time and computing)
 - ❖ CPU, memory space, disk space, disk I/O, network I/O, etc.

When do you hit an unsurmountable limit with batch learning?

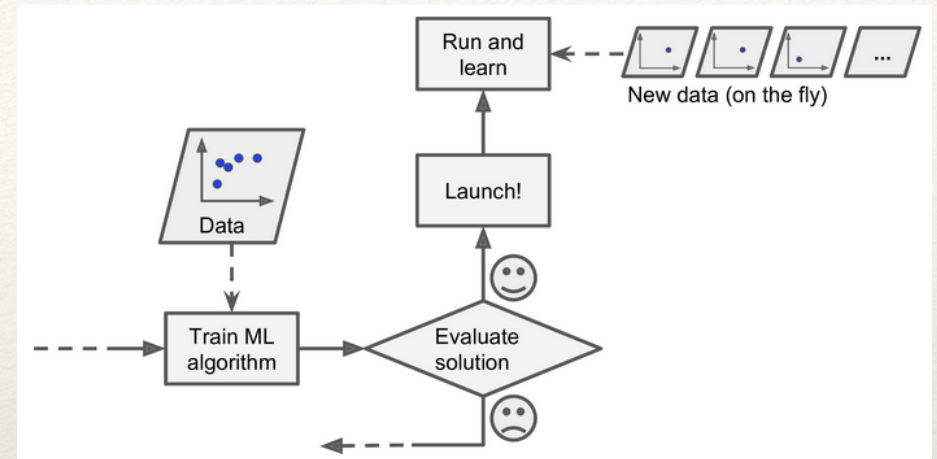
- **Big (Volume-wise) data + automation** requirements → requires plenty of money!
- same as above + **limited resources** (e.g. smartphone app, or rover on Mars..) → might be even impossible!

Solutions here might come from using different algos, that are capable of learning incrementally..

Online learning

In **online learning**, you **train the system incrementally**

- by feeding it data instances sequentially
 - ❖ either individually or by small groups called “mini-batches”
- **each learning step is fast and cheap**, so the system can learn about new data on the fly, as it arrives



Points of strength:

- perfect for systems that receive **data as a continuous flow**
 - ❖ and need to adapt to change rapidly or autonomously
- good option if you have **limited computing resources**
 - ❖ once an online learning system has learned about new data instances, you can discard them
 - ❖ Good choice to train systems on huge datasets that cannot fit in one machine's main memory (aka “out-of-core learning”): the algo loads part of the data, runs a training step on that data, and repeats the process until it has run on all of the data

Supervised, Unsupervised, Reinforcement Learning

Batch learning versus Online learning

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

Instance-based vs Model-based Learning

Classification of ML algos based on **how they generalise**.

Note that “**generalisation**” is key to success of a ML system

- data → training → ability to make predictions on previously unseen data
- key of applied ML (and all its art!) is to perform well on new instances!

The 2 main approaches to generalisation are:

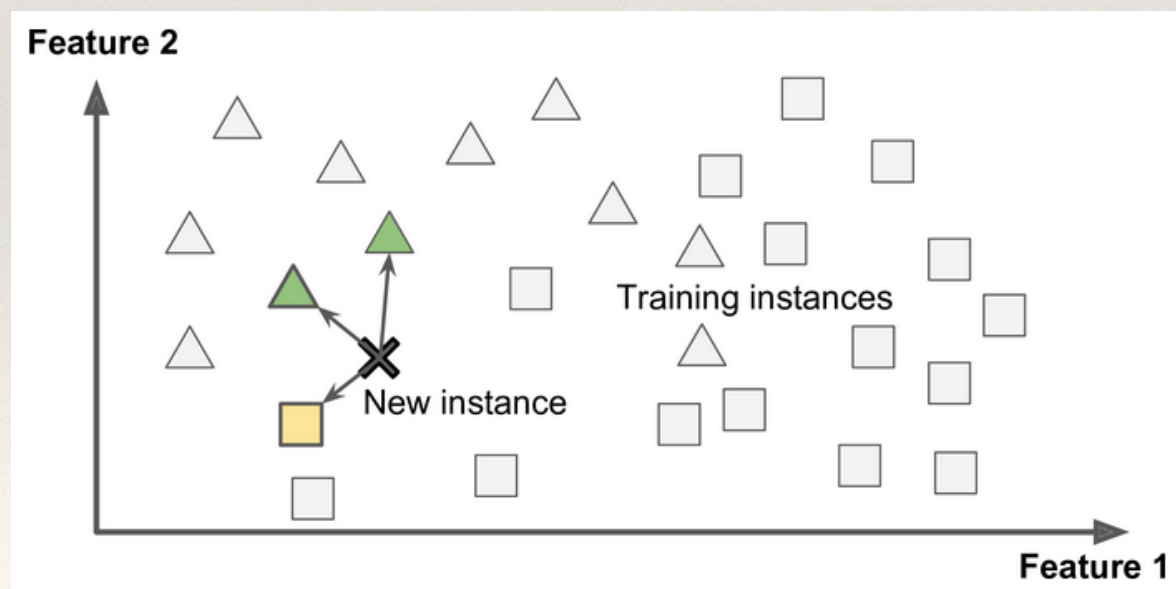
- **Instance-based Learning**
- **Model-based Learning**

Instance-based Learning

The most trivial form of learning is simply to learn by similarity

- with $[A, B]$ as options, classify an instance as $[A]$ if that instance is similar to a previous one in the training sample that had an $[A]$ label
- of course, this requires a “measure of similarity” between instances

This is **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.



