Big data science Day 5

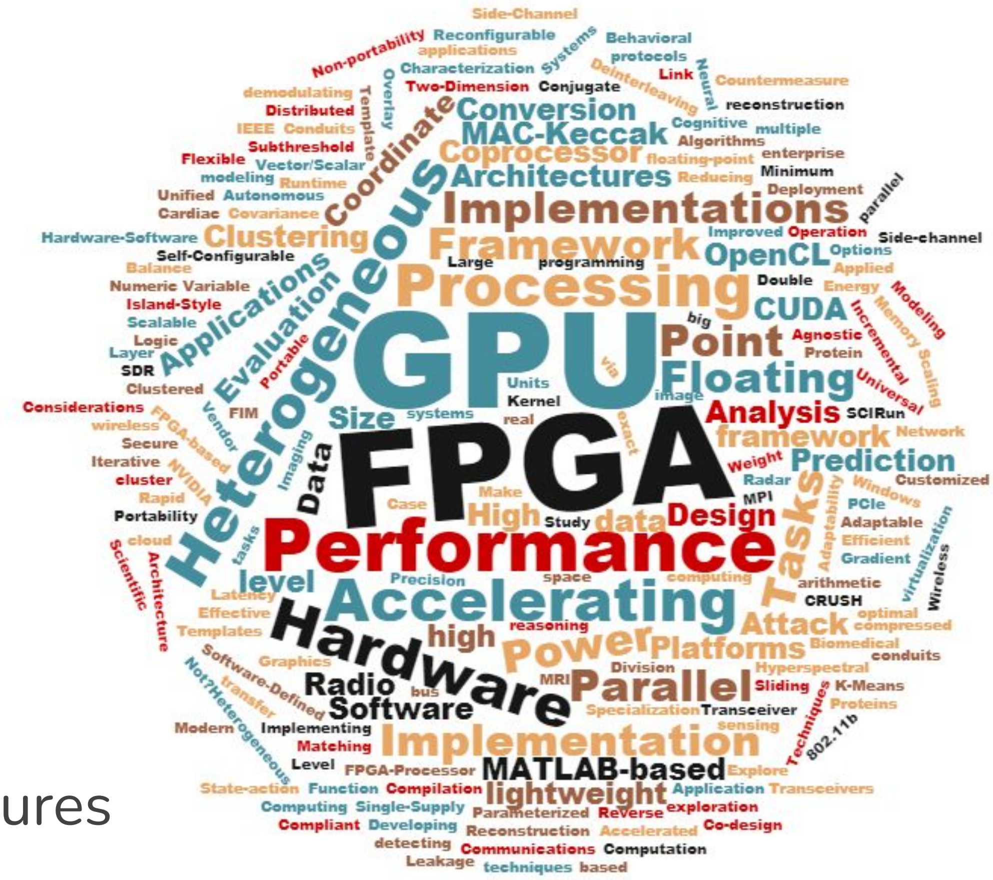
F. Legger - INFN Torino
https://github.com/Course-bigDataAndML/MLCourse-2223

We learned:

- Big data
- Analytics
- Machine learning
- Deep learning

Today

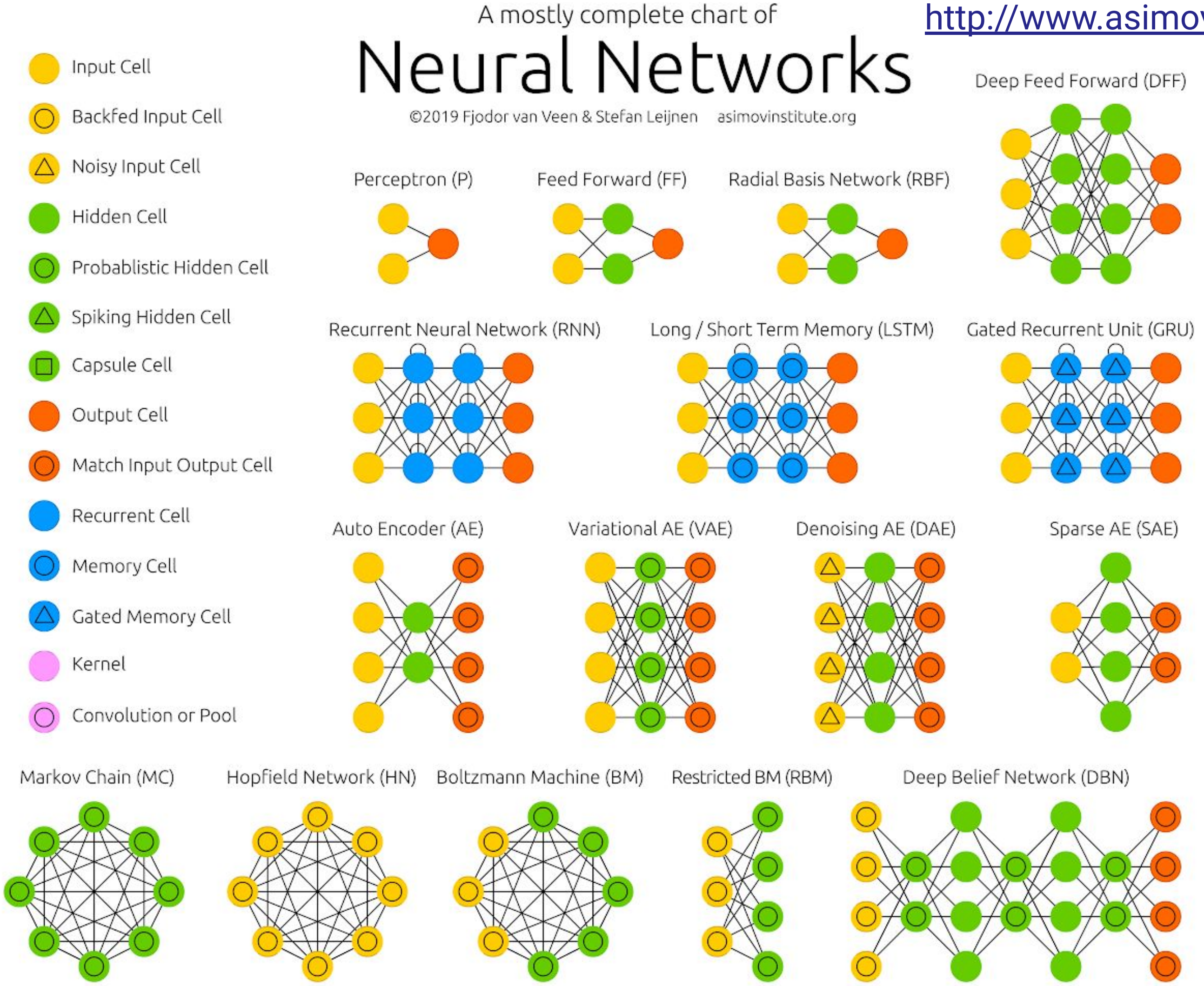
- NN models
- Parallelisation
- Heterogeneous architectures

