The Abdus Salam International Centre for Theoretical Physics

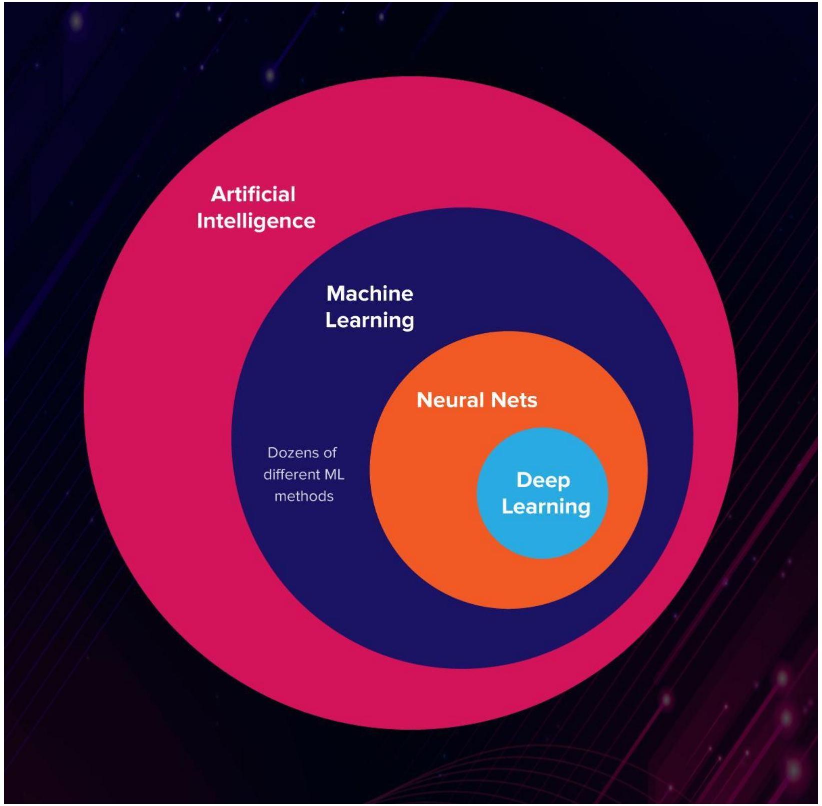
Intro to DL

Serafina Di Gioia
PostDoctoral Researcher @ ICTP

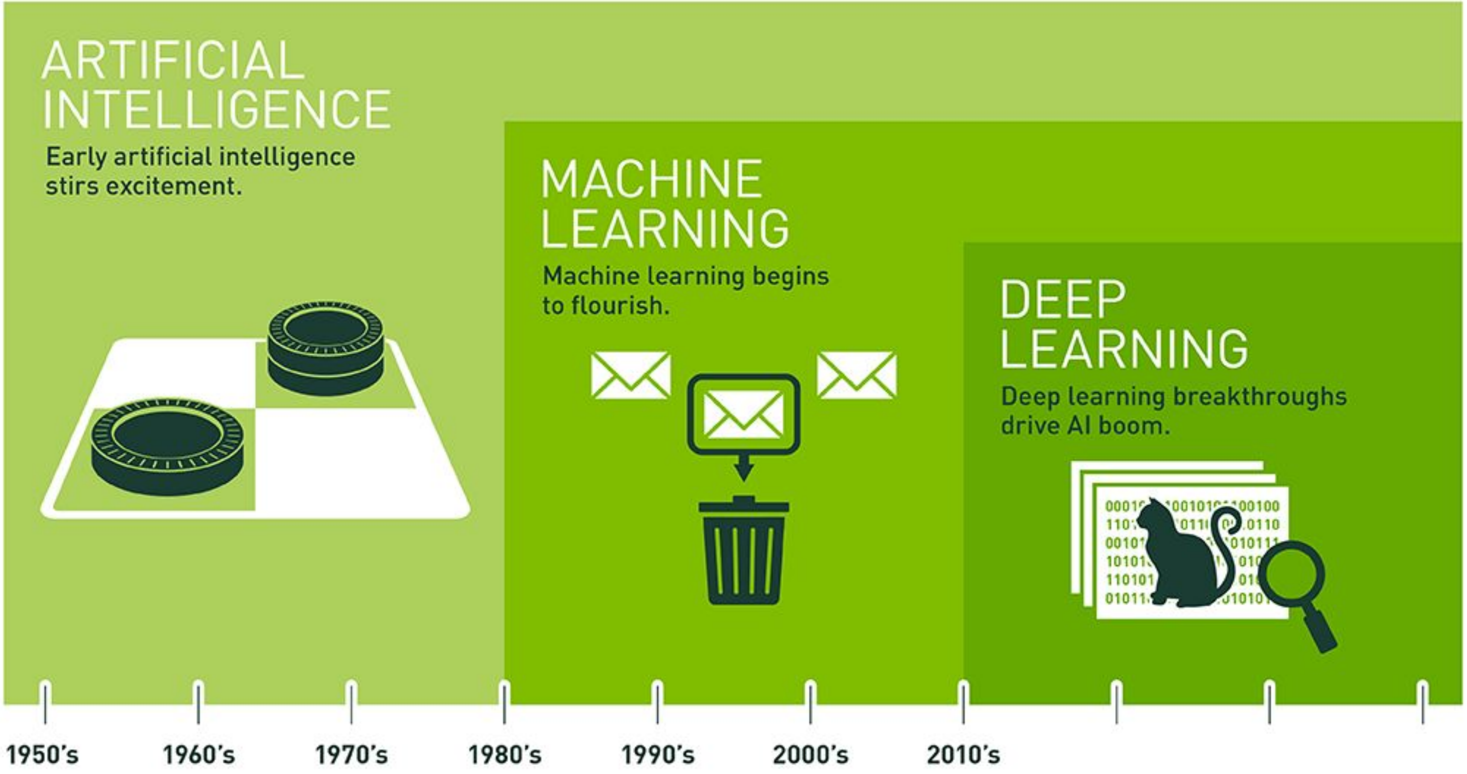
Where is DL in the picture?

Deep learning:

a type of machine learning based on artificial neural networks in which multiple layers of processing are used to extract progressively higher level features from data.



From LISP to the DL revolution…

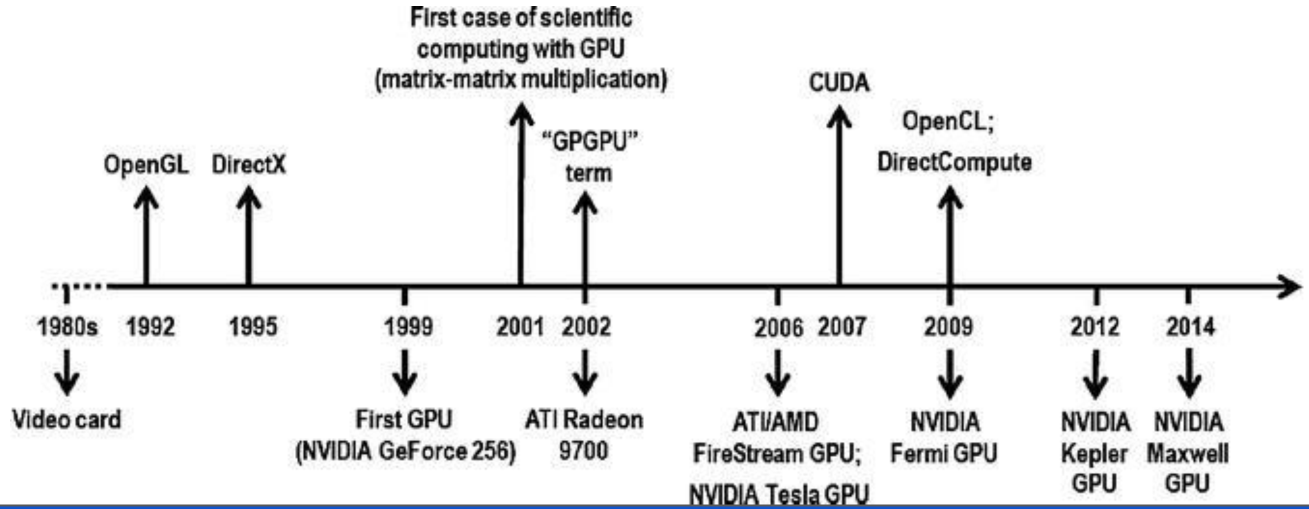


Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

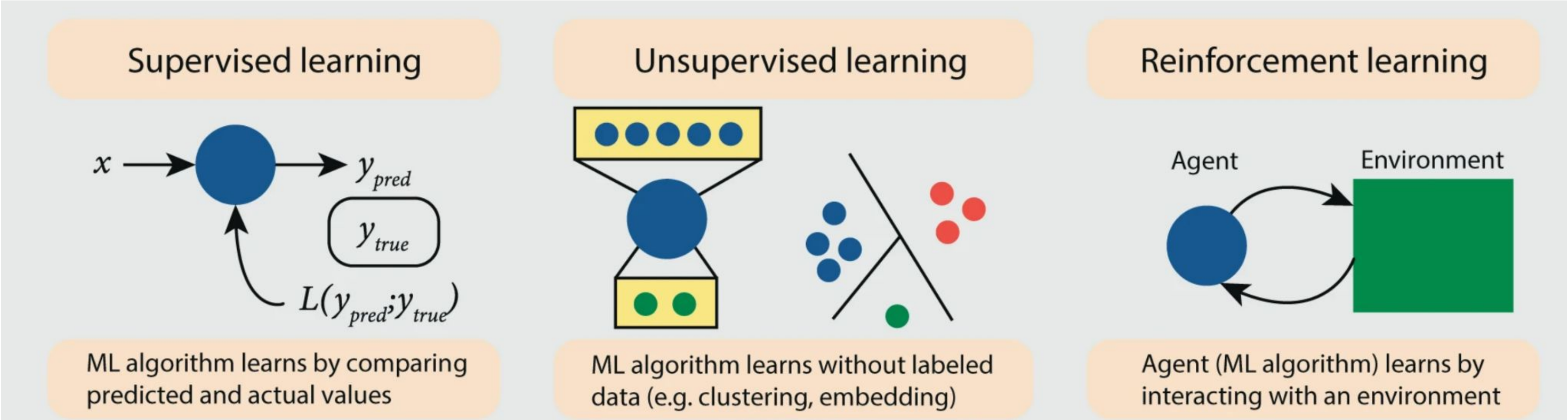
Main ingredients for DL breakthrough

- large datasets available (e.g IMAGENET)
- GPUs development (in particular, CUDA introduction)
- increased involvement of developers from CV and scientific communities

The DL era starts few years after that CUDA came to light

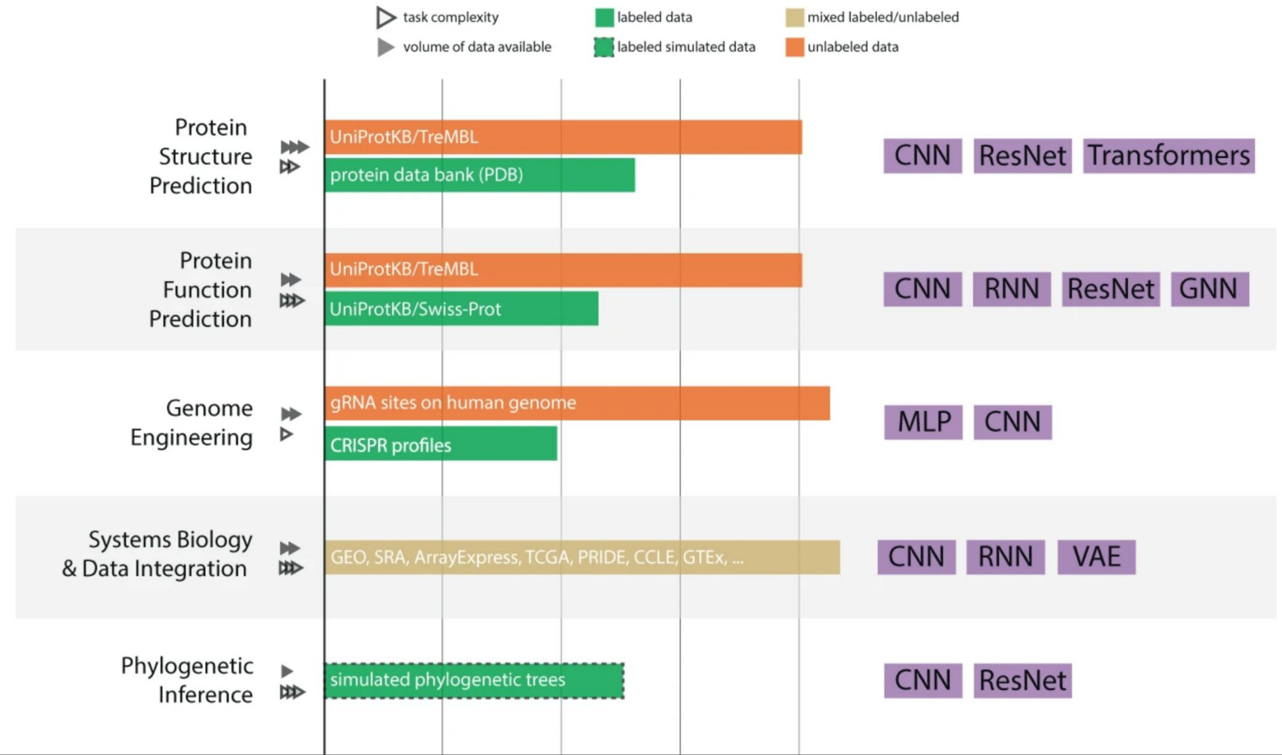


ML scenarios



Learning type	Model building	Examples
Supervised	Algorithms or models learn from labeled data (task-driven approach)	Classification, regression
Unsupervised	Algorithms or models learn from unlabeled data (Data-Driven Approach)	Clustering, associations, dimensionality reduction
Semi-supervised	Models are built using combined data (labeled + unlabeled)	Classification, clustering
Reinforcement	Models are based on reward or penalty (environment-driven approach)	Classification, control

