Big data science Day 3

F. Legger - INFN Torino

https://github.com/Course-bigDataAndML/MLCourse-INFN-2022

Yesterday

- Big data
- Analytics
- Machine learning

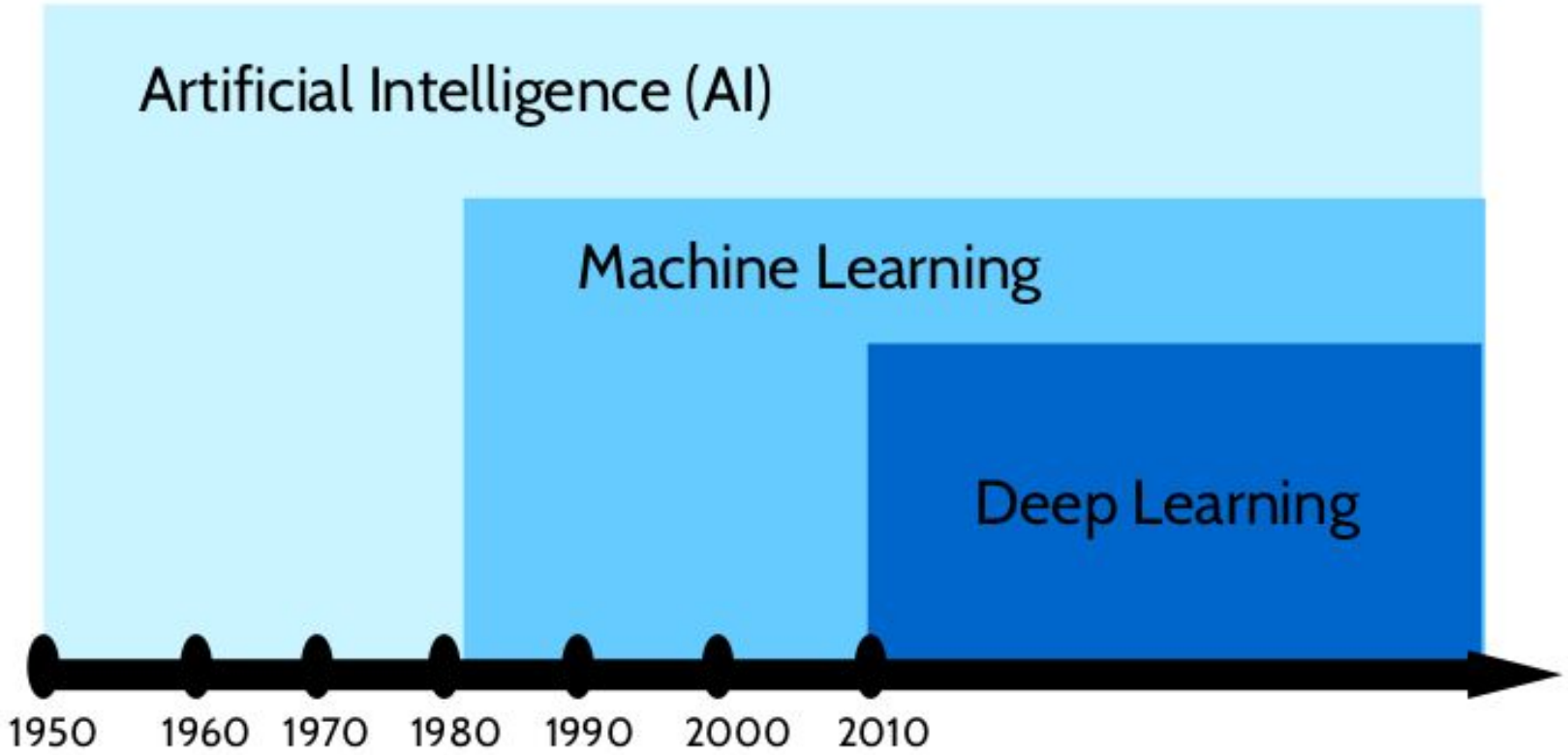
Today

- **Deep learning**
- History, current models and future directions
- Parallelisation in ML
- Heterogeneous architectures

Summary from yesterday

- 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:
  - Find algorithm that will give the best results
  - Train model
  - Tune hyperparameters
  - Do cross-validation
  - Do inference

Deep Learning is a subfield of ML concerned with algorithms inspired by the structure and function of the brain called artificial neural networks [Jason Brownlee]

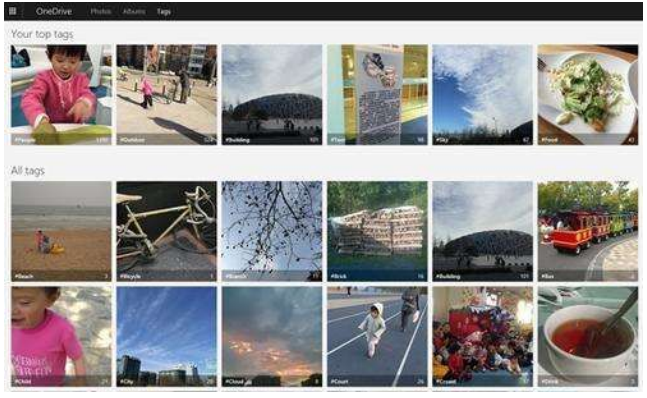


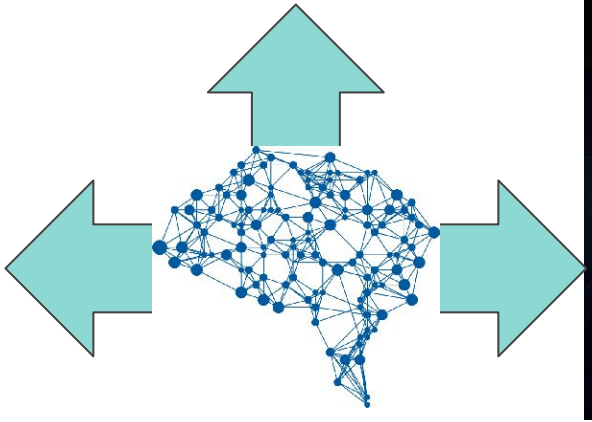
Machine translation
Real-time translation into Mandarin Chinese (2012)

Strategy games
DeepMind beats Go world champion (2017)

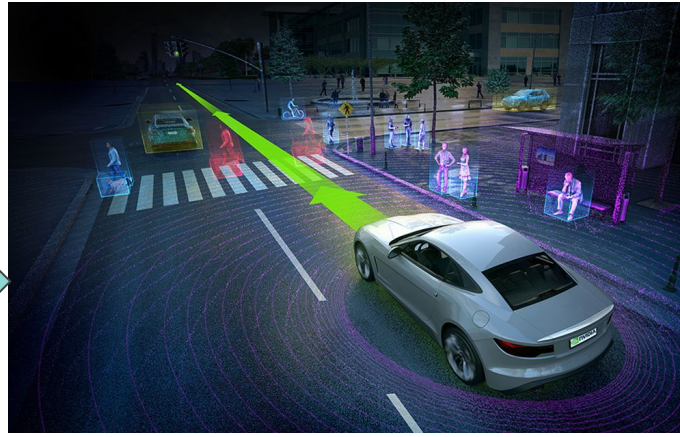
Creativity

Visual recognition



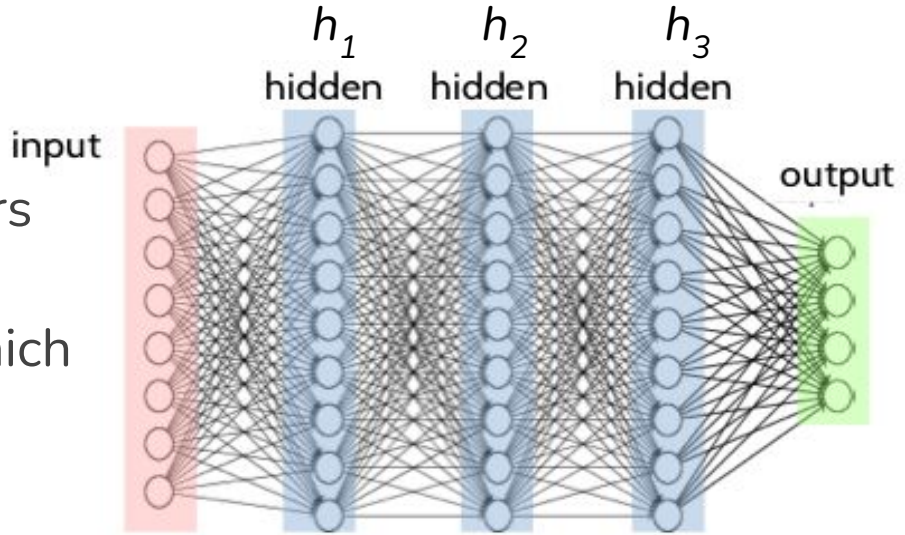


Self driving cars



Deep Learning

- Neural network with several layers
  - Deep vs shallow
- A family of parametric models which learn non-linear hierarchical representations:



$$a_L(\mathbf{x};\boldsymbol{\Theta}) = h_L(h_{L-1}(...(h_1(\mathbf{x},\boldsymbol{\theta}_1),\boldsymbol{\theta}_{L-1}),\boldsymbol{\theta}_L)$$