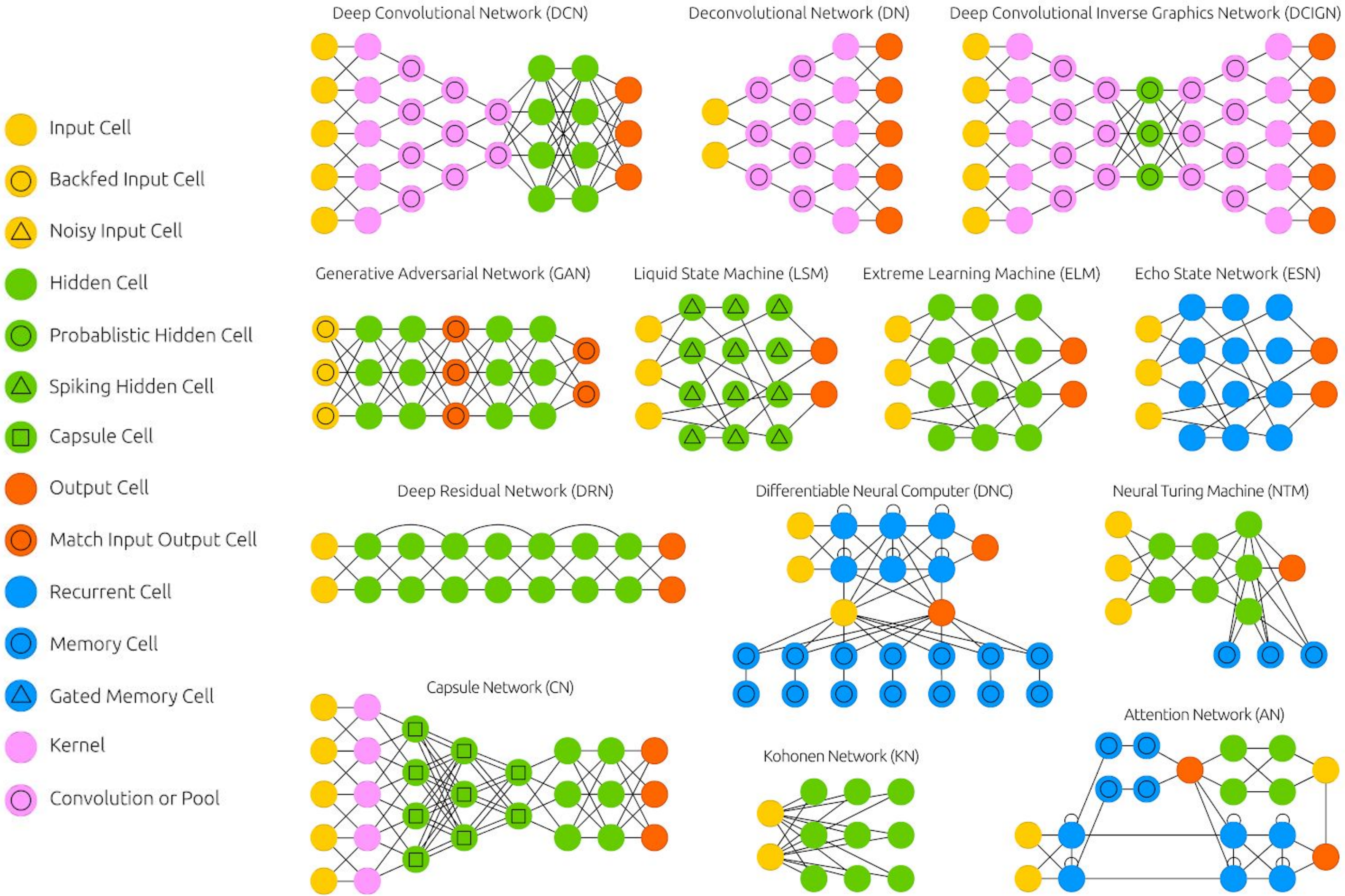


Recap:

- **Machine learning (ML)**: family of algorithms with ability to automatically learn and improve from experience without being explicitly programmed
  - potential to approximate linear and non-linear relationships
- For a given problem:
  - *Choose architecture*
  - *Train model*
  - *Tune hyperparameters*
  - *Do cross-validation*
  - *Do inference*

http://www.asimovinstitute.org/neural-network-zoo/

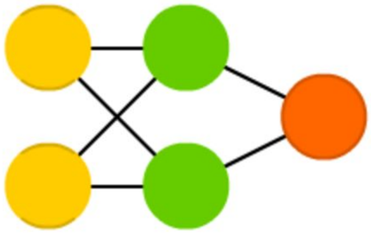



Input Cell
Backfed Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Capsule Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Gated Memory Cell
Kernel
Convolution or Pool
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Differentiable Neural Computer (DNC)
Neural Turing Machine (NTM)
Capsule Network (CN)
Kohonen Network (KN)
Attention Network (AN)

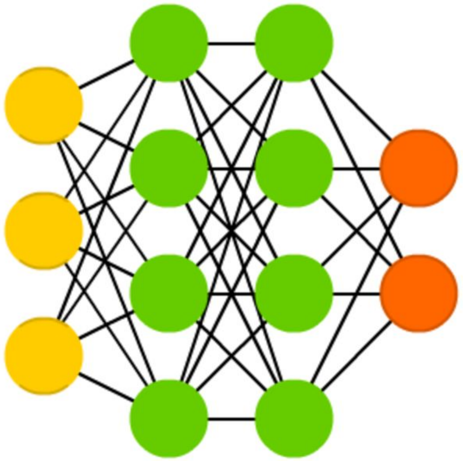
Feed Forward

- Supervised, simplest form of NN
  - Used in many ML tasks, speech, image recognition, classification, computer vision
  - Easy to implement and combine with other type of ML algorithms
- input/outputs are vectors of fixed length
- data passes through input nodes and exits on the output nodes
- typically trained with back propagation
- **DFF** is a FF NN with more than one hidden layer

Feed Forward (FF)

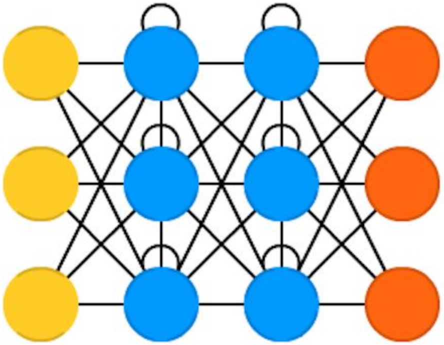


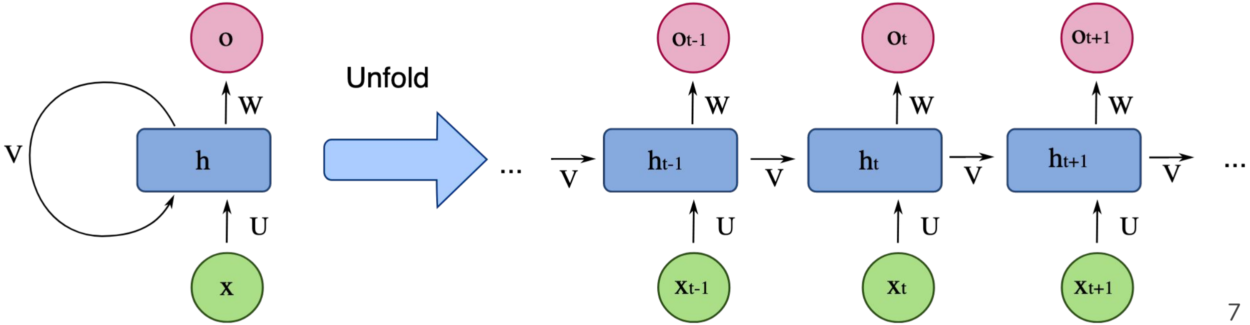
Deep Feed Forward (DFF)



Recurrent Neural Network (RNN)

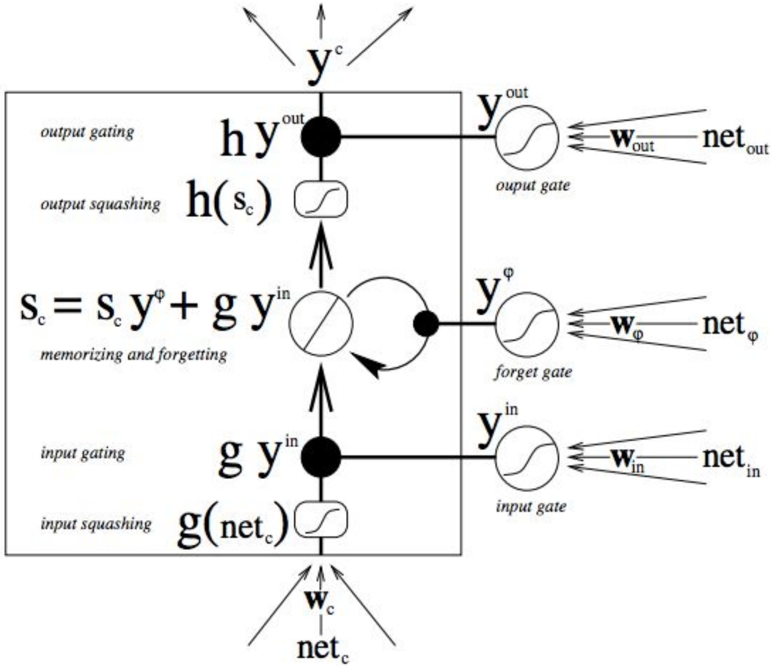
- FFNN with **Recurrent Cells**: cell that receives its own output with fixed delay
- **RNN**s permit to operate on **sequences of vectors**
  - context is important, decision from past iterations can influence current state
  - a word can be analyzed only in context of previous words or sentences
- RNNs, once unfolded in time, can be seen as **very deep FF networks** in which all the layers share the same weights
  - The parameters to be learned are shared by all time steps in the network
  - The gradient at each output depends also on the calculations of the previous time step

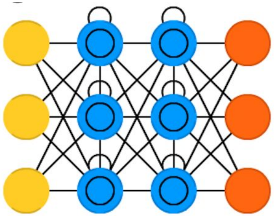




Long/Short Term Memory (LSTM)

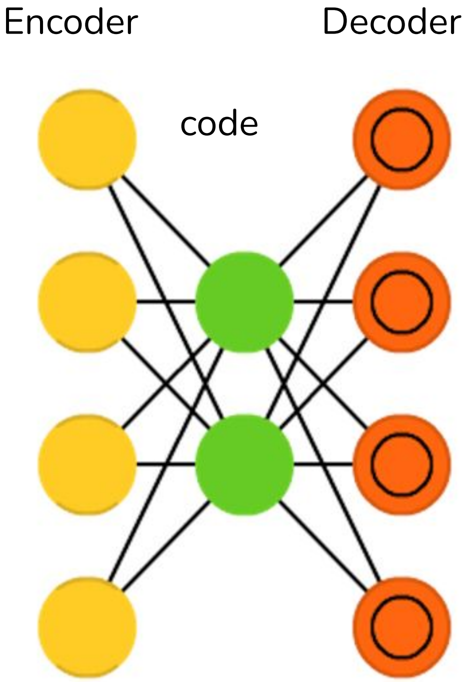
- RNNs not really capable of learning long term dependencies
  - Due to vanishing gradient with increasing time steps
- A common LSTM unit is made of a **cell**, an **input**, an **output** and a **forget** gate
  - The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell