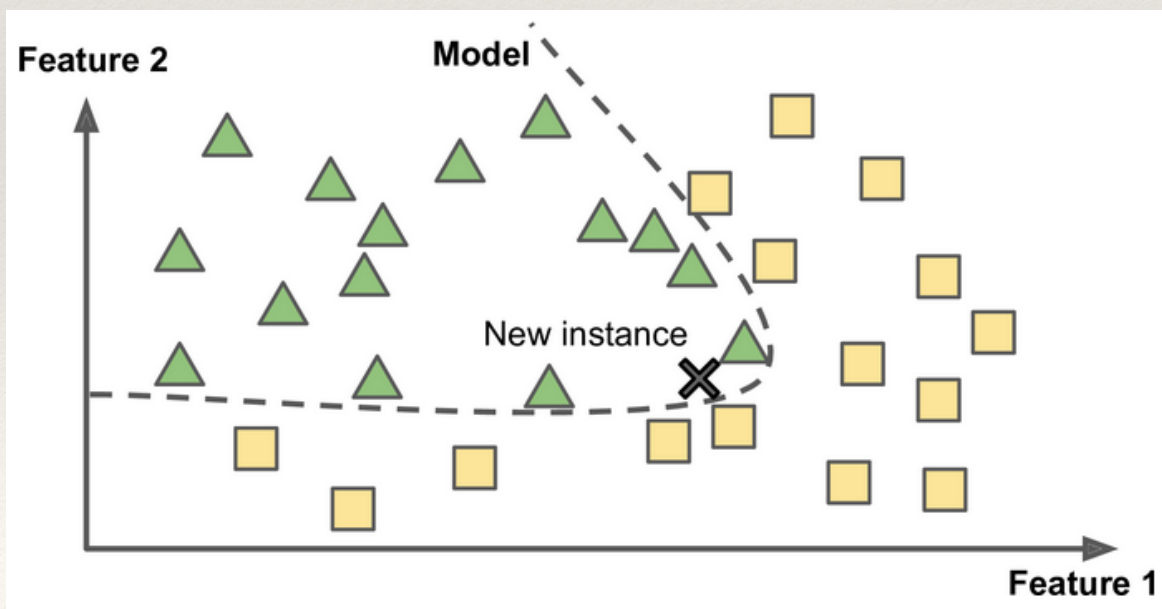
The new instance will be classified as a triangle **because most of most similar** (close-by in the features space) **training instances belong to that class**

Model-based Learning

In **model-based learning**, instead, a “model” for the training examples is built, then it is used to make predictions

- a completely different way to generalise from a set of examples

You rely on a **model selection**, i.e. you reason on your data and make an **hypothesis** as of how each data **features** contribute to its **label**, and apply such model to make predictions



The new instance will be classified as a triangle **because a model exists that explain why triangles do populate that portion of the features space**



Challenges in ML

Main challenges in ML

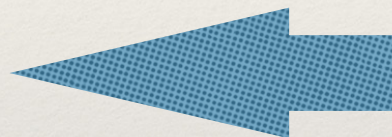
Basically, related to:

- bad data
- bad algos
- training technicalities

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: *how does a baby learn to spot car vs bus vs motorbikes?*

- try, dad gives labels, try, dad corrects, try, try, try, ...
- you need traffic! quite examples, i.e. many cars, busses, motorbikes..

ML is like a baby learning as above, but way more stupid!

- well.. it can do in seconds what a baby takes months to learn..
- .. but needs **PLENTY** of examples to learn and perform in even simpler tasks
 - ❖ (and at the same time the baby has learnt a lot of other things, btw)

Needs for large volumes of data (**the Volume "V" of Big Data..**) is a NEED for well performing Machine/Deep Learning applications

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

Bad data

Caution: not talking about the quality of the data (that comes later)..

.. but about their **representativeness** of the problem such data is expected to describe

- e.g. are they complete? Example: YouTube search engine favours popularity..

Crucial to **use a training set that is representative of the problem you want to describe and eventually generalise to**

- if the sample is too small → “sampling noise” (i.e. non-representative data as a result of chance). Even if the sample is very large, if the sampling method is flawed it will still be non-representative. This is called “**sampling bias**”

Main challenges in ML: Poor-Quality Data

Bad data

ML systems are “**garbage in garbage out**”

If your training data is full of errors, outliers, noise (e.g. due to poor-quality measurements), the system will find it hard to detect any underlying patterns, so less likely to achieve decent performances

It is worth the effort to spend time cleaning up your training data. Best data science teams spend here a significant portion of their overall time on a ML project.

E.g. some instances missing a few features

- ignore the attribute altogether
- ignore the instances
- fill in the missing features (e.g. median): bias?
- train one model with the attribute and one model without it



[credits: xkcd.com]

Main challenges in ML: Irrelevant Features

Bad data

Again, ML systems are “garbage in garbage out”. But here, focus is on **features** and the **learning process** itself

Success in a ML project largely depends on your ability to feed a training process with features in your data that enable an effective learning process

- please “help the ML system to help you”! ©

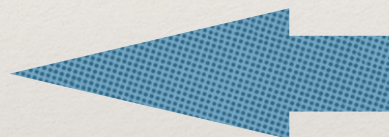
This process is important mainly in non-NN ML models, and it includes the so-called “**feature engineering**” at large, involving:

- **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** by gathering new data

Main challenges in ML

Basically, related to:

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



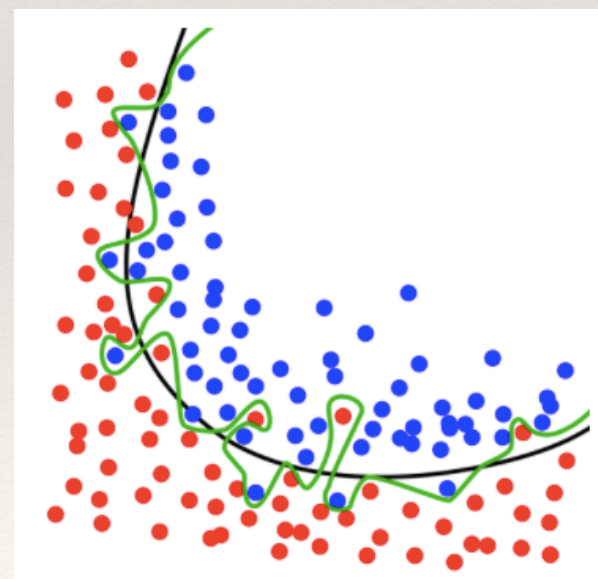
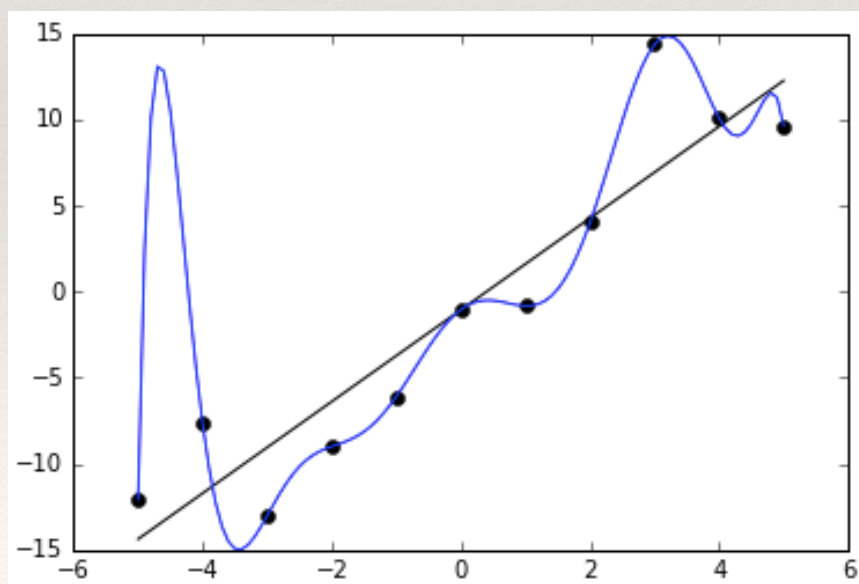
Main challenges in ML: Overfitting the Training Data

Bad algos

Overfitting means that a model performs well on the training data, but it does not generalise well to new, previously unseen data.

- The more complex a model is (DNN, or just high-degrees polynomials) the more it is able to detect and describe subtle patterns in the training data. But if the training set is noisy, or if it is too small (i.e. sampling noise), then the model is likely to detect patterns in the noise itself.

The cures include: **reducing the nb attributes, apply regularization, collect more training data** (in some cases..)



Main challenges in ML: Underfitting the Training Data

Bad algos

Underfitting (the opposite of overfitting) occurs when your model is too simple to learn the underlying structure of your training data

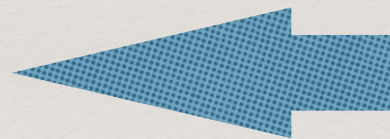
The main options to cure this are:

- selecting a **more powerful model**, with more parameters
- **feature engineering**: feeding better features to the learning algo
- **reducing the constraints** on the model (e.g. reducing the regularization hyperparameter)
 - ❖ careful, as this may open doors to overfitting..

Main challenges in ML

Basically, related to:

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



Training and Testing a model

Training technicalities

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 is to **split your data** into two sets: the **training set** and the **test set**

- train your model using the training set, and test it using the test set
 - ❖ the error rate you will experience on new cases will be called “generalisation error”
 - ❖ You can estimate this by evaluating your model on the test set
 - ❖ This value tells you how well your model will perform on instances it has never seen before
- If the generalisation error is high, it means that your model is overfitting the training data

More refined options do exist..

Model selection and Hyper-parameter tuning

Training technicalities

You are hesitating between 2 ML models..

- train both and compare how well they generalise using the test set

Ok, this helped to pick one. Now, to fight overfitting, you apply regularisation. Which λ (regularization parameter) do you choose?

- you train 100 models with 100 values of λ , you find the λ that produces a model with the lowest generalisation error (e.g. 5%), and then you pick your model. You launch this to production and measure a 15% error. What?!
- ❖ The problem is that you measured the generalization error multiple times on the test set, and you adapted the model and hyperparameters to produce the best model for that particular set. So presumably it will not perform well on new data

Solution: “holdout validation”, with a **validation** set

- you train multiple models with various hyperparameters on the “reduced” training set (i.e. the full training set minus the validation set), and you select the model that performs best on the validation set.
- Then, the holdout validation process is over, you 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

Training technicalities

The aforementioned solution usually works quite well. However:

- if the validation set is too small, then model evaluations will be imprecise: you may end up selecting a suboptimal model by mistake
- if the validation set is too large, then the remaining training set will be much smaller than the full training set. This is bad because the final model will be trained on the full training set, it is not ideal to compare candidate models trained on a much smaller training set.
 - ❖ it is like selecting the best marathon runner from his performance on the first 500 m ...