input

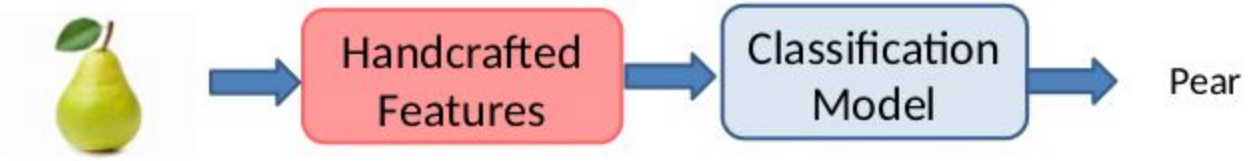
parameters of the network

non-linear activation function

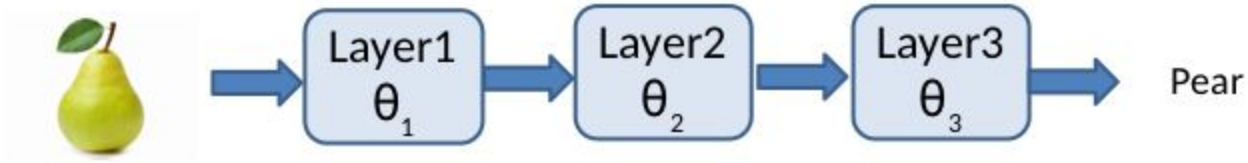
parameters of layer L

Learning hierarchical representations

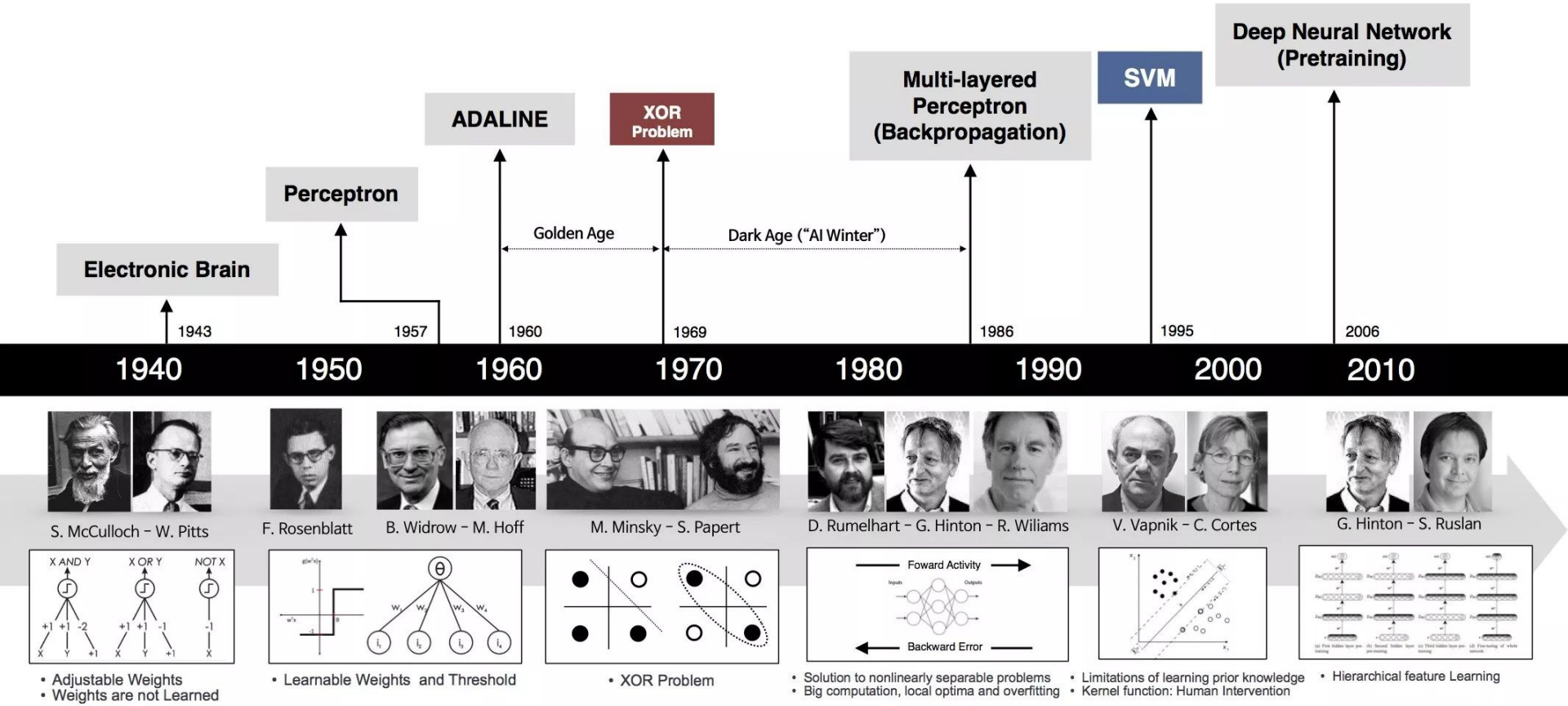
- Traditional framework



- Deep Learning



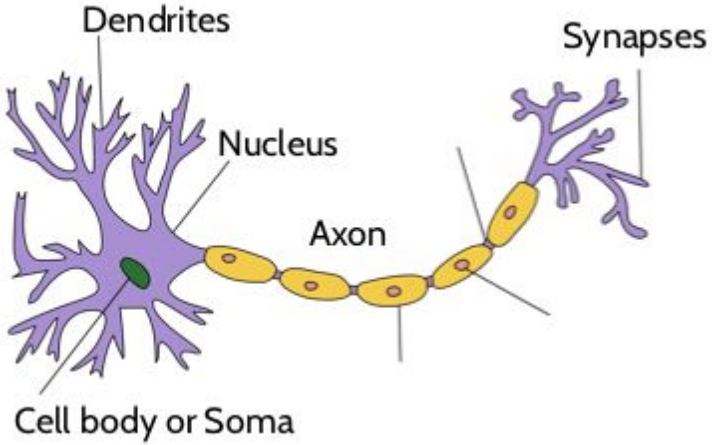
Brief history of neural networks

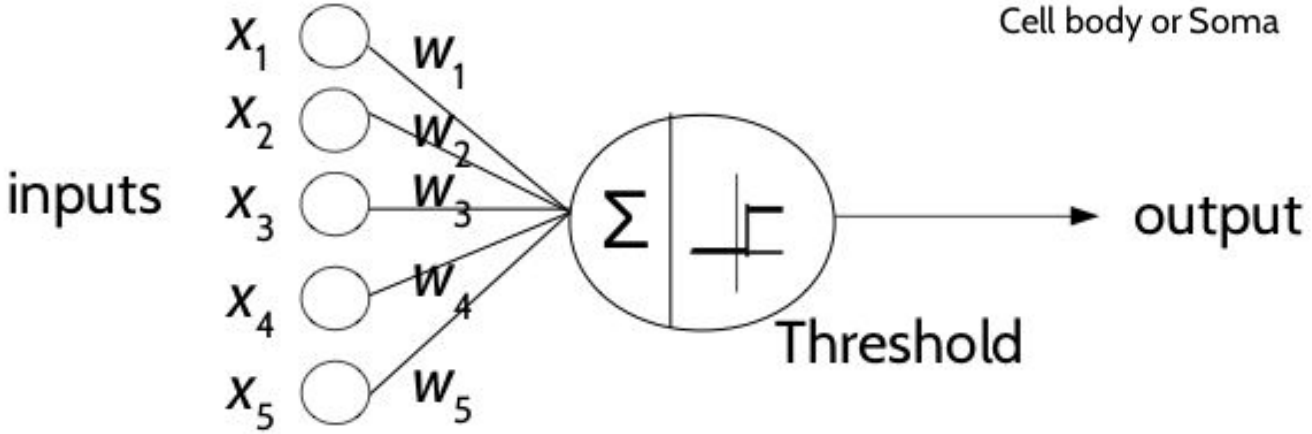


- Adjustable Weights
- Weights are not Learned
- Learnable Weights and Threshold
- XOR Problem
- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting
- Limitations of learning prior knowledge
- Kernel function: Human Intervention
- Hierarchical feature Learning

1943 – McCulloch & Pitts Model

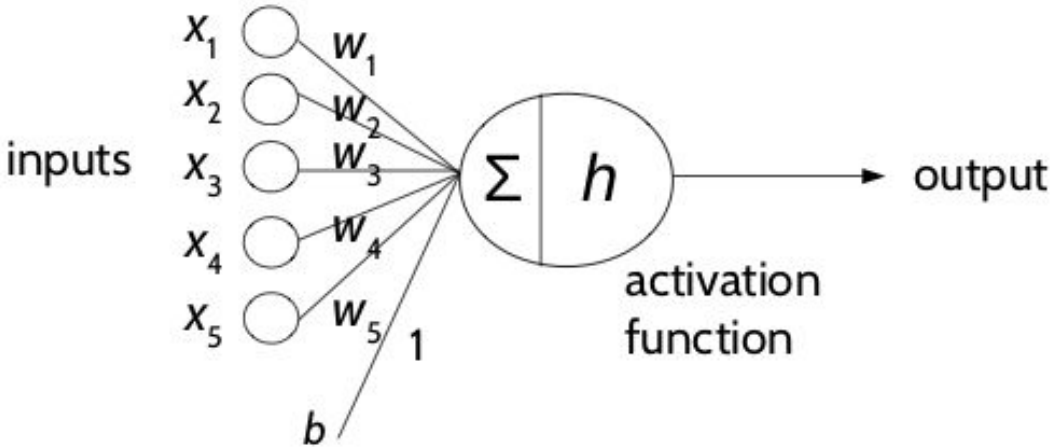
- Early model of artificial neuron
- Generates a binary output
- The weights values are fixed