Typical sizes of DL data sets

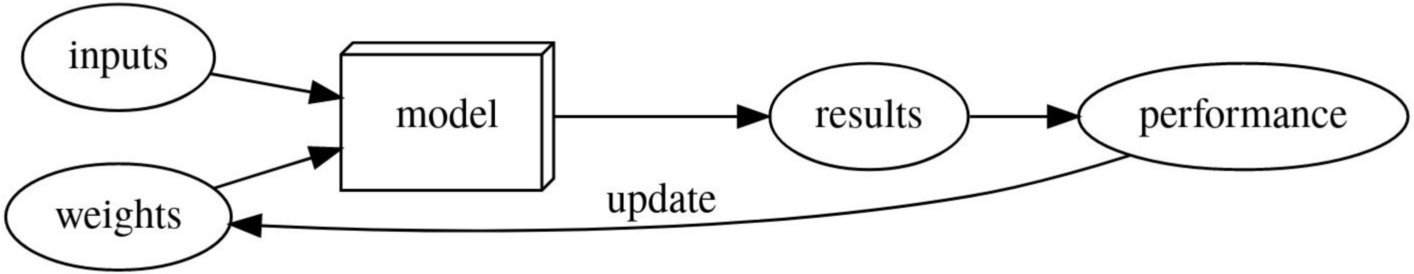


The increasing complexity of the new datasets, typical of big data epoch motivates the need for GPU-based libraries

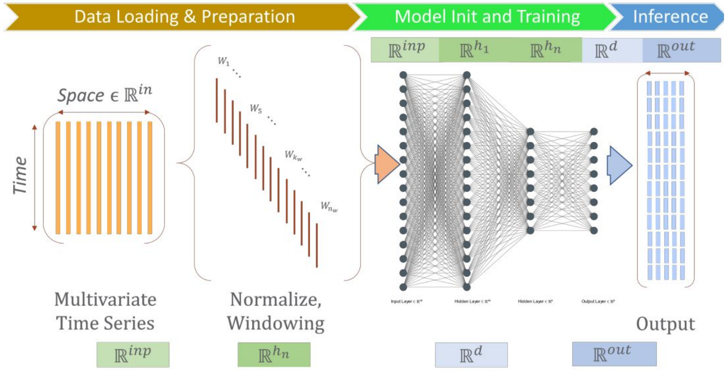
Building blocks of ML algorithms

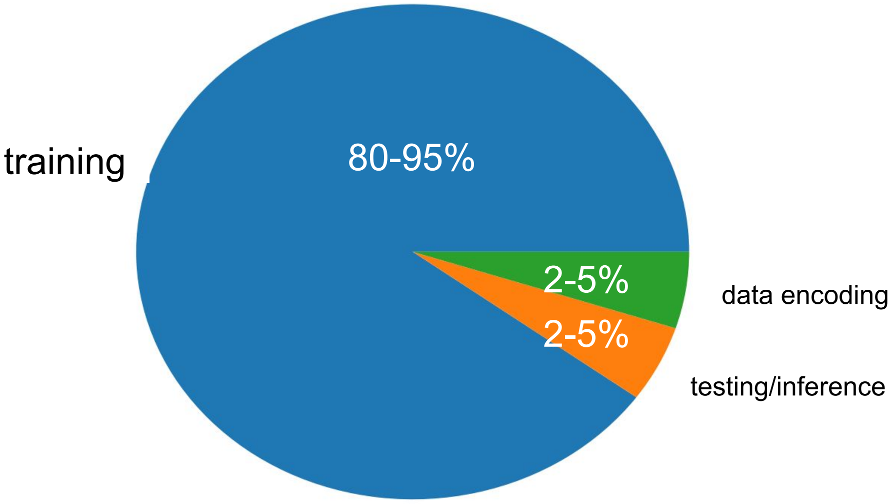
ML algorithms have three main components

1. **decision process**: based on some input data, which can be labeled or unlabeled, your algorithm will produce an estimate about a pattern in the data. This estimate can be used to solve a prediction or classification task
2. **error function**: it evaluates the prediction of the model. If there are known examples, an error function can make a comparison to assess the accuracy of the model.
3. **Model Optimization Process**: If the model can fit better to the data points in the training set, then weights are adjusted to reduce the discrepancy between the known example and the model estimate. The algorithm will repeat this “evaluate and optimize” process, updating weights autonomously until a threshold of accuracy has been met.

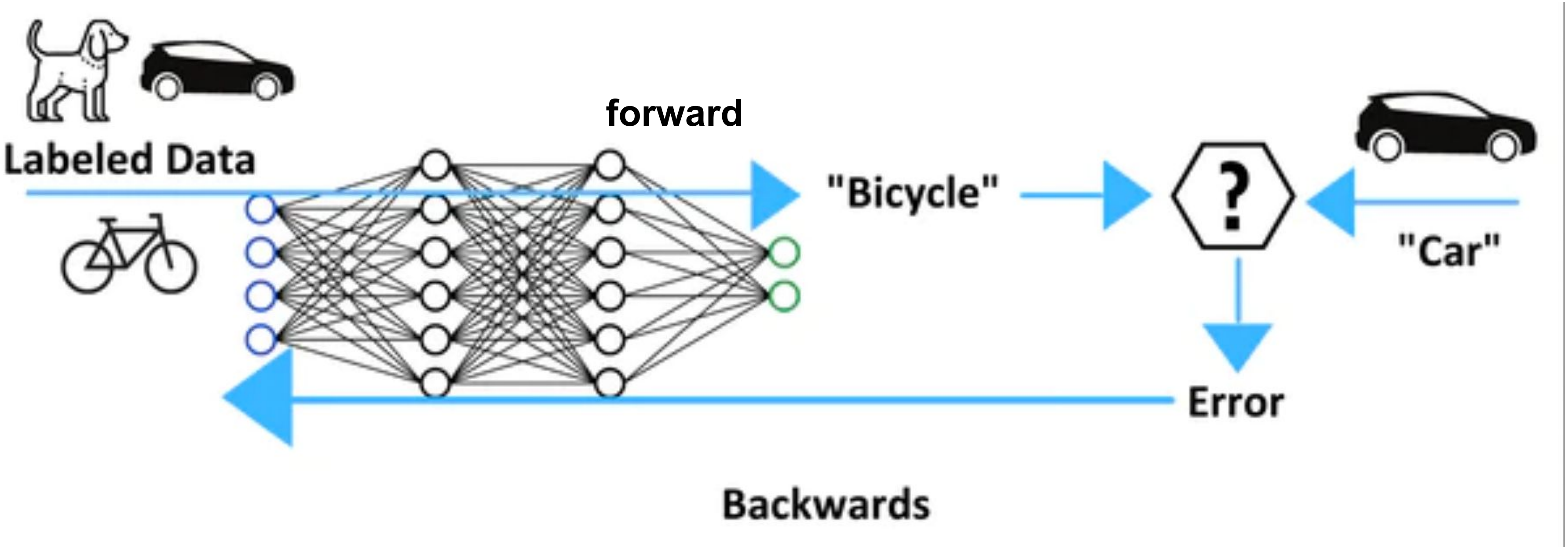


Analyzing the computational workload of DL models





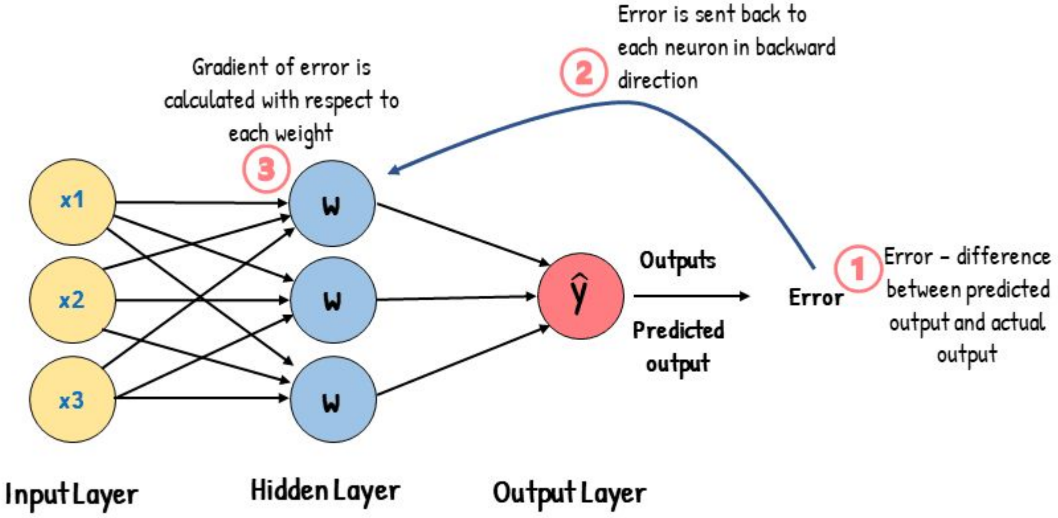
Training a DL model (in a supervised setting)

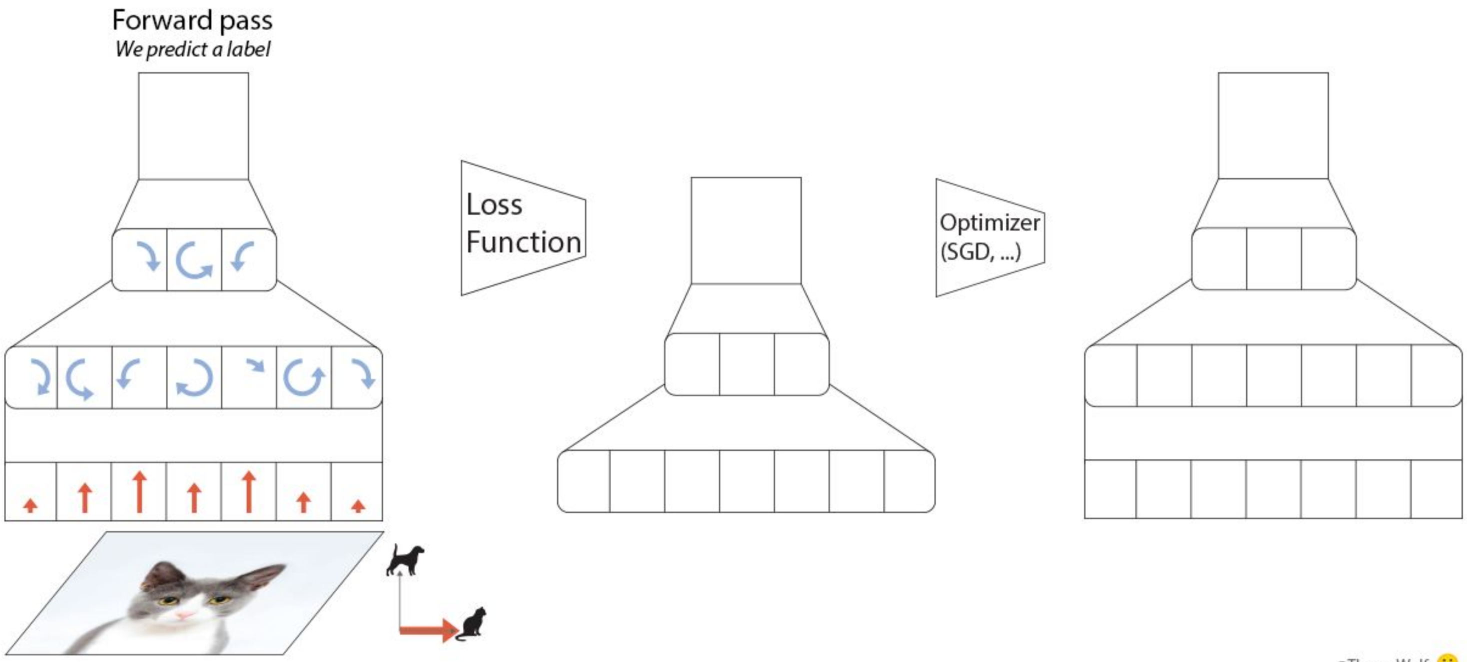


Back-propagation

The workhorse of DL is the Backpropagation algorithm (Rumelhart et al., 1986).

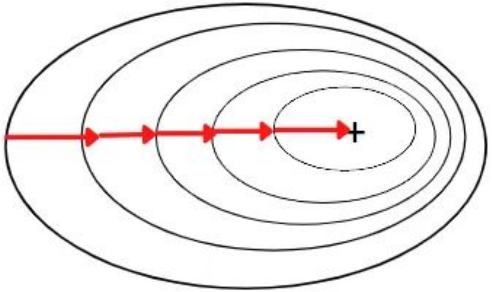
It allows for efficient gradient computation by recursively applying the chain rule of calculus. It owes his name to the presence of a ‘backward pass’ of an error signal through the neural network.