Auto Encoders

- used for classification, clustering and feature compression (**unsupervised** learning)
- Compress (encode) information automatically
  - An **encoder** is a deterministic mapping *f* that transforms an input vector *x* into hidden representation *y*
  - A **decoder** maps back the hidden representation *y* to the reconstructed input *z* via *g*
- **Autoencoder**: compare the reconstructed input *z* to the original input *x* and tries to minimize the reconstruction error

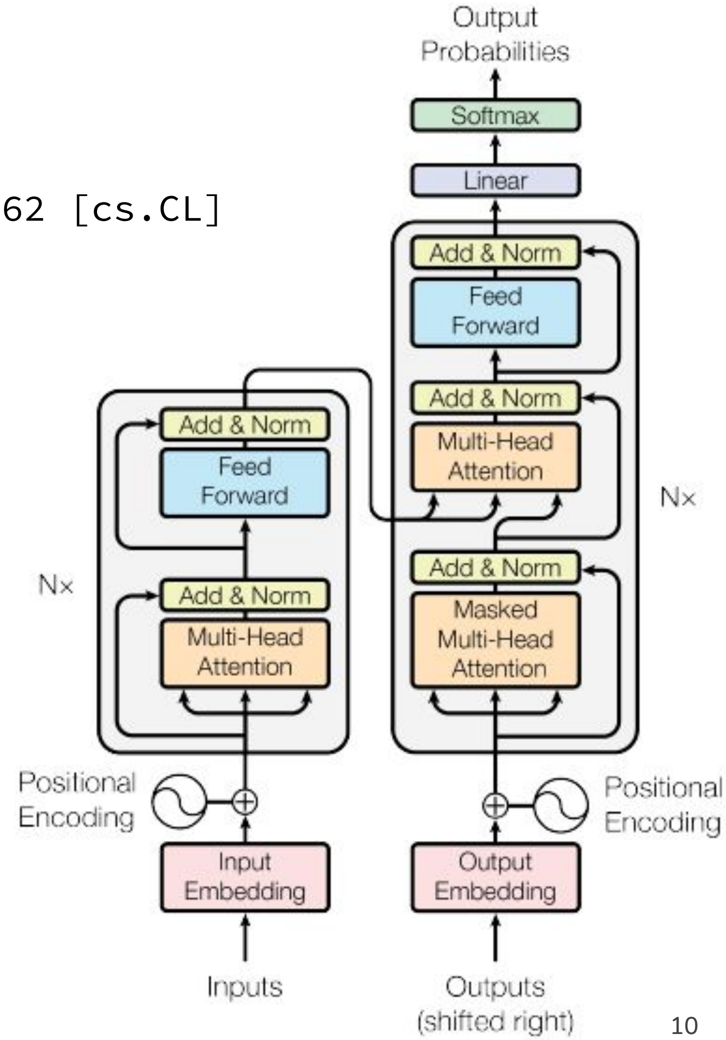


Transformers

arXiv:1706.03762 [cs.CL]

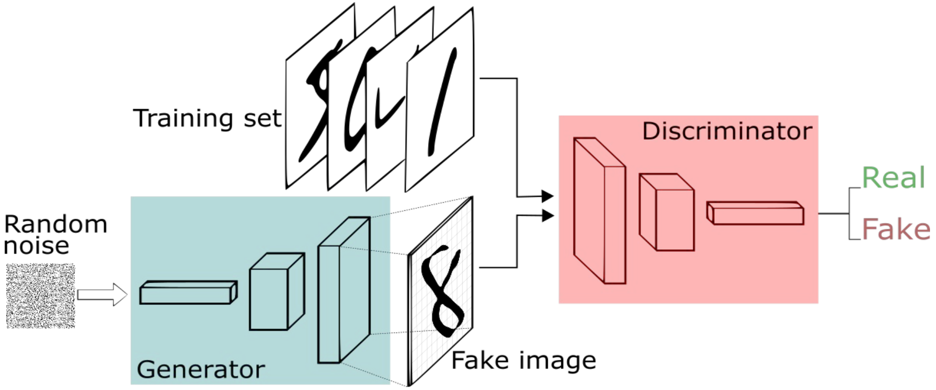
- All you need is **attention**
  - transformers process the entire input all at once
  - the attention mechanism provides context for any position in the input sequence
- Self-attention: **query, key, value**:
  - The significance of each part of the input data is differentially weighted

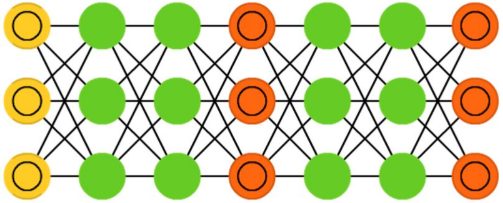
$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



Generative Adversarial Networks (GAN)

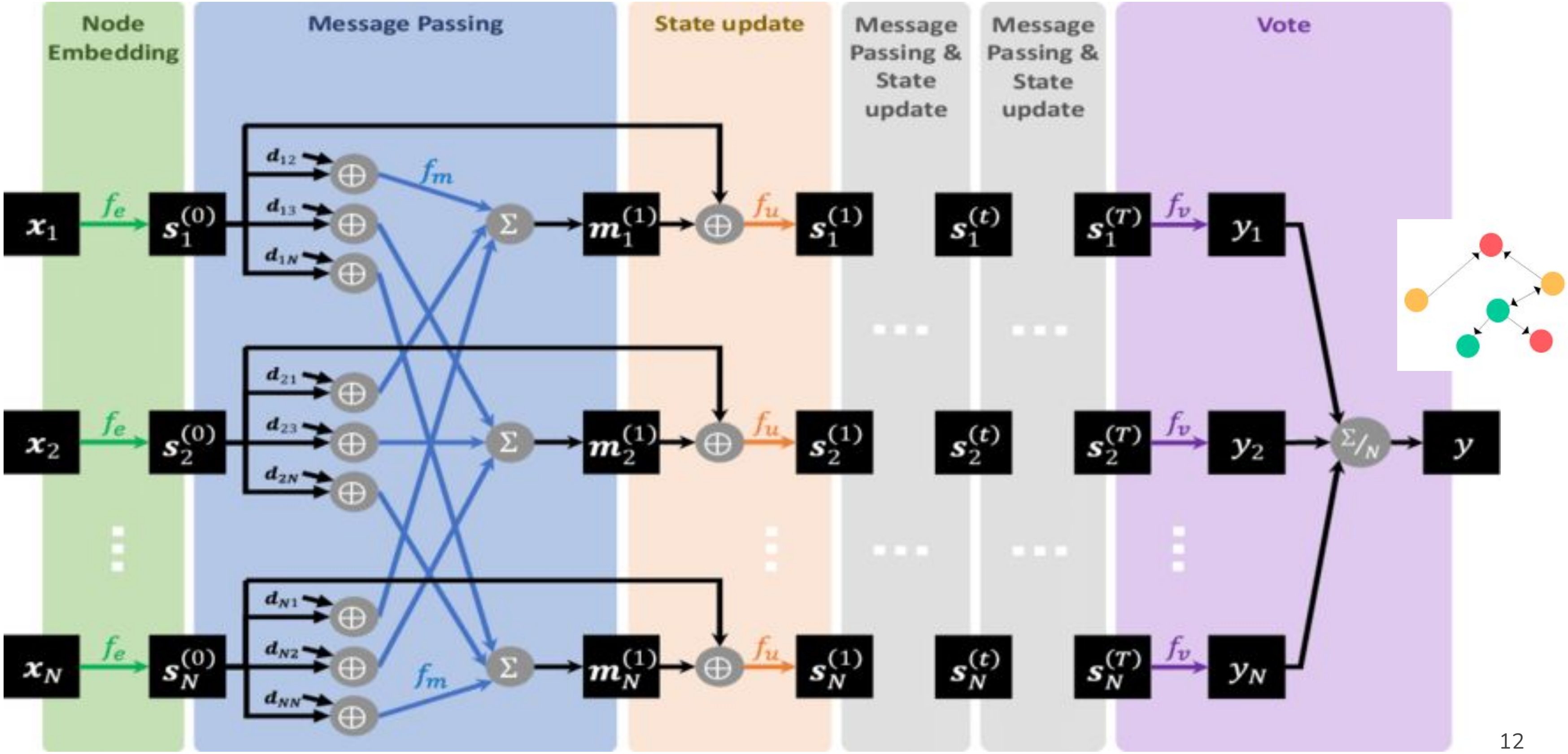
- GANs represent a huge family of double networks that are composed from a **generator** net and a **discriminator** net
  - The generator produces samples close to training samples
  - Discriminator net (adversary) differentiates between samples from the generative net and the training set
  - Use error feedback to improve task of both nets, until discriminator can no longer distinguish
- Can be used to generate samples of data without prior knowledge of the data