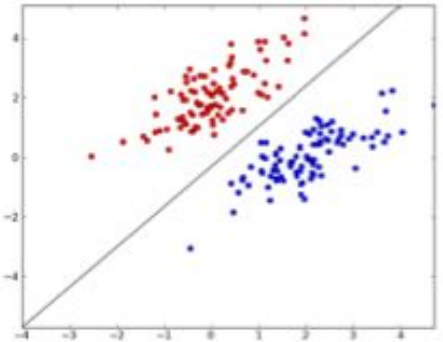




1958 – Perceptron by Rosemblatt

- Perceptron as a machine for linear classification
- Main idea: Learn the weights and consider bias.
  - One weight per input
  - Multiply weights with respective inputs and add bias
  - If result larger than **threshold** return 1, otherwise 0

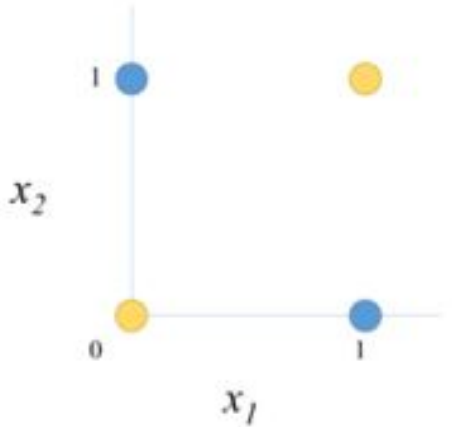


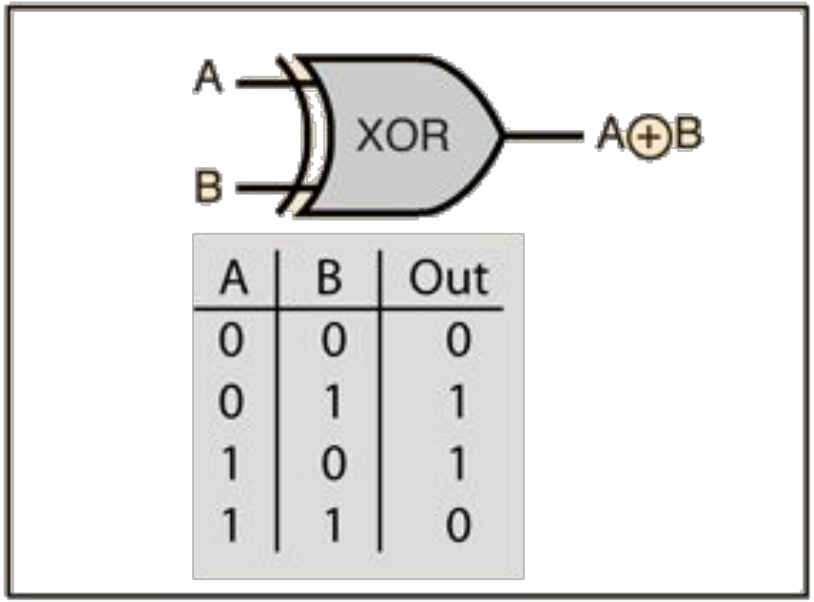


First NN winter

- 1970- Minsky. The XOR cannot be solved by perceptrons.

- Neural models cannot be applied to complex tasks.



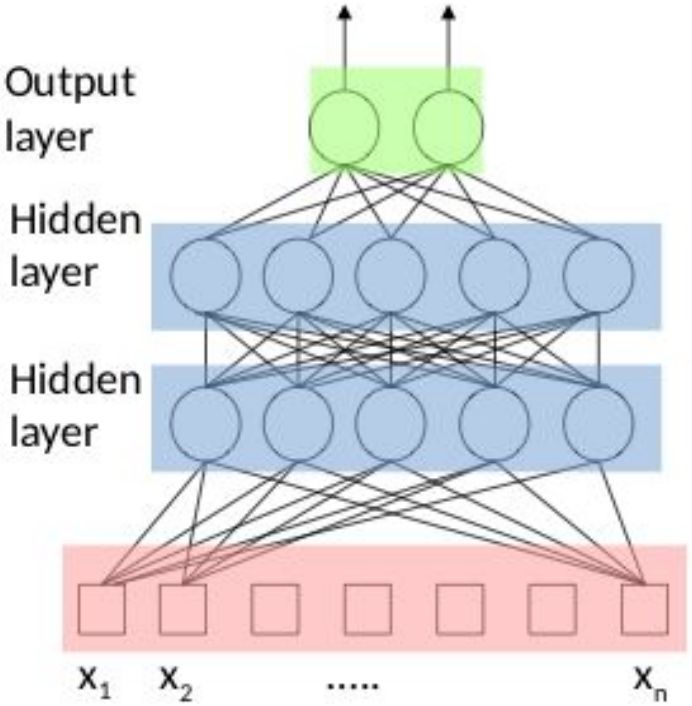


A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

Multi-layer Feed Forward Neural Network

- 1980s. Multi-layer Perceptrons (MLP) can solve XOR.

- ML Feed Forward Neural Networks:
  - Densely connect artificial neurons to realize compositions of non-linear functions
  - The information is propagated from the inputs to the outputs
  - The input data are usually *n*-dimensional feature vectors
  - Tasks: Classification, Regression



How to train it?

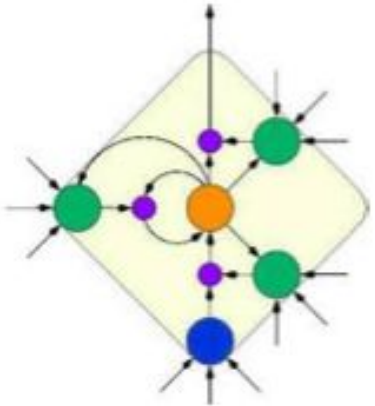
- Rosenblatt algorithm* not applicable, as it expects to know the desired target
  - For hidden layers we cannot know the desired target
- Learning MLP for complicated functions can be solved with **Back propagation** (1980)**
  - efficient algorithm for complex NN which processes large training sets

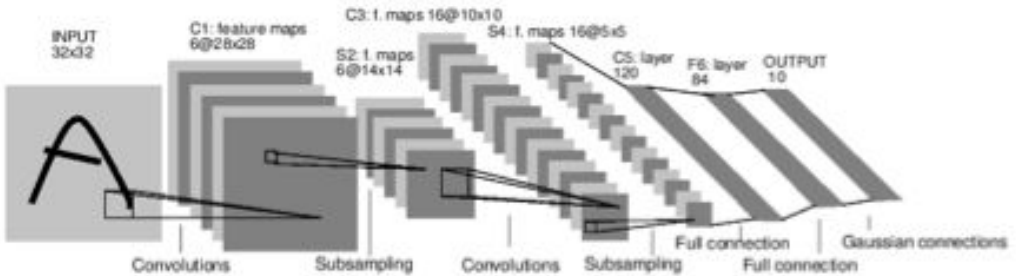
* Remember? Rosenblatt developed a method to train a single neuron

** See yesterday lecture

1990s - CNN and LSTM

- Important advances in the field:
  - Backpropagation
  - Recurrent Long-Short Term Memory Networks (Schmidhuber, 1997)
  - Convolutional Neural Networks - LeNet: OCR solved before 2000s (LeCun, 1998).