One way to solve this problem is to perform **repeated cross-validation**, using many small validation sets.

- Each model is evaluated once per validation set, after it is trained on the rest of the data. By averaging out all the evaluations of a model, we get a much more accurate measure of its performance.
- However, there is a drawback: the training time is multiplied by the number of validation sets: it increases the requirements on computing resources!



Tools and Frameworks in ML

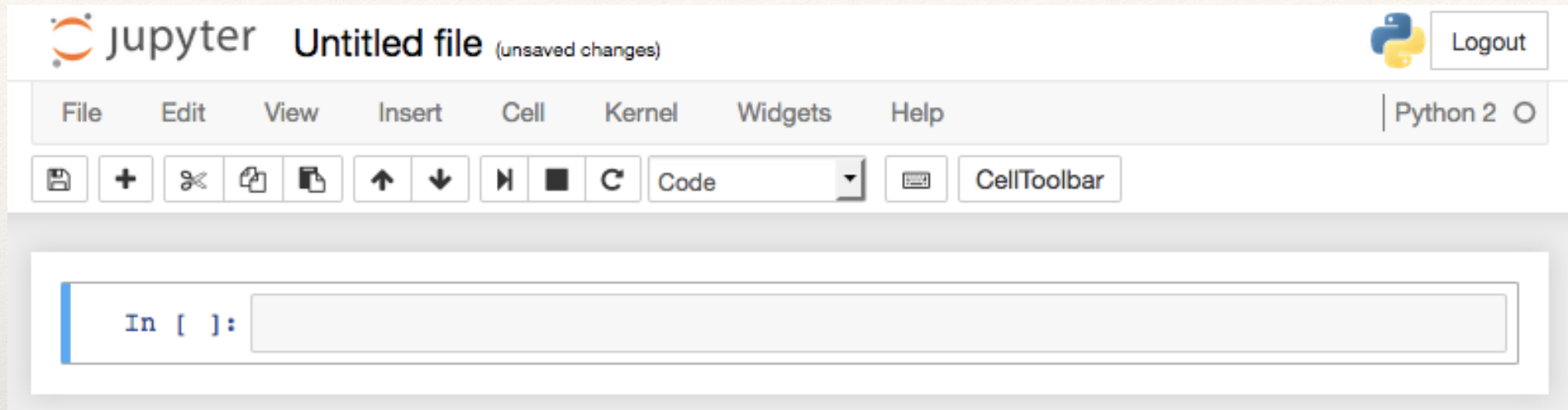
(and which general Python library and tools we will use today)

Jupyter

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

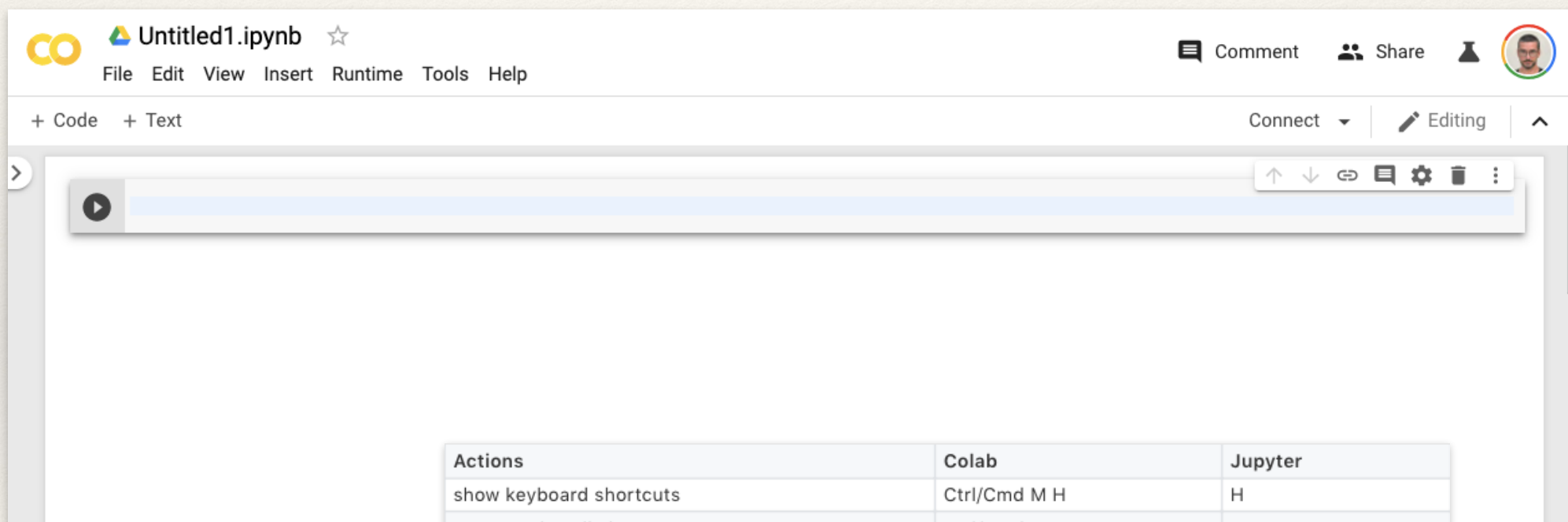
Google Colab(oratory)



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

(allow me oversimplifications here..)