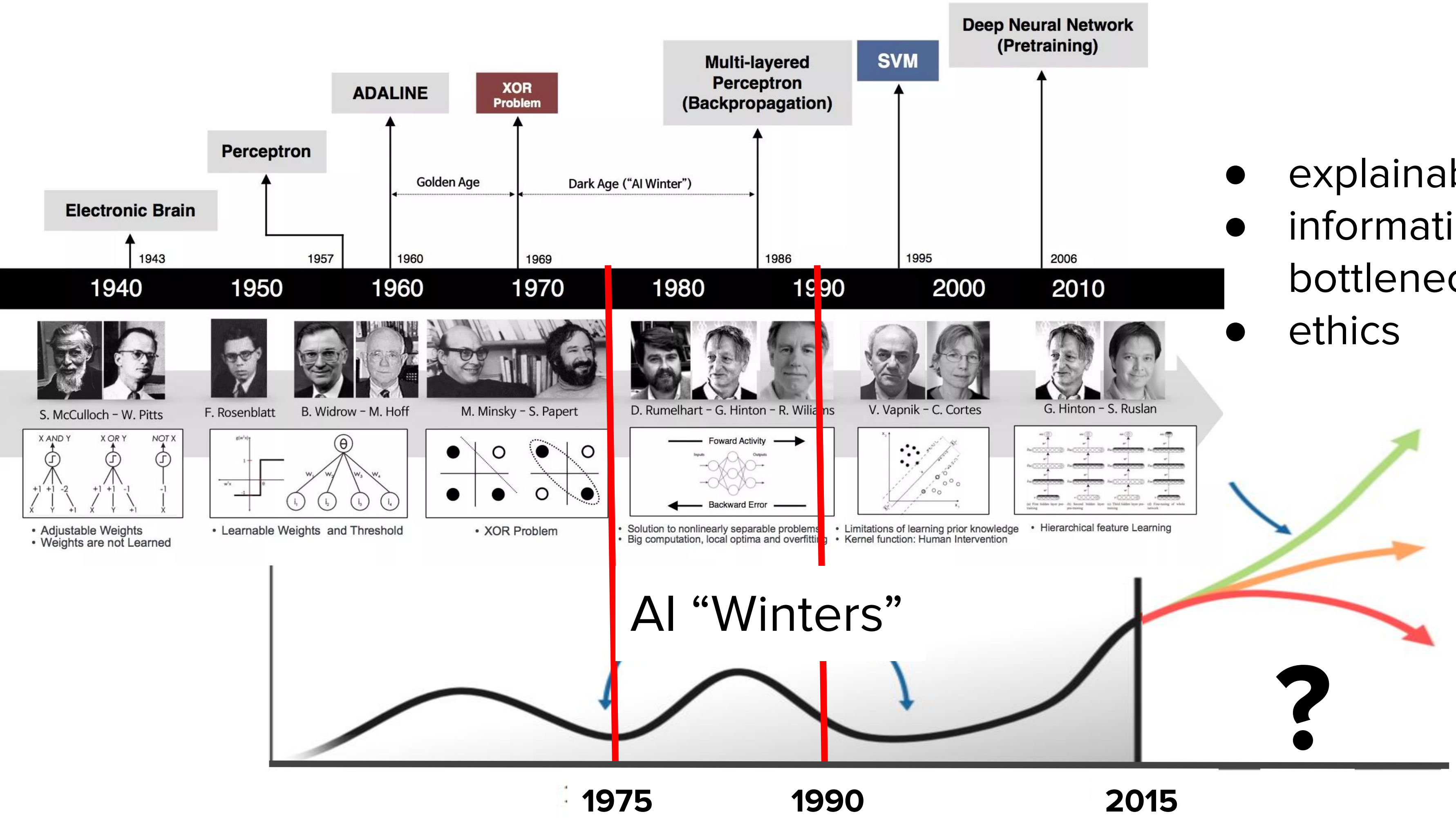
Graph Neural Network (GNN)

- Node prediction
- Edge prediction
- Graph prediction



NN evolution in the past 10 years

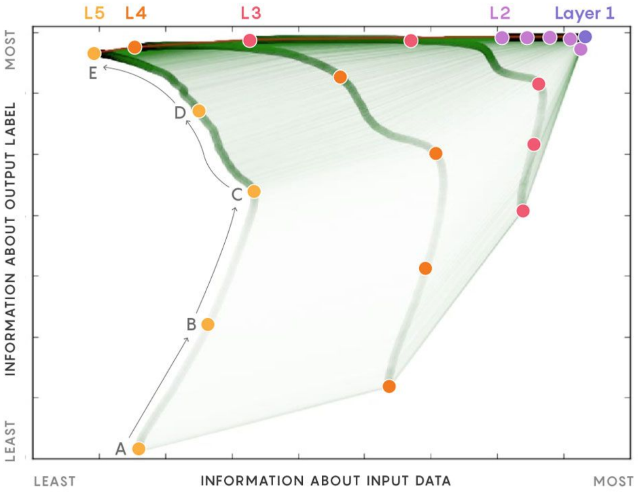
- 2009: handwriting recognition prizes with LSTM
- 2011 DanNet (CNN) beats humans in **visual pattern recognition**
- 2014: GANs, Alexa, Tesla AutoPilot
- 2015: FaceNet starts facial recognition programs
- 2016: AlphaGo **beats Go champion**
- 2017: Google **Translate** and Facebook Translate based on two LSTMs
- 2018: Cambridge analytica, deep fakes
- 2019: **DeepMind** defeats professional StarCraft players with RL + LSTM, Rubik’s Cube solved with a Robot Hand
- 2021: AI Colorectal Cancer Detection Technique Better than Pathologist
- 2022: **chatGPT** model produces “believable” open text


Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
XOR Problem
ADALINE
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch – W. Pitts
F. Rosenblatt
B. Widrow – M. Hoff
M. Minsky – S. Papert
D. Rumelhart – G. Hinton – R. Wiliams
V. Vapnik – C. Cortes
G. Hinton – S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning
AI "Winters"
?
1975
1990
2015

- explainable AI
- information bottleneck
- ethics

Information Bottleneck

- Hidden layers represent a Markov chain of topologically distinct representations
  - Information about the inputs decreases along the hidden layers
  - I(X, h_1) > ...> I(X, h_i) > I(X, h_i+1)
- In the first epochs, the network is trained to fully represent the input data; then, it learns to forget the irrelevant details by compressing the representation of the input



A **INITIAL STATE:** Neurons in Layer 1 encode everything about the input data, including all information about its label. Neurons in the highest layers are in a nearly random state bearing little to no relationship to the data or its label.

B **FITTING PHASE:** As deep learning begins, neurons in higher layers gain information about the input and get better at fitting labels to it.

C **PHASE CHANGE:** The layers suddenly shift gears and start to "forget" information about the input.

D **COMPRESSION PHASE:** Higher layers compress their representation of the input data, keeping what is most relevant to the output label. They get better at predicting the label.

E **FINAL STATE:** The last layer achieves an optimal balance of accuracy and compression, retaining only what is needed to predict the label.

Artificial General Intelligence

- **Common sense**:
  - Current systems may be easily fooled by just slight changes in the input data (for example image taken from another viewpoint)
  - Embed coordinate systems, whole-part relationship (capsules)
- **Abstract concepts**:
  - Current models may be able to distinguish between a jet and a tau, but do not know what a particle is
- **Creativity**:
  - Current models highly specialised and engineered to solve specific problems

[Murray Shanahan, Geoff Hinton]

Self Supervised Learning

- Supervised learning needs many labeled data
- Reinforced learning:
  - Not practical to train in real world (when no simulation is available)
  - takes longer than an average human for a machine to learn a new task
- **Self supervised learning:** Predict everything from everything else - learn representations, rather than learning specific tasks
  - Very large networks trained with large amount of data
  - Fill_ing the bl_anks - Word2Vec, Transformer architecture for NLP
  - Not (yet) so successful for continuous problems (image, video)

[Yann LeCun]