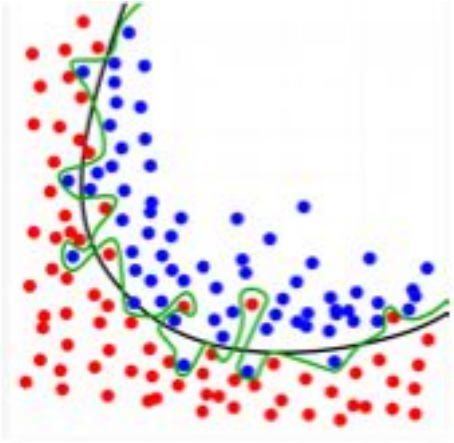


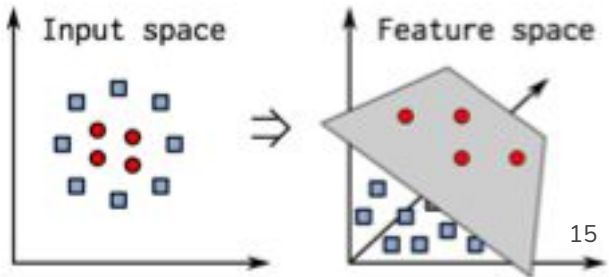


OCR: Optical character recognition

Second NN Winter

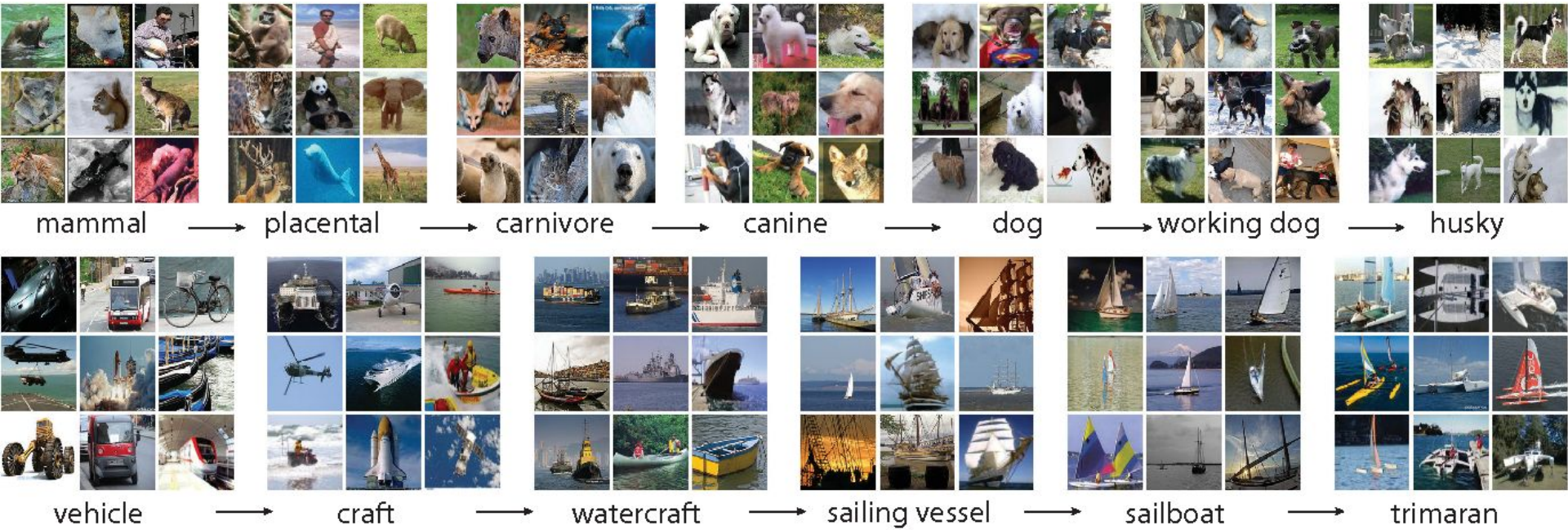
- NN cannot exploit many layers
  - Overfitting
  - Vanishing gradient (with NN training you need to multiply several small numbers → they become smaller and smaller)
- Lack of processing power (no GPUs)
- Lack of data (no large annotated datasets)
- Kernel Machines (e.g. SVMs) suddenly become very popular





ImageNet

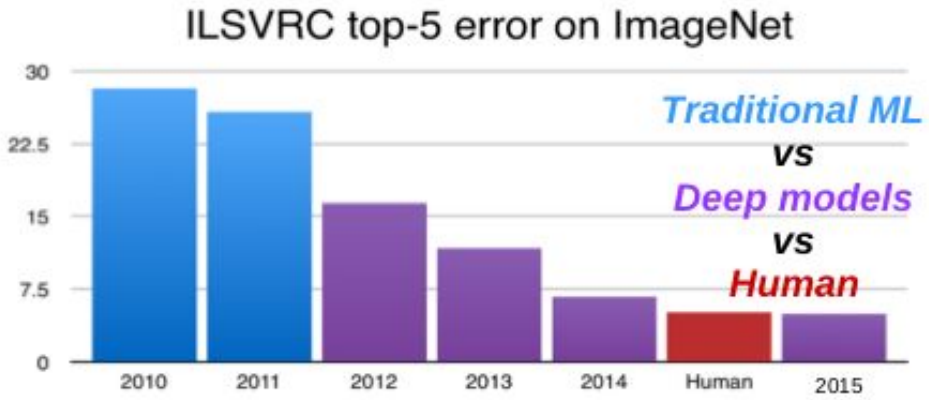
A Large-Scale Hierarchical Image Database (2009)



https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5206848&tag=1

2012 - AlexNet

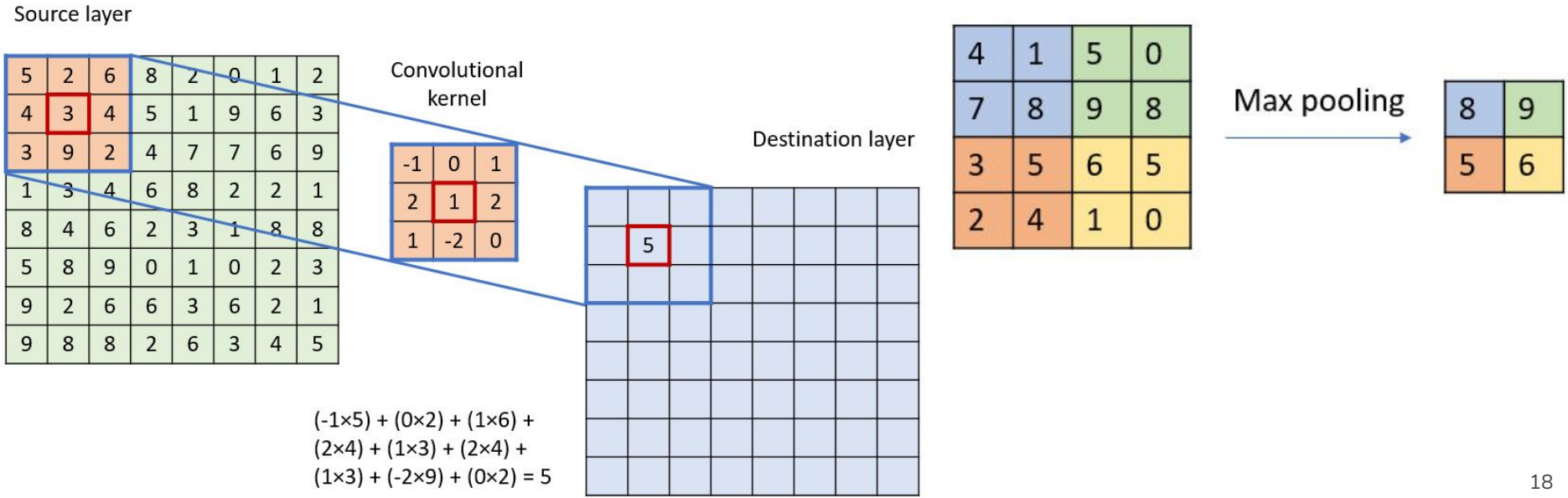
- Hinton's group implemented a CNN similar to LeNet [LeCun1998] but...
  - Trained on ImageNet (1.4M images, 1K categories)
  - With 2 GPUs
  - Other technical improvements (ReLU, dropout, data augmentation)



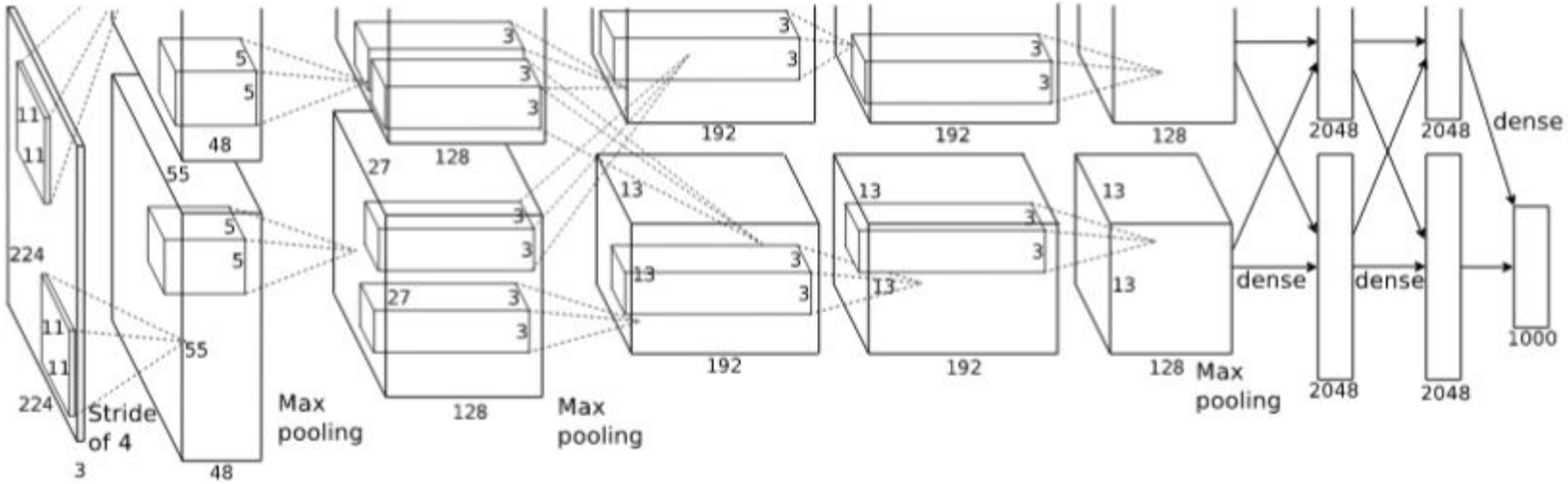
https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Convolutional Neural Networks (CNN)

- **Convolutional** layer: two functions produce a third that describes how the shape of one is changed by the other
- **pooling** layer: reduce dimensionality



AlexNet - CNN



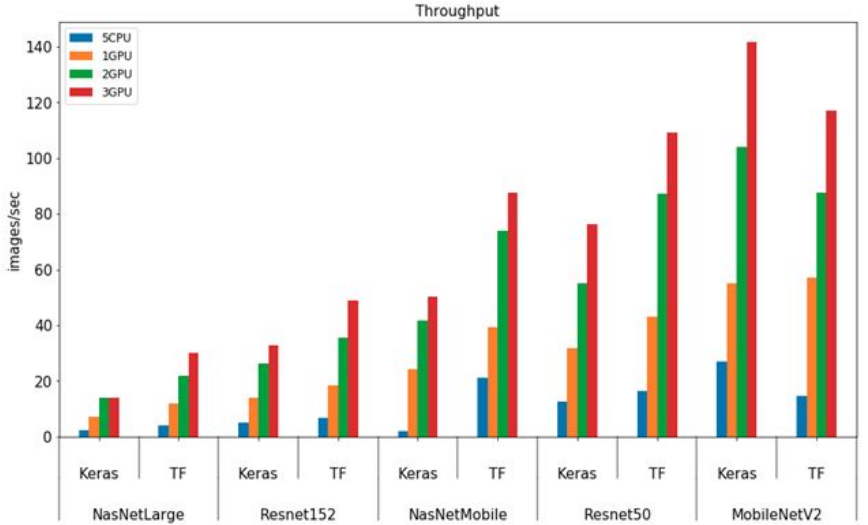
- 60M parameters
- Limited information exchange between GPUs

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Why Deep Learning now?