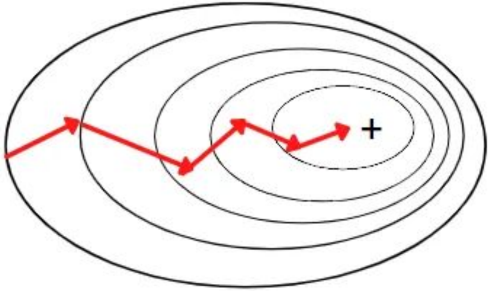

Forward pass
We predict a label
Loss
Function
Optimizer
(SGD, ...)
@Thom_Wolf

Gradient Descent variants

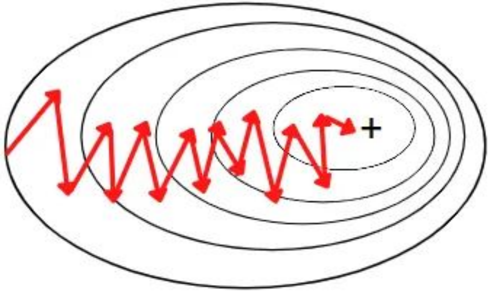
Batch Gradient Descent



Mini-Batch Gradient Descent



Stochastic Gradient Descent **(SGD)**



Alternative Optimizers

Other optimizers have been proposed to enhance the speed and convergence of the training process:

• **SGD with Momentum** (Polyak, 1964): Speeds up gradient descent by adding a fraction of the previous update to the current one.

• **RMSprop** (Hinton, 2012): Adapts the learning rate for each parameter based on recent gradient magnitudes.

• **Adam** (Kingma and Ba, 2015): Combines momentum and adaptive learning rates for more efficient optimization.

Hyperparameters importance

Hyperparameters are an overlooked but crucial factor in DL practise. They include:

- learning rate
- training steps/epochs
- batch size
- Optimizer choice-setting

How to choose them?

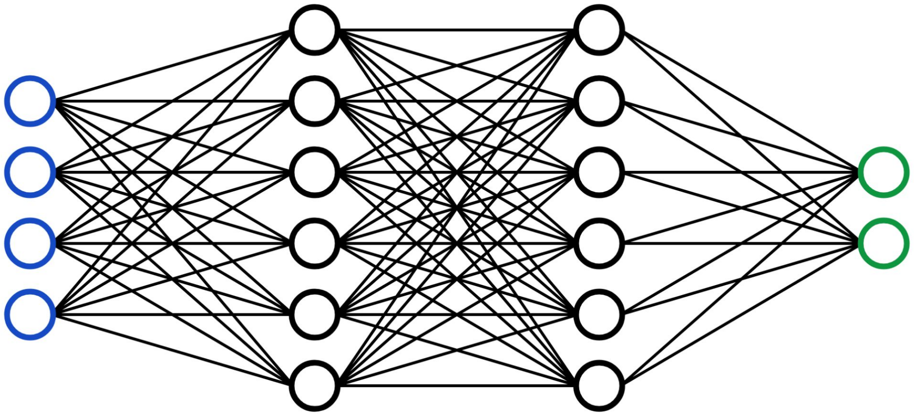
- Experience
- Trial and error

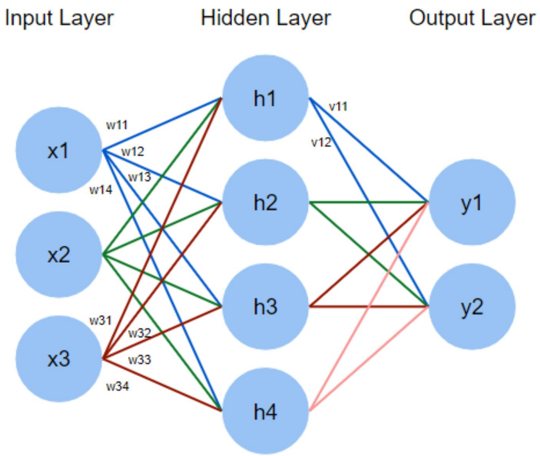
Universal approximation theorem

Universal approximation theorem (Hornik, 1991):

A feedforward neural network with at least one hidden layer can approximate any continuous function to any desired accuracy, given enough neurons and the right activation function.

Linear Neural Network (lNN)





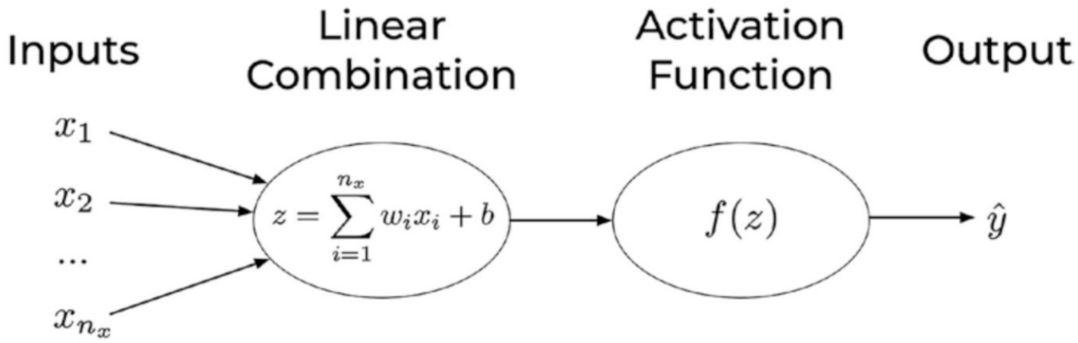
$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} * \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \end{bmatrix} = \begin{bmatrix} h'_1 & h'_2 & h'_3 & h'_4 \end{bmatrix}$$

Alias of nn.linear() in Pytorch: torch.mm(inputs, linear.weight.T).add(linear.bias)

Artificial neuron

A neuron has two main components:

- The weights (w_i) (the bias b is sometime included in the weights)
- The *activation function* (the f



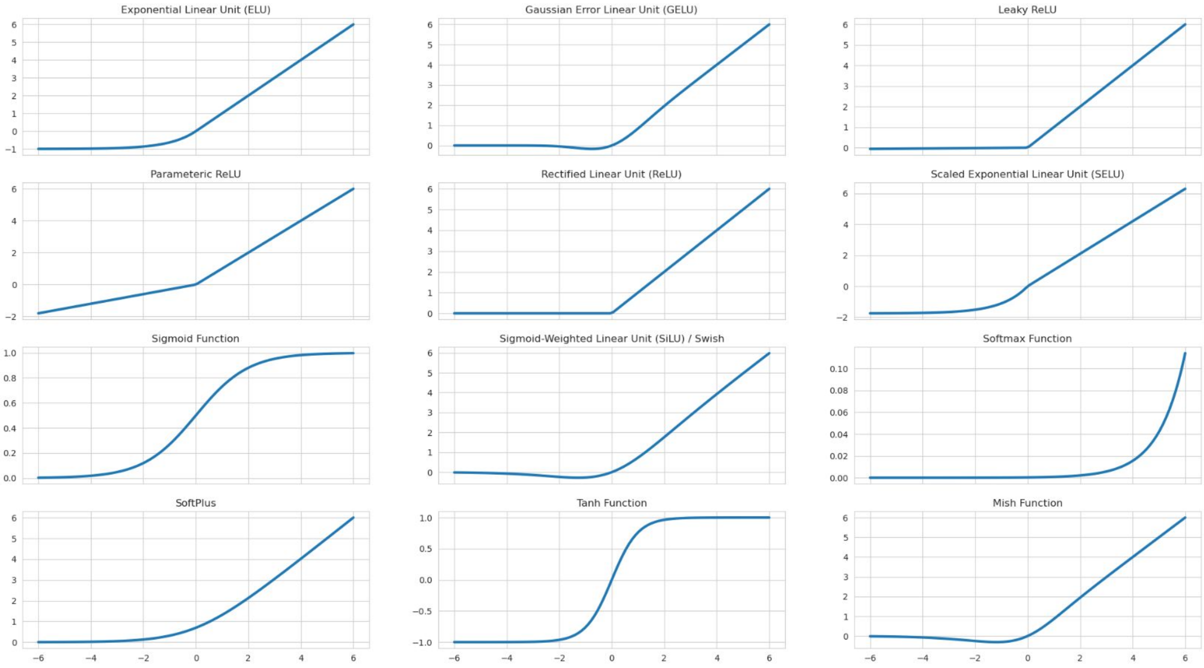
Which portion of a neural network model is responsible for the type of problem that can be solved?

Two main components are responsible for the type of problem that can be solved:

- The output activation function
- The loss function

The optimiser is not related in any way to the type of problem solved (it does not depend on the type of the response variable).

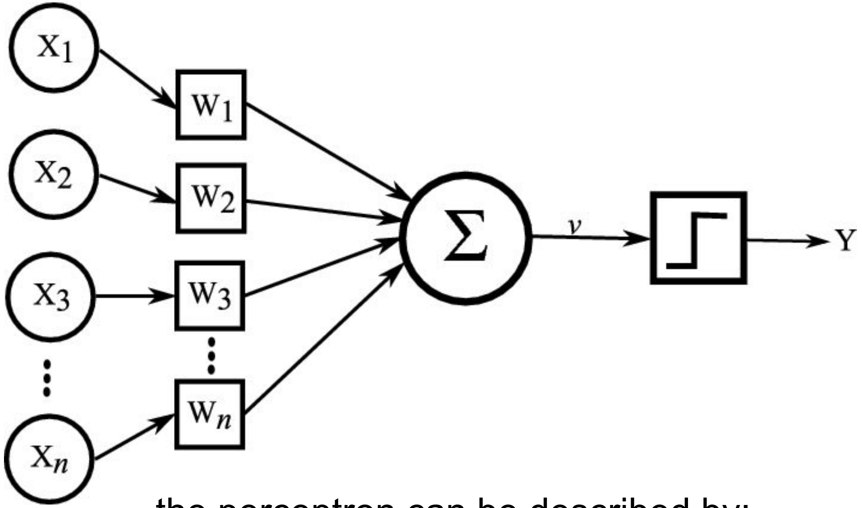
Types of activation functions



Perceptron

Rosenblatt said:

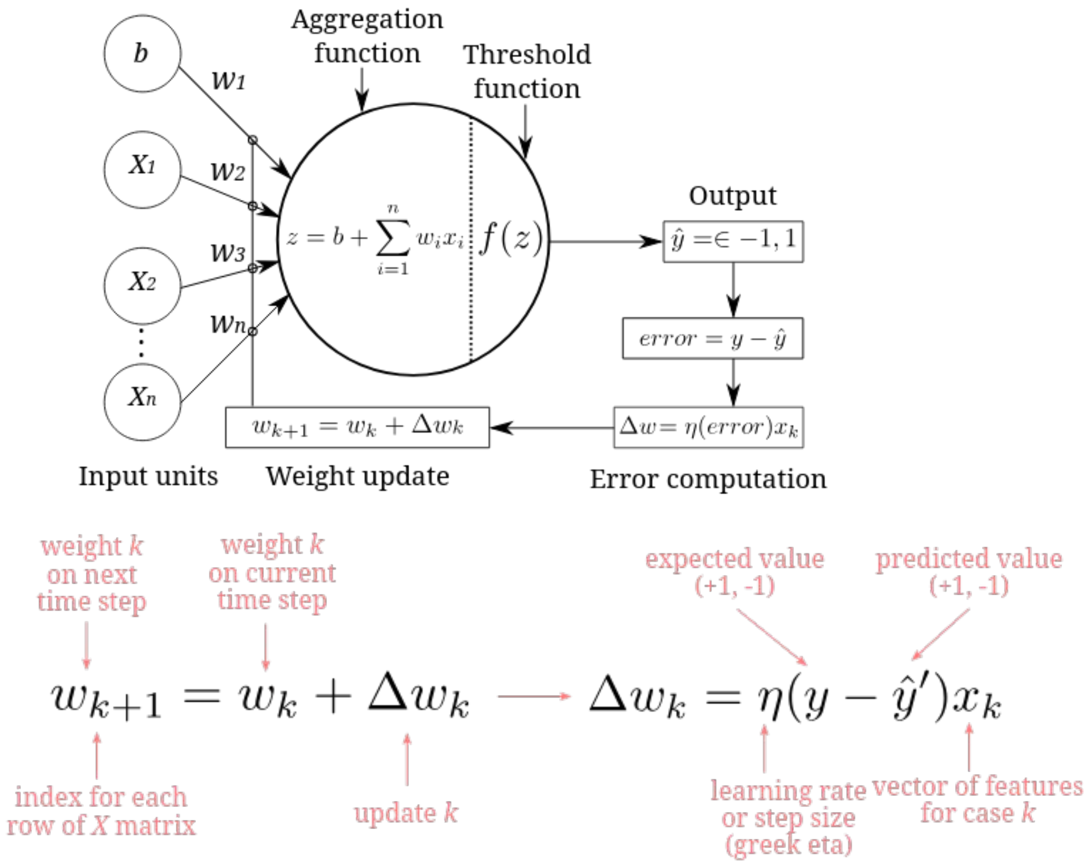
"A perceptron is first and foremost a ***brain model, not an invention for pattern recognition***. As a brain model, its utility is in enabling us to determine the physical conditions for the emergence of various psychological properties. It is by no means a "complete" model, and we are fully aware of the simplifications which have been made from biological systems; but it is, at least, an analyzable model."