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 -

Proper way to get started: **remember you are on the cloud!**

- **create** a new Python notebook
 - ❖ or **open an existing notebook** from gdrive, github, etc.. or upload your ipynb from your laptop
- **Save** a copy on your own gdrive, and **code** from that one
 - ❖ all future changes will be saved

Colab: select your Runtime

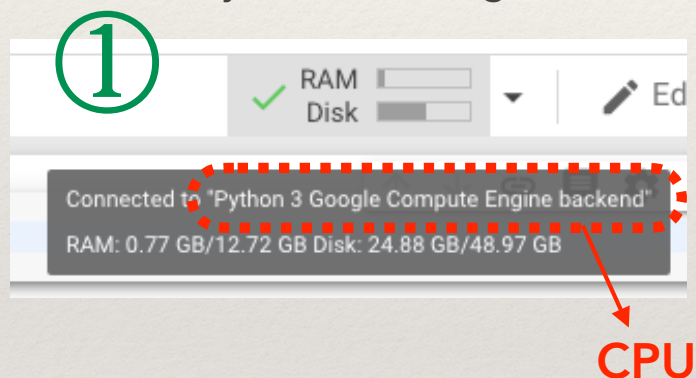


Once you open a Colab notebook,
you get a Colab **Runtime**

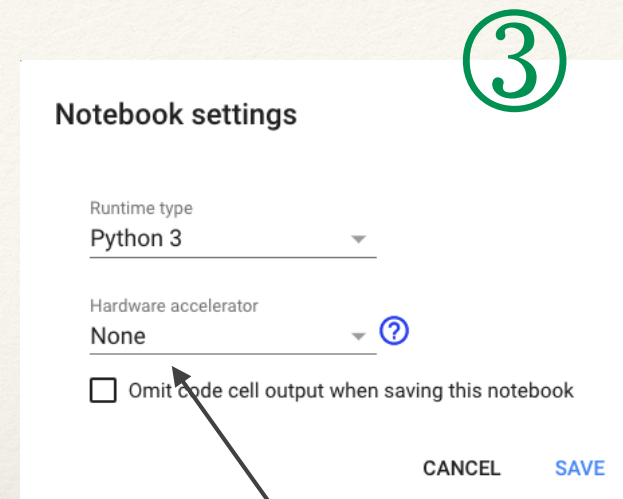
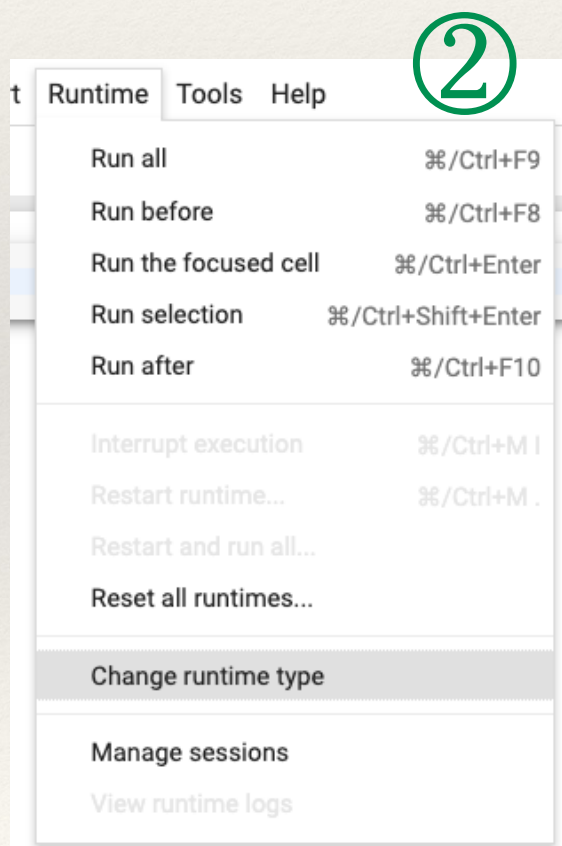
- this is the free Google VM dedicated to you for a while

By default, the initial Runtime is CPU-only

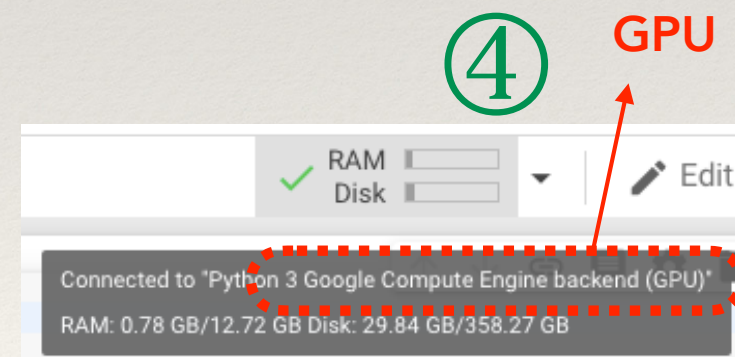
- but you can change this:



colab VM states are:
Connecting,
Allocating,
Initialising, and its
actual RAM/Disk
monitoring widget