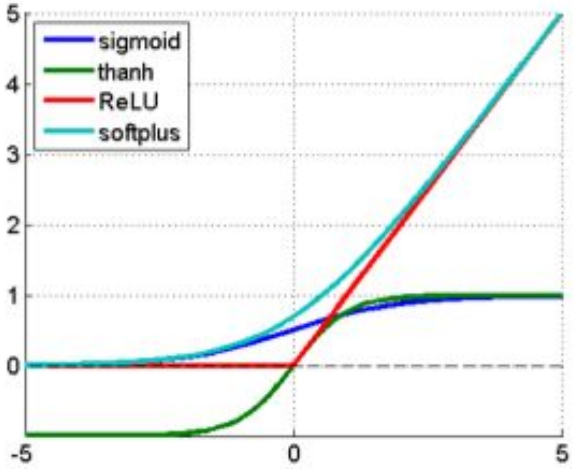
- Three main factors:
  - Better hardware
  - Big data
  - Technical advances:
    - Layer-wise pretraining
    - Optimization (e.g. Adam, batch normalization)
    - Regularization (e.g. dropout)

    ....



Rectified Linear Units (2010)

$$f(x) = \max(0, x)$$



- More efficient gradient propagation: (derivative is 0 or constant)
- More efficient computation: (only comparison, addition and multiplication).
- Sparse activation: e.g. in a randomly initialized networks, only about 50% of hidden units are activated (having a non-zero output)

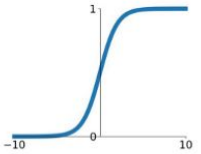
https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf

Activation functions

- Classification: sigmoid functions
  - sigmoids and tanh functions are sometimes avoided due to the vanishing gradient problem
- **ReLU** function is a general activation function
  - dead neurons in our networks -> the leaky ReLU
  - ReLU function should only be used in the hidden layers
  - As a rule of thumb, start with ReLU

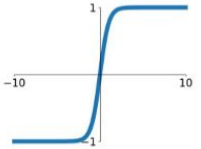
Sigmoid

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



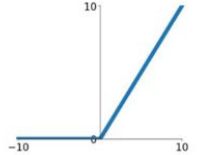
tanh

$$\tanh(x)$$



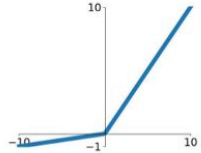
ReLU

$$\max(0, x)$$



Leaky ReLU

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

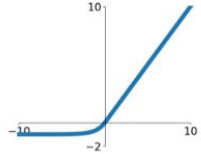


Maxout

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

ELU

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



Regularization

- One of the major aspects of training the model is overfitting -> the ML model captures the noise in your training dataset
- The **regularization** term is an addition to the loss function which helps generalize the model
  - **L1** or Lasso regularization adds a penalty which is the sum of the absolute values of the weights
    - L1+MSE

$$Min(\sum_{i=1}^{n}(y_i - w_i x_i)^2 + p\sum_{i=1}^{n}|w_i|)$$

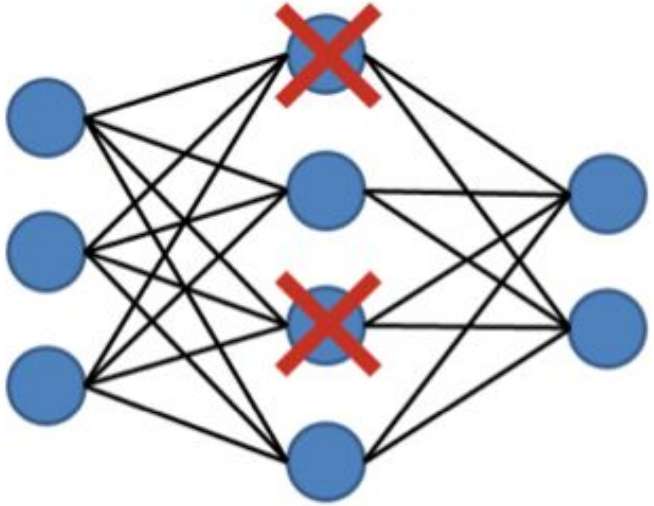
  - **L2** or Ridge regularization adds a penalty which is the sum of the squared values of weights
    - L2+MSE

$$Min(\sum_{i=1}^{n}(y_i - w_i x_i)^2 + p\sum_{i=1}^{n}(w_i)^2)$$

- **Early Stopping** is a time regularization technique which stops training based on given criteria

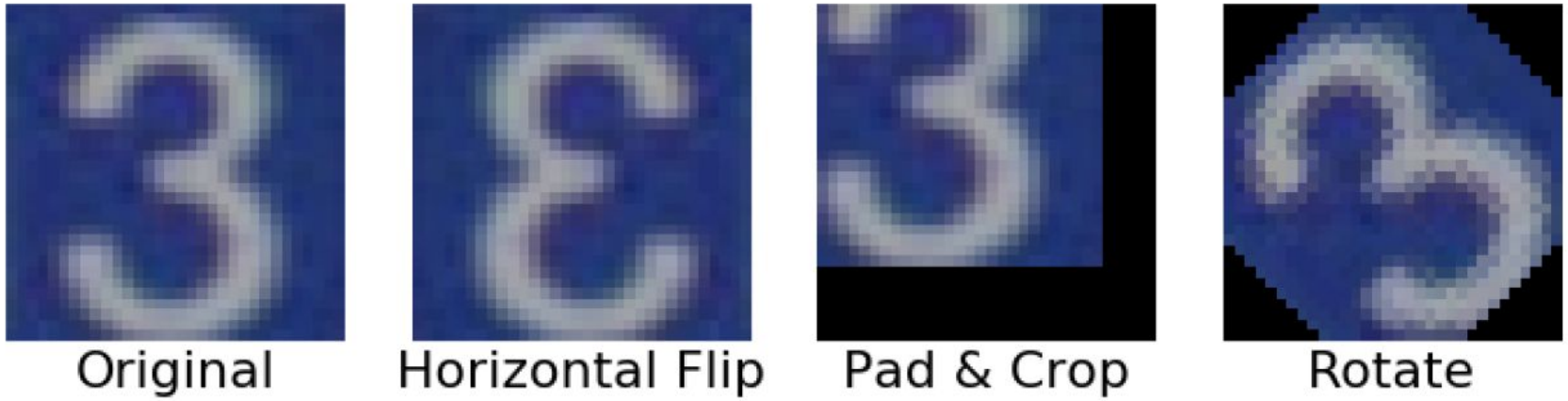
Regularization - Dropout

- For each instance drop a node (hidden or input) and its connections with probability *p* and train
- Final net just has all averaged weights (actually scaled by 1-*p*)
- As if ensembling 2^n different network substructures



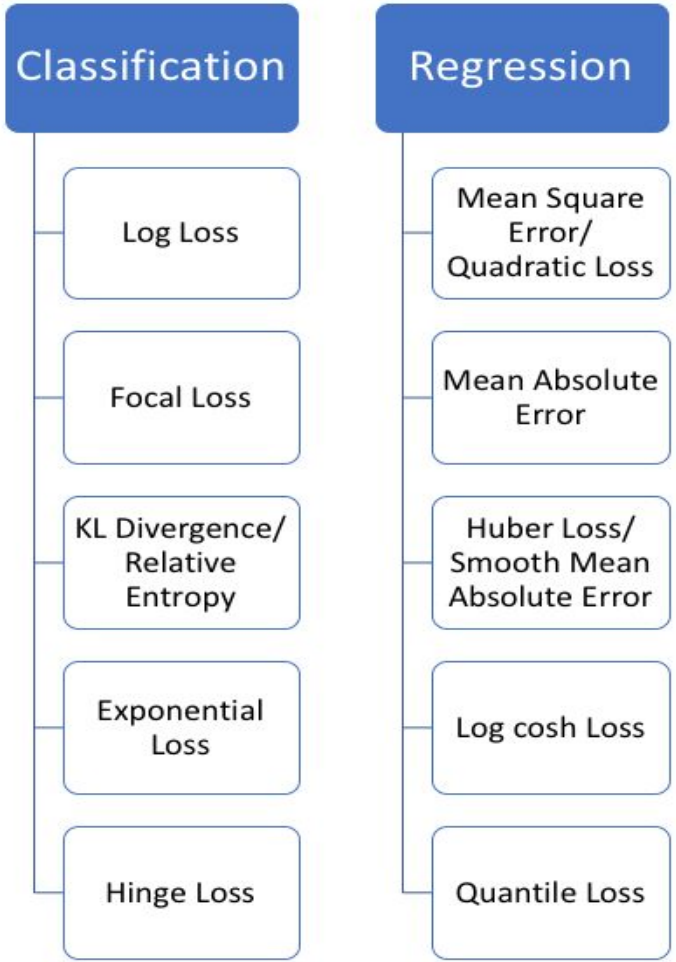
Data augmentation

- Techniques to significantly increase the diversity of data available for training models, without actually collecting new data
- Data augmentation techniques such as cropping, padding, and horizontal flipping are commonly used to train large neural networks