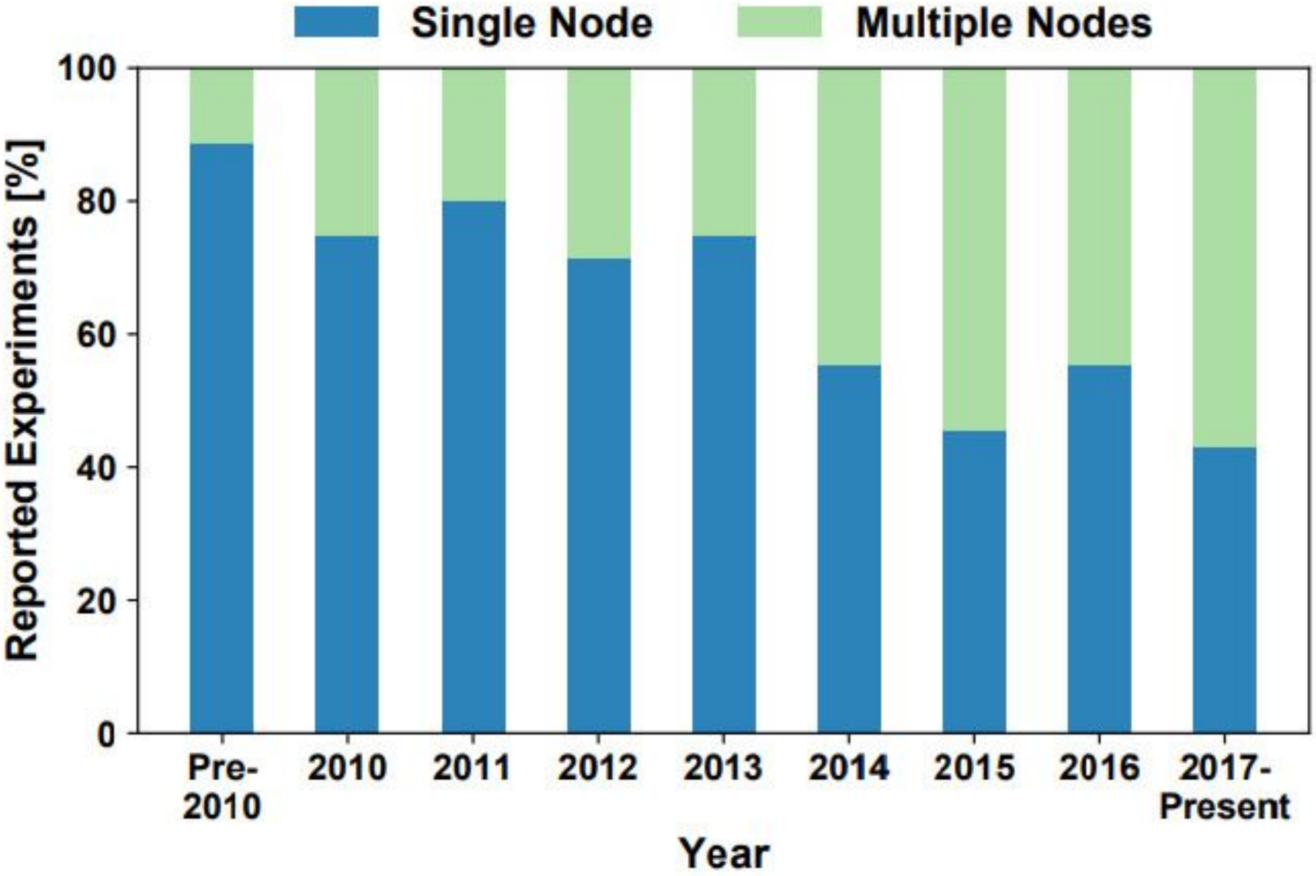
Consciousness Prior

- Current deep learning:
  - **System 1:** fast, unconscious task solving
- Future deep learning:
  - **System 2:** slow, conscious task solving like reasoning, planning
- How?
  - Learn by predicting in abstract space
  - Learn representations (low dimensional vector), derived using attention from a high dimensional vector
  - The prior: the factor graph (joint distribution between a set of variables) is sparse

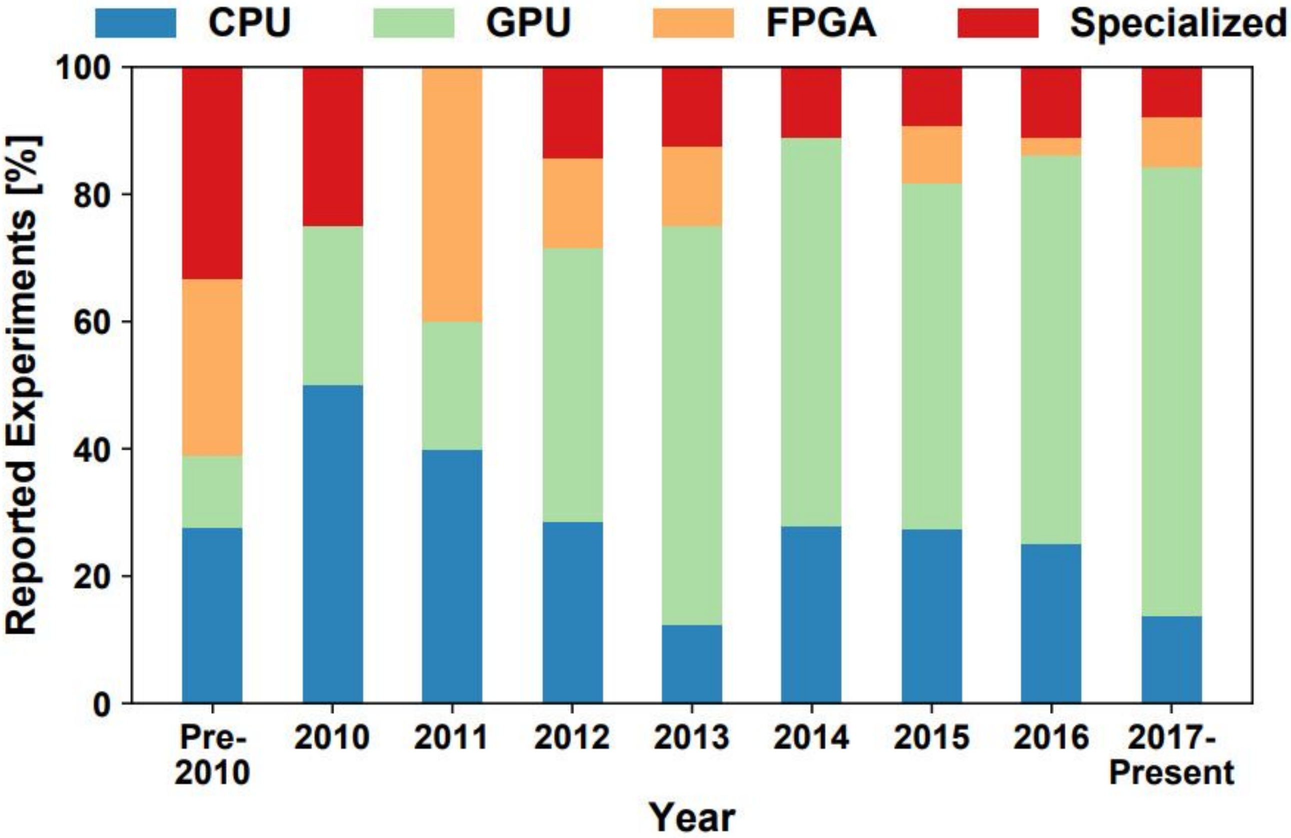
$$P(S) = \frac{\prod_j f_j(S_j)}{Z}$$

Yoshua Bengio, arXiv:1709.08568 [cs.LG]

- Most ML algorithms require significant amount of CPU, RAM and sometimes GPU in order to be applied efficiently
- **Does it fit on your laptop?**



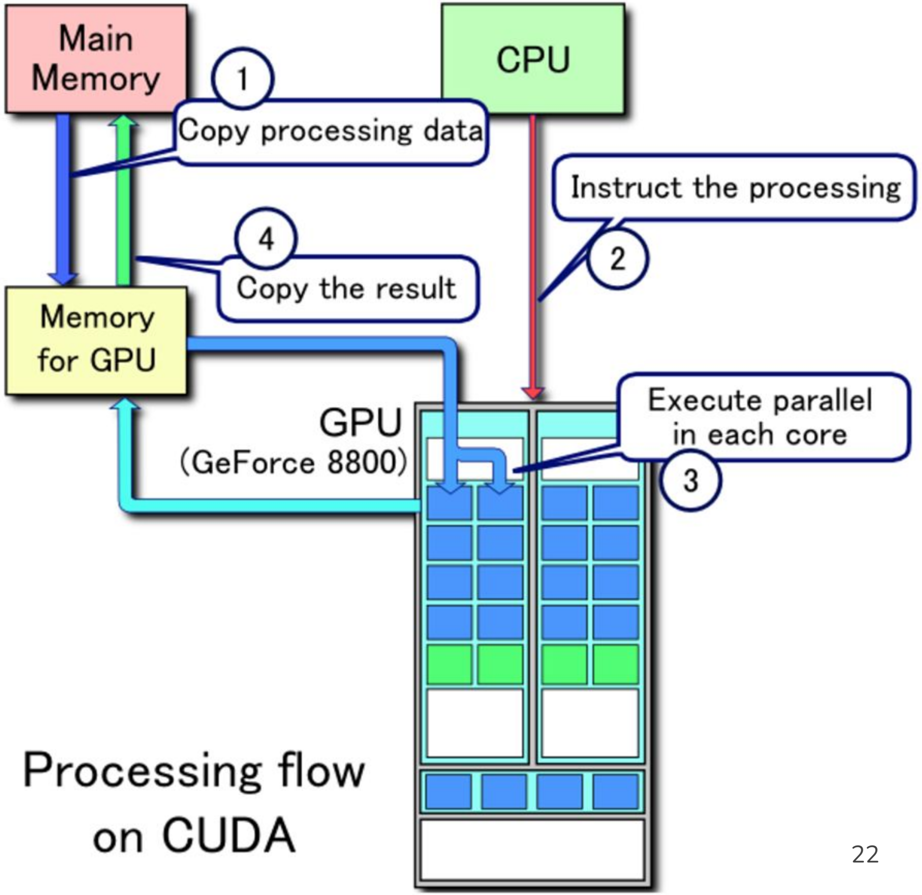
Fraction of non-distributed vs distributed deep learning over time



Use of hardware for deep learning

GPU (Graphical Processing Unit)

- Typically composed of thousands of logical cores
- Excels at matrix and vector operations
  - Gaming, rendering…and ML
- Main vendor: NVidia
  - CUDA: parallel computing platform and programming model



