The Abdus Salam International Centre for Theoretical Physics

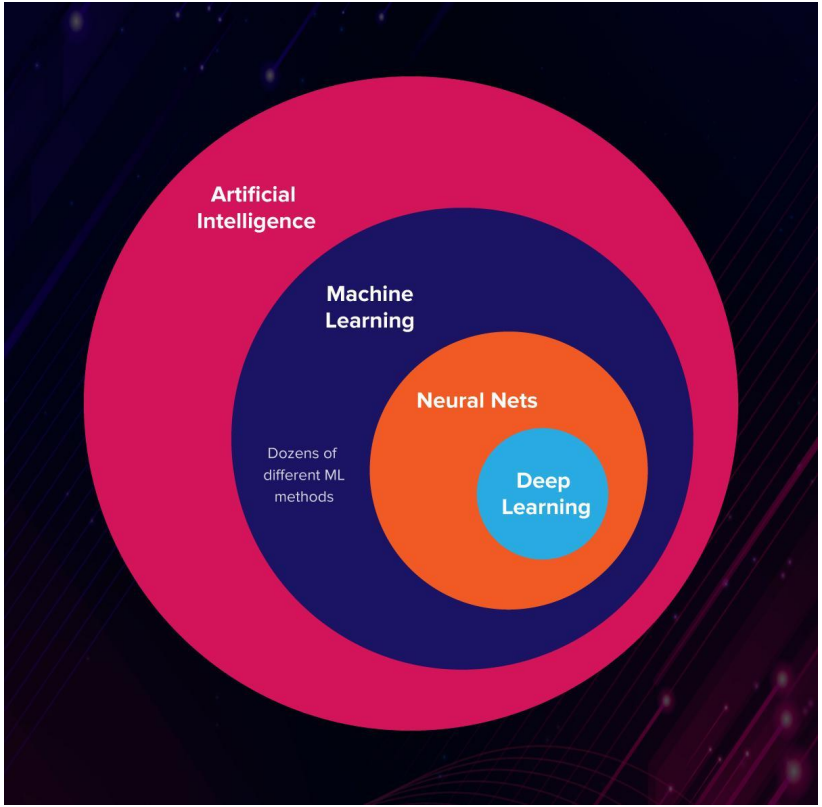
Intro to DL

Serafina Di Gioia
PostDoctoral Researcher @ ICTP

Where is DL in the picture?

Deep learning:

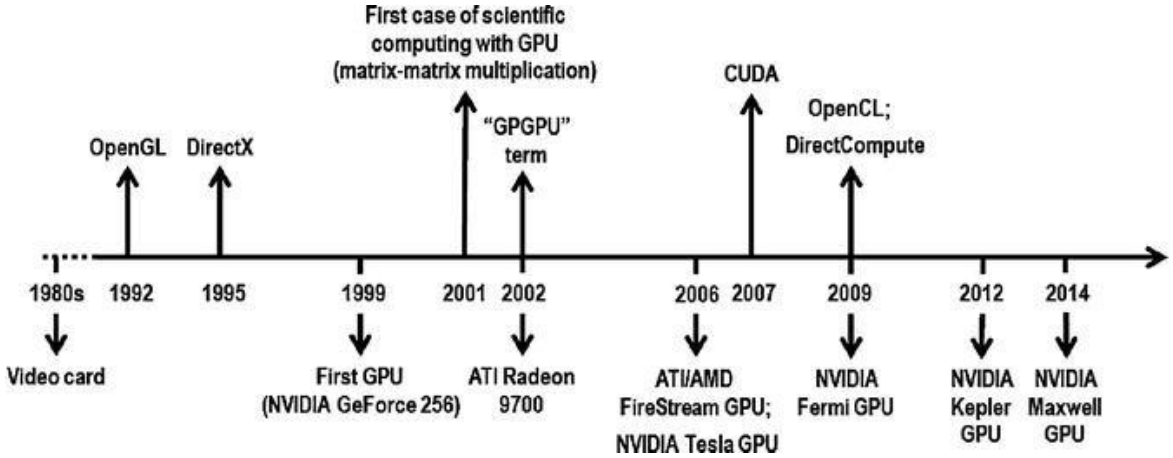
a type of machine learning based on artificial neural networks in which multiple layers of processing are used to extract progressively higher level features from data.



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



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)**

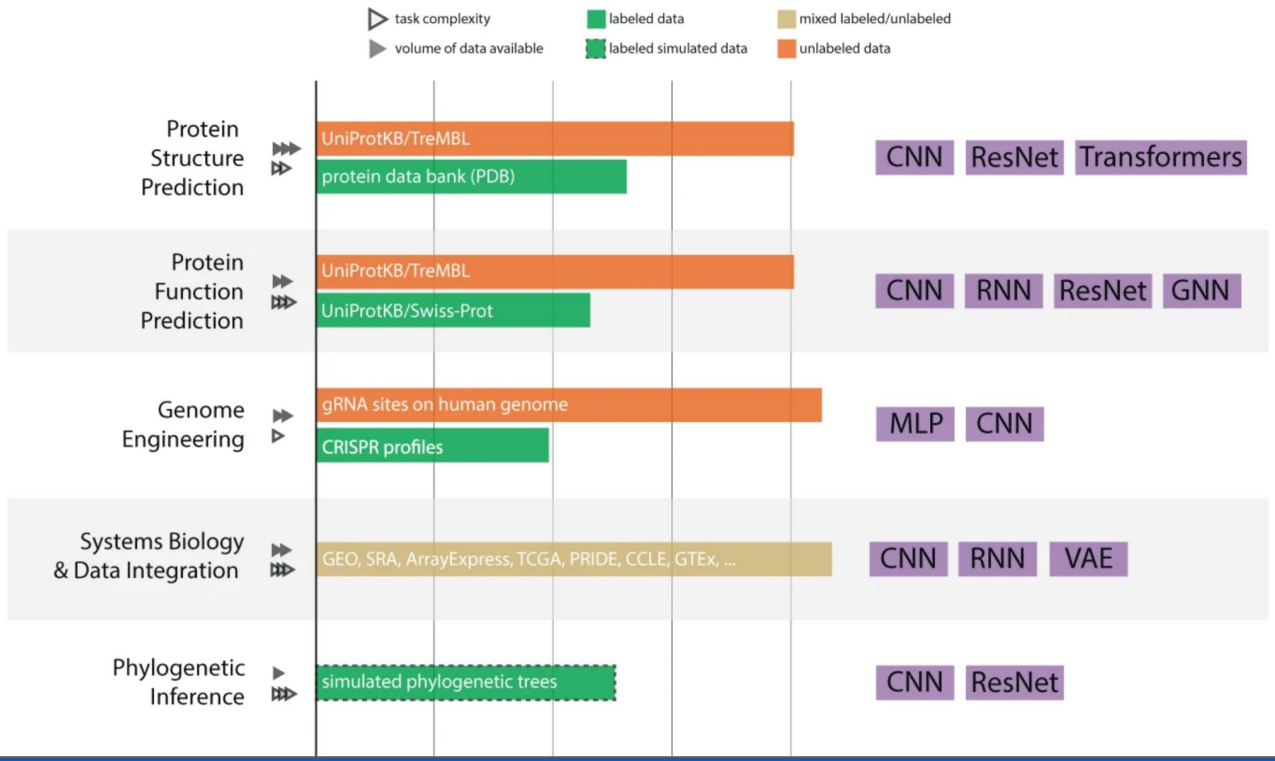
and we know that GPUs are very good in solving specific parallel tasks (e.g matrix multiplication) , thanks to

- +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:

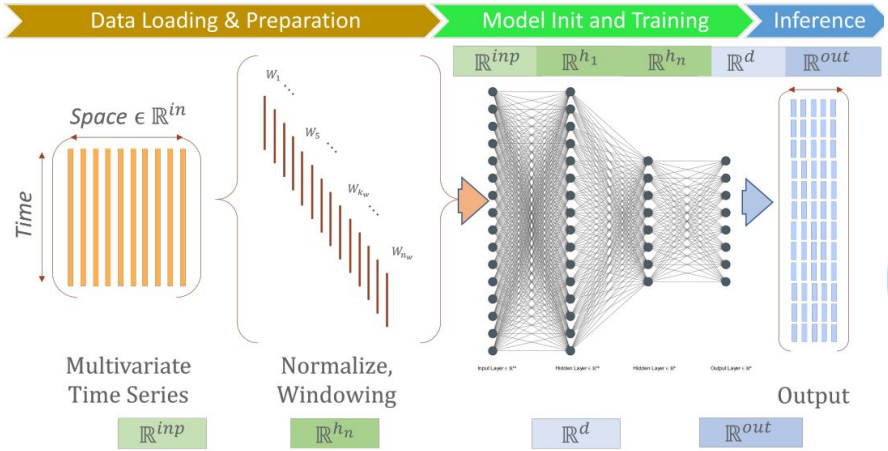
- GPUs might not be as efficient for extreme sparse networks, due to the overhead of managing sparse data structures.
- Some specialized sparse operations might not be as optimized as dense operations on GPUs.

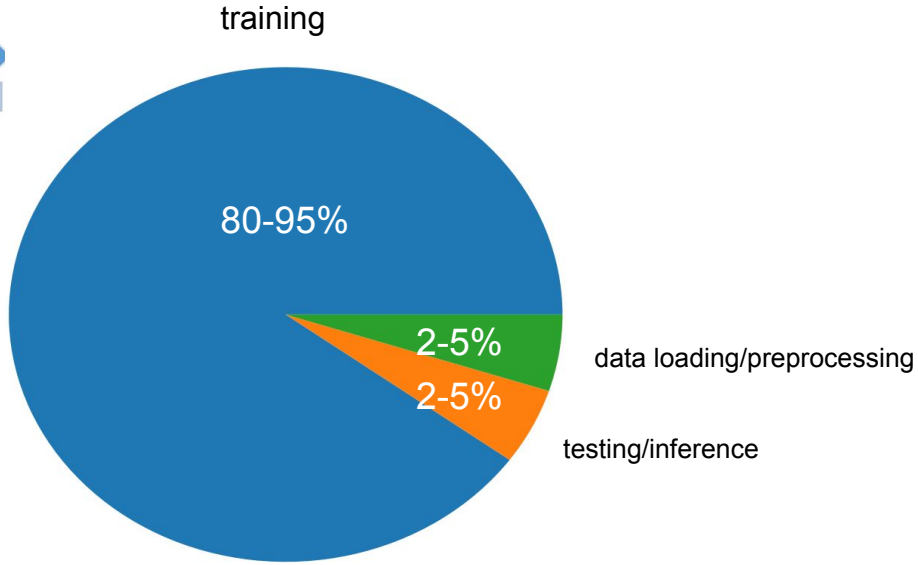
Typical sizes of DL data sets



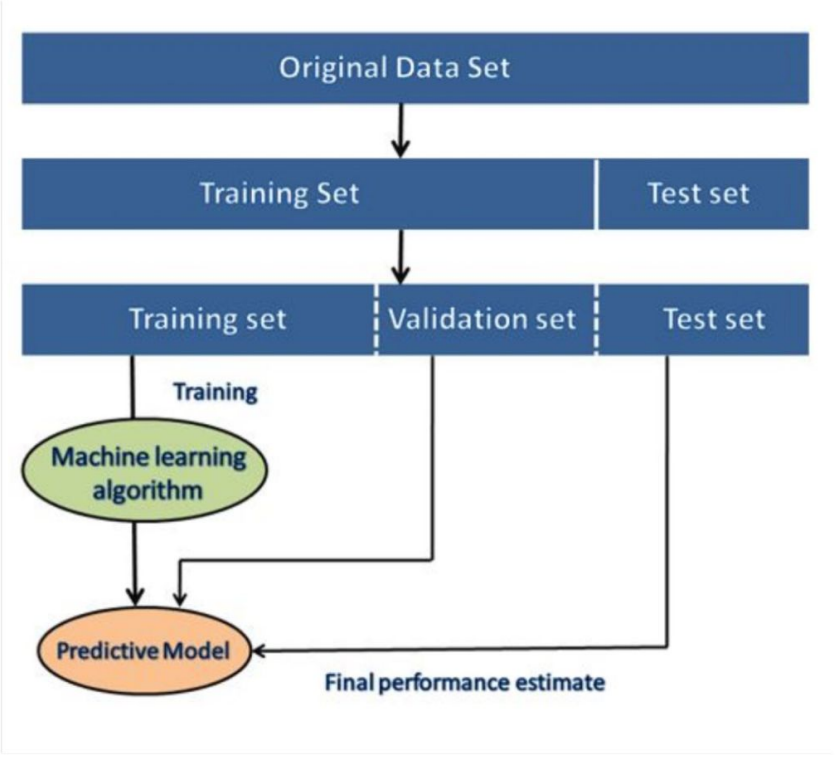
The increasing complexity of the new datasets, typical of big data epoch motivates the need for GPU-based libraries

Analyzing the computational workload of DL models





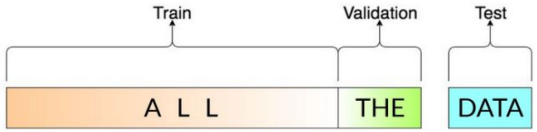
Training/Validation/Test splitting



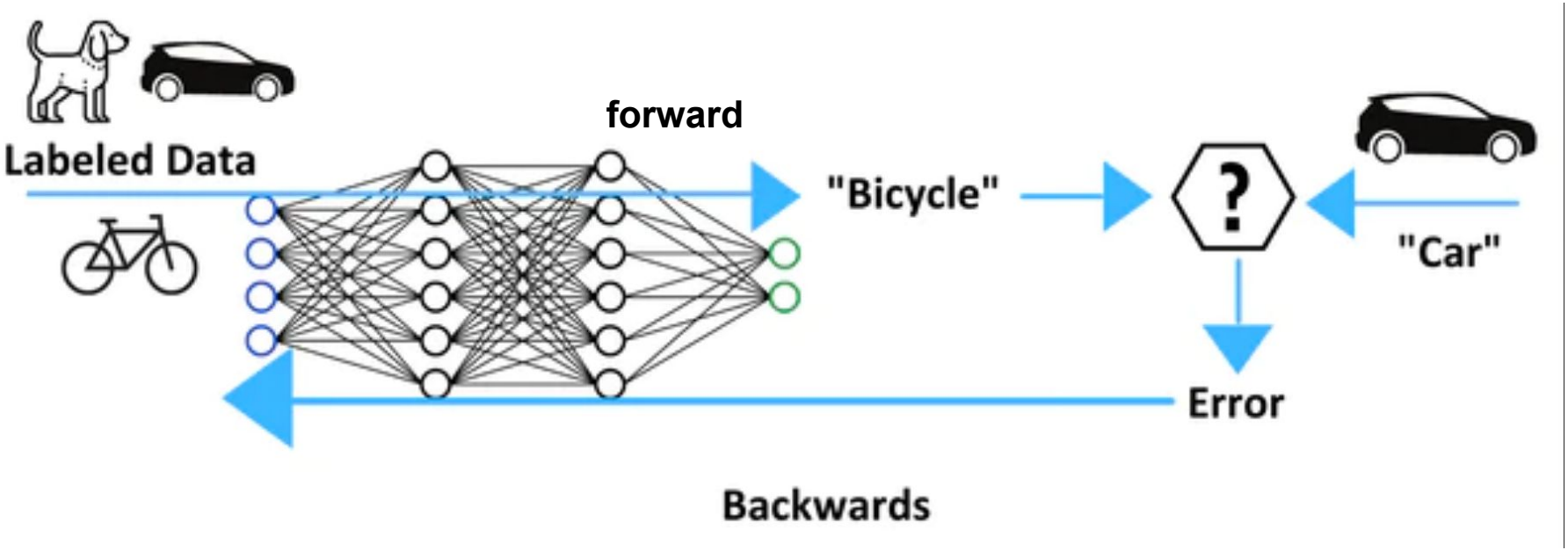
Usual split:

In Climate applications:

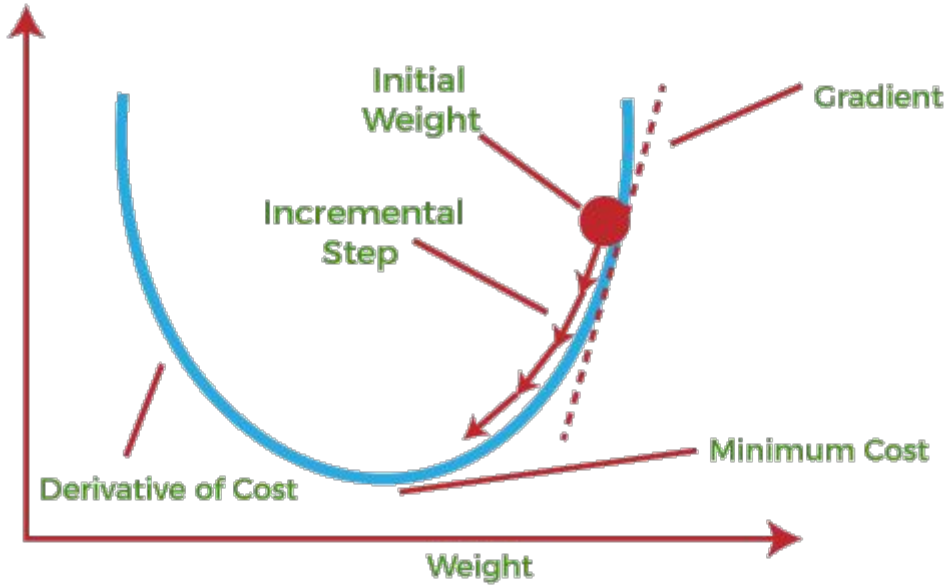
- Training: 1900-1980
- Validation: 1981-1990
- Testing: 1991- 2000



Training a DL model (in a supervised setting)



Solving optimization with Gradient descent



Given the cost function

$$J = -\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

During gradient descent the next location depends on the negative
gradient of J multiplied by the learning rate λ.