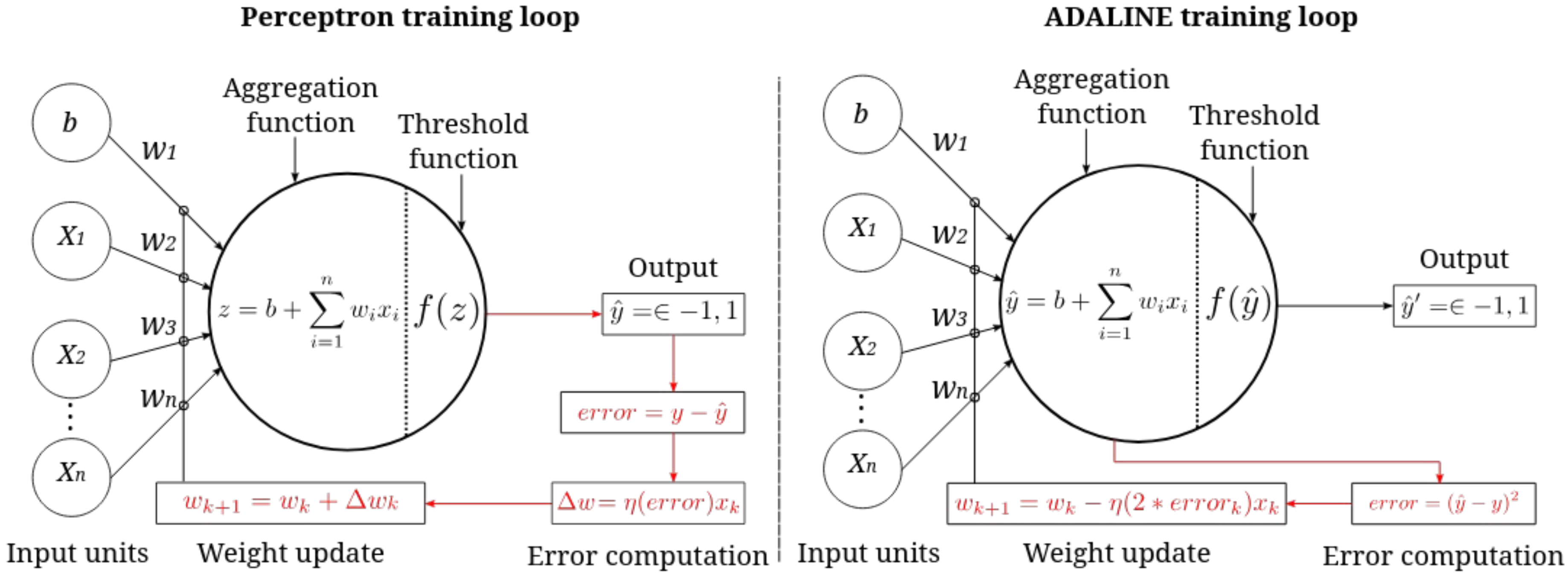
the perceptron can be described by:

- a linear function that aggregates the input signals
- a threshold-activation function that determines if the response neuron fires or not
- a learning procedure to adjust connection weights

Learning procedure for the Perceptron

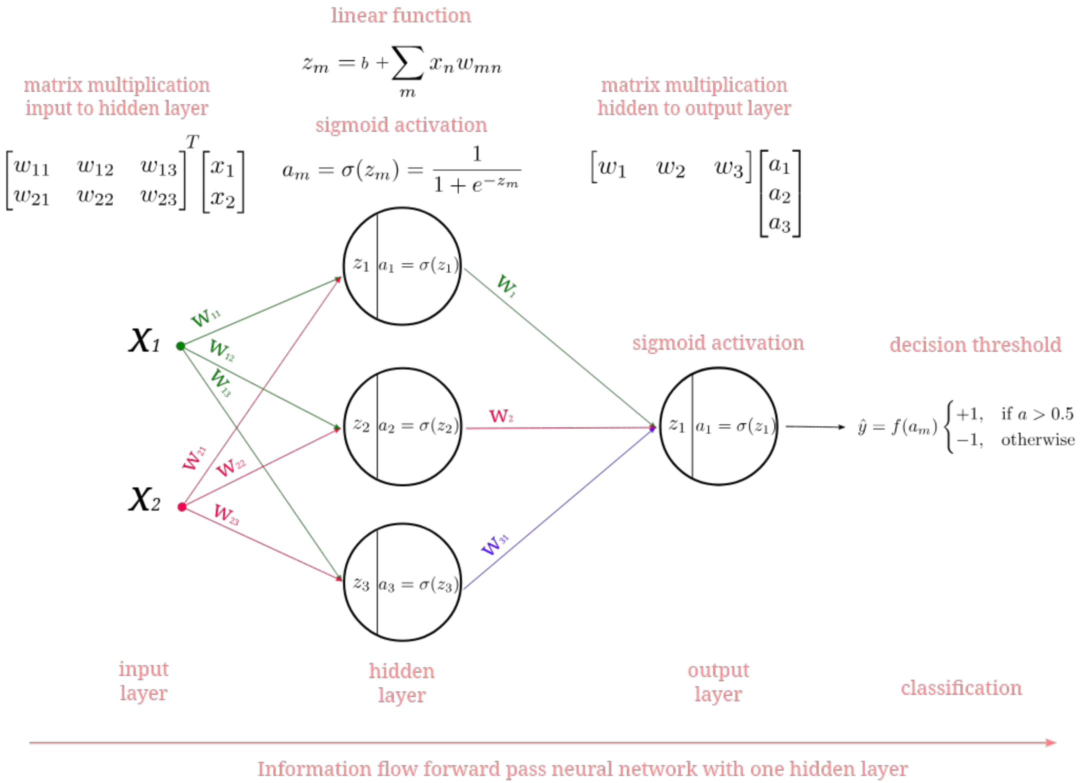


Adaline vs Perceptron

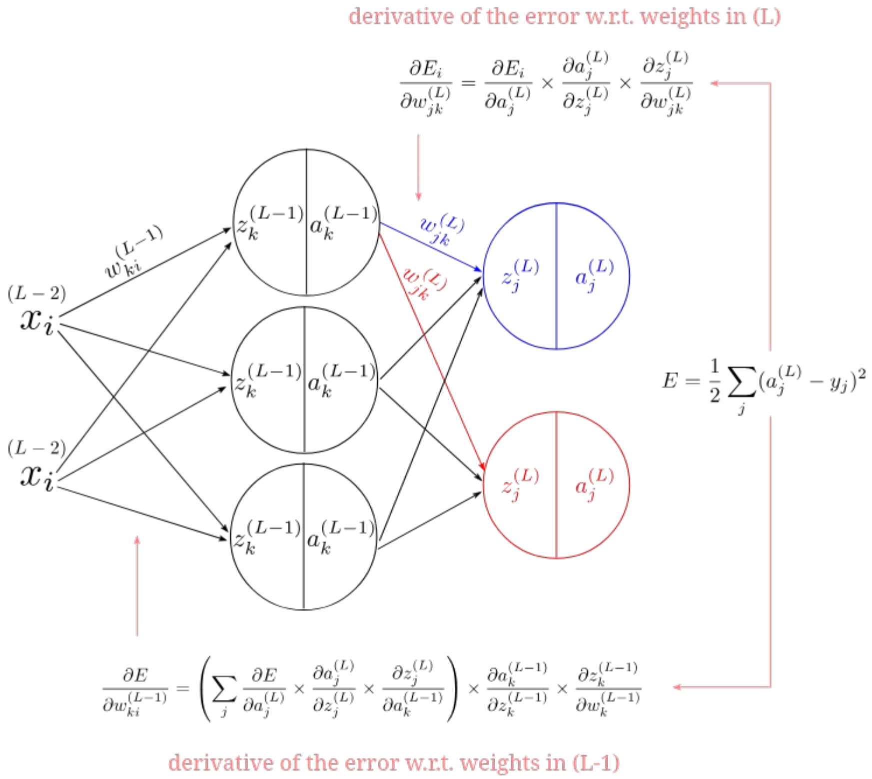


it introduces SGD in the training

Multi-layer Perceptron



Back-propagation in multi-layer perceptron



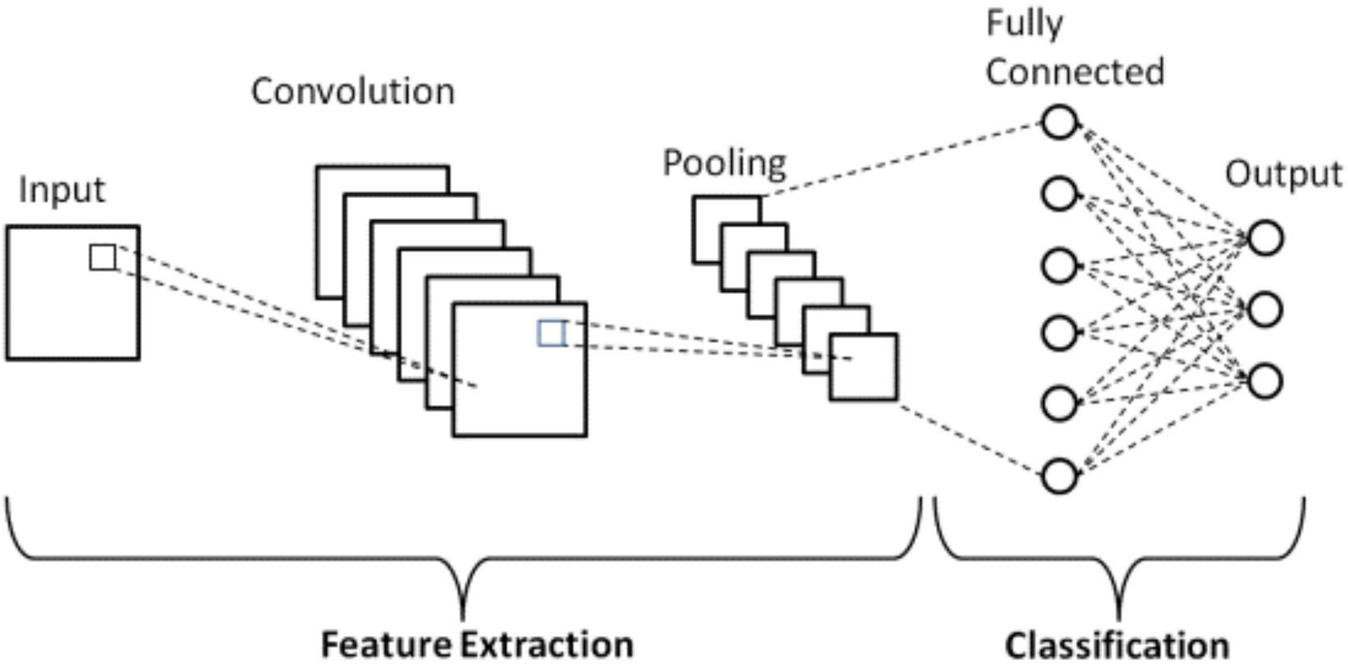
Formula for the weights update in the L-layer:

$$w_{jk}^L = w_{jk}^L - \eta \times \frac{\partial E}{\partial w_{jk}^L}$$

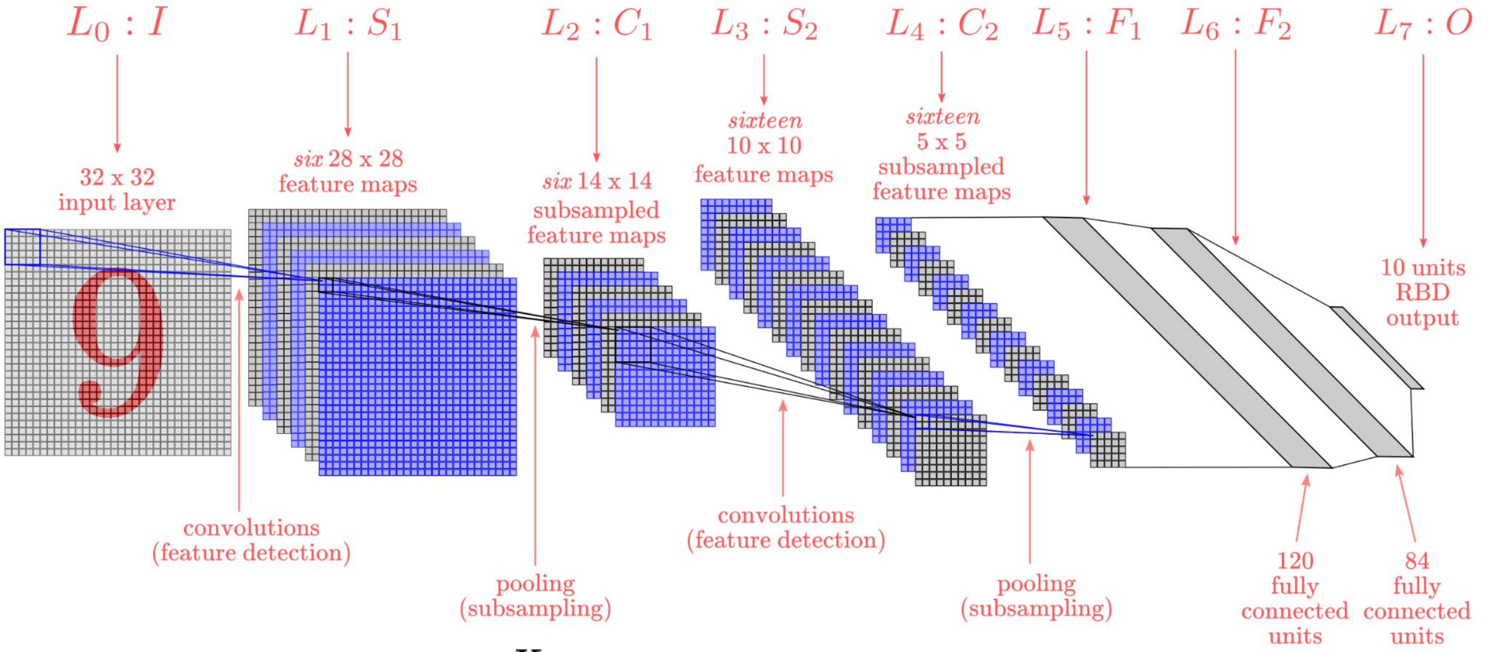
Formula for the bias update in the L-layer:

$$b^{(L)} = b^{(L)} - \eta \times \frac{\partial E}{\partial b^{(L)}}$$

Convolutional Neural Networks (CNN)