Loss functions

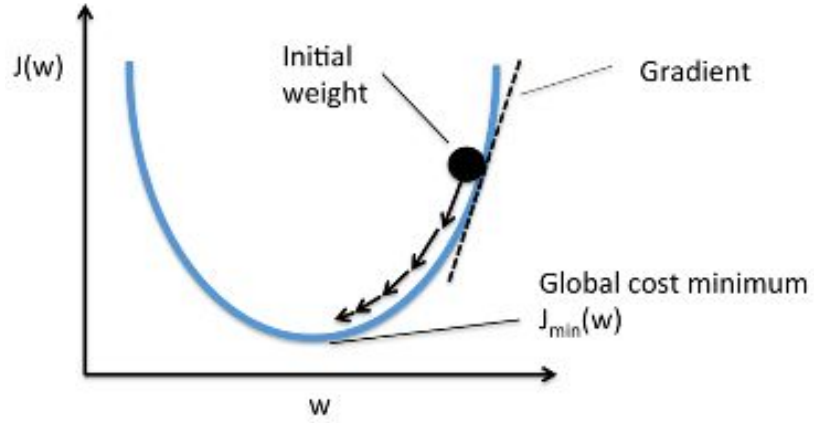
- https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0
- https://www.wikiwand.com/en/Loss_functions_for_classification



Stochastic Gradient Descent (SGD)

- Start with randomly initialised weights
- **update equation**:

$$w_1 \leftarrow w_1 - \eta \frac{\partial L}{\partial w_1}$$



- **η** is a constant called the **learning rate** that controls how fast we train

- If $\frac{\partial L}{\partial w_1}$ is positive, w_1 will decrease, which makes L decrease.
- If $\frac{\partial L}{\partial w_1}$ is negative, w_1 will increase, which makes L decrease.

- **Stochastic** vs **Batch (BGD)**-> the parameters are updated using only one single training instance (usually randomly selected) in each iteration vs the whole training set (== batch)

Mini-batch gradient descent

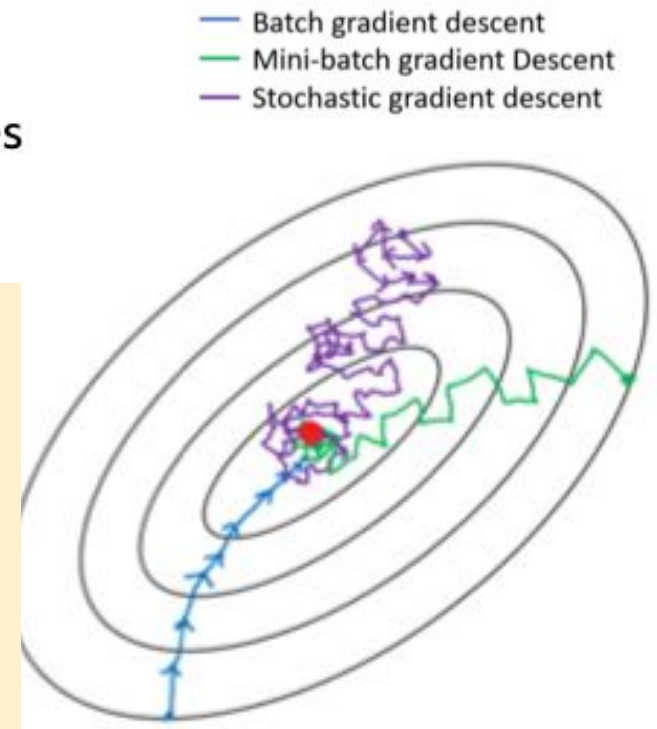
- Use mini-batch **sampled** in the dataset for gradient estimate.
- Sometimes helps to escape from local minima
- Noisy gradients act as regularization
- Variance of gradients increases when batch size decreases
- Not clear how many sample per batch

What happens in **one epoch** for SGD:

1. Take a random sample
2. Feed it to Neural Network
3. Calculate its gradient
4. Use the gradient we calculated in step 3 to update the weights
5. Repeat steps 1–4 for all the examples in training dataset

What happens in **one epoch** for Mini-BGD**:**

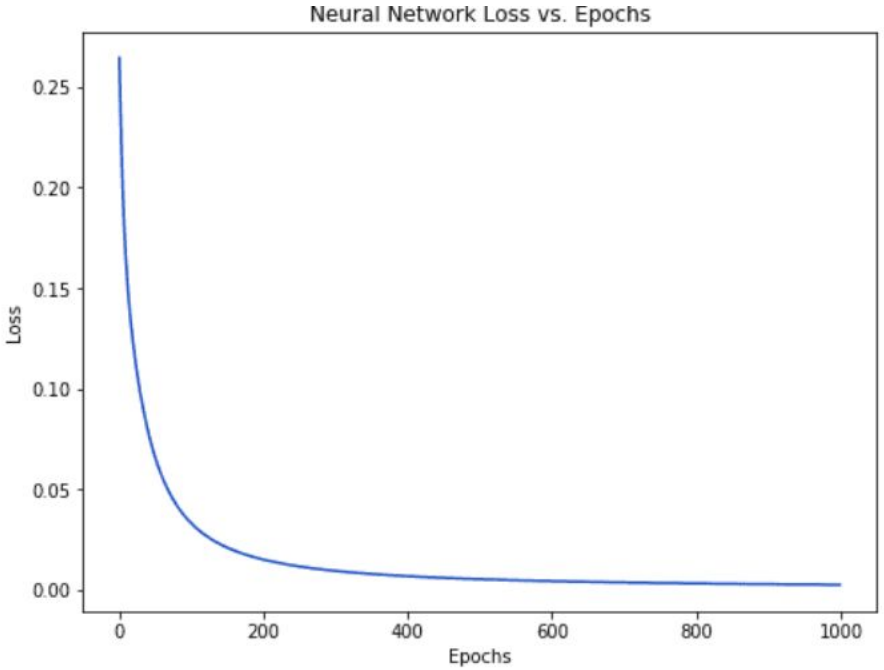
1. Pick a mini-batch
2. Feed it to Neural Network
3. Calculate the mean gradient of the mini-batch
4. Use the mean gradient we calculated in step 3 to update the weights
5. Repeat steps 1–4 for the mini-batches we created



Take home messages

- **Gradient descent** is an iterative learning algorithm that uses a training dataset to update a model.
  - **BGD**: Batch Size = Size of Training Set
  - **SGD**: Batch Size = 1
  - **Mini-BGD**. 1 < Batch Size < Size of Training Set

- The **batch size** is a hyperparameter of gradient descent that controls the number of training samples to work through before the model's internal parameters are updated.