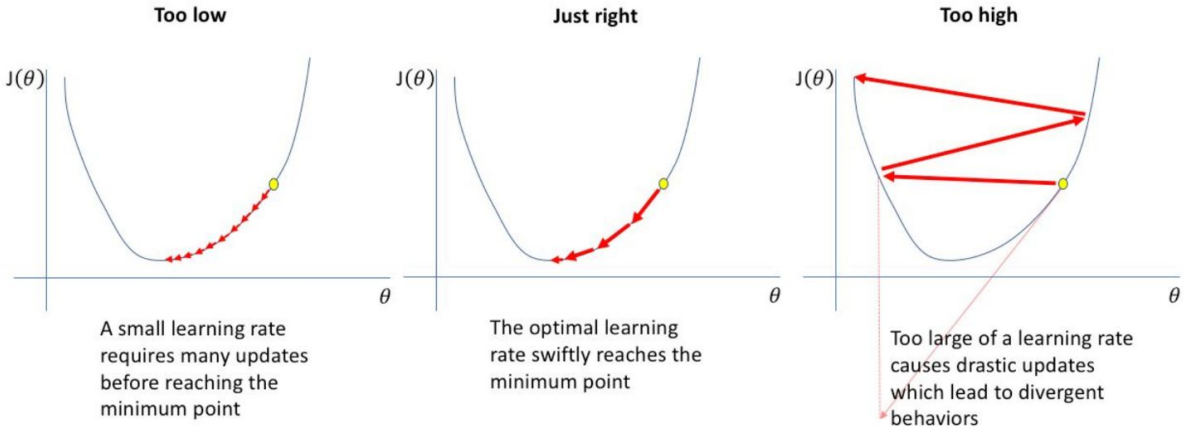
$$J_{t+1} = J_t - \lambda \nabla J_t$$

Importance of learning rate

Identify the right learning rate.

A learning rate too large may cause the solution to skip between local minima.

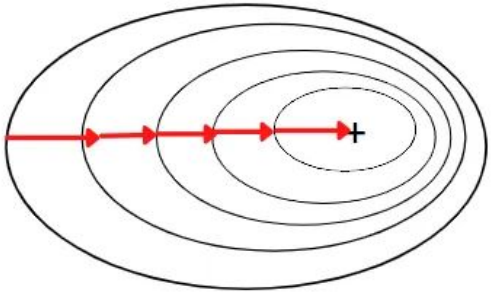
A learning rate too small may get stuck in one local minimum, and also may take a longer to learn.



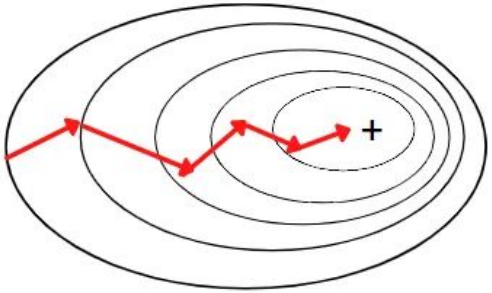
https://www.jeremyjordan.me/nn-learning-rate/

Gradient Descent variants

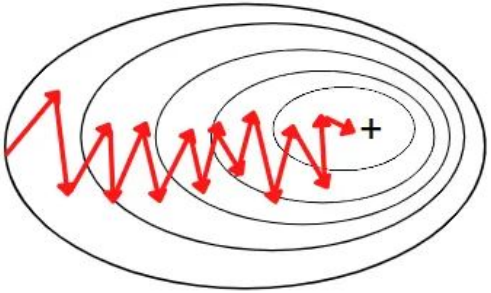
Batch Gradient Descent



Mini-Batch Gradient Descent



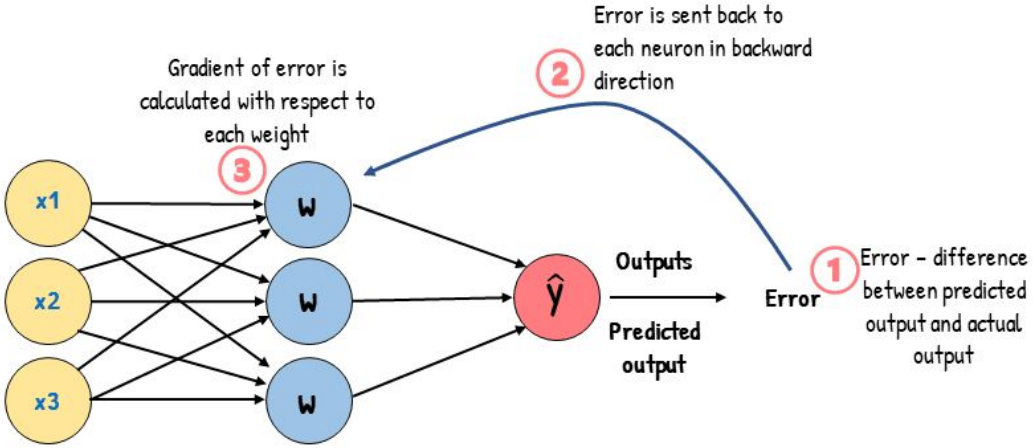
Stochastic Gradient Descent (SGD)



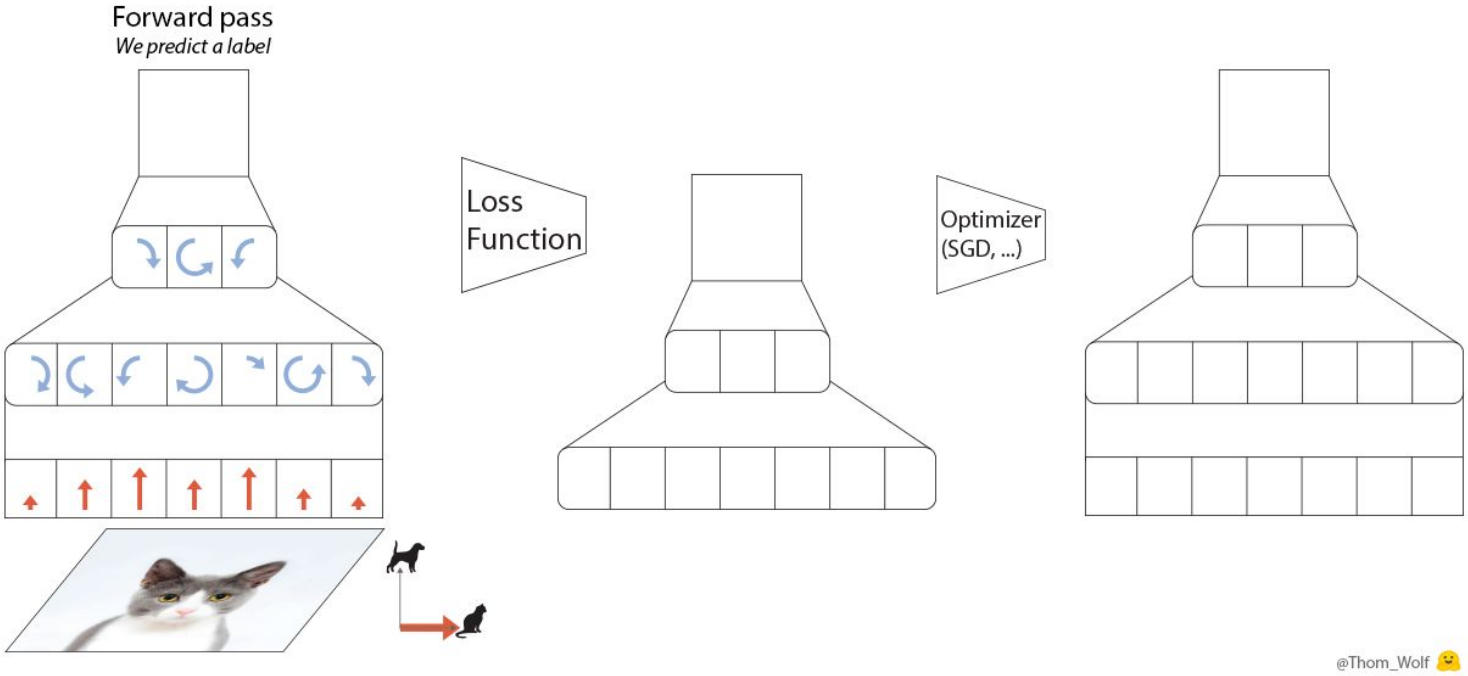
Back-propagation

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.



Training of a DL model in action



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.

Hyper-parameters importance

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

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

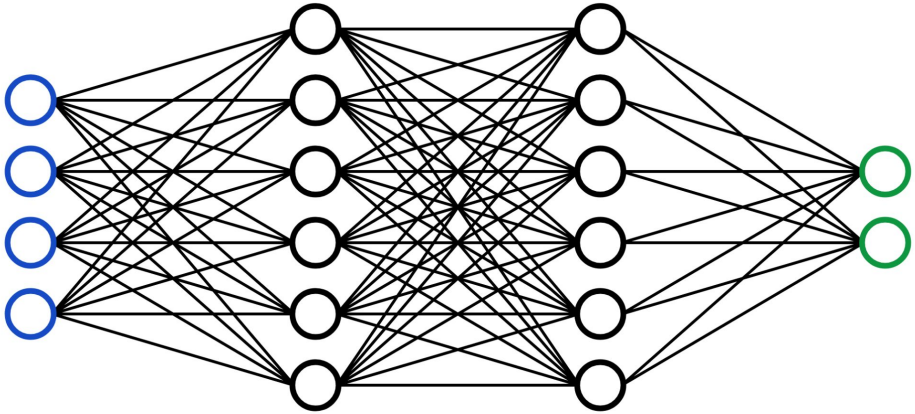
How to choose them?

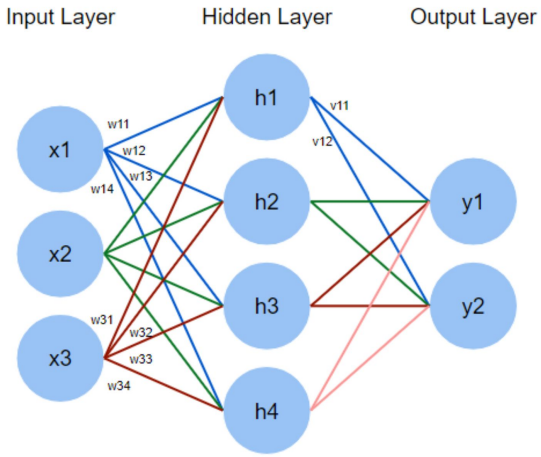
• Experience

• Trial and error

or AutoML

Feed-forward Neural Network (INN)





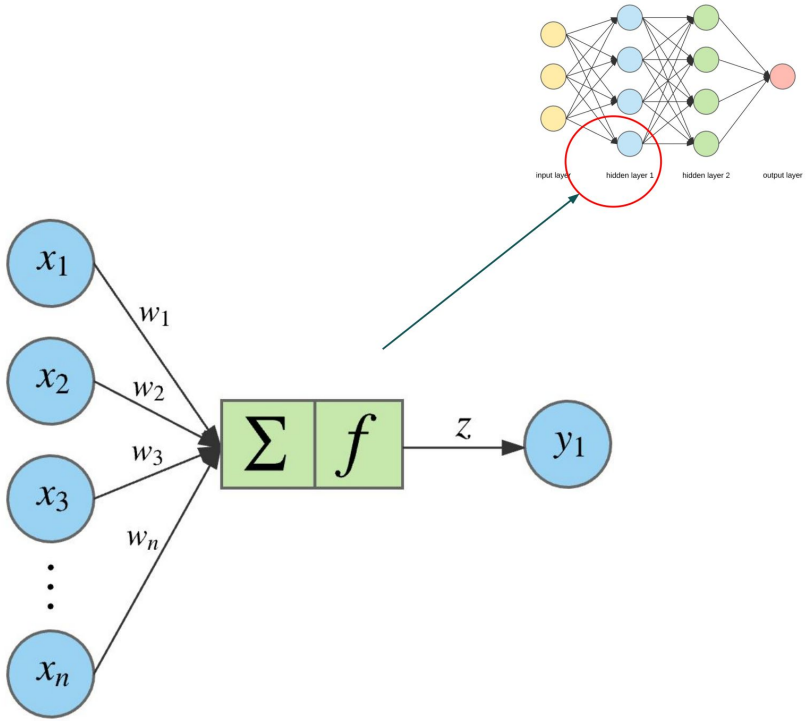
$$[\,x_1 \quad x_2 \quad x_3\,] * \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \end{bmatrix} = [\,h'_1 \quad h'_2 \quad h'_3 \quad h'_4\,]$$

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

What happens in the hidden layers?

In each individual node the values coming in are weighted and summed together with the bias terms and the sum is multiplied by activation:

$$z = f_{activation}(w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n + b)$$



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 (e.g. MSE, RMSE, MAE for regression

  Cross-Entropy or FocaLoss for classification)

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).

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.