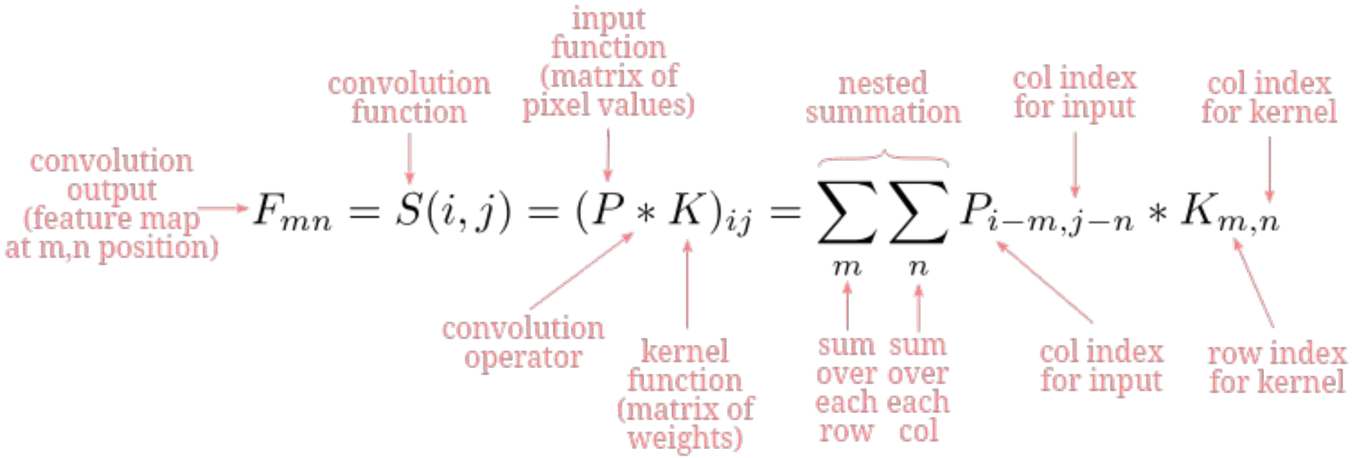


LeNet-5



$$E(W) = \frac{1}{P}\sum_{p=1}^{X}(\hat{y}_{D^p}(X^p, W) + \log(e^{-j} + \sum_i e^{-\hat{y}(X^p, W)}))$$

The Convolution step



Actually, several deep learning libraries like [MXNet]() and [Pytorch]() **DO NOT implement convolutions** but a closely related operation called **cross-correlation**

$$F_{mn} = S(i,j) = (P \star K)_{ij} = \sum_m \sum_n P_{i+m,j+n} \star K_{m,n}$$

The convolution operation is simply a matrix multiplication

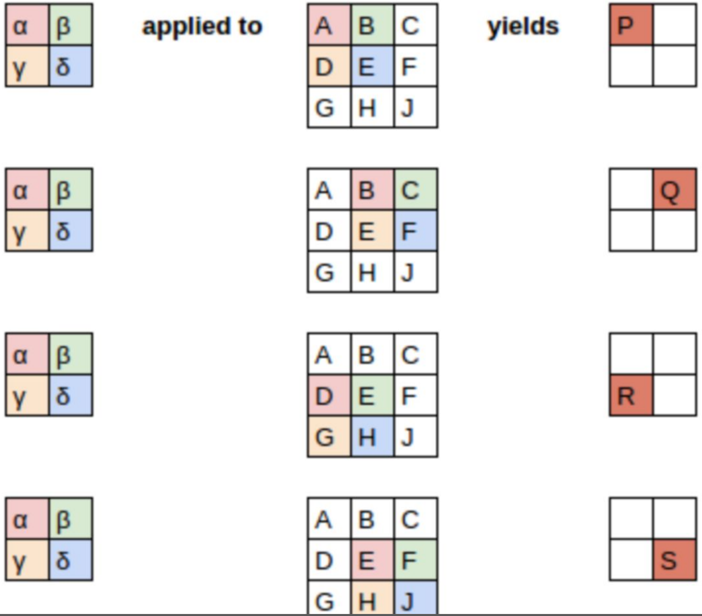
Let's take a look at basic element of CNN: convolution layer

Consider the case where we are applying (2,2) kernel

α	β
γ	δ

to a (3,3) matrix:

A	B	C
D	E	F
G	H	J

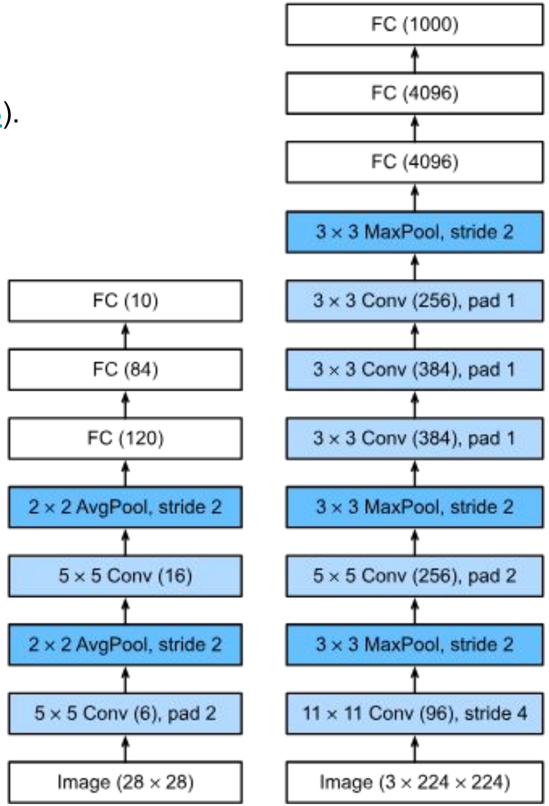


The convolution can be rewritten as

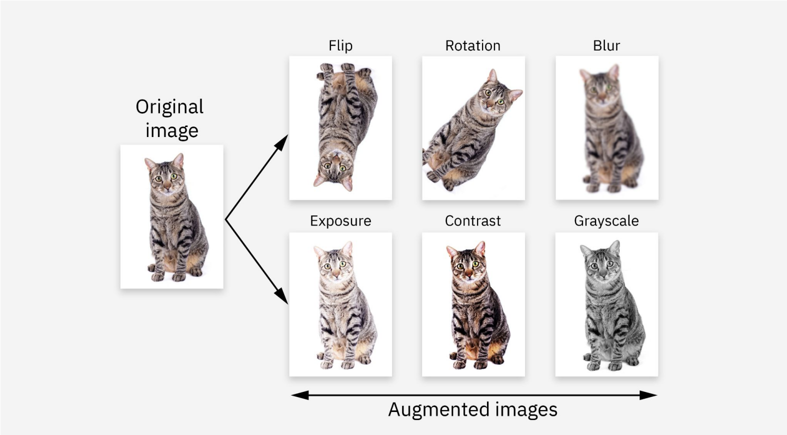
$$
\begin{array}{c}
\begin{bmatrix}
\alpha & \beta & 0 & \gamma & \delta & 0 & 0 & 0 & 0 \\
0 & \alpha & \beta & 0 & \gamma & \delta & 0 & 0 & 0 \\
0 & 0 & 0 & \alpha & \beta & 0 & \gamma & \delta & 0 \\
0 & 0 & 0 & 0 & \alpha & \beta & 0 & \gamma & \delta
\end{bmatrix} \\
\begin{matrix} A & B & C & D & E & F & G & H & J \end{matrix}
\end{array}
*
\begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ J \end{bmatrix}
+
\begin{bmatrix} b \\ b \\ b \\ b \end{bmatrix}
=
\begin{bmatrix}
\alpha A+\beta B+0C+\gamma D+\delta E+0F+0G+0H+0J+b \\
0A+\alpha B+\beta C+0D+\gamma E+\delta F+0G+0H+0J+b \\
0A+0B+0C+\alpha D+\beta E+0F+\gamma G+\delta H+0J+b \\
0A+0B+0C+0D+\alpha E+\beta F+0G+\gamma H+\delta J+b
\end{bmatrix}
=
\begin{bmatrix}
\alpha A+\beta B+\gamma D+\delta E+b \\
\alpha B+\beta C+\gamma E+\delta F+b \\
\alpha D+\beta E+\gamma G+\delta H+b \\
\alpha E+\beta F+\gamma H+\delta J+b
\end{bmatrix}
=
\begin{bmatrix} P \\ Q \\ R \\ S \end{bmatrix}
$$

AlexNet

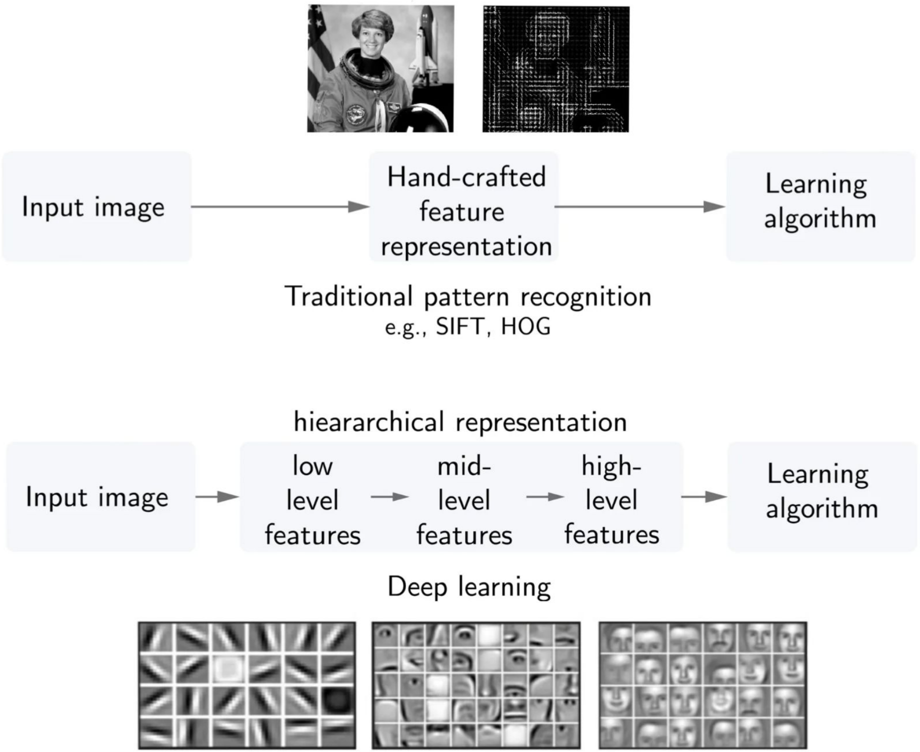
(Russakovsky *et al.*, 2013).



The training procedure of AlexNet used for the first time data augmentation:

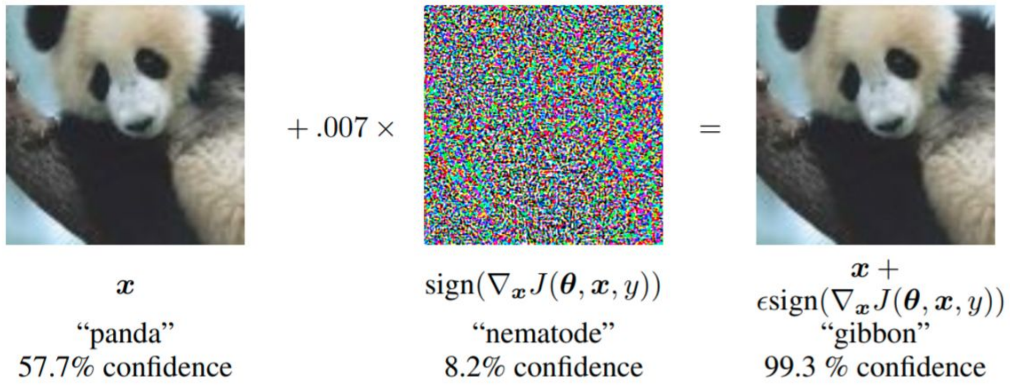


Hierarchical representation learning



Limitations of CNNs

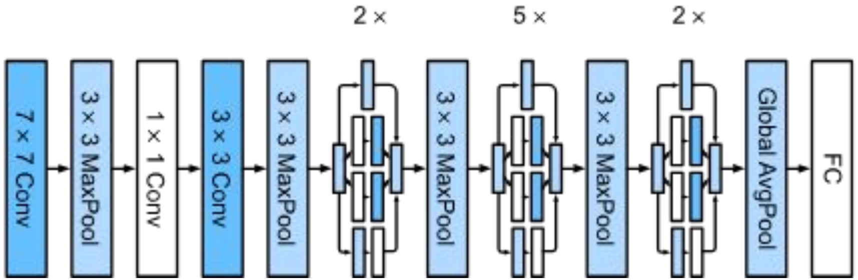
- they are sensible to ADVERSARIAL ATTACKS:



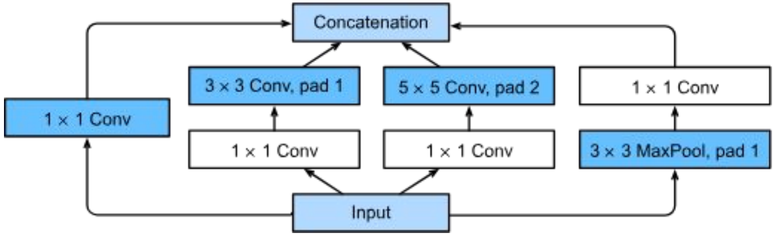
- contain unrealistic features
- require heavy computation for the training

GoogleNet (and the rise of inception blocks)

In 2014, *GoogLeNet* won the ImageNet Challenge ([Szegedy *et al.*, 2015](#))