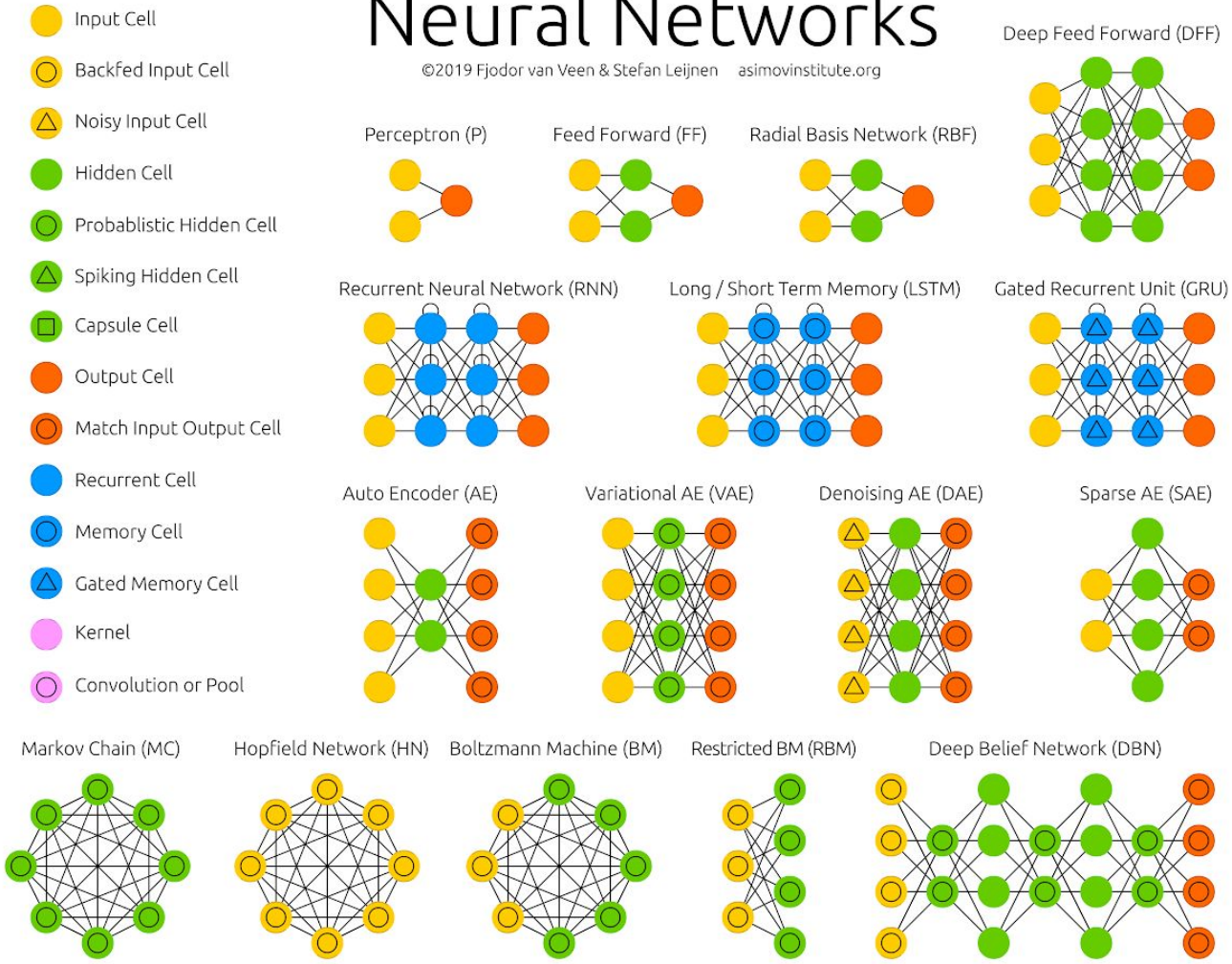


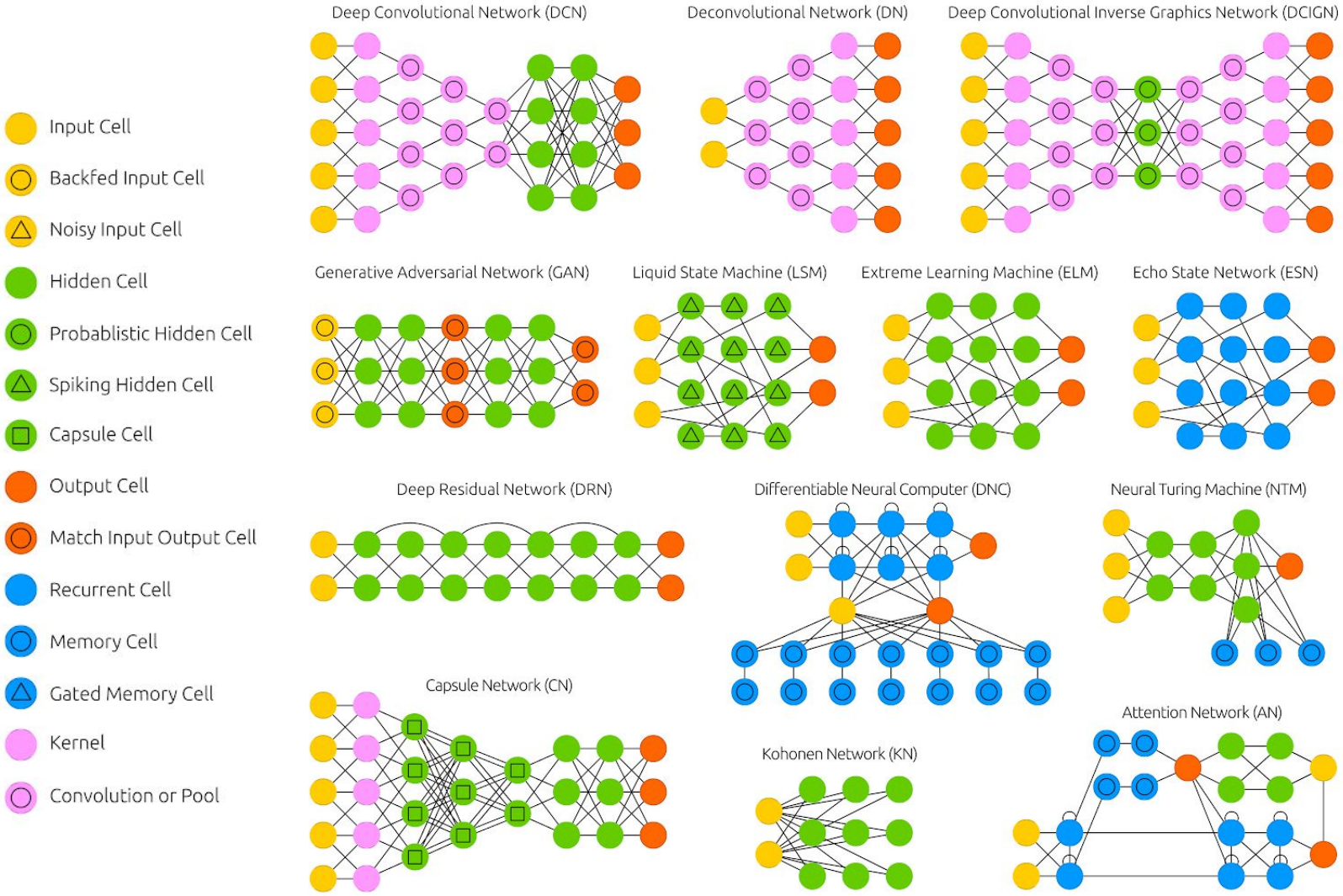
- The number of **epochs** is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset

A mostly complete chart of

Neural Networks

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

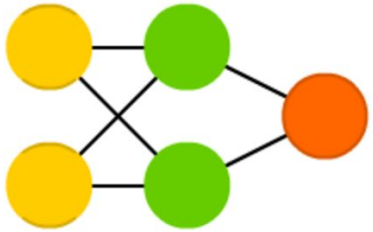



Input Cell
Backfed Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Capsule Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Gated Memory Cell
Kernel
Convolution or Pool
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional 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)

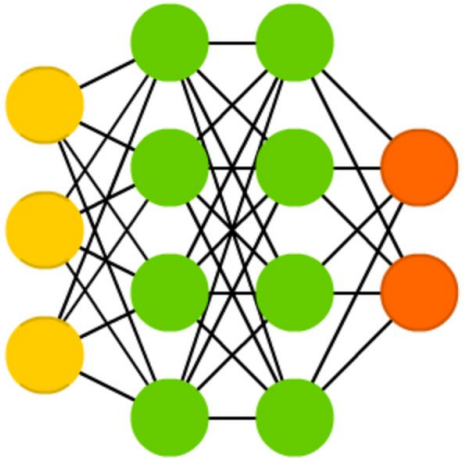
Feed Forward

- 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 backpropagation
- **DFF** is a FF NN with more than one hidden layer

Feed Forward (FF)

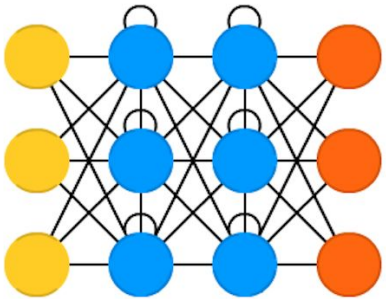


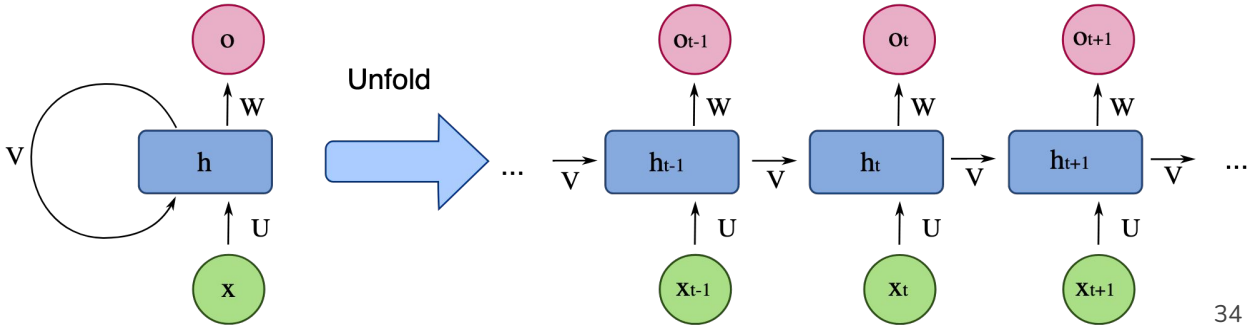
Deep Feed Forward (DFF)



Recurrent Neural Network (RNN)

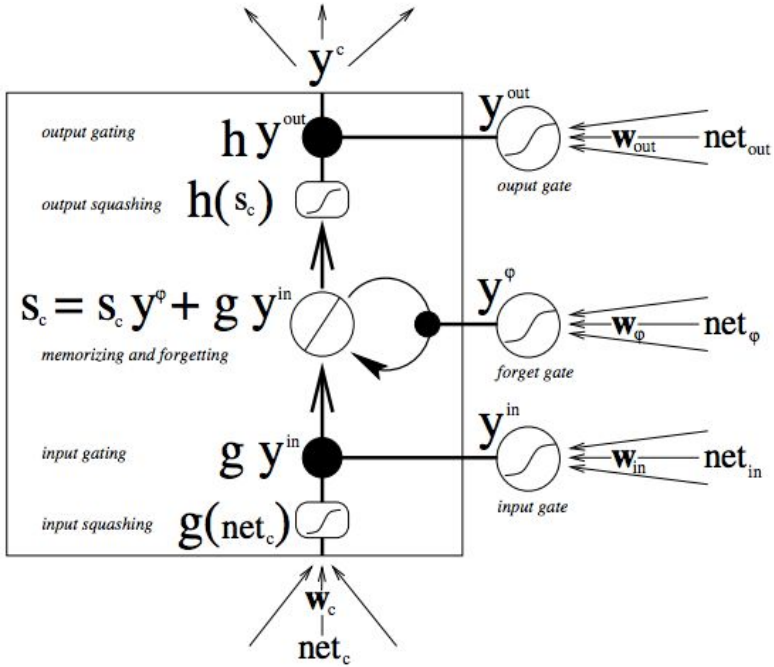
- FFNN with **Recurrent Cells**: cell that receives its own output with fixed delay
- RNNs permit to operate on **sequences of vectors**
  - context is important, decision from past iterations can influence current state
  - a word can be analyzed only in context of previous words or sentences
- RNNs, 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