Feed-forward NN in pytorch

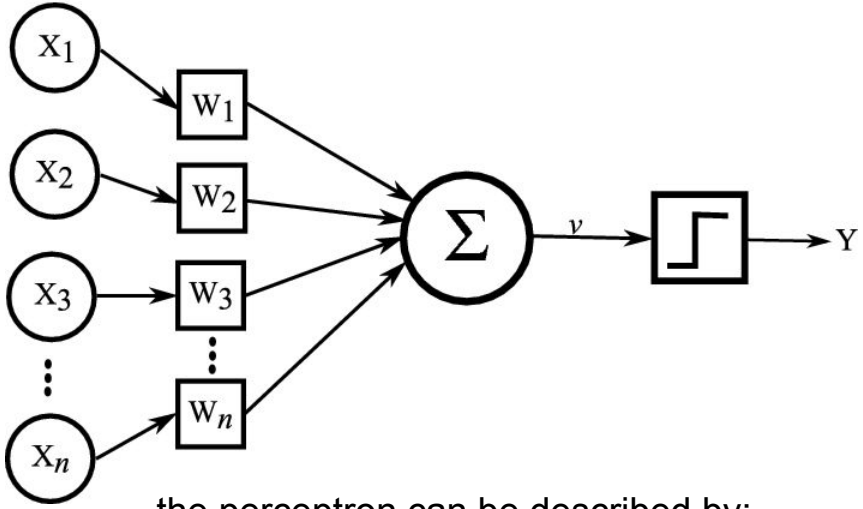
```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

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."



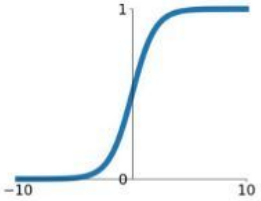
the perceptron can be described by:

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

Types of activation functions

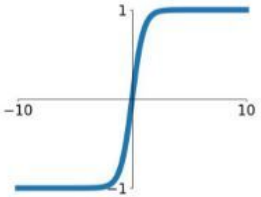
Sigmoid

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



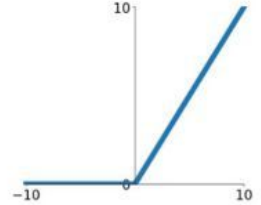
tanh

$$\tanh(x)$$



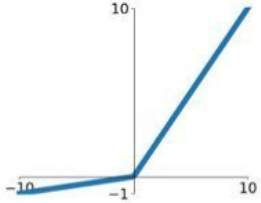
ReLU

$$\max(0, x)$$



Leaky ReLU

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

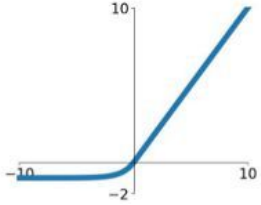


Maxout

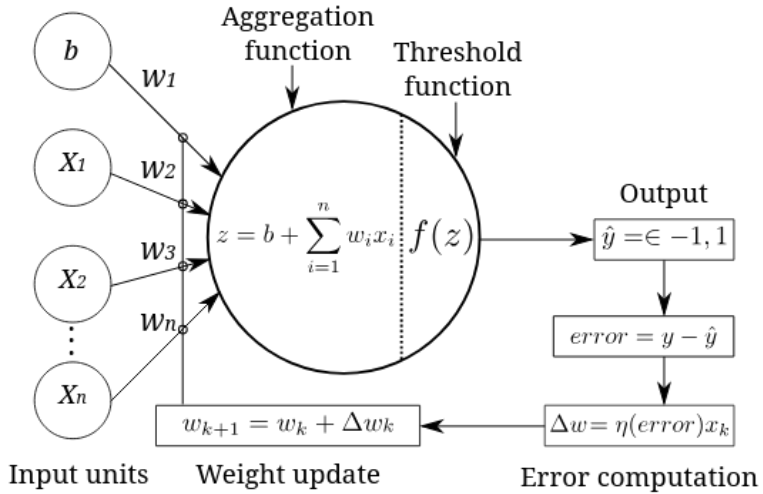
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

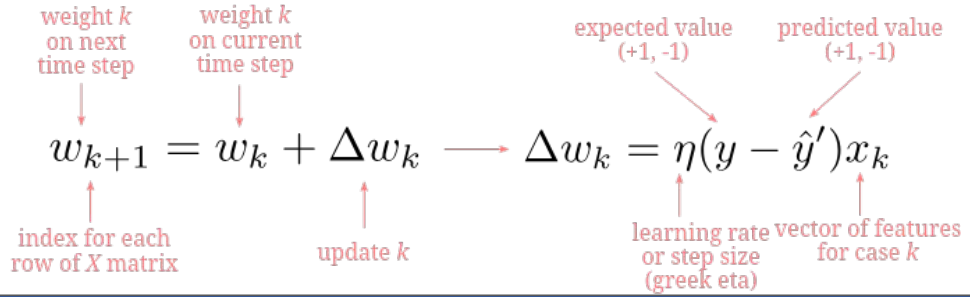
ELU

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

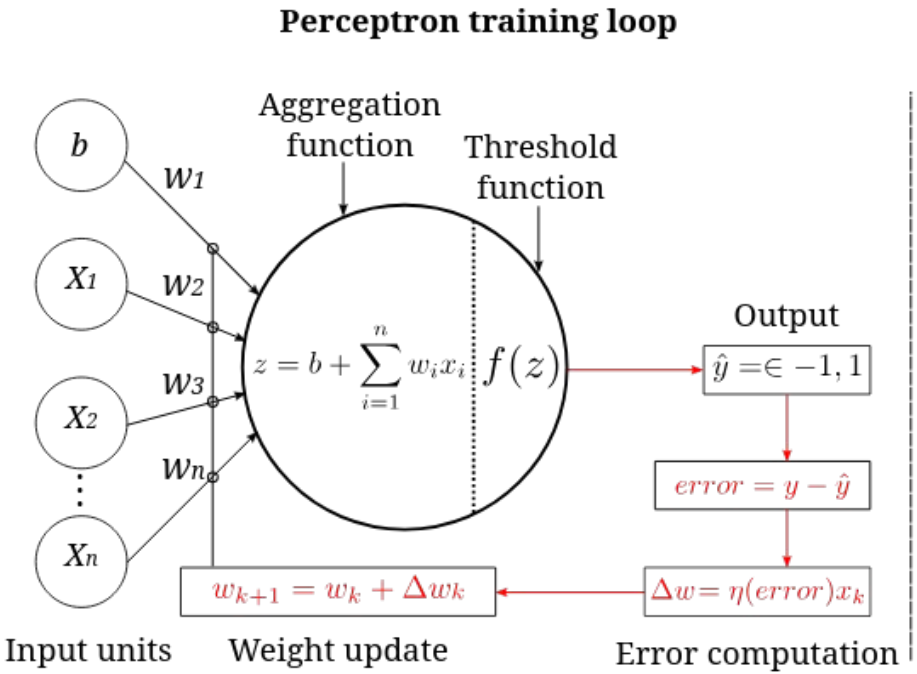


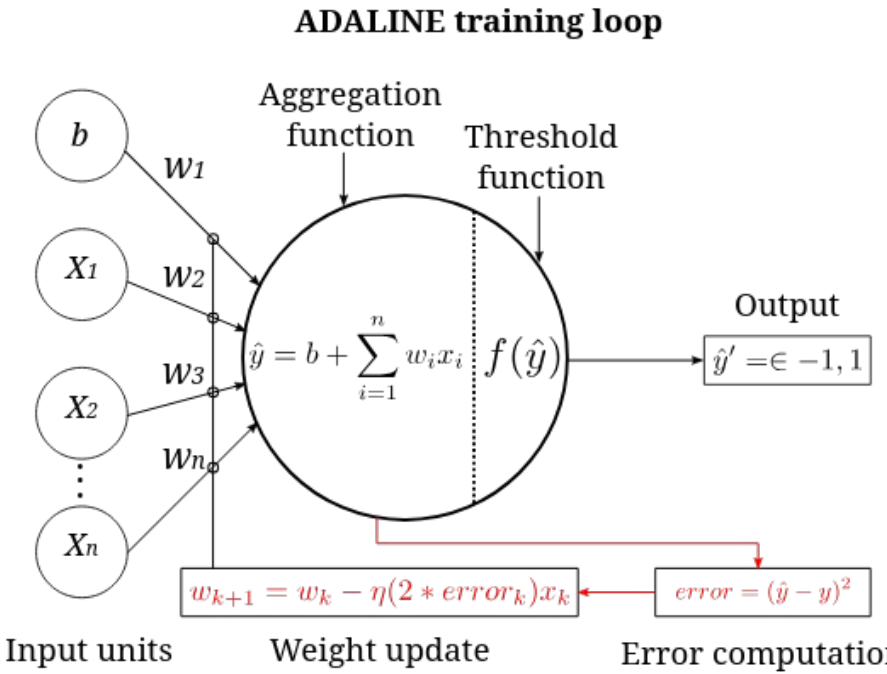
Learning procedure for the Perceptron





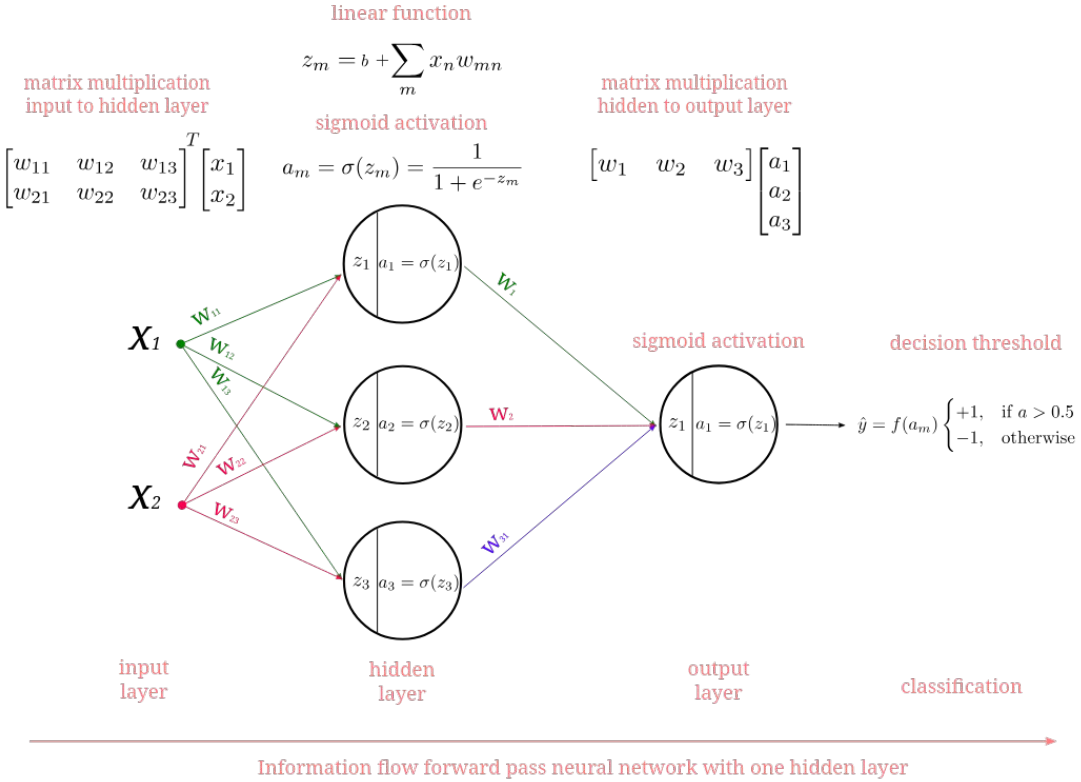
Adaline vs Perceptron



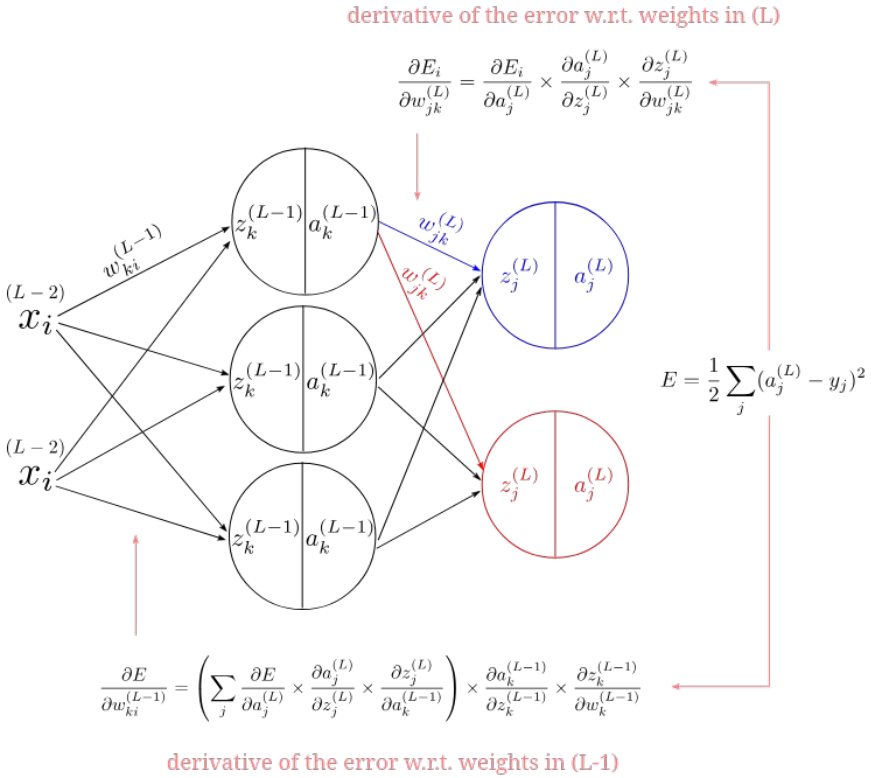


it introduces SGD in the training

Multi-layer Perceptron



Back-propagation in multi-layer perceptron



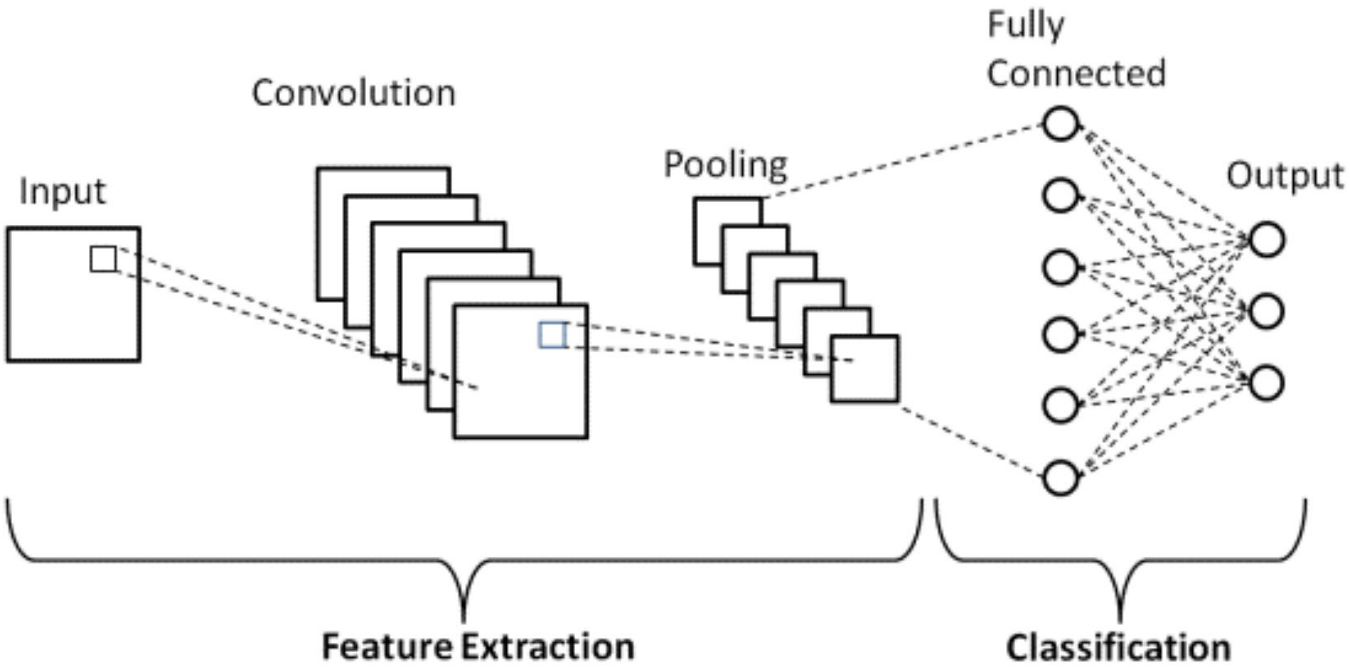
Formula for the weights update in the L-layer:

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

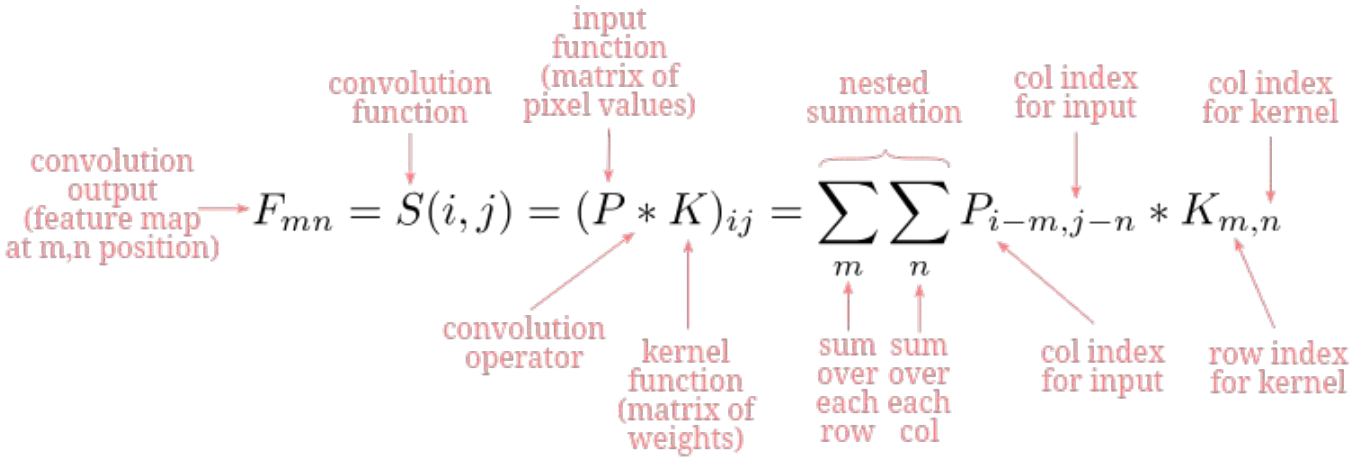
Formula for the bias update in the L-layer:

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

Convolutional Neural Networks (CNN)



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

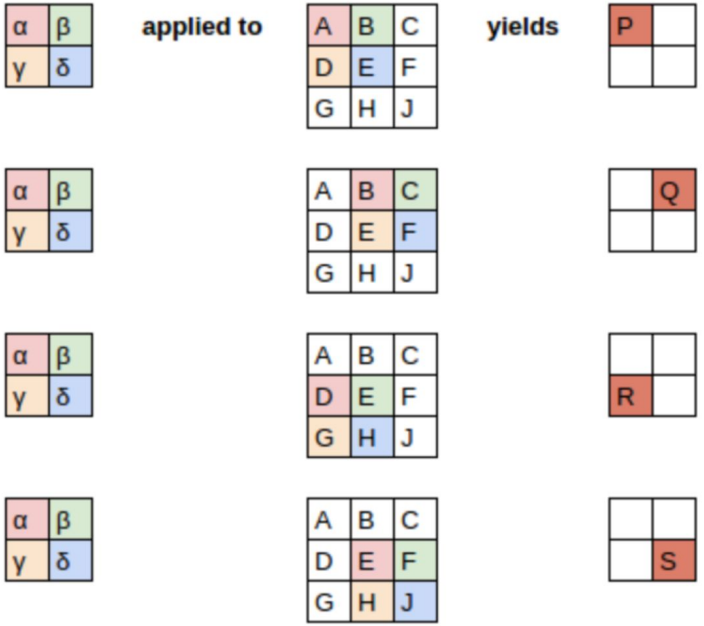
Let's take a look at basic element of CNN: convolution layer

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

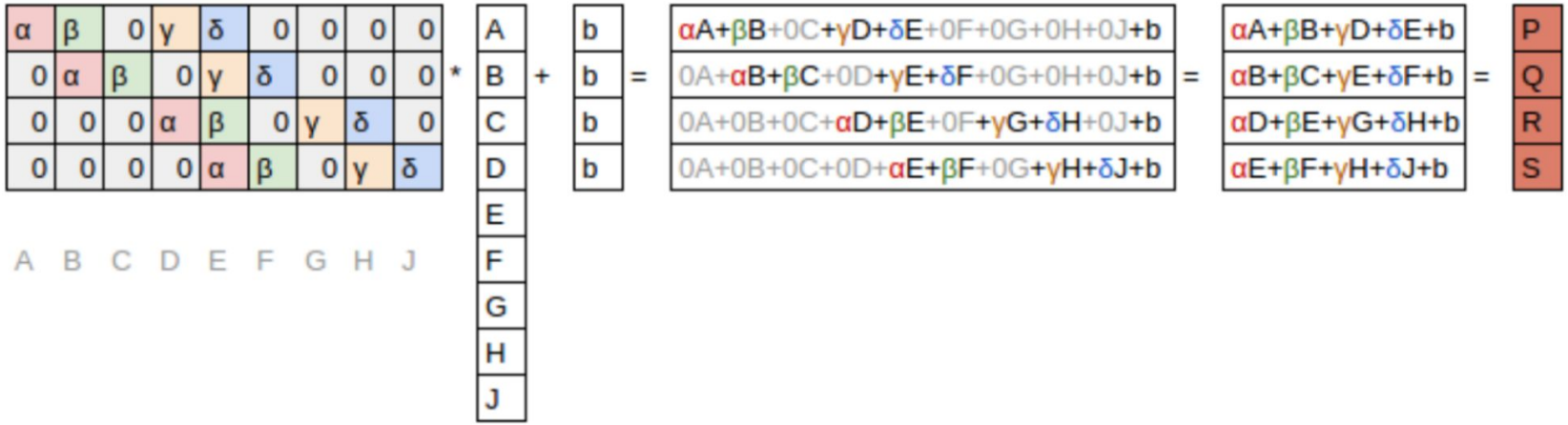
α	β
γ	δ

to a (3,3) matrix:

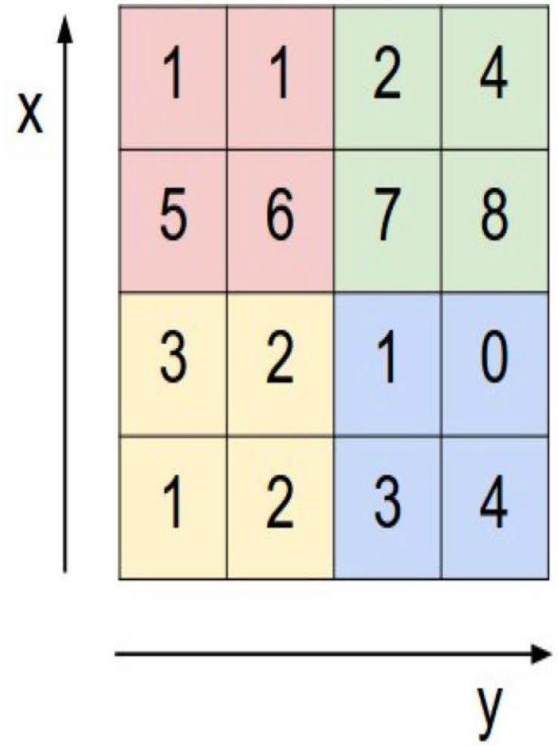
A	B	C
D	E	F
G	H	J



The convolution can be rewritten as



Pooling



max pool with 2x2 filters and stride 2

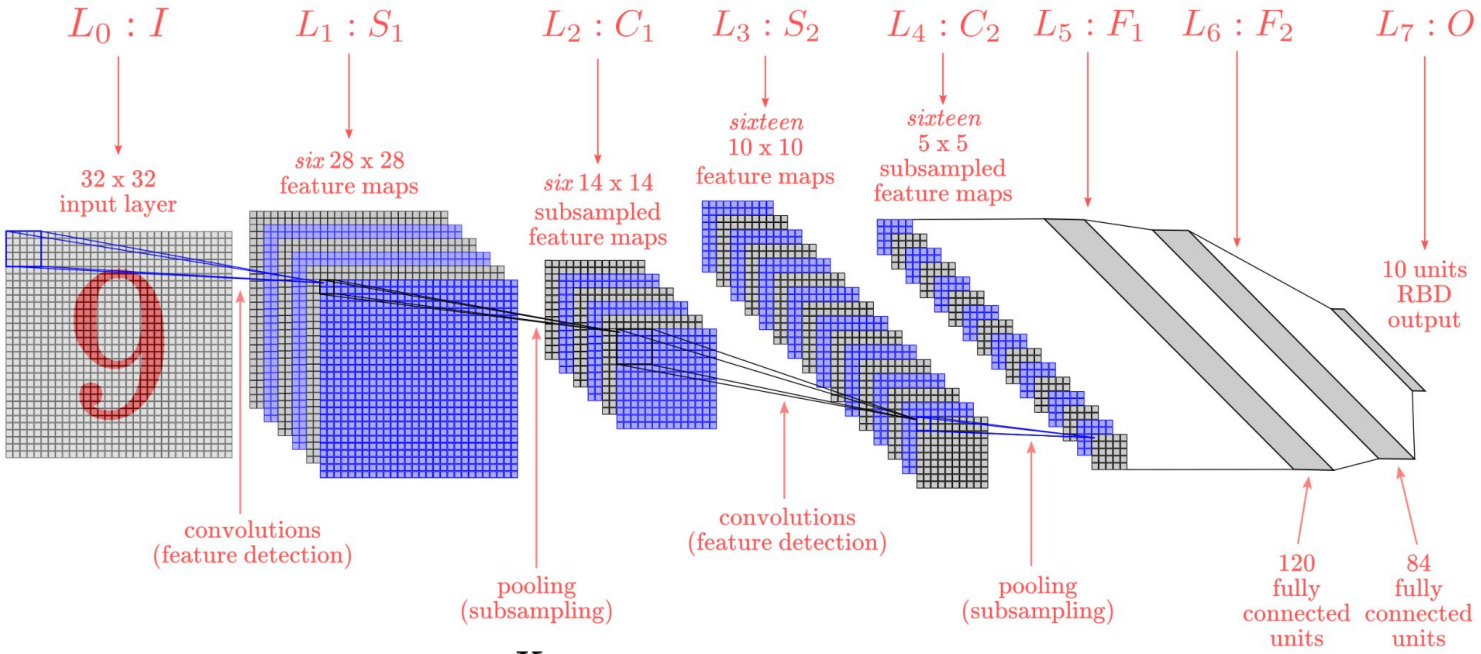
```
tf.keras.layers.MaxPool2D(
    pool_size=(2,2),
    strides=2
)
```

6	8
3	4

1) Reduced dimensionality
2) Spatial invariance

Max Pooling, average pooling

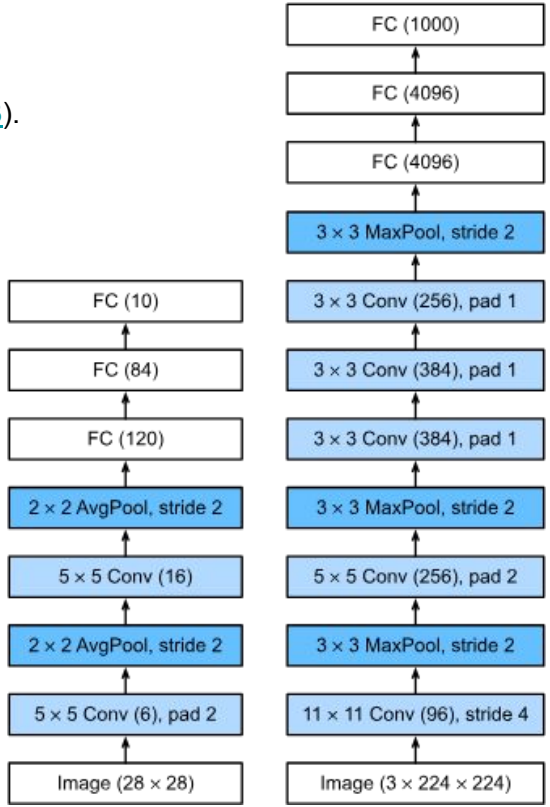
LeNet-5



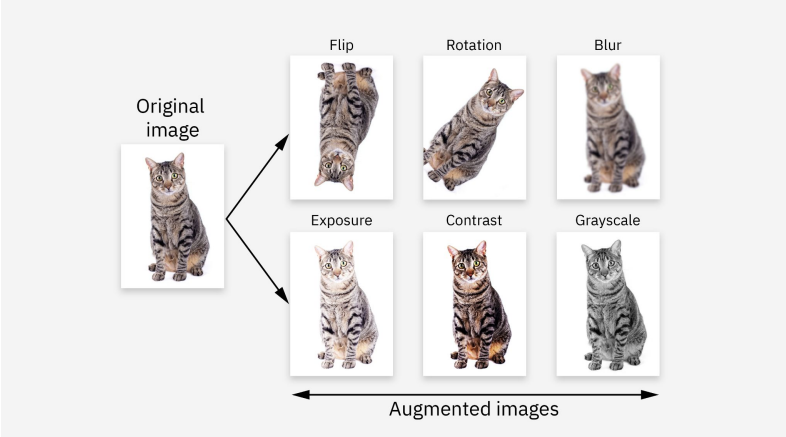
$$E(W) = \frac{1}{P}\sum_{p=1}^{X}(\hat{y}_{D^p}(X^p, W) + \log(e^{-j} + \sum_{i} e^{-\hat{y}(X^p, W)}))$$

AlexNet