Options

- **NVIDIA CUDA**
  - Market leader, large user community, best support from DL frameworks
  - Can be used in python, C/C++, fortran, Matlab
- **AMD HIP**
  - Limited support for pyTorch and Tensorflow
  - Slightly behind in terms of performances
- **INTEL**
  - Xeon Phi: Poor support
  - Habana Gaudi **(AI processor)**-based AWS EC2 instances available since Oct 2021
- **Google TPU**
  - Good performances, more powerful than cloud GPUs
  - best for training, for prototype and inference better use cheaper alternatives
- **Amazon AWS and Microsoft Azure**
  - Powerful, easy to scale, expensive

How to choose hardware

- **GPUs** have high performance for floating point arithmetic but
  - limited amount of memory available
  - rate at which data can be moved from CPU to GPU
  - memory bandwidth for GPUs must be seen relative to the amount of FLOPS: if one has more floating point units, a higher bandwidth is needed to keep them occupied
- **Matrix operations:** typically bandwidth cost is larger than multiplication cost
  - Particularly important for many small multiplications

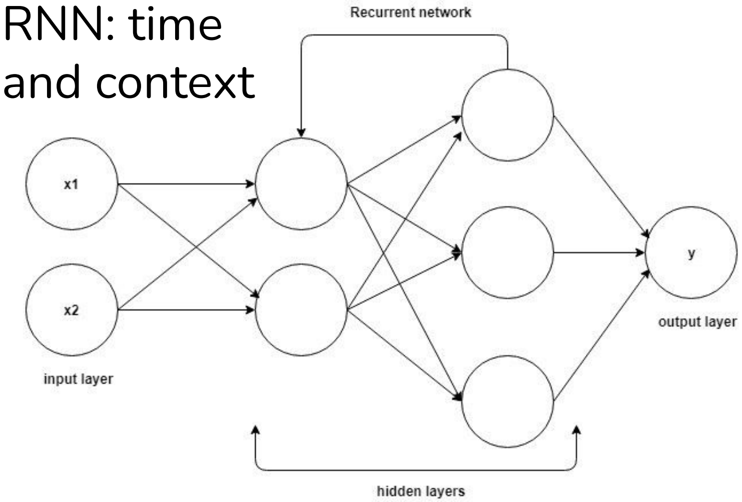
How to choose hardware for NN

- CNNs need a lot of computing power (need high number of cores, FLOPS)
    - Remember AlexNet?
- LSTM and RNN are made of small matrix multiplication: memory bandwidth matters
- Be aware of power consumption and overheating when placing multiple GPUs close to one another!

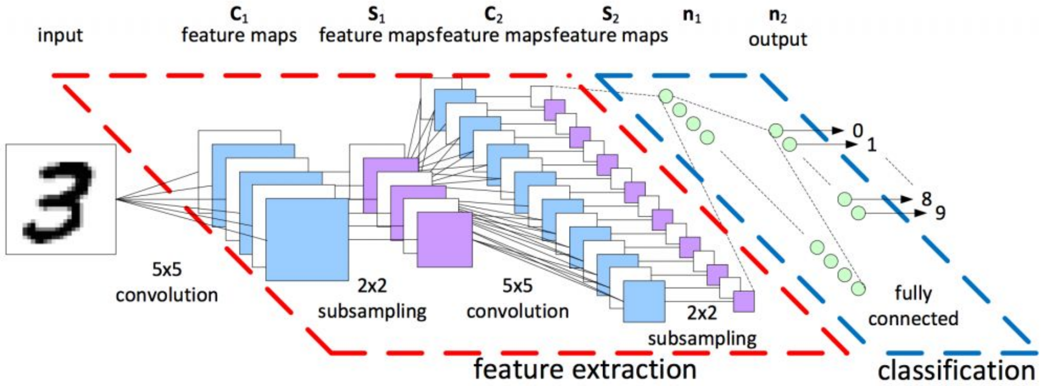
Parallelisation on multiple GPUs

- ‘Easy’ for RNNs and CNNs
- Fully connected networks with transformers poorer performances
- multiple GPUs can be used for tasks trivial to parallelise such as hyperparameter scan