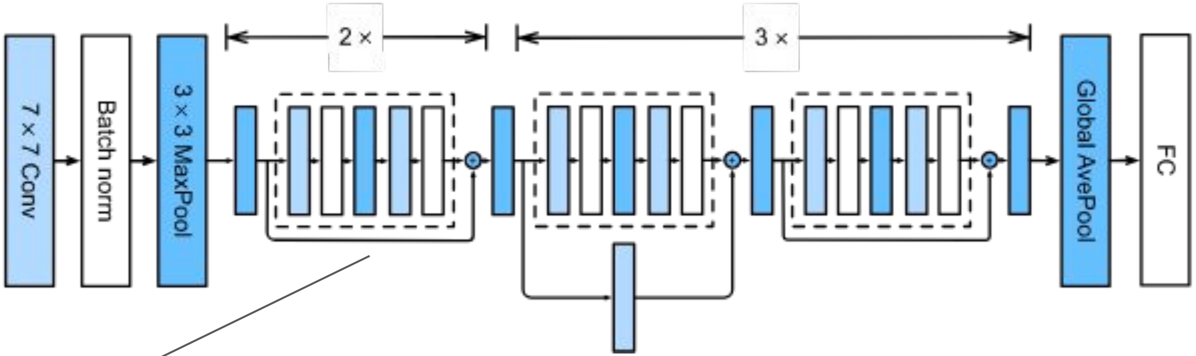


inception block

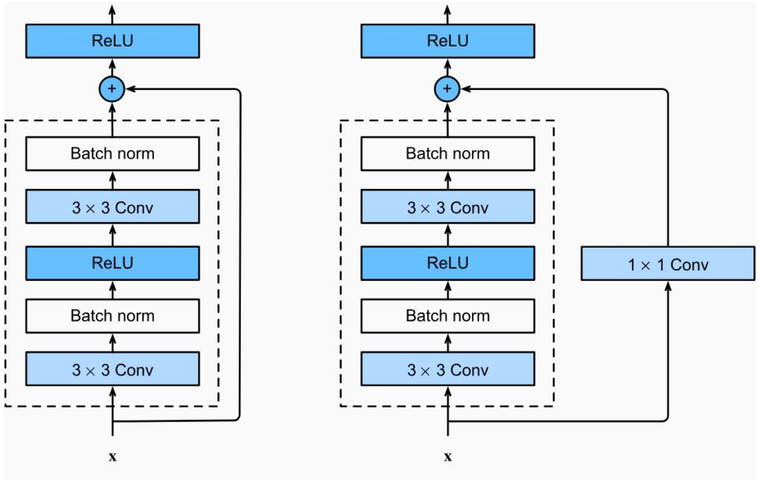


ResNet

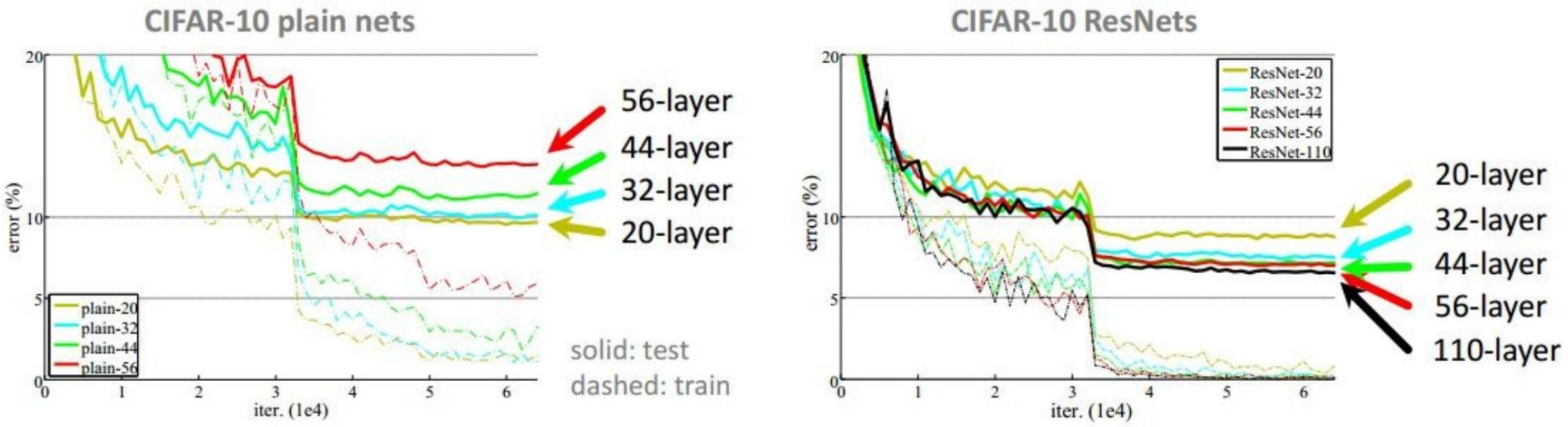
[He et al., 2015]



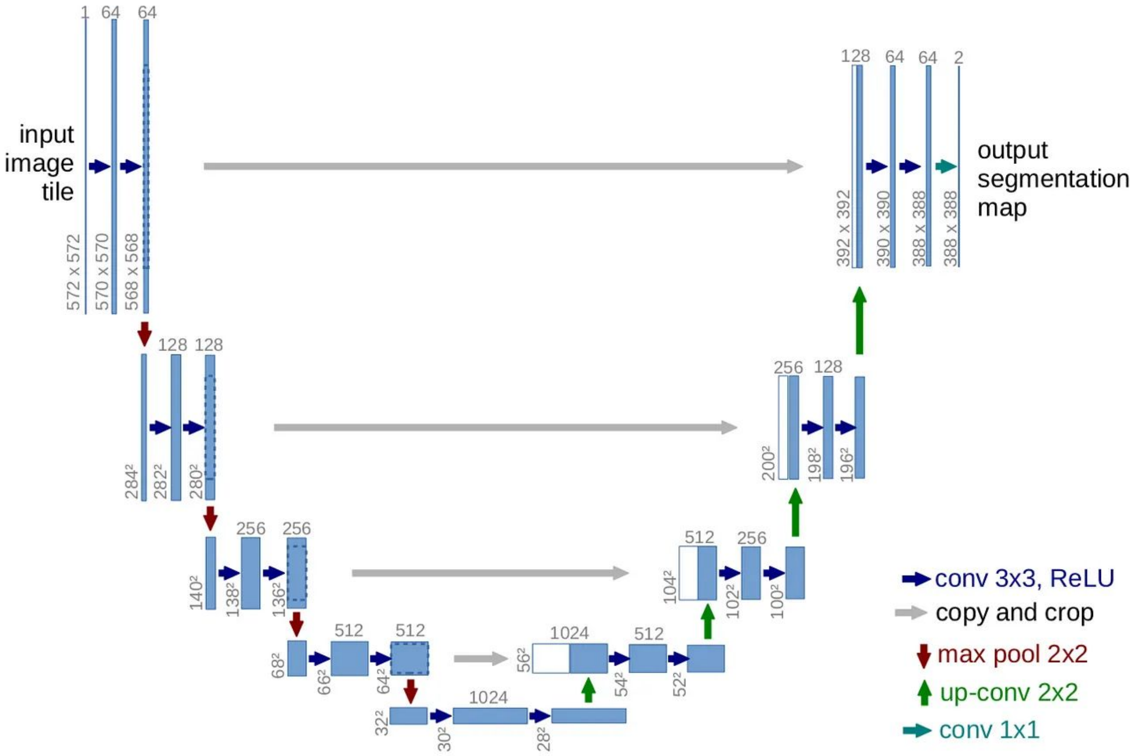
residual block



CIFAR experiments

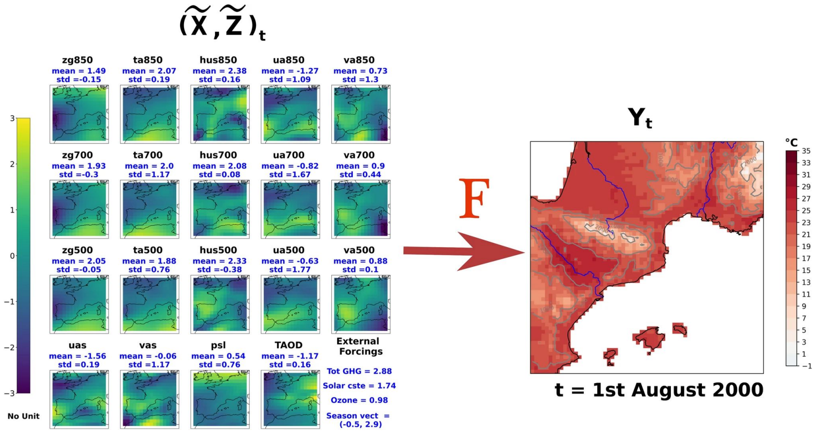


UNet


1 64 64
input image tile
572 x 572
570 x 570
568 x 568
128 128
284²
282²
280²
256 256
140²
138²
136²
512 512
68²
66²
64²
1024
32²
30²
28²
1024 512
56²
54²
52²
512 256
104²
102²
100²
256 128
200²
198²
196²
128 64 64 2
392 x 392
390 x 390
388 x 388
388 x 388
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

UNet applications in Climate studies

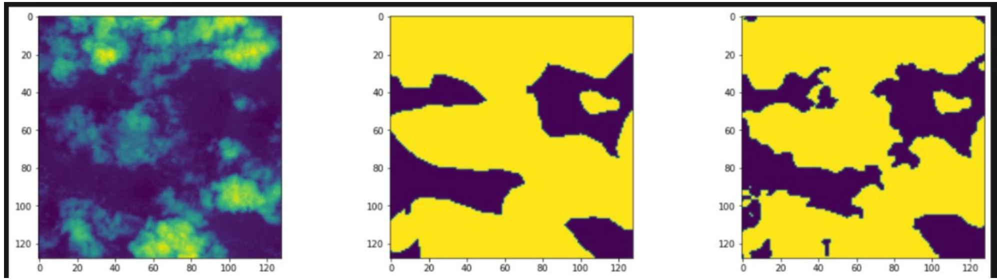
RCM emulator/downscaling



Doury et al. (2022-2024)

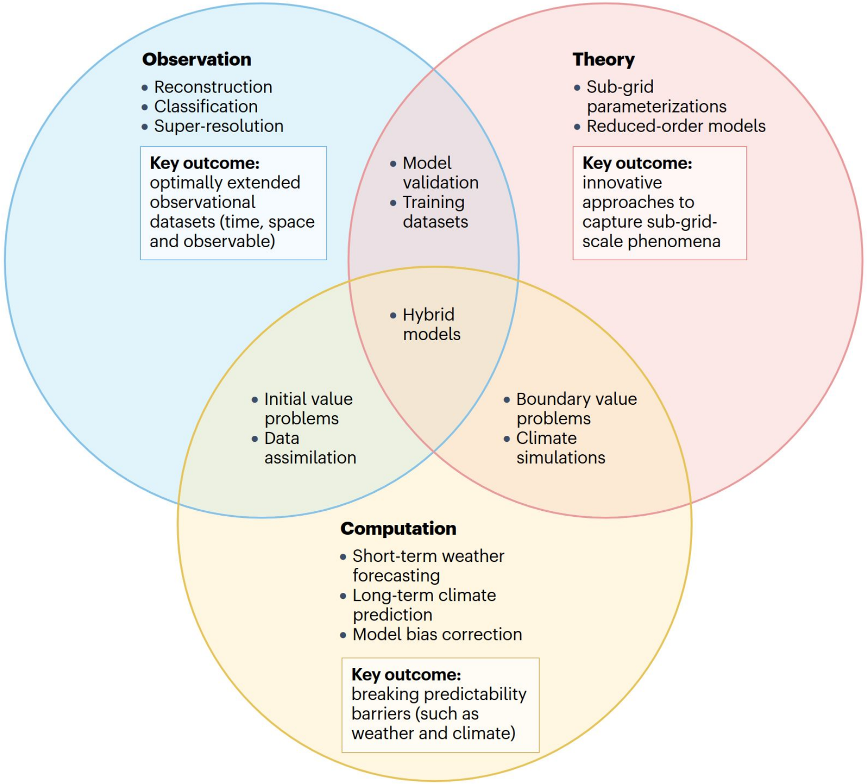
resol: 150 km -> 12.5 km

semantic segmentation in satellite data (e.g. clouds)