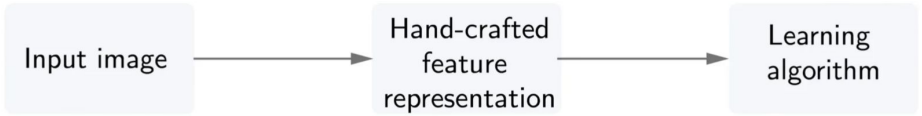
(Russakovsky *et al.*, 2013).



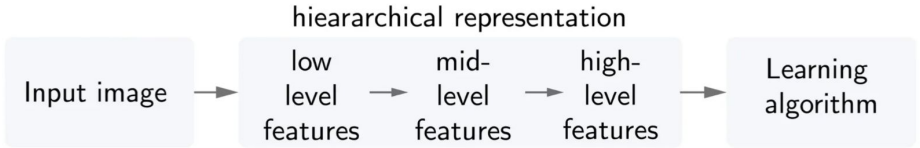
The training procedure of AlexNet used for the first time data augmentation:



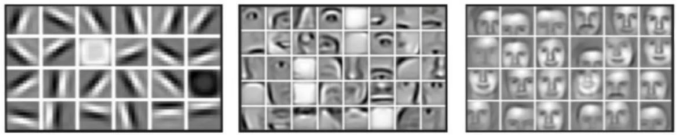
Hierarchical representation learning



Traditional pattern recognition
e.g., SIFT, HOG

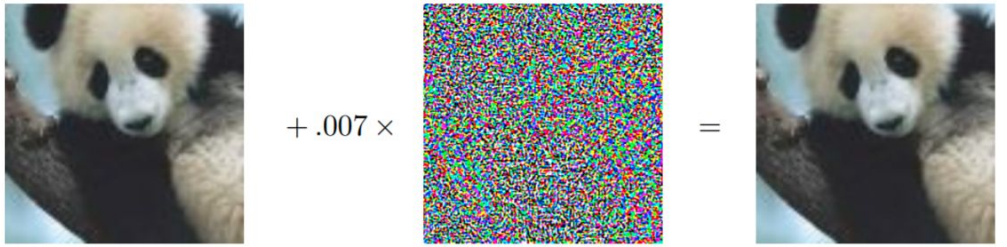


Deep learning



Limitations of CNNs

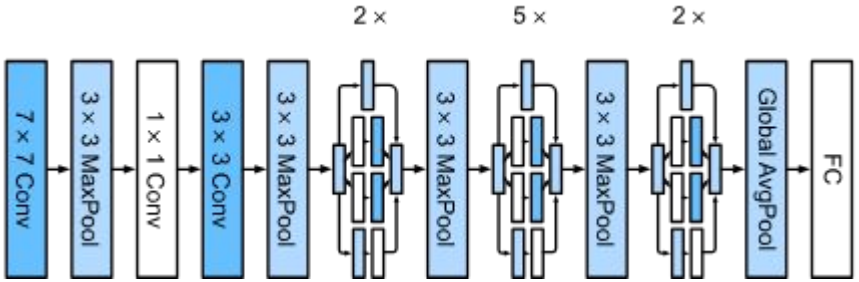
- they are sensible to ADVERSARIAL ATTACKS:



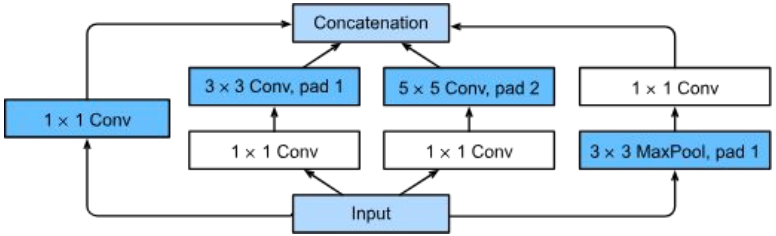
- contain unrealistic features
- require heavy computation for the training

GoogleNet (and the rise of inception blocks)

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

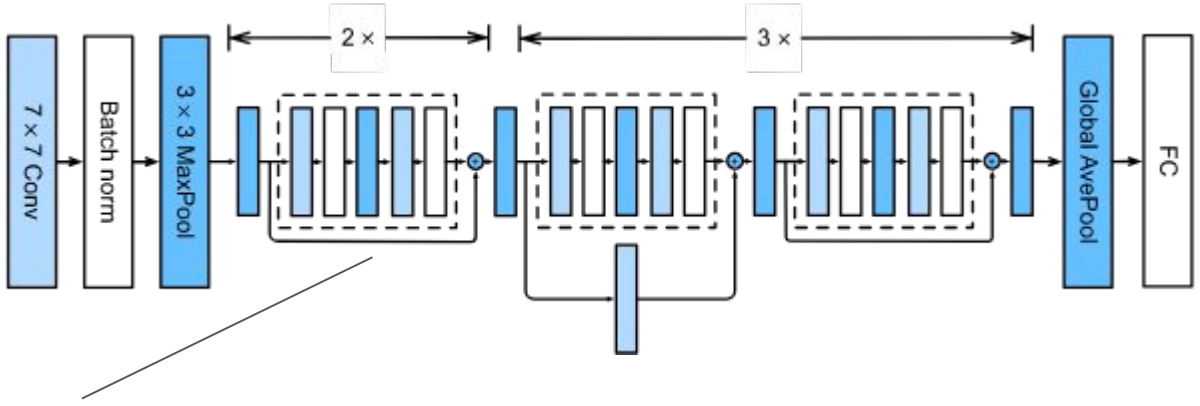


inception block

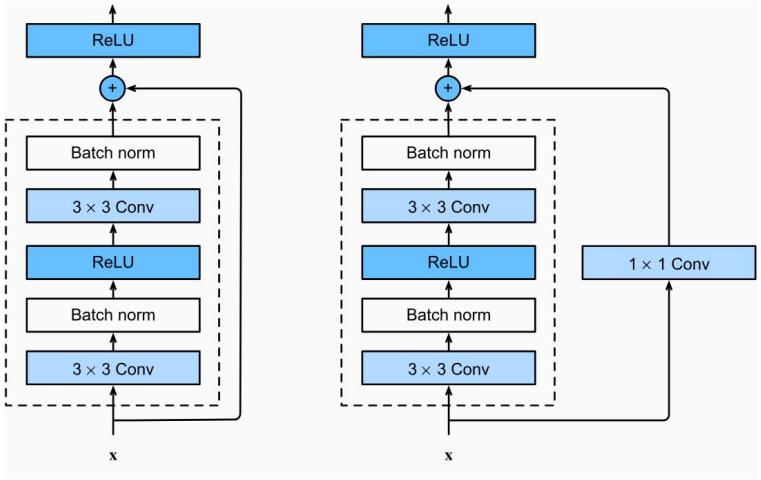


ResNet

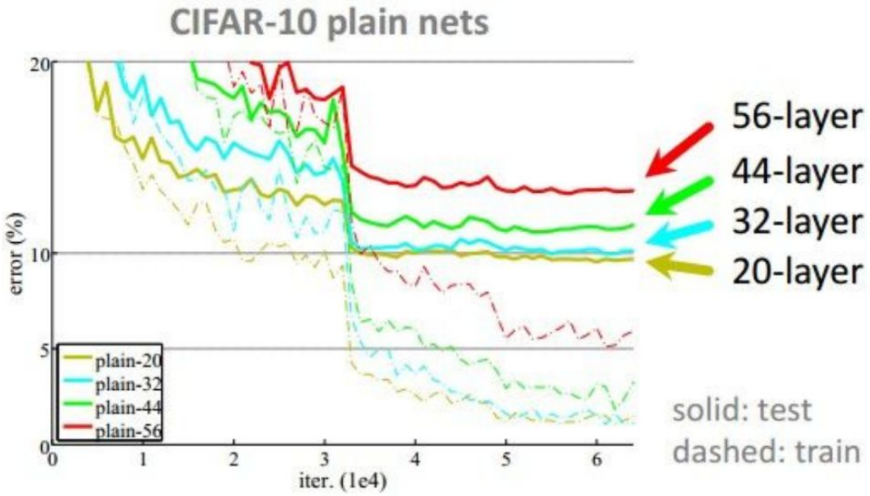
[He et al., 2015]

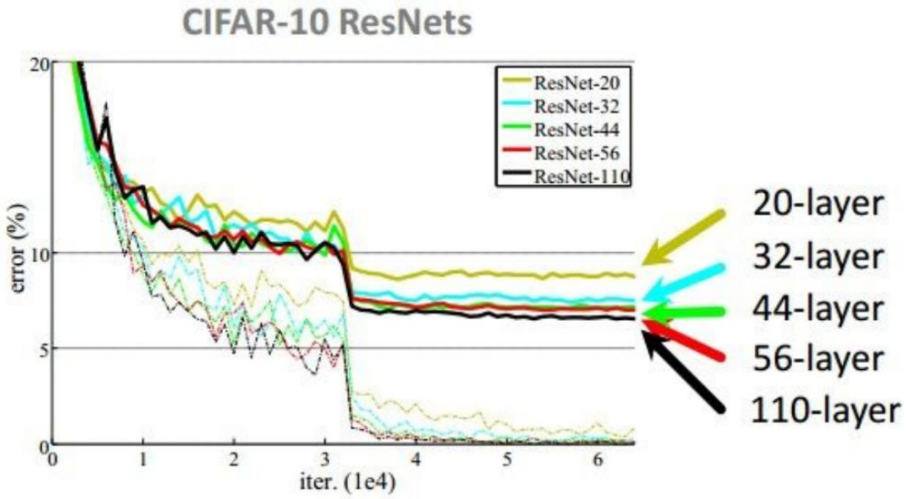


residual block

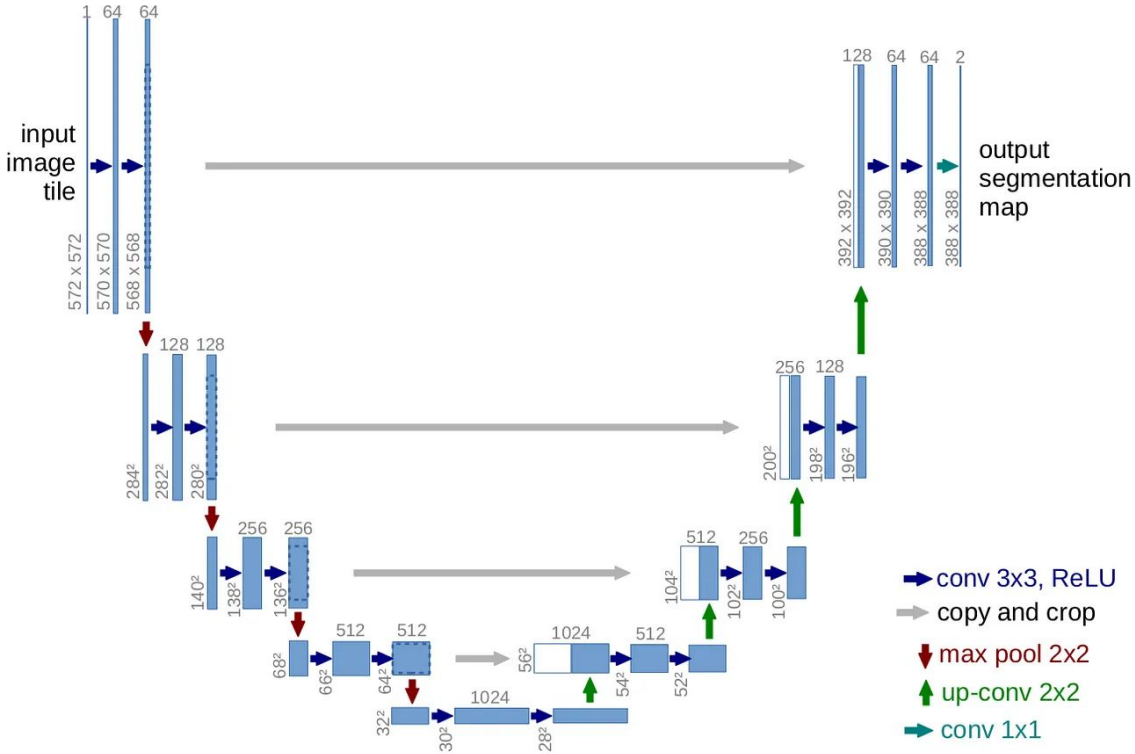


CIFAR experiments



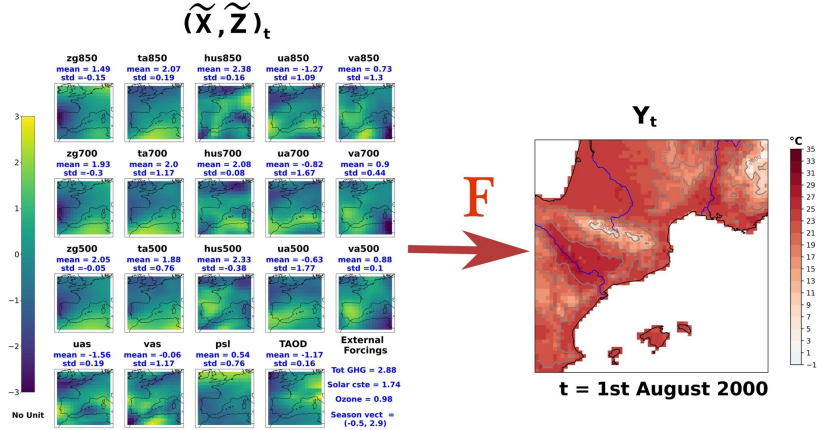


UNet


input
image
tile
output
segmentation
map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

UNet applications in Earth Sciences

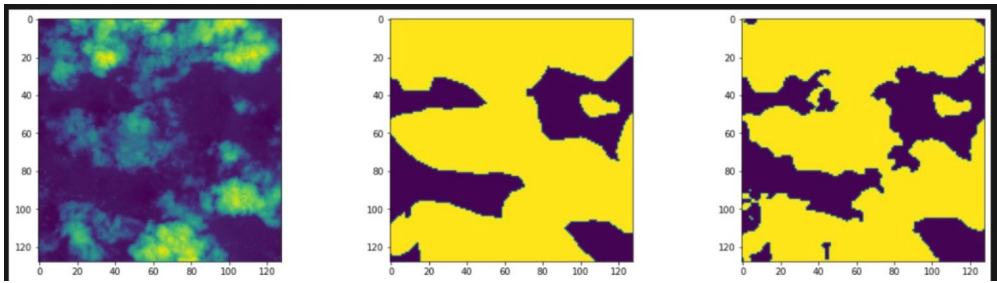
RCM emulator/downscaling



Doury et al. (2022-2024)

resol: 150 km -> 12.5 km

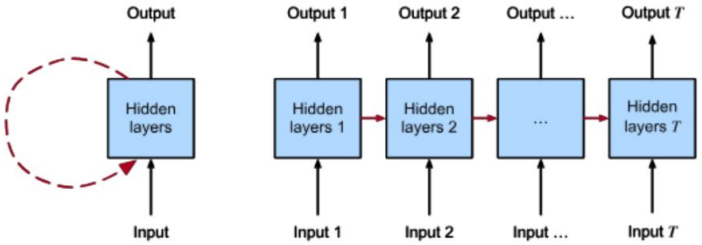
semantic segmentation in satellite data (e.g. clouds)



De Souza 2023

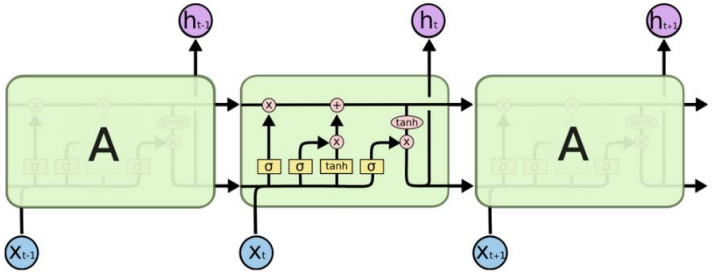
Deep Learning for time series data

RNN



Recurrent neural networks (RNNs) are deep learning models that capture the dynamics of sequences via *recurrent* connections, which can be thought of as cycles in the network of nodes

LSTM

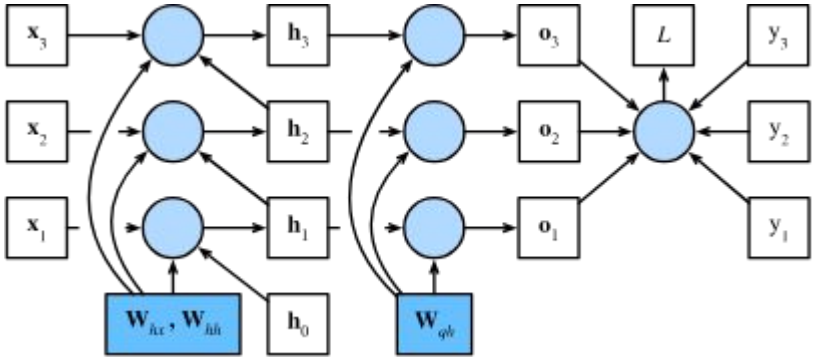


LSTM unit is a cell with 3 gates:
- Input gate
- Forget gate
- Output gate

Backpropagation through time

$$\mathbf{h}_t = \mathbf{W}_{\text{hx}}\mathbf{x}_t + \mathbf{W}_{\text{hh}}\mathbf{h}_{t-1},$$
$$\mathbf{o}_t = \mathbf{W}_{\text{qh}}\mathbf{h}_t,$$

where $\mathbf{W}_{\text{hx}} \in \mathbb{R}^{h \times d}$, $\mathbf{W}_{\text{hh}} \in \mathbb{R}^{h \times h}$, and $\mathbf{W}_{\text{qh}} \in \mathbb{R}^{q \times h}$



In the backward pass we estimate:

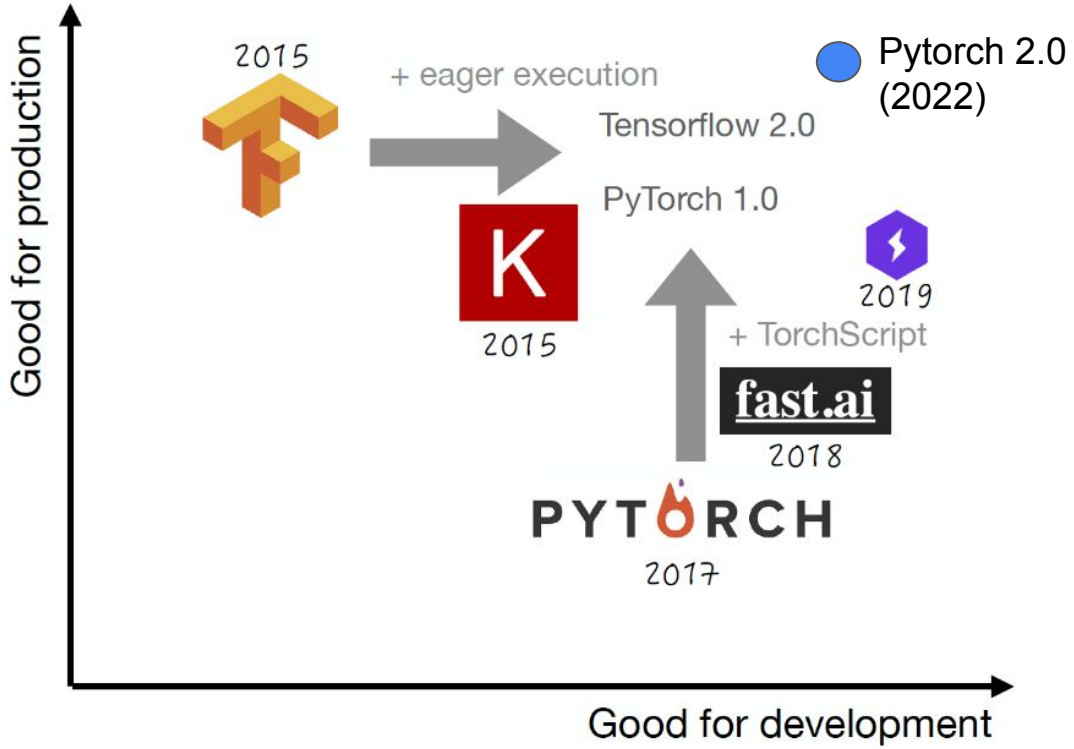
$$\frac{\partial L}{\partial \mathbf{W}_{\text{qh}}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \mathbf{o}_t}, \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}_{\text{qh}}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{o}_t}\mathbf{h}_t^\top,$$