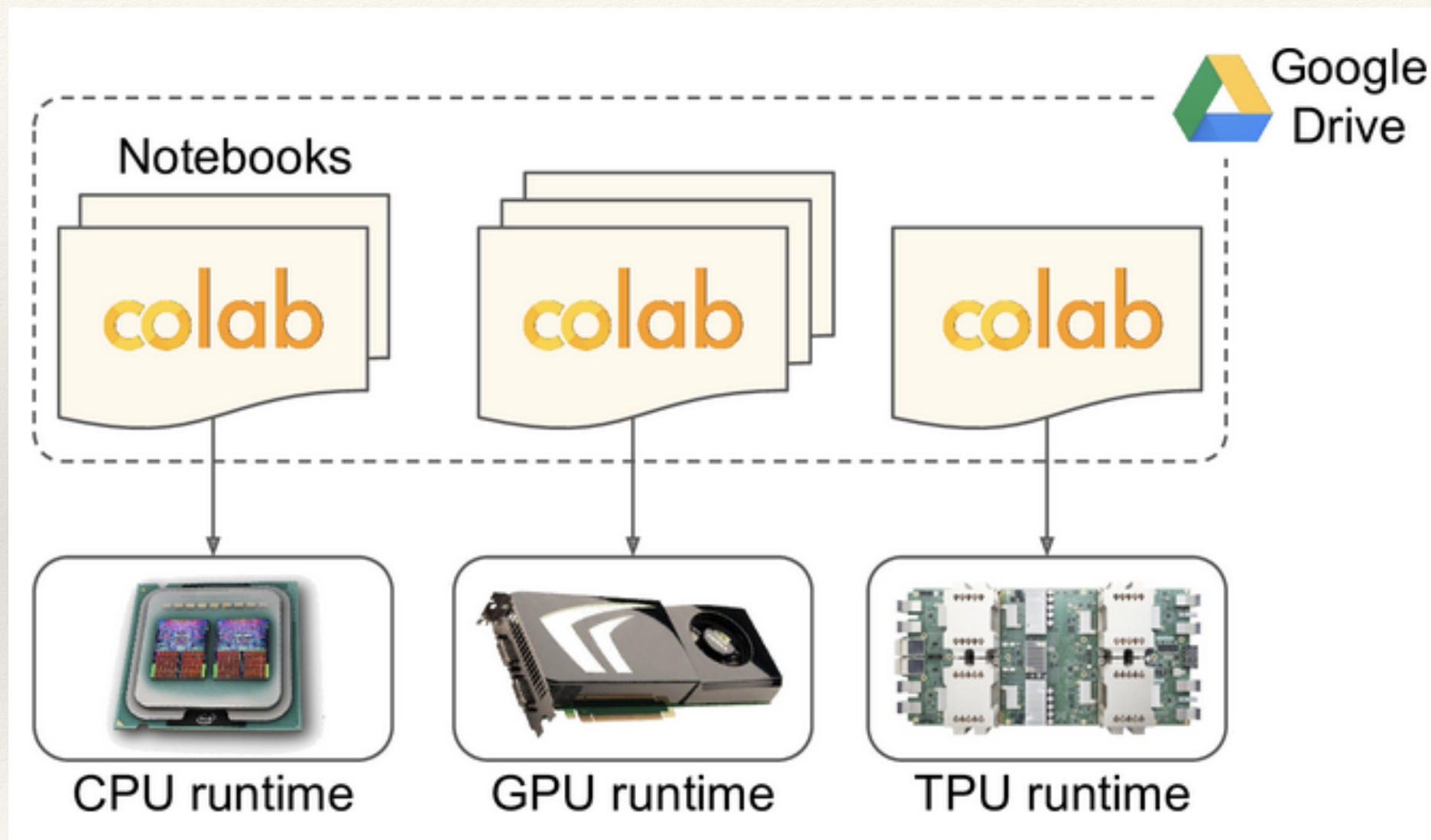


select GPU here



(NOTE: yes, also TPUs are available..)

Colab Runtimes



As from the Colab FAQs:

- “Colaboratory is intended for interactive use. Long-running background computations, particularly on GPUs, may be stopped. Please do not use Colaboratory for cryptocurrency mining.”

The web interface will automatically disconnect you from the Colab Runtime after **~30 minutes** of unattended connection

- you can have it again, but when you reconnect to the Colab Runtime, it may have been reset (so make sure you download what you needed before moving to something else)