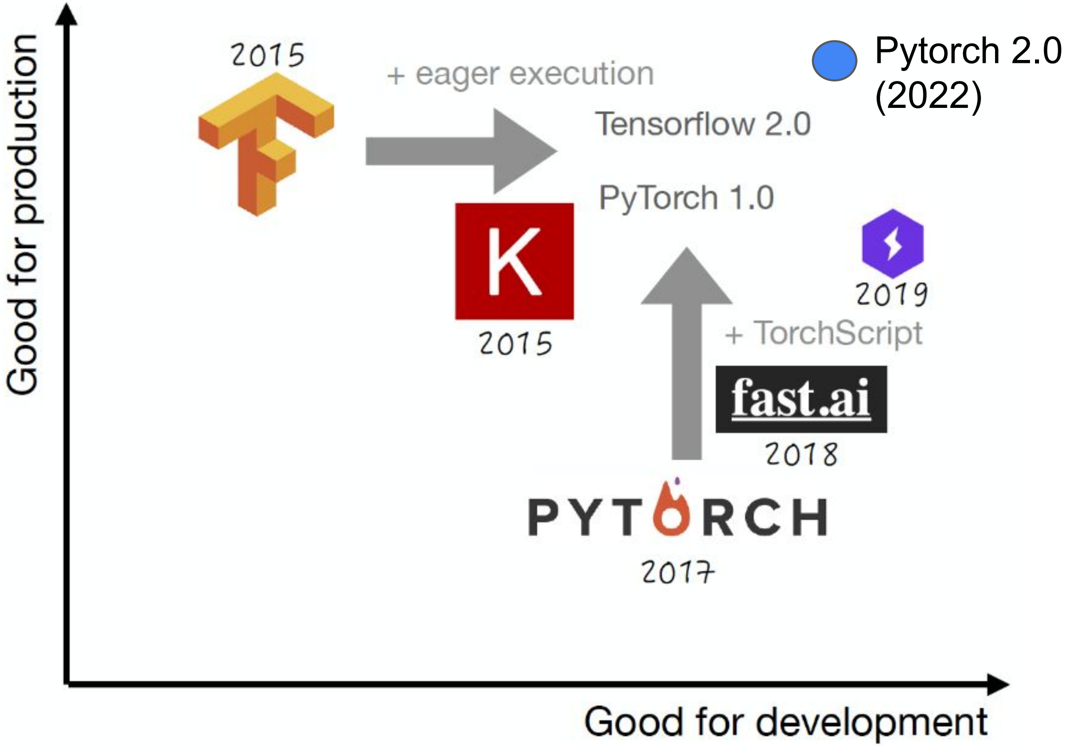


De Souza 2023

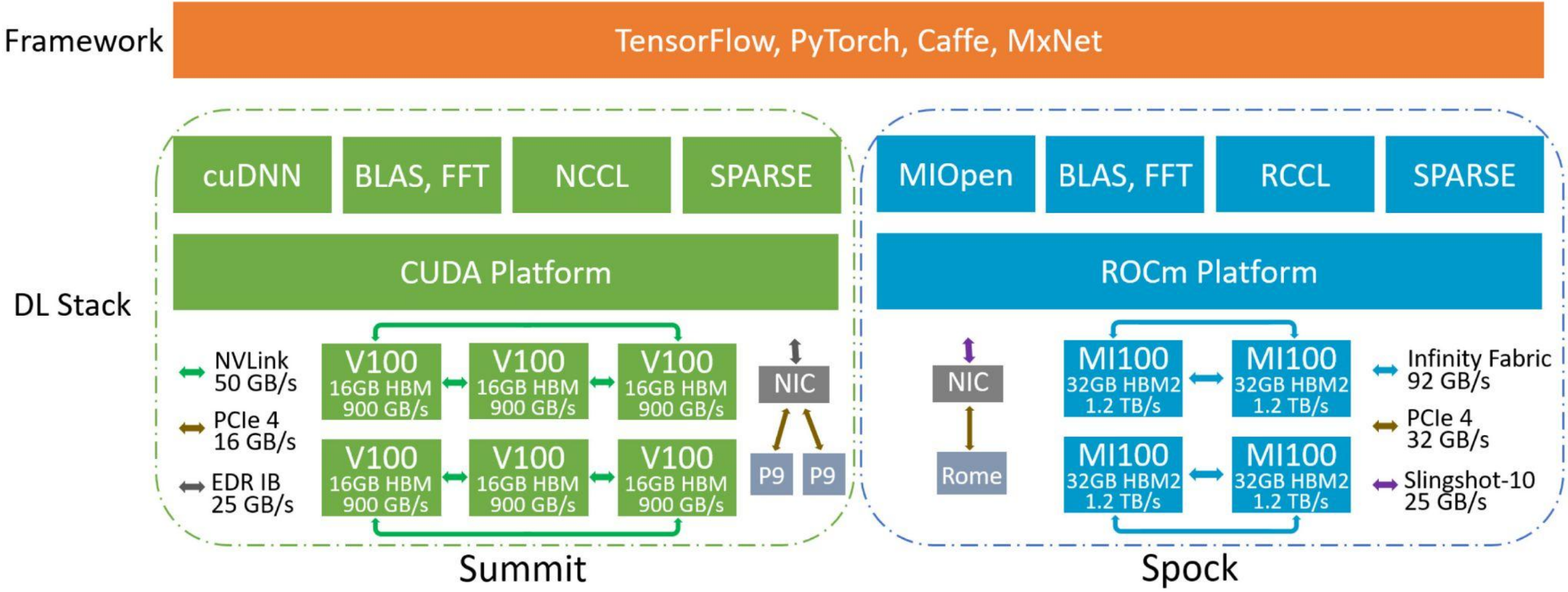
AI 4 climate


Observation
• Reconstruction
• Classification
• Super-resolution
Key outcome: optimally extended observational datasets (time, space and observable)
Theory
• Sub-grid parameterizations
• Reduced-order models
Key outcome: innovative approaches to capture sub-grid-scale phenomena
• Model validation
• Training datasets
• Hybrid models
• Initial value problems
• Data assimilation
• Boundary value problems
• Climate simulations
Computation
• Short-term weather forecasting
• Long-term climate prediction
• Model bias correction
Key outcome: breaking predictability barriers (such as weather and climate)

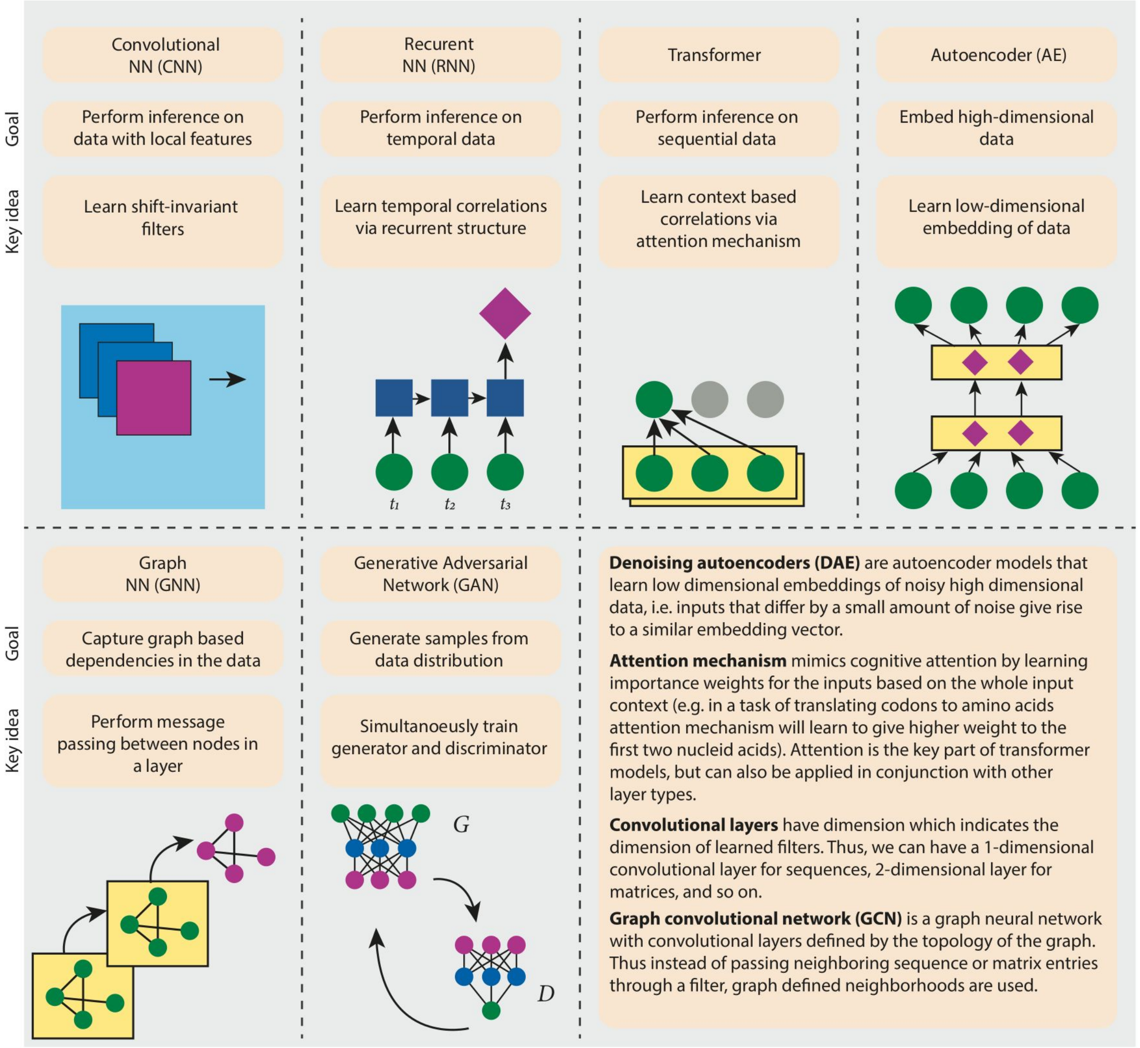
Python most used libraries/frameworks for DL



What’s behind Pytorch/Tensorflow?



Diving into the world of DL models



Why GPUs for DL?

1) **Neural networks are embarrassingly parallel algorithm**
2) **most of the operations performed in DL models can be rewritten as matrix multiplications**
3) **big datasets require to perform big matrix computation (**extremely slow on CPU with respect to GPU**)**
4) **well established libraries, with specific classes for ML objects (e.g. cuDNN, more recently tensorRT)**

and we know that GPUs are very good in solving specific parallel tasks (e.g matrix multiplication) , thanks to

- +1000 cores (>100K threads)
- **SIMD / SIMT**
- **high memory bandwidth**
- **newer GPUs have also tensor cores (particularly suited to tensor ops typical of NNs), and mixed precision**

However, also GPUs have limitations:

- GPUs might not be as efficient for extreme sparse networks, due to the overhead of managing sparse data structures.
- Some specialized sparse operations might not be as optimized as dense operations on GPUs.

Different strategies for Multi-GPUs training

we can identify 5 different categories of parallelism

`model parallelism`

`tensor parallelism`

`data parallelism`

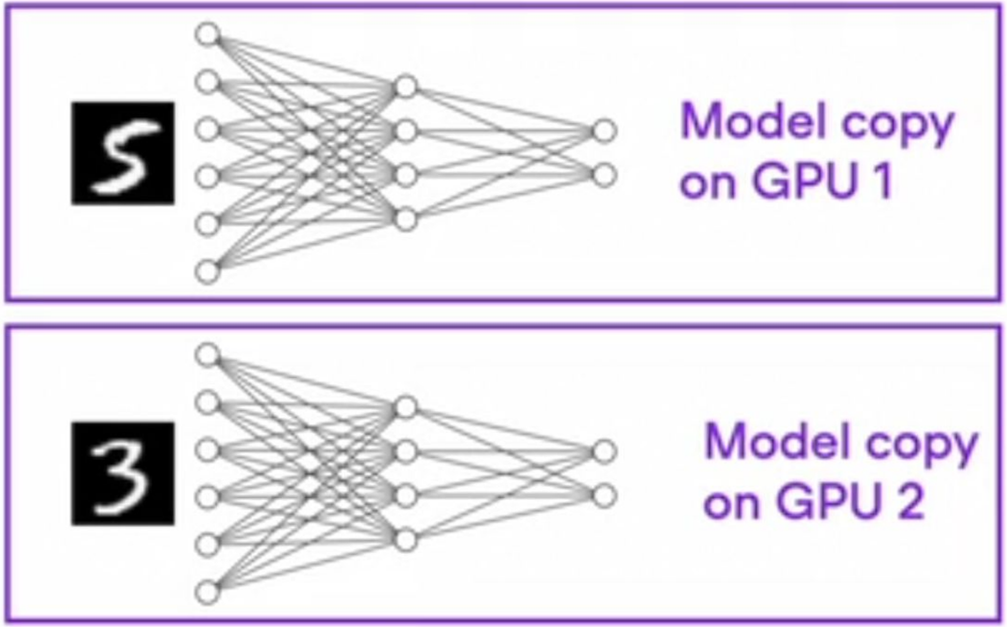
`sequence parallelism`

`pipeline parallelism`

Data Parallelism

In this framework we split batches to train DL model

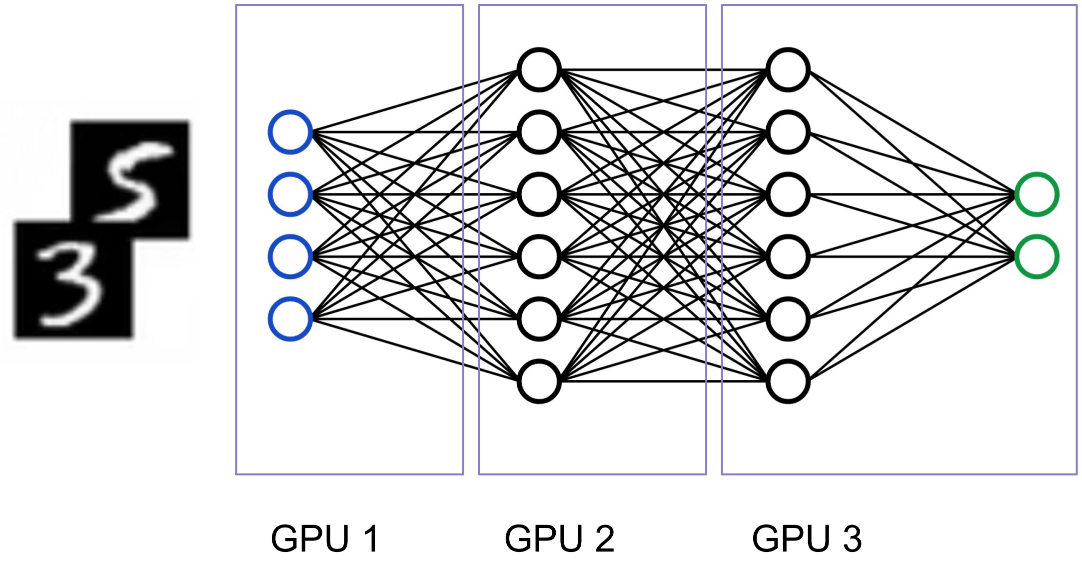
into different GPUs



Model parallelism

In this parallelism framework we choose to put different layers of the NN on different GPUs