$$\frac{\partial L}{\partial \mathbf{W}_{\text{hx}}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \mathbf{h}_t}, \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{\text{hx}}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t}\mathbf{x}_t^\top,$$

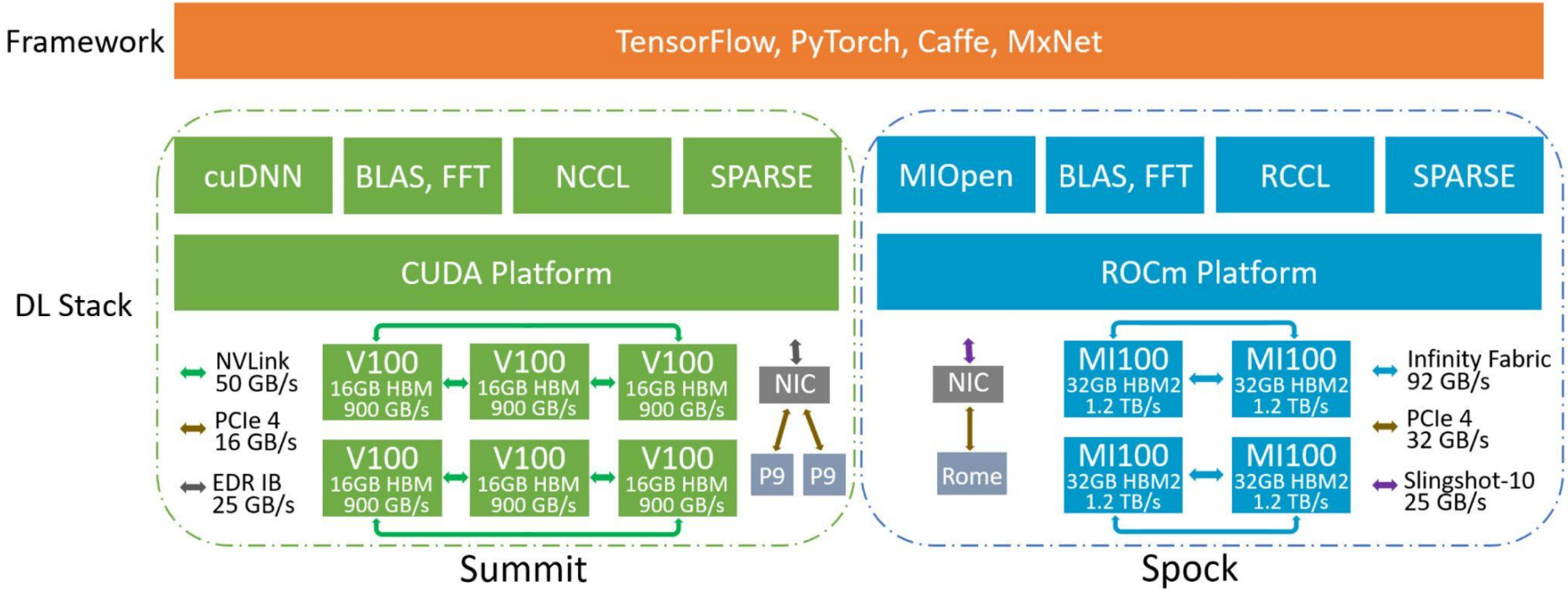
$$\frac{\partial L}{\partial \mathbf{W}_{\text{hh}}} = \sum_{t=1}^{T} \text{prod}\left(\frac{\partial L}{\partial \mathbf{h}_t}, \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_{\text{hh}}}\right) = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbf{h}_t}\mathbf{h}_{t-1}^\top,$$

$$\frac{\partial L}{\partial \mathbf{h}_t} = \text{prod}\left(\frac{\partial L}{\partial \mathbf{h}_{t+1}}, \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}\right) + \text{prod}\left(\frac{\partial L}{\partial \mathbf{o}_t}, \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t}\right) = \mathbf{W}_{\text{hh}}^\top \frac{\partial L}{\partial \mathbf{h}_{t+1}} + \mathbf{W}_{\text{qh}}^\top \frac{\partial L}{\partial \mathbf{o}_t}.$$

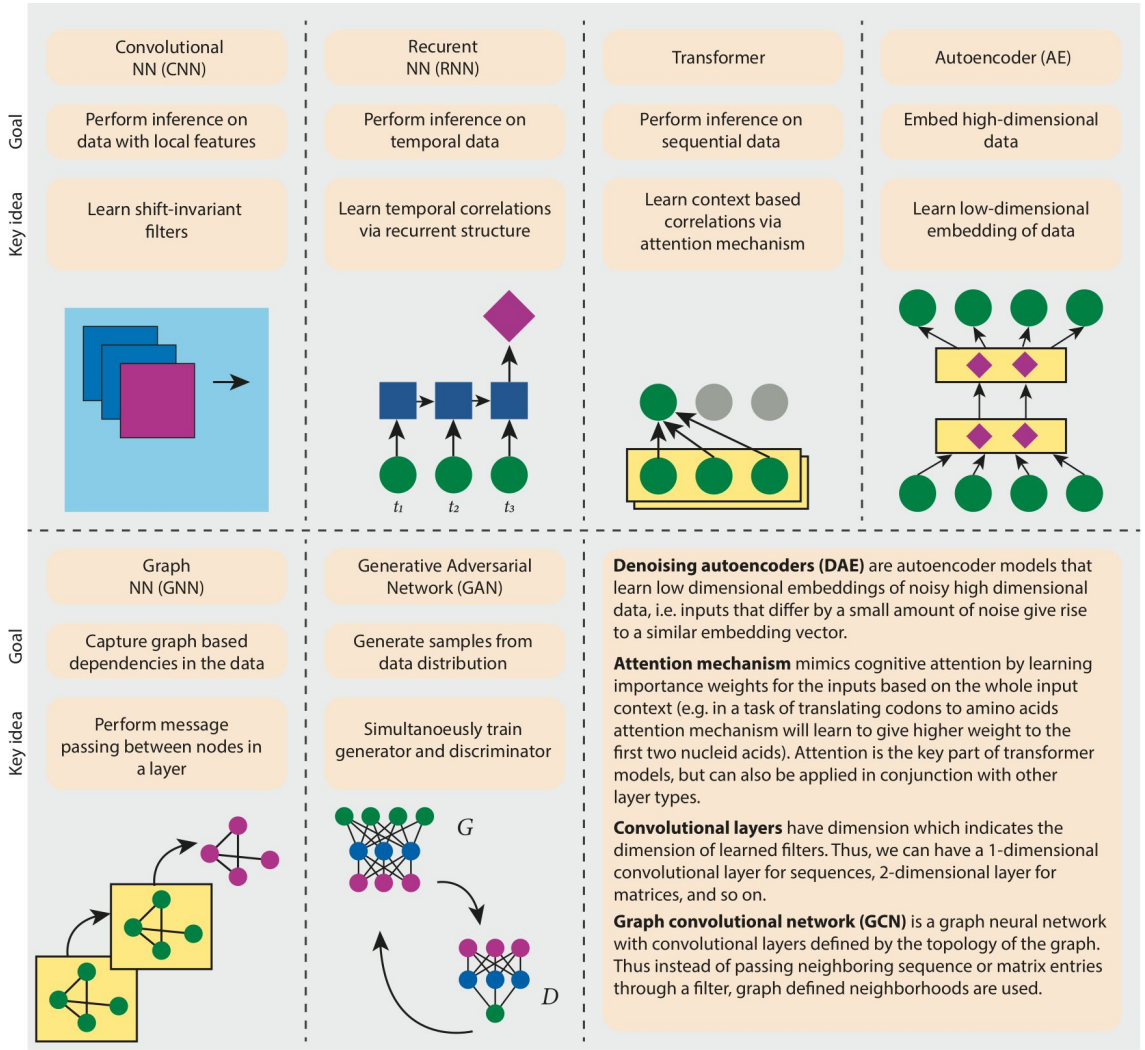
Python most used libraries/frameworks for DL



What's behind Pytorch/Tensorflow?



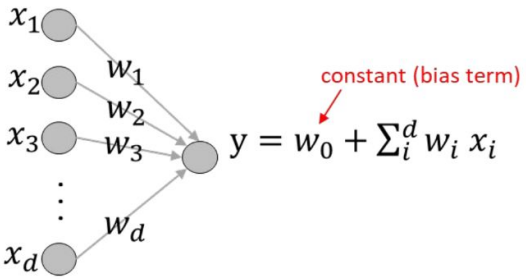
Diving into the world of DL models



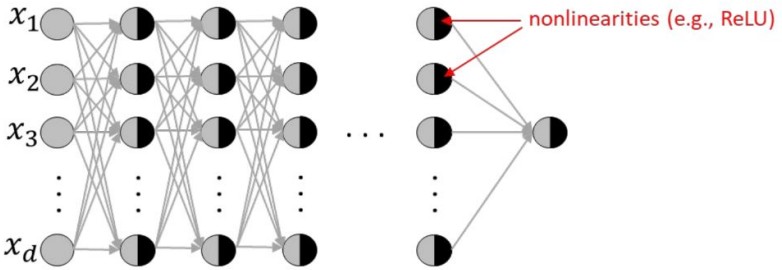
Explainable AI for Science (XAI)

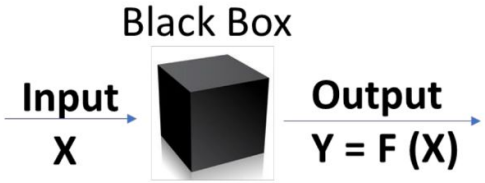
Scientists need to understand what the AI model is doing; what the decision-making process is.

Linear model: inherently interpretable

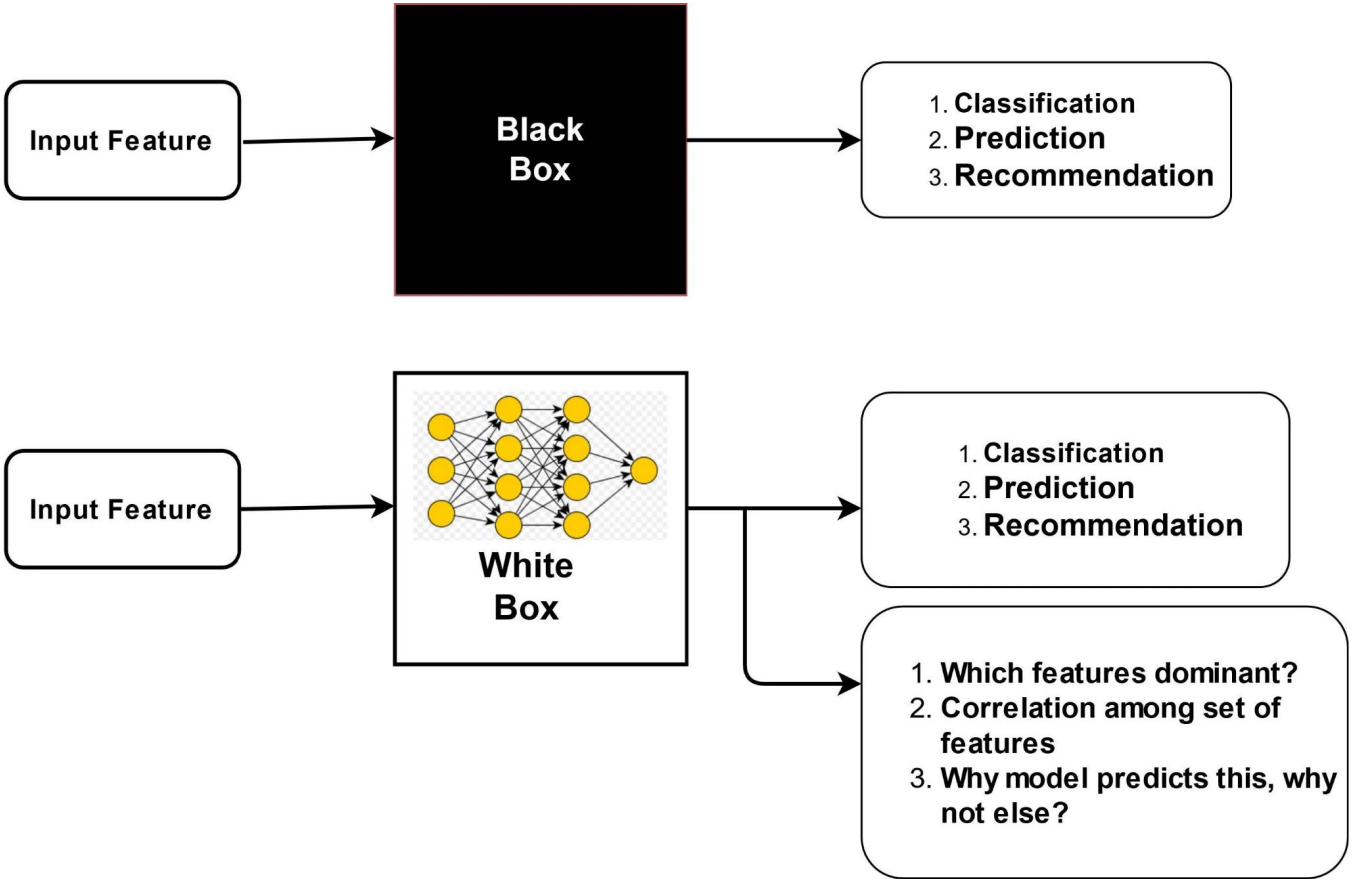


Neural Network: not inherently interpretable





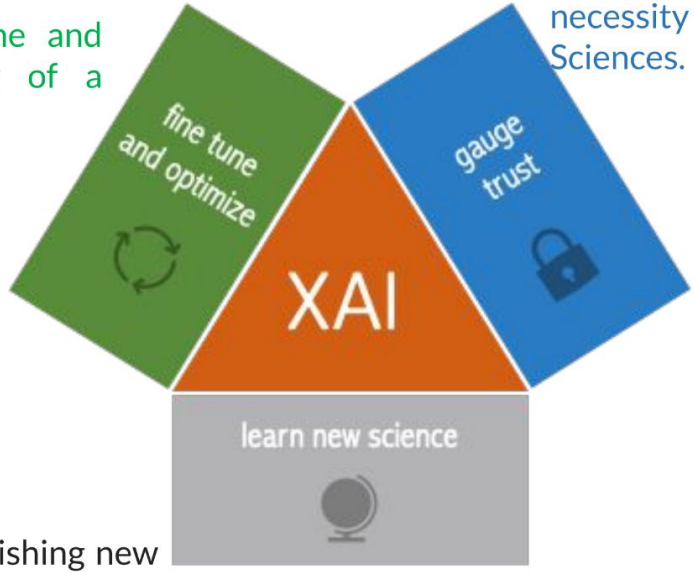
AI vs XAI


Input Feature
Black Box
1. Classification
2. Prediction
3. Recommendation
Input Feature
White Box
1. Classification
2. Prediction
3. Recommendation
1. Which features dominant?
2. Correlation among set of features
3. Why model predicts this, why not else?

Advantages of XAI for Science

XAI helps calibrate model trust and physically interpret the network, which is a necessity in many applications in Earth Sciences.

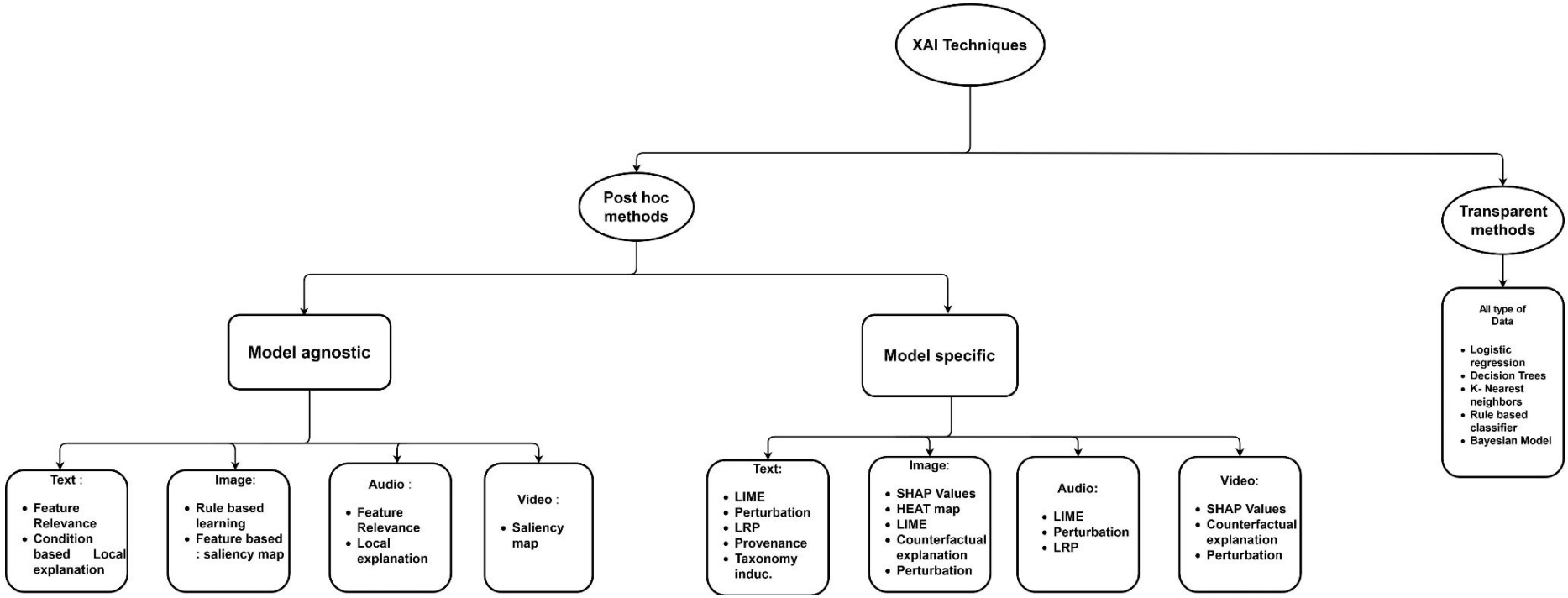
XAI may help fine-tune and optimize the strategy of a flawed model



XAI may help accelerate establishing new science, like investigating new climate teleconnections and gaining new insights.

From *Mamalakis et al. (2022)* 207

XAI methods


XAI Techniques