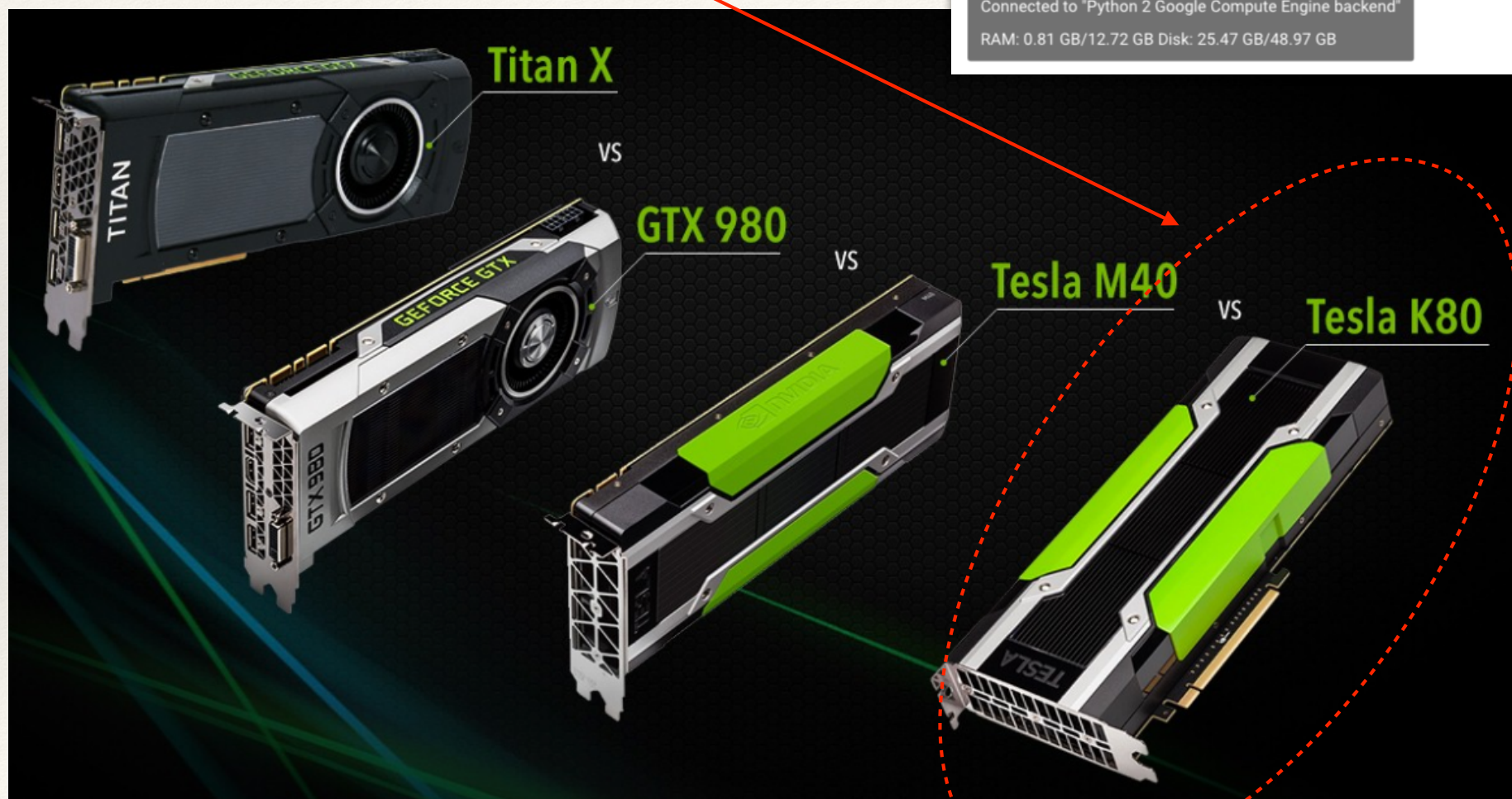
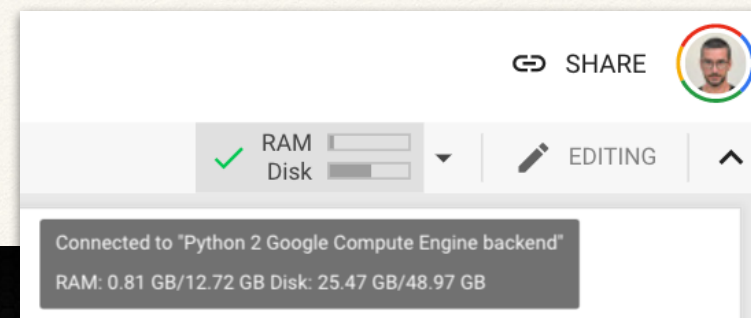
Even if you stay connected and actually code, the Colab Runtime will automatically shut down after **12 hours**

- this is meant to prevent long-running computations

Still a neat tool for experimenting on GPUs (and for free)..

Which GPU am I getting in Colab?

Tesla K80, 12 GB RAM,
up to 12 hrs in a row.



[DISCLAIMER: this information might be outdated in the future, pretty soon..]

NumPy

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 waves¹ and in the first imaging of a black hole². 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/>

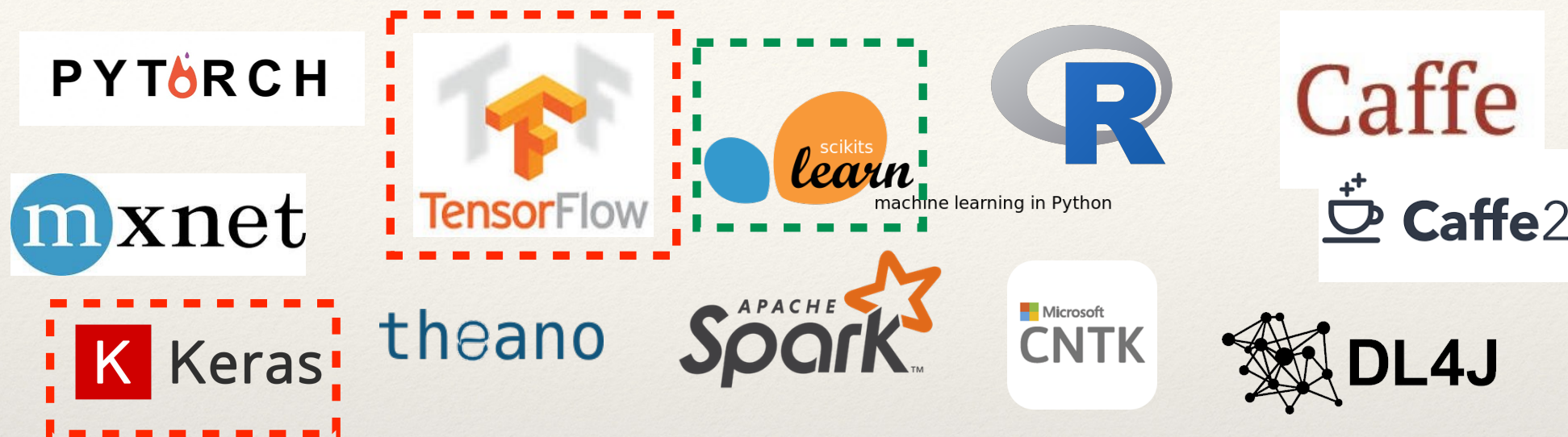
Pandas is an open source library providing **high-performance, easy-to-use data structures and data analysis tools** for Python.