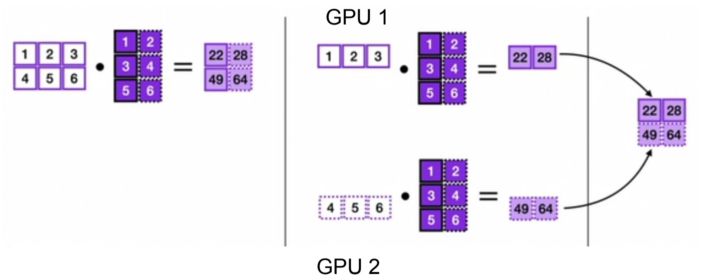
to work around GPU memory limits

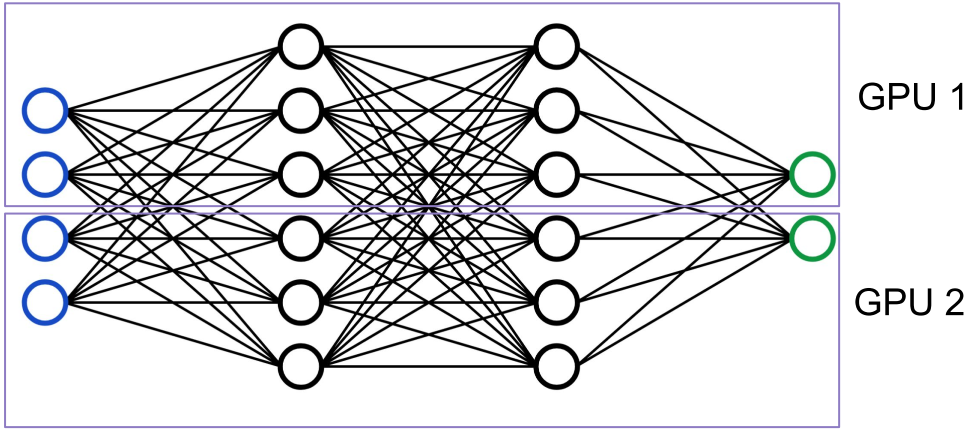


Tensor parallelism

In this framework we split the tensor operation done at each layer among different GPUs

similarly to what we would have done for matmul





More complex strategies for DL training

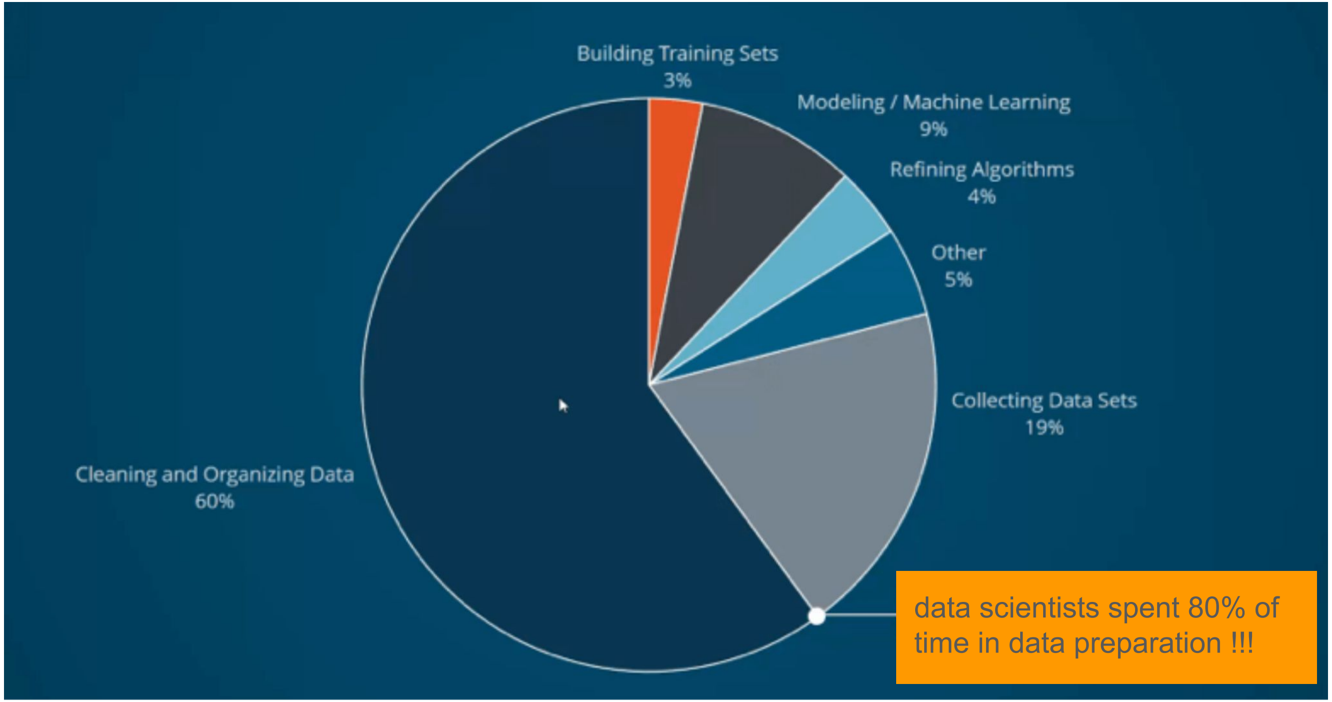
Sequence parallelism and pipeline parallelism frameworks

are obtained combining the previous approaches,

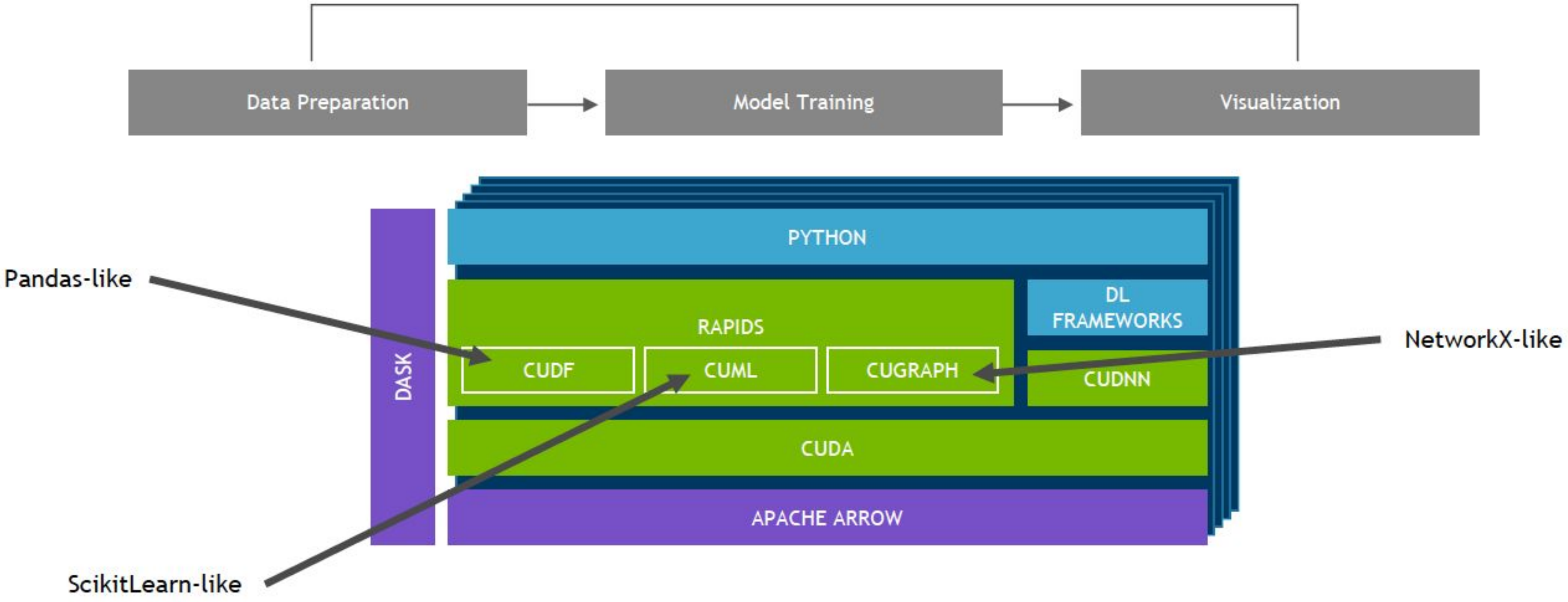
and are typically applied to DL models dealing with

spatio-temporal data.

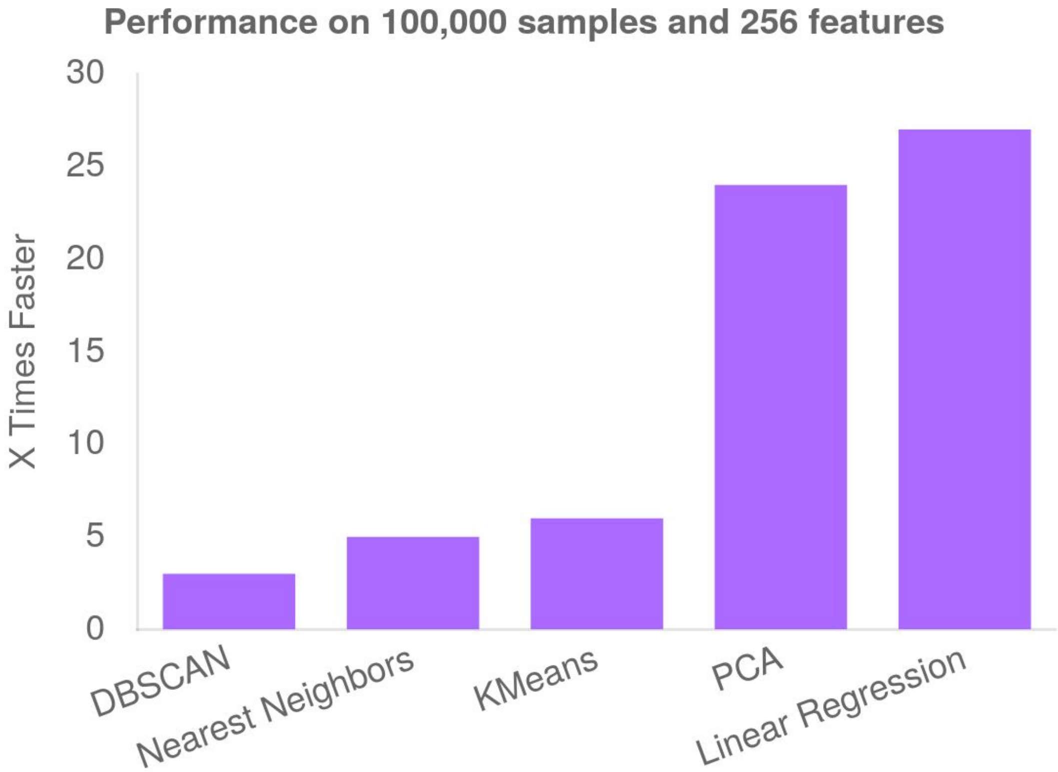
Do we need GPUs also for other ML tasks?



GPU-based libraries outside of Pytorch

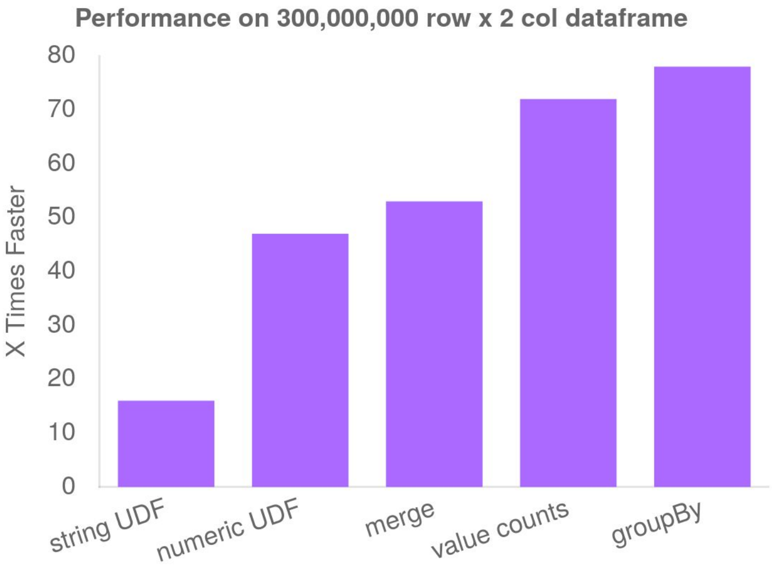


GPU for classical ML



* Benchmark on AMD EPYC 7642 (using 1x 2.3GHz CPU core) w/ 512GB and NVIDIA A100 80GB (1x GPU) w/ scikit-learn v1.2 and cuML v23.02

GPUs for data preprocessing



* Benchmark on AMD EPYC 7642 (using 1x 2.3GHz CPU core) w/ 512GB and NVIDIA A100 80GB (1x GPU) w/ pandas v1.5 and cuDF v23.02

References

Bishop, C. M. (1995). Neural networks for pattern recognition. Oxford university press.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.

Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions

Junqi Yin et. al, 2021, *Comparative evaluation of deep learning workloads for leadership-class systems*

NVIDIA Booklet on GPU development, 2021