Post hoc methods
Transparent methods
Model agnostic
Model specific
All type of Data
• Logistic regression
• Decision Trees
• K- Nearest neighbors
• Rule based classifier
• Bayesian Model
Text :
• Feature Relevance
• Condition based Local explanation
Image:
• Rule based learning
• Feature based : saliency map
Audio :
• Feature Relevance
• Local explanation
Video :
• Saliency map
Text:
• LIME
• Perturbation
• LRP
• Provenance
• Taxonomy induc.
Image:
• SHAP Values
• HEAT map
• LIME
• Counterfactual explanation
• Perturbation
Audio:
• LIME
• Perturbation
• LRP
Video:
• SHAP Values
• Counterfactual explanation
• Perturbation

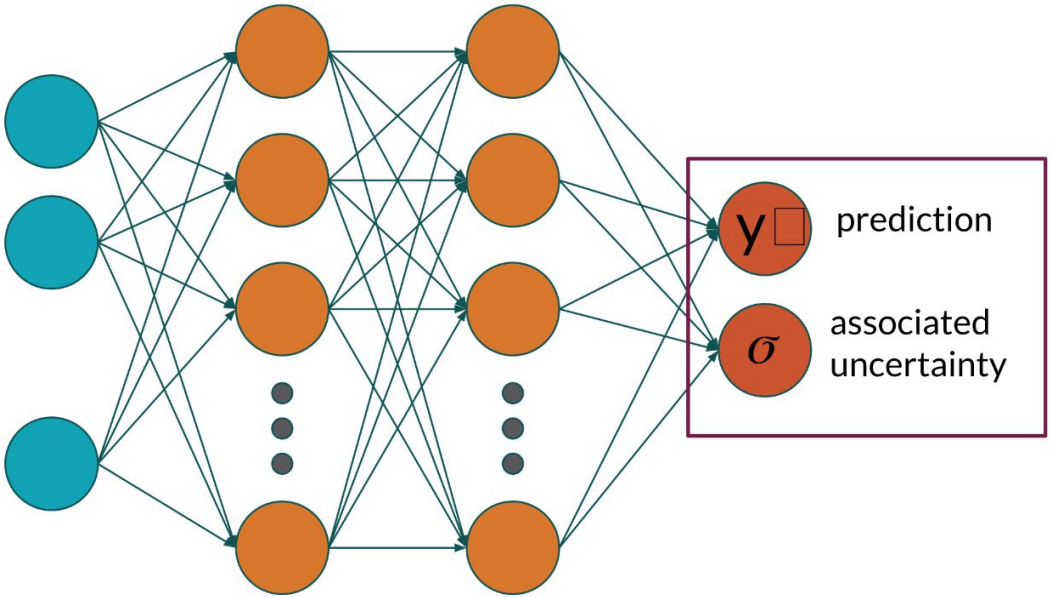
Libraries for XAI

Captum (website: https://captum.ai/tutorials/)

How can we add uncertainty to the predictions of DL model for regression task?

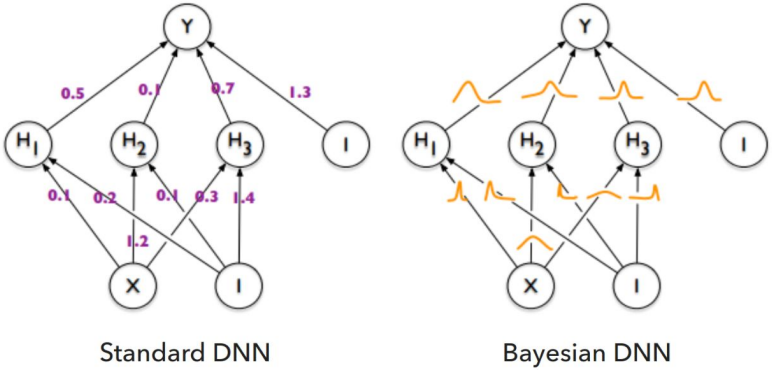
Rather than the network outputting a single number as its estimate….

We want the network to output an estimate and an uncertainty range



Bayesian Deep Learning

- In Bayesian deep learning we model posterior distribution over the weights of neural networks
- In theory, leads to better predictions and well-calibrated uncertainty



Standard DNN

Bayesian DNN

"Weight Uncertainty in Neural Networks" by Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, Daan Wierstra

Bayesian learning by steps…

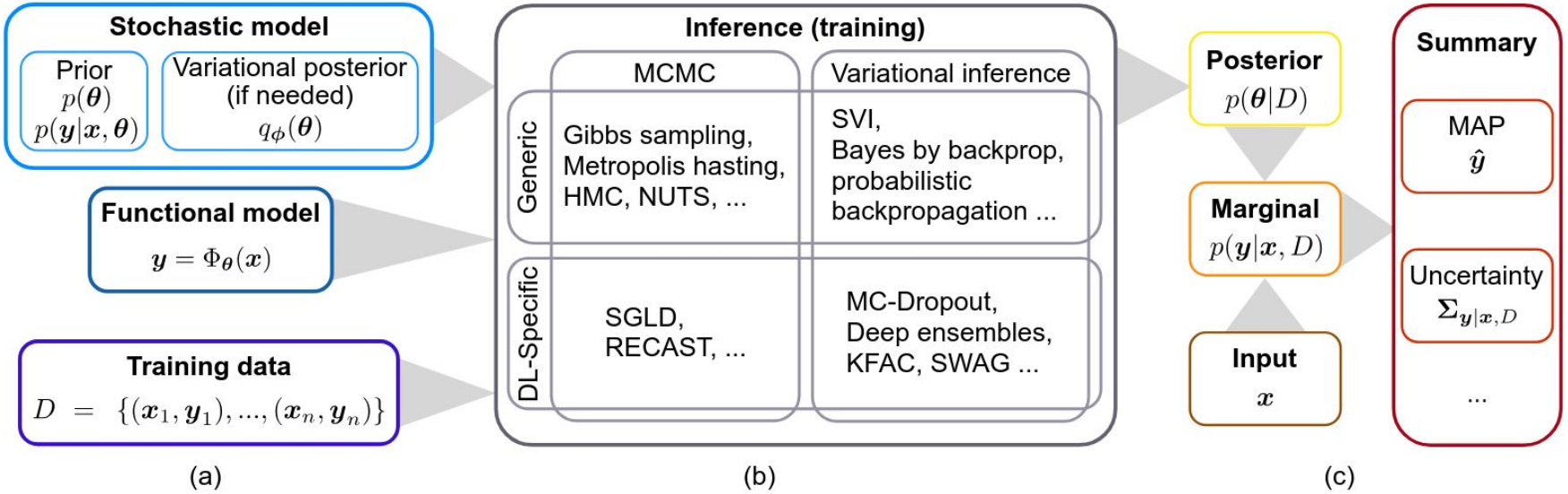
2 steps to model uncertainty over parameters of the model:

Step 1: introduce a prior distribution $p(w)$ over parameters

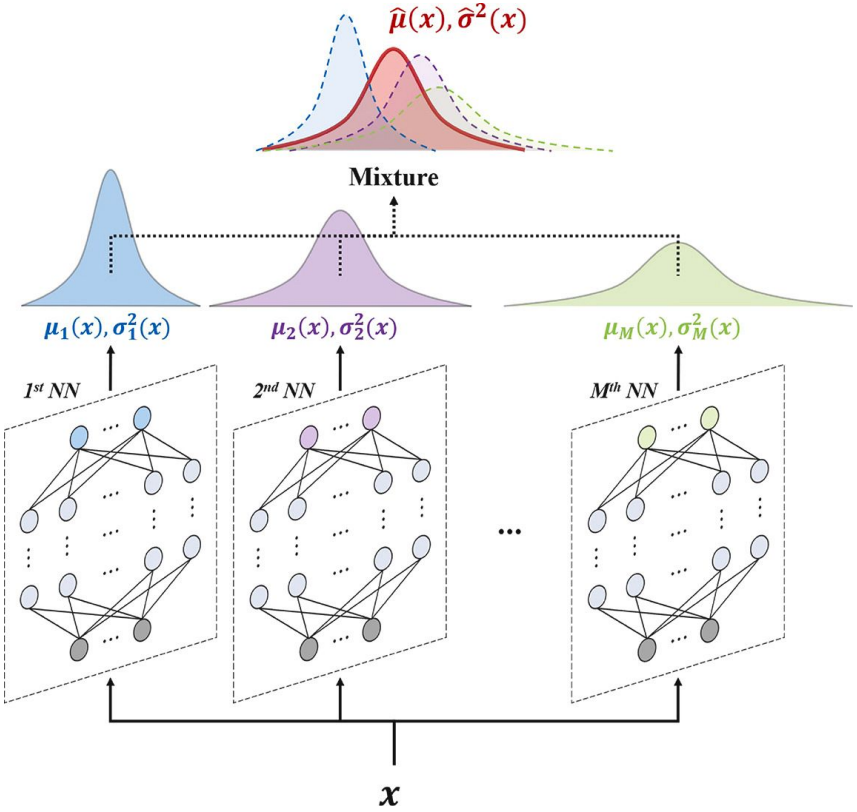
Step 2: Compute posterior $p(w|D)$ using Bayes rule

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)}$$

Workflow to train Bayesian NN



Deep ensemble



"Towards reliable uncertainty quantification via deep ensemble in multi-output regression task"

Sunwong, Kwanjung, ELSEVIER

Challenges of Bayesian DL

Bayesian inference for deep neural networks is extremely challenging

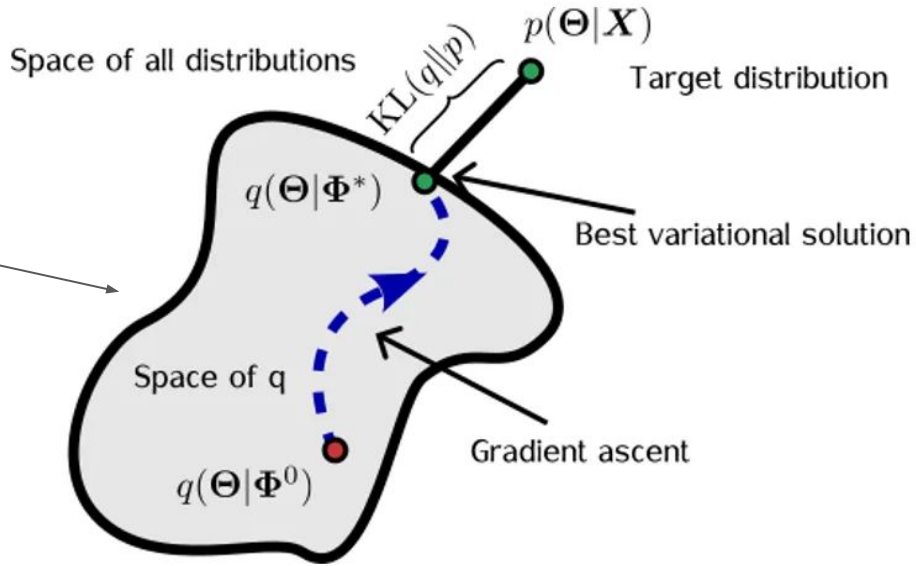
- Posterior is intractable — Is the likelihood correct?
- Millions of parameters — What do these parameters mean?
- Large datasets — Can we run MCMC for 1 million steps on ImageNet??
- Unclear which priors to use — Is the prior correct?

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)} = \frac{p(D|w)p(w)}{\int_{w'} p(D|w')p(w')dw'}$$

Approximate Bayesian Inference

Posterior Approximation:

- Laplace Approximation
- Variational Inference