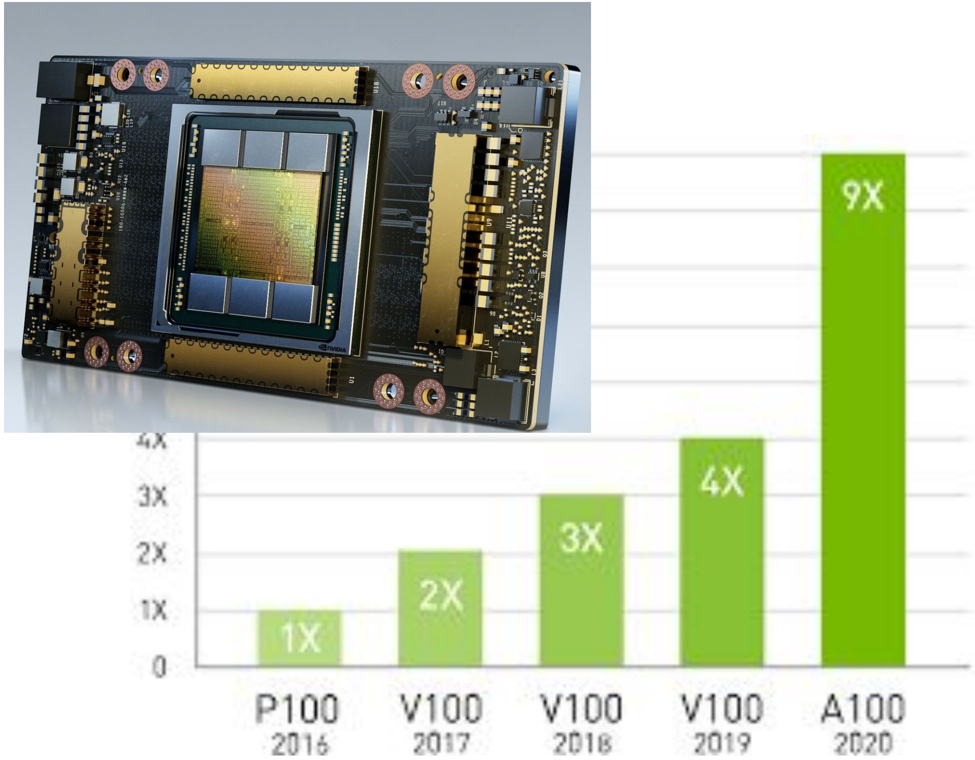
RNN: time and context

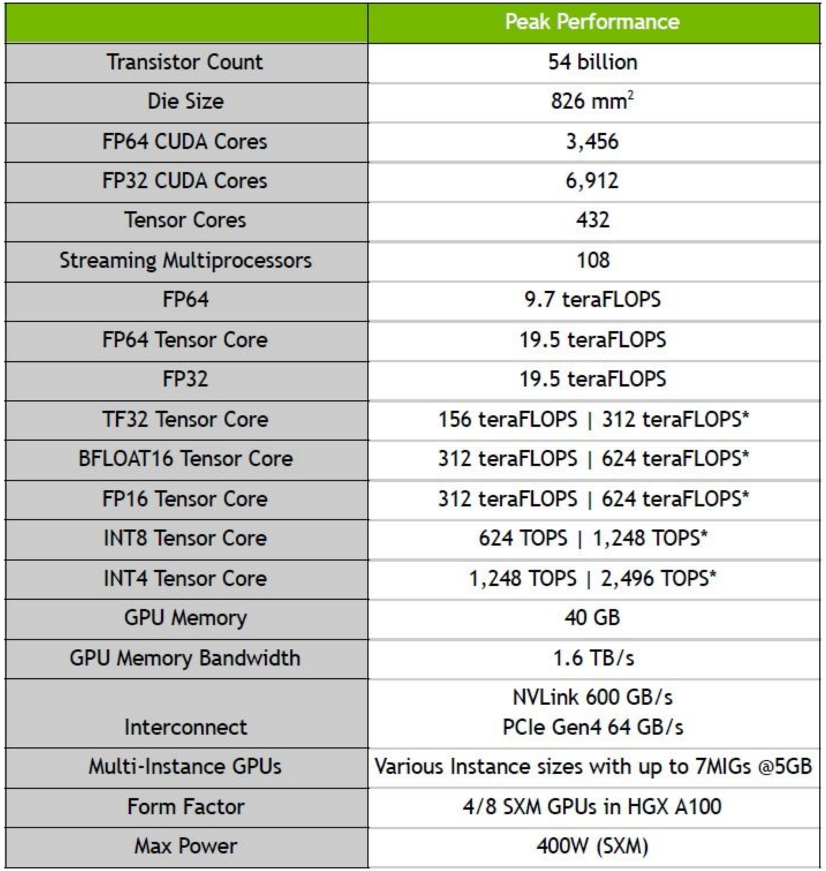


CNN: image recognition



NVIDIA GPUs

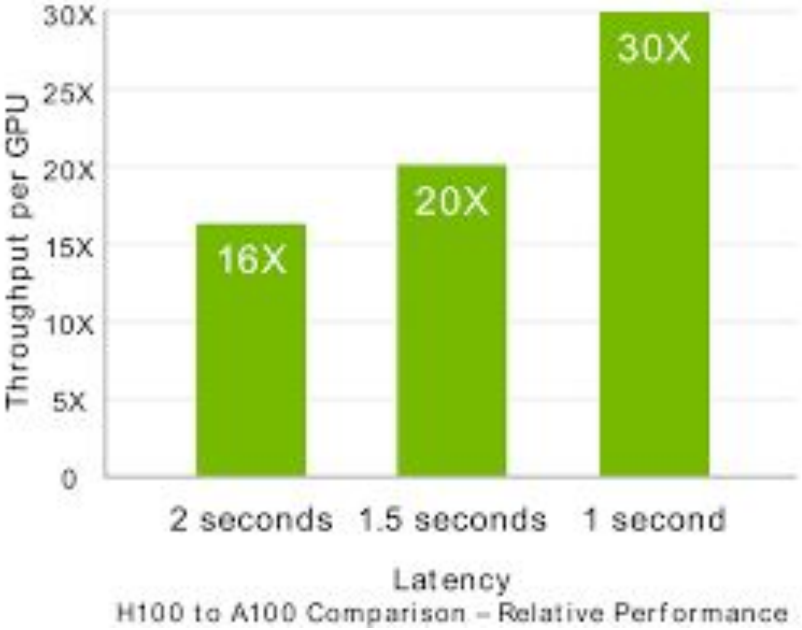




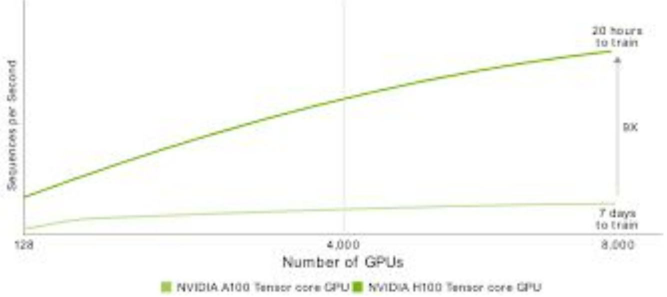
	Peak Performance
Transistor Count	54 billion
Die Size	826 mm^2
FP64 CUDA Cores	3,456
FP32 CUDA Cores	6,912
Tensor Cores	432
Streaming Multiprocessors	108
FP64	9.7 teraFLOPS
FP64 Tensor Core	19.5 teraFLOPS
FP32	19.5 teraFLOPS
TF32 Tensor Core	156 teraFLOPS | 312 teraFLOPS*
BFLOAT16 Tensor Core	312 teraFLOPS | 624 teraFLOPS*
FP16 Tensor Core	312 teraFLOPS | 624 teraFLOPS*
INT8 Tensor Core	624 TOPS | 1,248 TOPS*
INT4 Tensor Core	1,248 TOPS | 2,496 TOPS*
GPU Memory	40 GB
GPU Memory Bandwidth	1.6 TB/s
Interconnect	NVLink 600 GB/s PCIe Gen4 64 GB/s
Multi-Instance GPUs	Various Instance sizes with up to 7MIGs @5GB
Form Factor	4/8 SXM GPUs in HGX A100
Max Power	400W (SXM)

With its multi-instance GPU (MIG) technology, A100 can be partitioned into up to seven GPU instances, each with 10GB of memory.

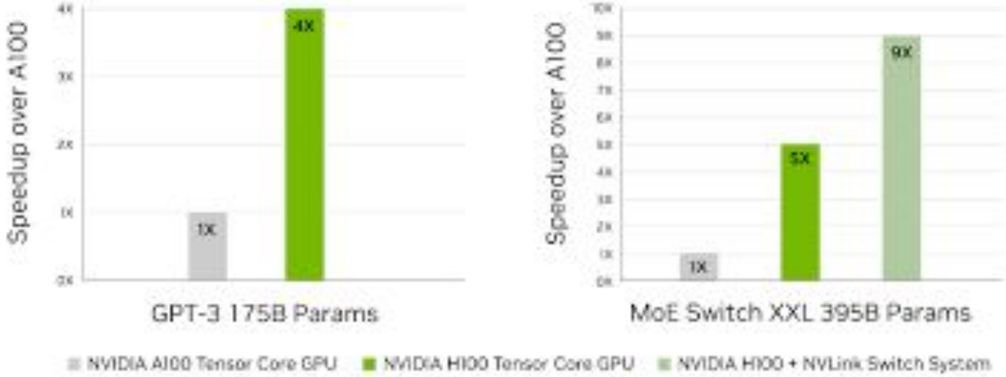
NVIDIA GPUs: latest kid on the block



Megatron Chatbot Inference
(530 billions parameters)



MOE transformer training



Local training

- Both model and data fit on a single machine (**multi cores + GPU**)
    - Multi-core processing:
        - *Embarrassingly parallel process*: use the cores to process multiple images at once, in each layer
        - Use multiple cores to perform SGD of multiple mini-batches in parallel.
    - Use GPU for computationally intensive subroutines like matrix multiplication.
    - Use both multi-core processing and GPU where all cores share the GPU and computationally intensive subroutines are pushed to the GPU.

Distributed training

- Data or model stored across multiple machines
- **Data parallelism**: data is distributed across multiple machines
  - data is too large to be stored on a single machine or to achieve faster training
  - large batch of input data split across a collection of workers, each holding the full model
    - the forward pass involves no communication
    - the backward pass involves aggregating the gradients computed by each individual worker with respect to its separate part of the “global batch”

Data parallelism

- **Synchronous update**: all loss gradients in a given mini-batch are computed using the same weights and full information of the average loss in a given mini-batch is used to update weights
- **Asynchronous update**: as soon as a machine finishes computing updates, the parameters in the driver get updated. Any machine using the parameters will fetch the updated parameters from the server.
- **scaling out**: the "global batch size" (i.e. the total number of samples across all workers that are seen in a single forward pass) increases
  - Affects convergence of DL algorithms and does not reach same accuracy levels that can be obtained with smaller batch sizes

Model parallelism

- model layers split across a collection of workers
- the batch size stays constant, and large models that would not fit the memory of a single device can be trained
- active communication also in the forward pass, thus requiring a lot more communication than a data parallel approach
- unless the advantages of model parallelism are absolutely critical (model doesn’t fit on one machine), most research on large-scale training is done using data parallelism.
- Schemes involving a mix of data and model parallelism also exist: *hybrid parallelism*.

Hardware issues - not only compute

- **Communication** bottlenecks during gradient-aggregation, due to high ratio between the number of parameters and amount of computation
- **Memory** bottlenecks for large networks, particularly when using memory-limited GPUs
- **I/O contention** with large data sets, or data sets with many small samples, and the data has to be physically stored on shared storage facilities.
- On large system such as **HPCs**, data are typically stored on shared file systems
  - Might be faster to copy data locally to worker node

Architectural choices

- **Your model doesn’t fit in the GPU memory?**
  - tune the model e.g. by reducing its connectivity (if that is acceptable) to fit the hardware
  - use model parallelism to distribute the model over multiple GPUs
  - use a data parallel approach on CPUs (if it does fit in CPU memory)
- **Your dataset doesn’t fit in the GPU memory?**
  - Make sure your data arrives fast enough to the GPU, otherwise revert to CPU
- Do you have access to a **system with a few GPU nodes, but thousands of CPUs**?
  - Development stage: run a limited number of examples (potentially even with a smaller model) on a single GPU, which allows quick development cycles
  - Production: may require CPUs because of memory constraints

That's all Folks!
SUPERB wallpapers

Hands-on test

- Evaluation might be based on:
  - Improve ML model trying to improve performances vs training time
  - Measure scaling performances vs complexity
- May include cross-validation and hyper-parameter tuning:
  - K-fold training
  - Grid search

Hands-on today

- Point your browser to: https://yoga.to.infn.it
- Open a terminal:
  - cd MLCourse-2223
  - git pull
  - cp Notebooks/Day4/* ../
- From JupyterHub Home tab:
  - start and run Keras.ipynb
  - Start and run BigDL.ipynb

Spare slides

Requirements for ML framework

- Run any ML algorithm of our choice in parallel
- Large datasets to be processed
- Multi-tenancy, so different users can request simultaneously ML pipelines
- An efficient resource management system for the cluster
- Support heterogeneous architectures
    - CPUs, GPUs, ...

ML as a Service (MLaaS)

CHALLENGES

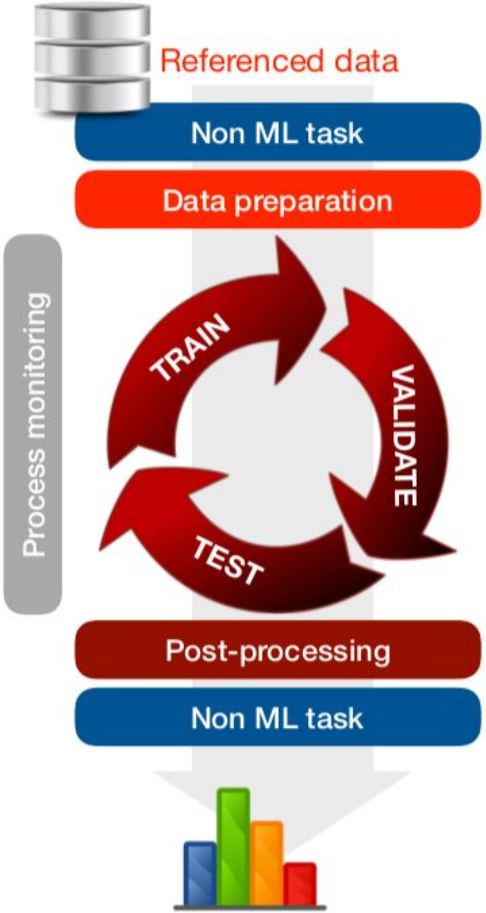
- Reconstruction
- Analysis
- Trigger
- Data quality
- Detector monitoring
- Computing operations
- Monte Carlo tuning
- ...