Matplotlib

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

ML libraries and frameworks


ML tools and frameworks



Nowadays, often tools define what you do and not only how...

- get familiar with Sklearn + Keras/Tensorflow + Pytorch + fast.ai + ...

Today: **scikit-learn + Keras/Tensorflow**



The screenshot shows the Scikit-learn website. At the top is the Scikit-learn logo and a navigation bar with links for Home, Installation, Documentation, and Examples. A search bar and a 'Fork me on GitHub' button are also present. The main banner features a grid of 24 small plots showing various machine learning model results, followed by the text 'scikit-learn Machine Learning in Python' and a list of bullet points: 'Simple and efficient tools for data mining and data analysis', 'Accessible to everybody, and reusable in various contexts', 'Built on NumPy, SciPy, and matplotlib', and 'Open source, commercially usable - BSD license'. Below the banner, three columns describe 'Classification', 'Regression', and 'Clustering' with their respective definitions, applications, and algorithms.

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.

Algorithms: SVM, nearest neighbors, random

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, ridge regression, Lasso, ...

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, ...

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!)

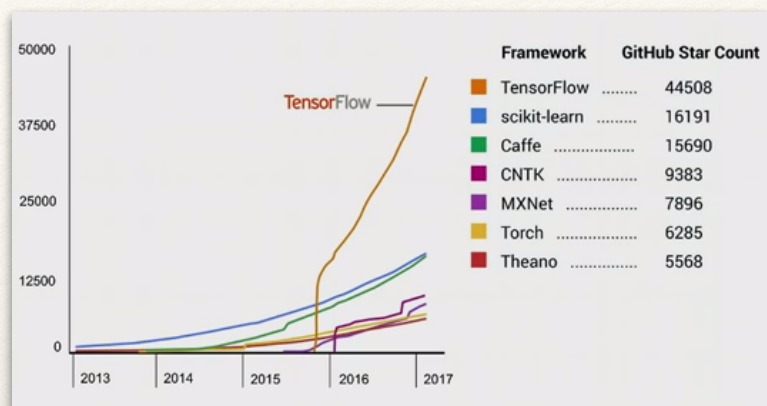
a good backend choice

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

on top of (e.g.) Tensorflow

Both quite popular, and widely adopted



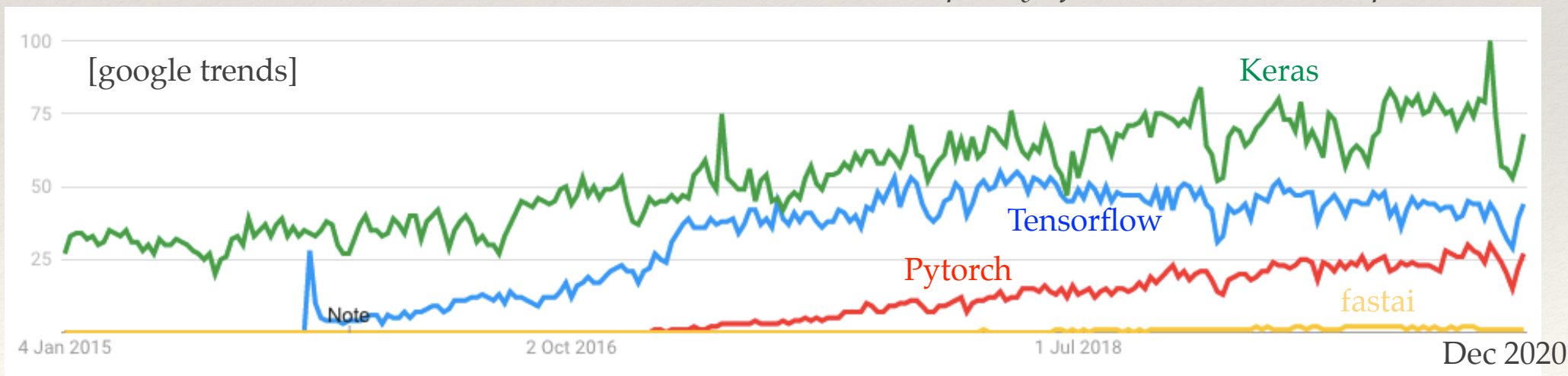
Bump in 2015 as TF became public

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..]



TensorFlow is an open source (since 2015) software library by the Google Brain team for **numerical computation using data flow graphs**.

- **Nodes** in the graph represent mathematical operations..
- .. while the graph edges represent the multidimensional data arrays (**tensors**) communicated between them.

In first approx: Tensorflow is not only about NNs..

.. but it is a perfect match to implement NNs efficiently.



(Most) popular tool to train and apply NNs

Python wrapper around multiple numerical computation libraries

- e.g. TensorFlow
- but, backends: TensorFlow, Theano, CNTK , ..

Pros:

- Hides most of the low-level operations that you don't want to care about

Cons:

- Sacrificing little functionality for much easier user interface

Main asset: **being able to go from idea to result with the least possible delay**

The labs will use Keras code too!

Ready?

It is time to start our first lab!