- Markov Chain Monte Carlo
- Geometrically Inspired Methods



Variational inference

We can find the best approximating distribution within a given family with respect to KL-divergence

- $KL(q||p) = \int_w q(w) \log \frac{q(w)}{p(w|D)} dw$
- Stochastic variational inference (Hoffman et al, '13, Kucelkibir, et al, '17, Graves, 2011)

$$ELBO(w) = E_{q(w)}(\log p(\mathcal{D}|w)) - KL(q(w)||p(w))$$

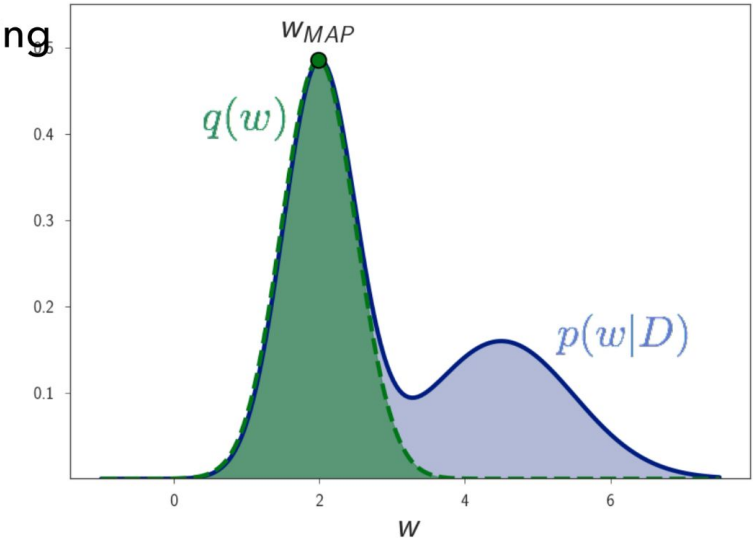
Traditionally... $q(w) = \mathcal{N}(\mu_i, \sigma_i^2)$

- Minimizing the KL divergence is "consistent" statistically (Wang & Blei, '19) & optimal in other settings (Knoblauch, et al, '19)

Laplace approximation

Approximate posterior with a Gaussian $\mathcal{N}(w|\mu, A^{-1})$

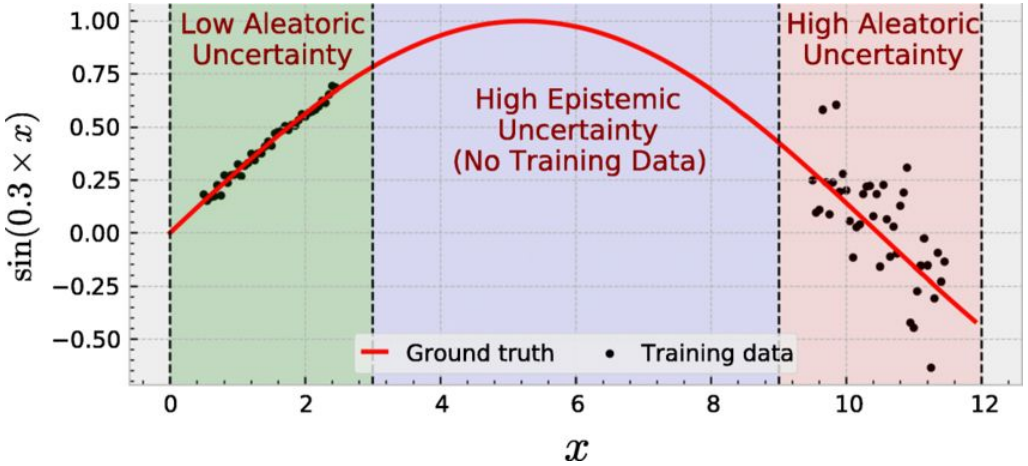
- $w = w_{MAP}$ mode (local maximum) of $p(w|D)$
- Approximate A with a KFAC (tri-diagonal) – Ritter et al., 2018a
- Application: Catastrophic forgetting (Ritter et al, 2018b)
- Originally from Mackay, '92



Aleatoric vs Epistemic uncertainty

We combine aleatoric and epistemic uncertainties via Bayesian Model Averaging (BMA):

$$p(y^*|x^*, D) = \int_w p(y^*|x^*, w)p(w|D)dw$$



Pros/cons of different methods

	Benefits	Limitations	Use cases	
MCMC (V.A)	Directly samples the posterior	Requires to store a very large number of samples	Small and average models	
Classic methods (HMC, NUTS)(§V-A)	State of the art samplers limit autocorrelation between samples	Do not scale well to large models	Small and critical models	Can be combined
SGLD and derivates (§V-E2a)	Provide a well behaved Markov Chain with minibatches	Focus on a single mode of the posterior	Models with larger datasets	
Warm restarts (§V-E2a)	Help a MCMC method explore different modes of the posterior	Requires a new burn-in sequence for each restart	Combined with a MCMC sampler	
Variational inference (V.B)	The variational distribution is easy to sample	Is an approximation	Large scale models	
Bayes by backprop (§V-C)	Fit any parametric distribution as posterior	Noisy gradient descent	Large scale models	Can be combined
Monte Carlo-Dropout (§V-E1)	Can transform a model using dropout into a BNN	Lack expressive power	Dropout based models	
Laplace approximation (§V-E2b)	By analyzing standard SGD get a BNN from a MAP	Focus on a single mode of the posterior	Unimodals large scale models	
Deep ensembles (§V-E2b)	Help focusing on different modes of the posterior	Cannot detect local uncertainty if used alone	Multimodals models and combined with other VI methods	