REQUIREMENTS

- Workflow definition
  - Results reproducibility
- Multi-tenancy (scheduling, authentication...)
- Parallel execution and scaling
- Data handling
- Ease of use and management
- ...

IMPLEMENTATION

- Lightweight virtualization
- Modularity
- Flexibility
- Heterogeneous back-end infrastructures
- ...



Referenced data

Non ML task

Data preparation

Process monitoring

TRAIN

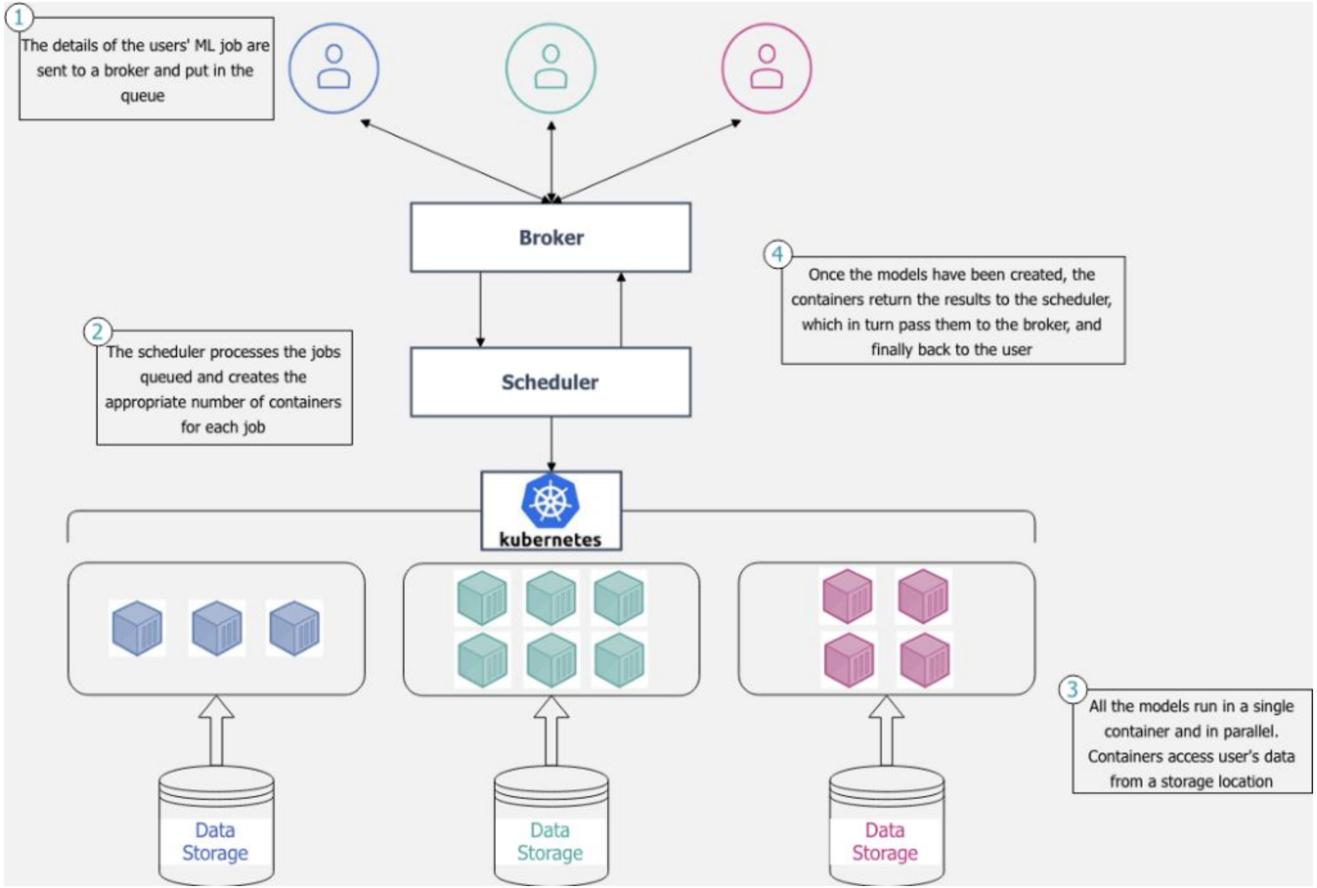
VALIDATE

TEST

Post-processing

Non ML task

Example architecture

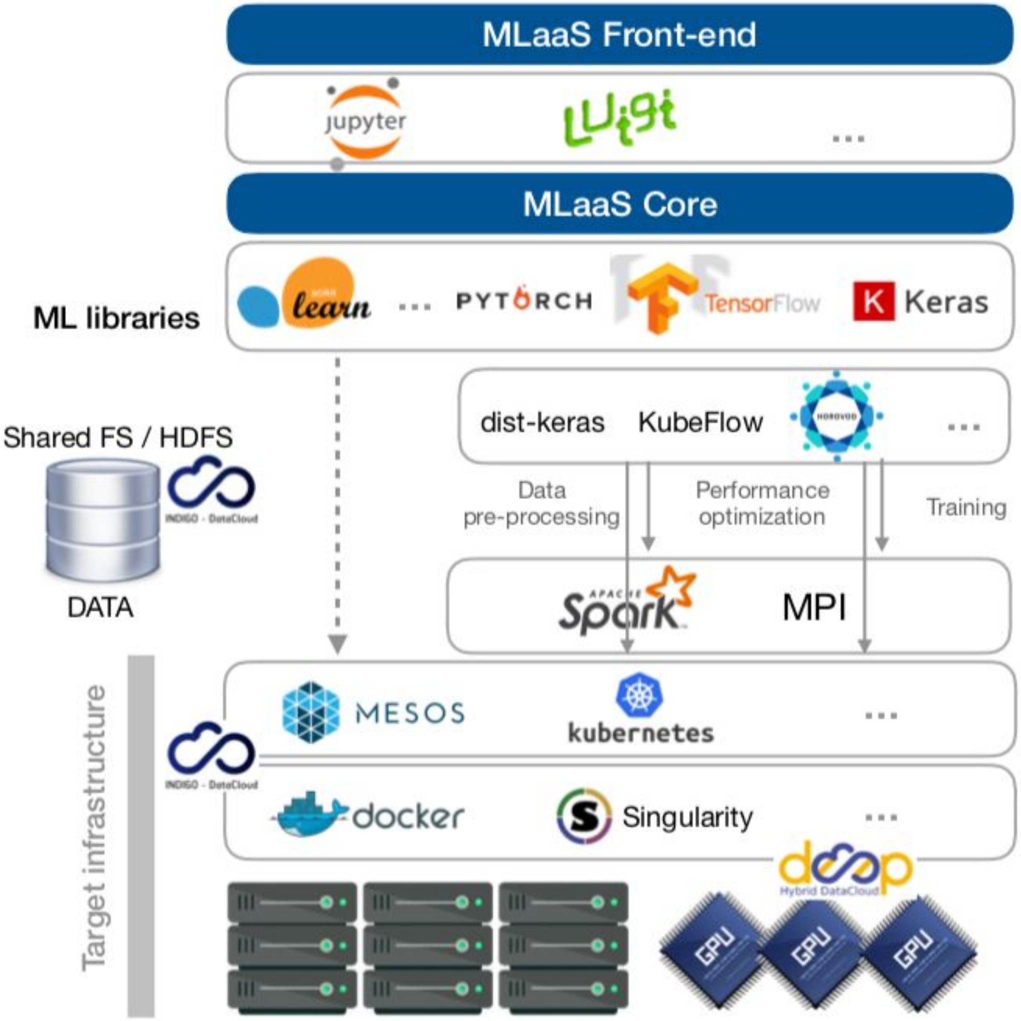


1 The details of the users' ML job are sent to a broker and put in the queue

2 The scheduler processes the jobs queued and creates the appropriate number of containers for each job

3 All the models run in a single container and in parallel. Containers access user's data from a storage location

4 Once the models have been created, the containers return the results to the scheduler, which in turn pass them to the broker, and finally back to the user



MLaaS Front-end

jupyter Luigi ...

MLaaS Core

ML libraries

learn ... PYTORCH TensorFlow Keras

dist-keras KubeFlow HOROVOD ...

Data pre-processing

Performance optimization

Training

Shared FS / HDFS

DATA

Spark MPI

MESOS kubernetes ...

docker Singularity ...

deep Hybrid DataCloud

GPU GPU GPU

Target infrastructure

INDIGO - DataCloud

- Workflow definition
- Process Monitoring
- Authentication

INDIGO - DataCloud

Deep Learning framework

Distributed DL libraries

Cluster framework (parallelize task)

Orchestrator (schedule on resources)

Packetization and virtualization

Resources:

- Bare metal
- Infrastructure as a Service