References

“A Simple Baseline for Bayesian Uncertainty in Deep Learning,” Maddox, Garipov, Izmailov, Vetrov, Wilson,

https://arxiv.org/abs/1902.02476, NeurIPS, 2019

- Code: https://github.com/wjmaddox/swa_gaussian

“Subspace Inference for Bayesian Deep Learning,”

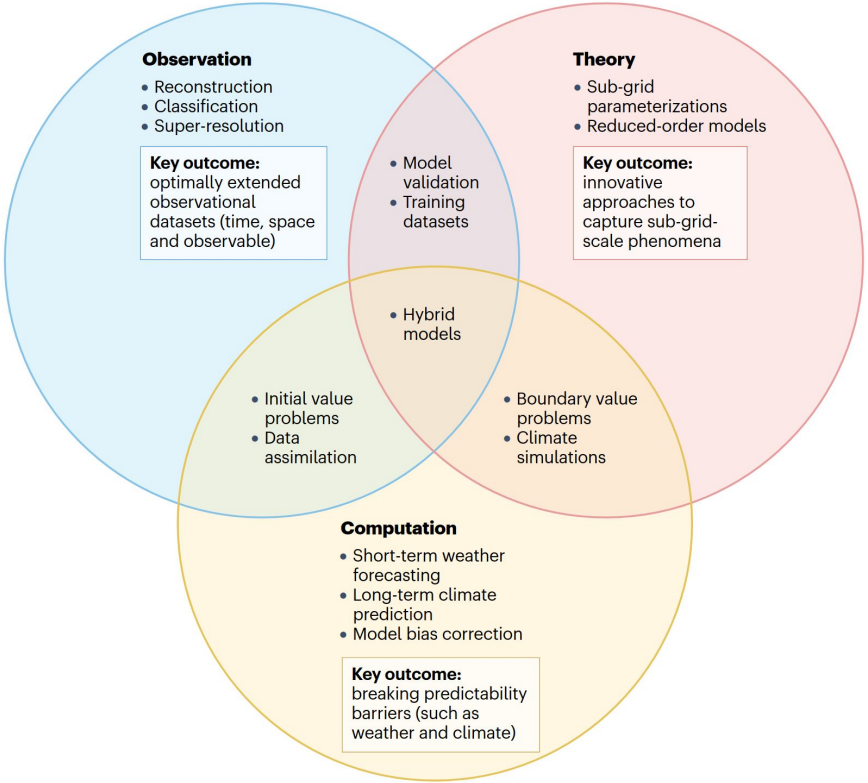
Izmailov, Maddox, Kirichenko, Garipov, Vetrov, Wilson,

https://arxiv.org/abs/1907.07504, UAI, 2019.

- Code: https://github.com/wjmaddox/drbayes

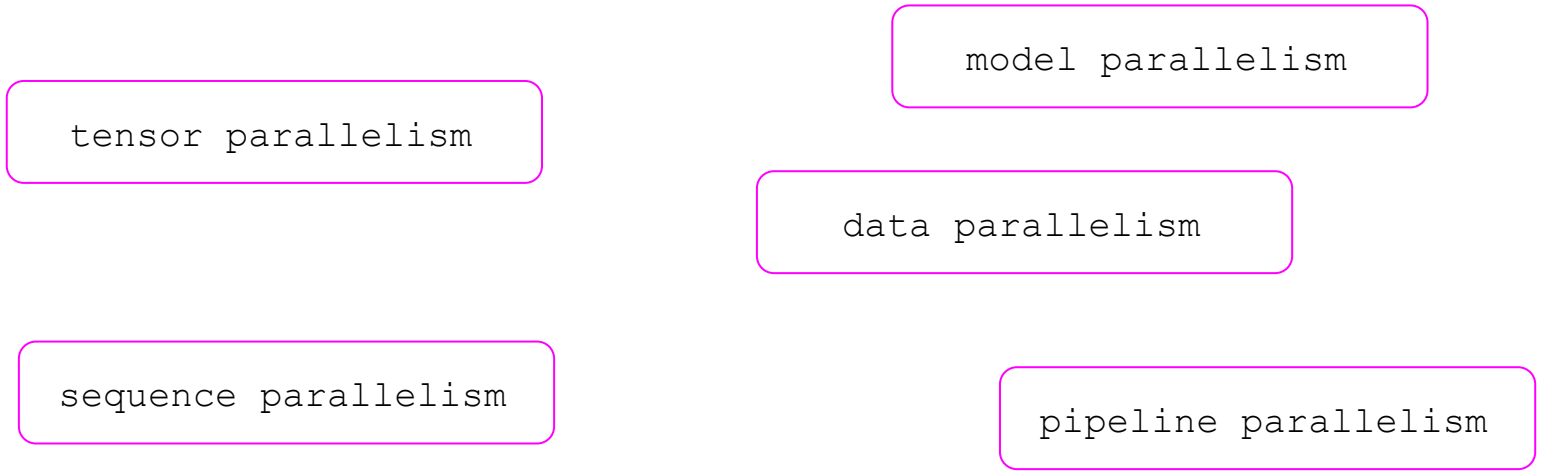
Method	Accuracy	Calibration	Train time	Test time	Code
Ensembles (Lakshminarayanan et al, '17)	**high**	Often more overconfident	K times standard training	K times slower	**Train K models**
Swag (Maddox et al, '19)	Slightly better than MAP	**Less overconfident**	**Standard training**	K times slower	**Store models at train time**
Dropout (Gal & Gharamani, '16)	About the same as MAP	Slightly less overconfident	**Standard training**	K times slower	**Apply dropout at test time**
VOGN (Osawa et al, '19)	Slightly worse than MAP?*	Less overconfident	2x standard training	K times slower	Modify Adam

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)

Different strategies for parallel training of DL models

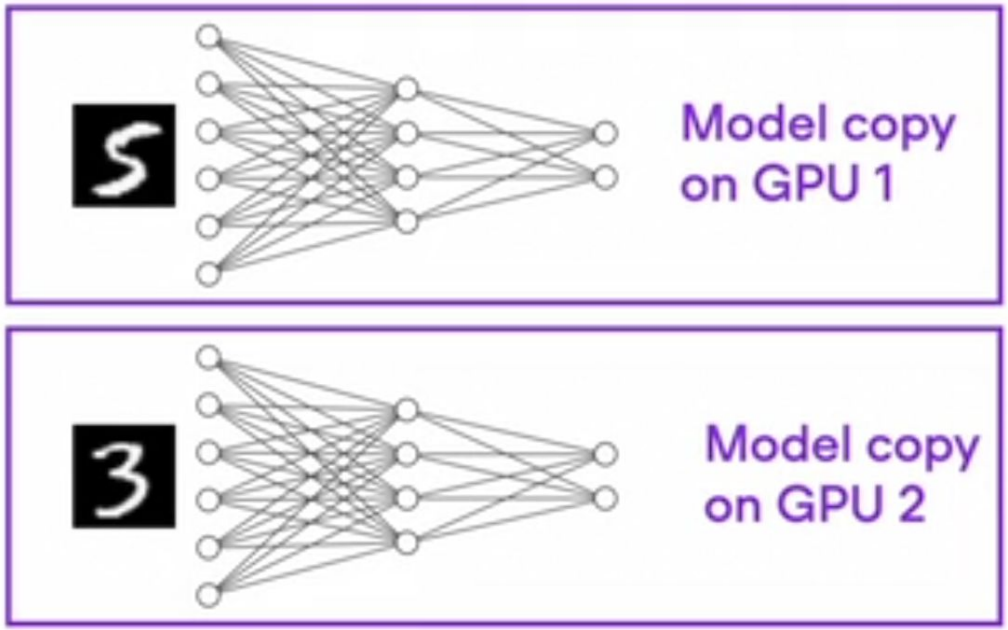
we can identify 5 different categories of parallelism



Data Parallelism

In this framework we split batches to train DL model

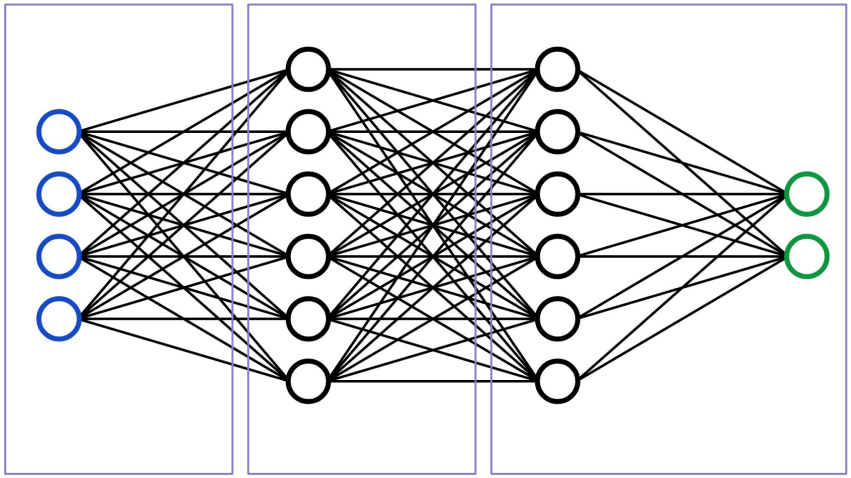
into different GPUs



Model parallelism

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

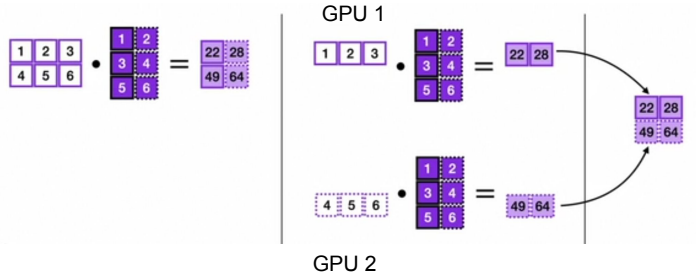
to work around GPU memory limits

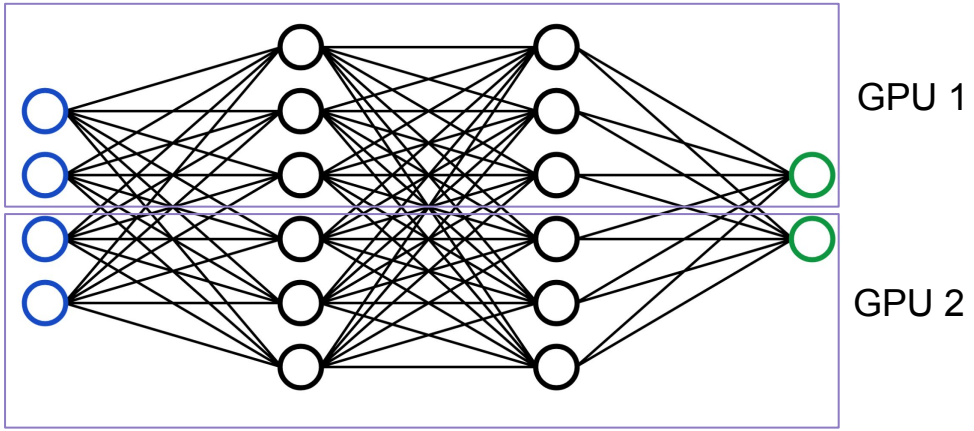


Tensor parallelism

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

similarly to what we would have done for matmul

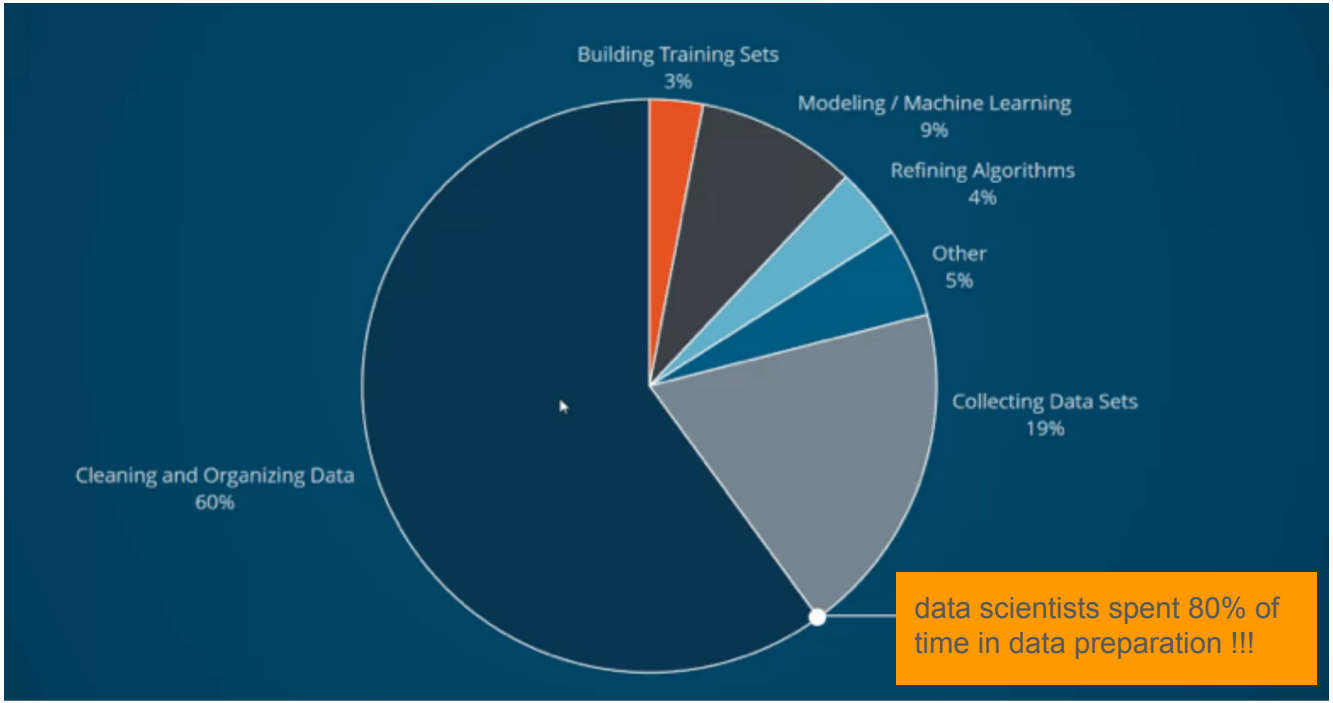




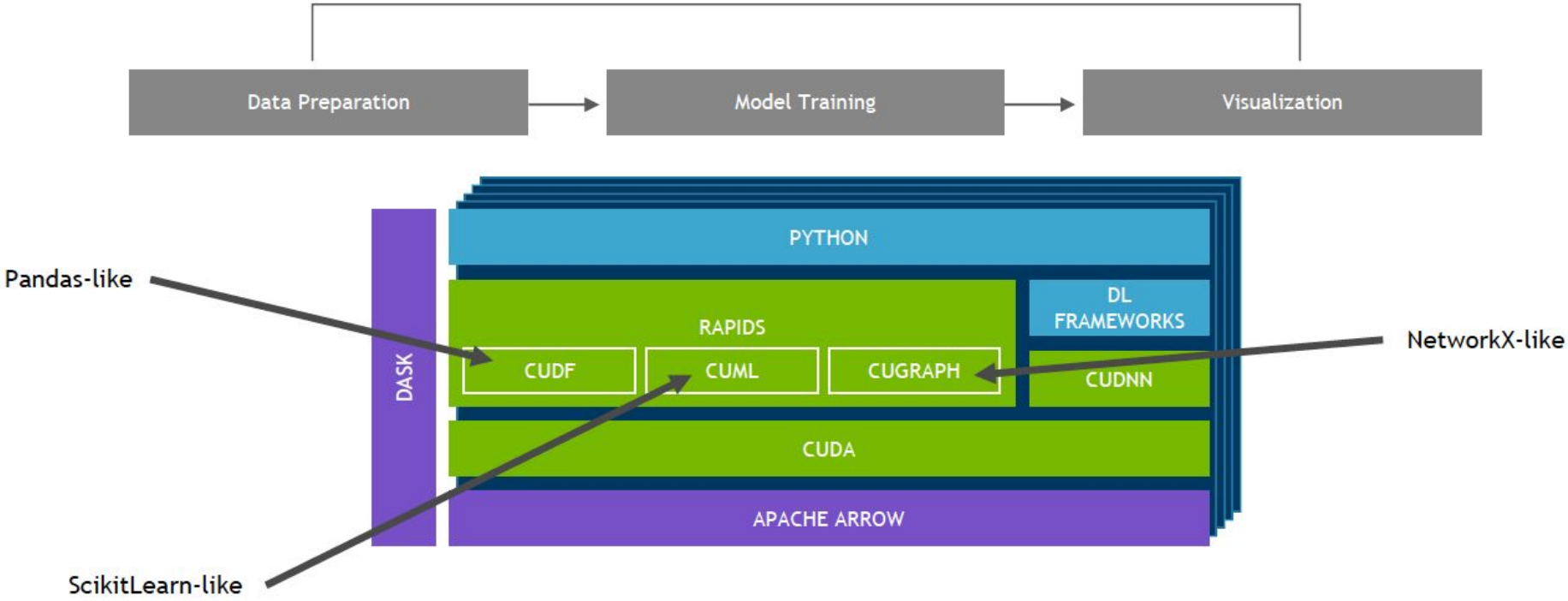
More complex strategies for DL training

Sequence parallelism and pipeline parallelism frameworks are obtained combining the previous approaches, and are typically applied to DL models dealing with spatio-temporal data.

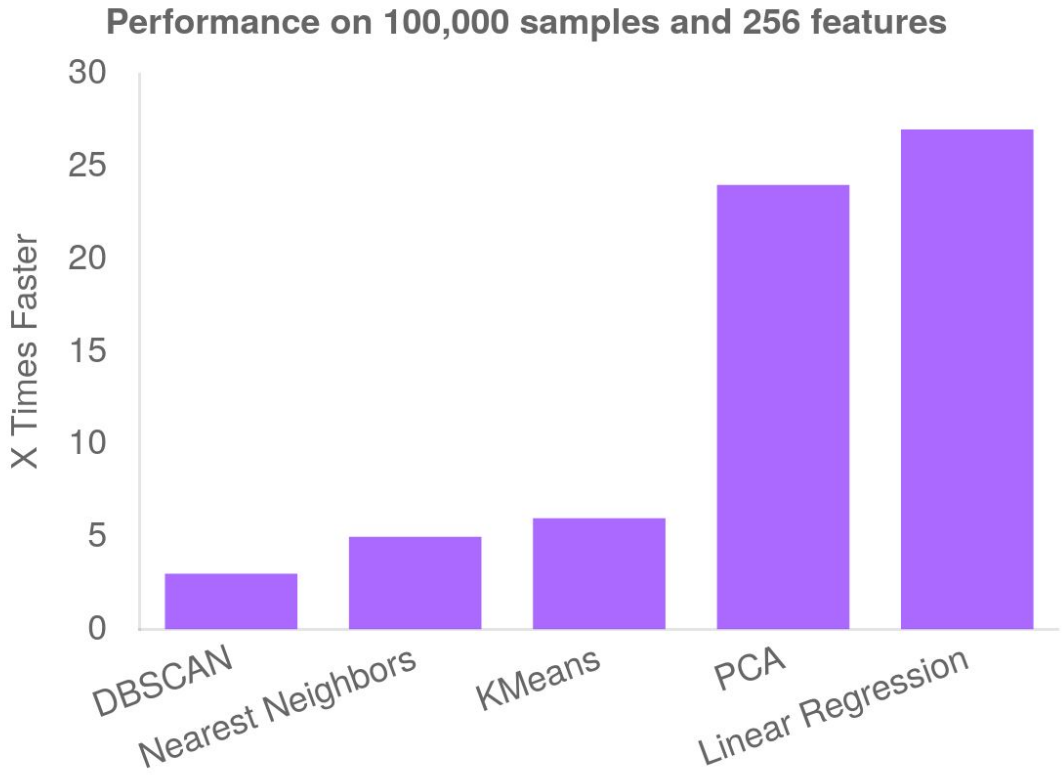
Do we need GPUs also for other ML tasks?



GPU-based libraries outside of Pytorch

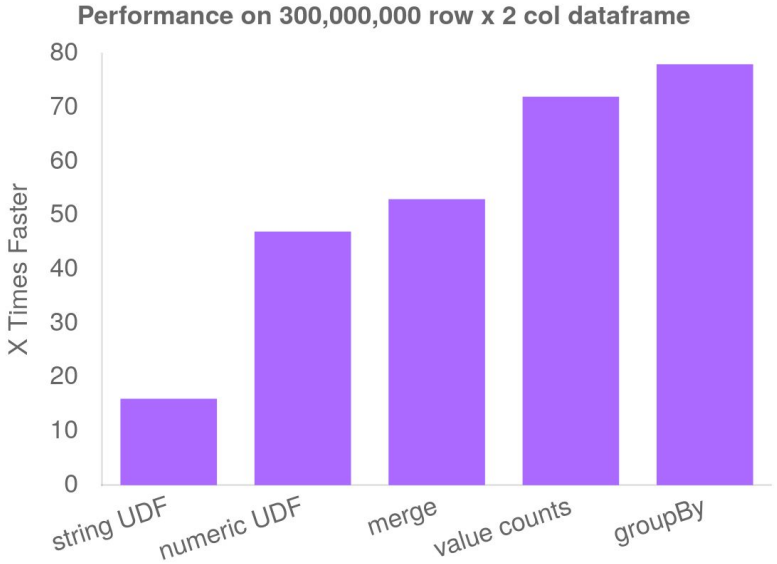


GPU for classical ML



* 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

https://github.com/Sera91/SMR4054--ICTP