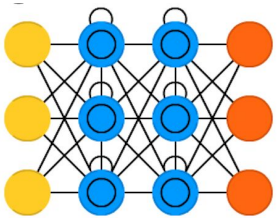




Long/Short Term Memory (LSTM)

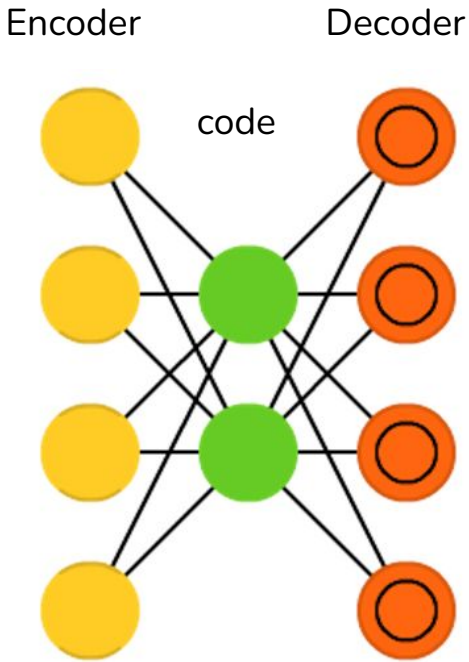
- RNNs not really capable of learning long term dependencies
  - Due to vanishing gradient with increasing time steps
- A common LSTM unit is made of a **cell**, an **input**, an **output** and a **forget** gate
  - The cell remembers values over arbitrary time intervals and the three gates regulate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

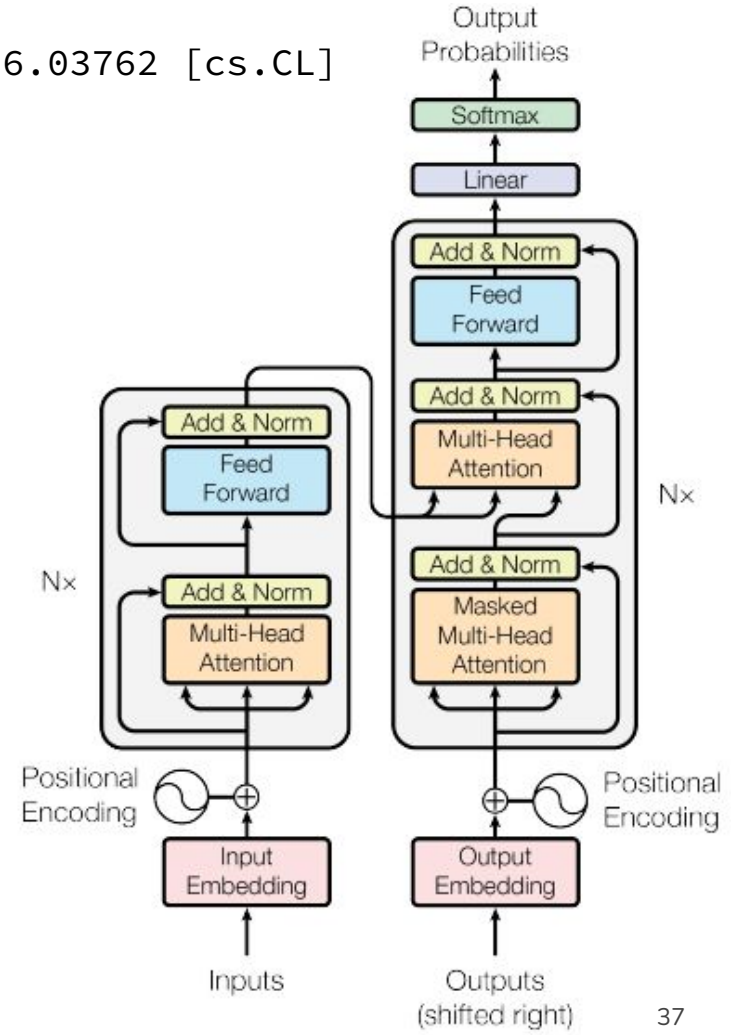


Transformers

arXiv:1706.03762 [cs.CL]

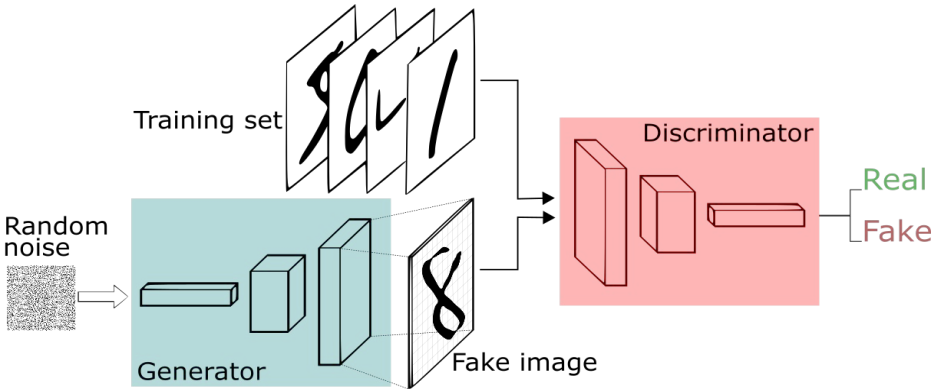
- All you need is **attention**
  - transformers process the entire input all at once
  - the attention mechanism provides context for any position in the input sequence
- Self-attention: **query, key, value**:
  - The significance of each part of the input data is differentially weighted

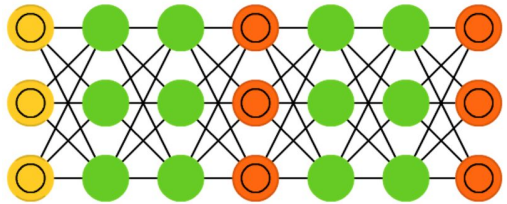
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



Generative Adversarial Networks (GAN)

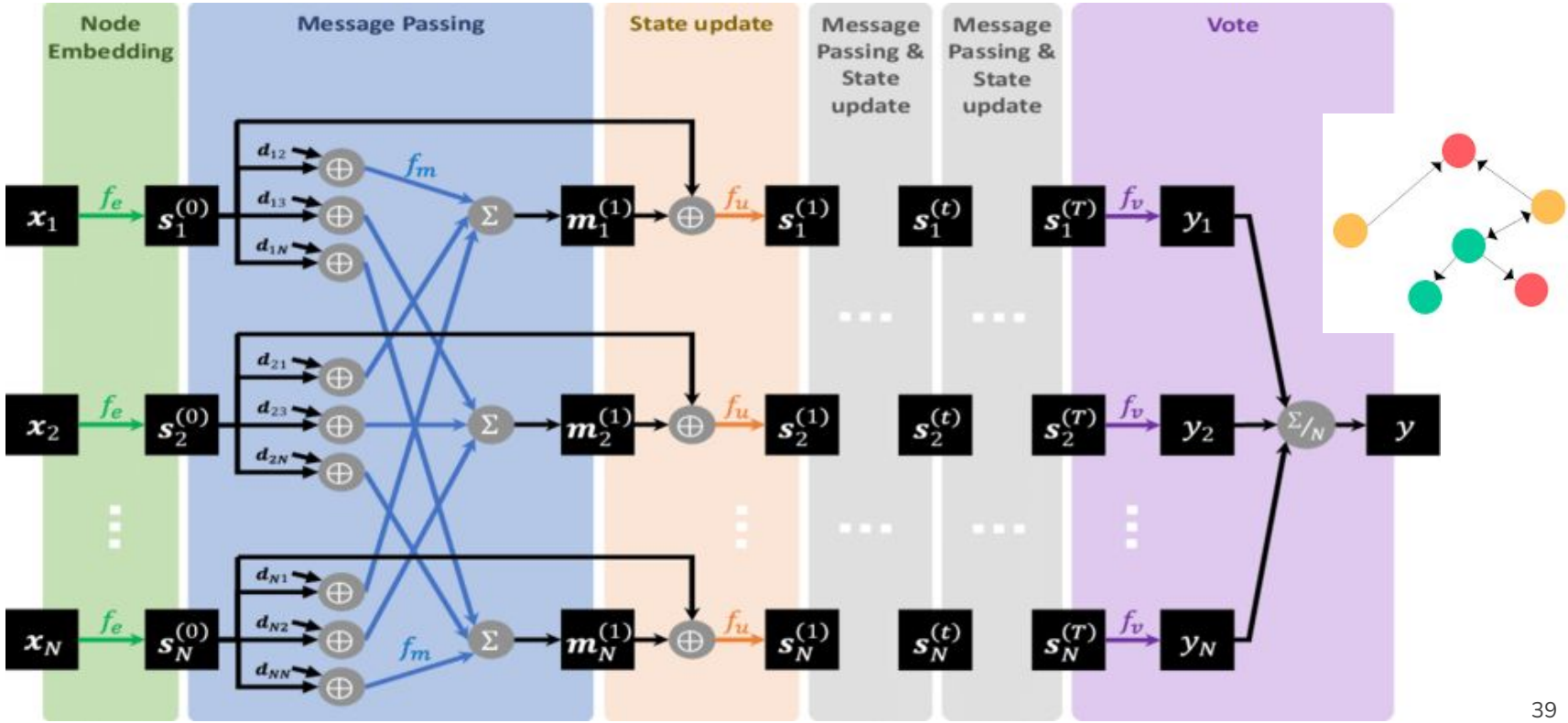
- GANs represent a huge family of double networks that are composed from a **generator** net and a **discriminator** net
  - The generator produces samples close to training samples
  - Discriminator net (adversary) differentiates between samples from the generative net and the training set
  - Use error 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





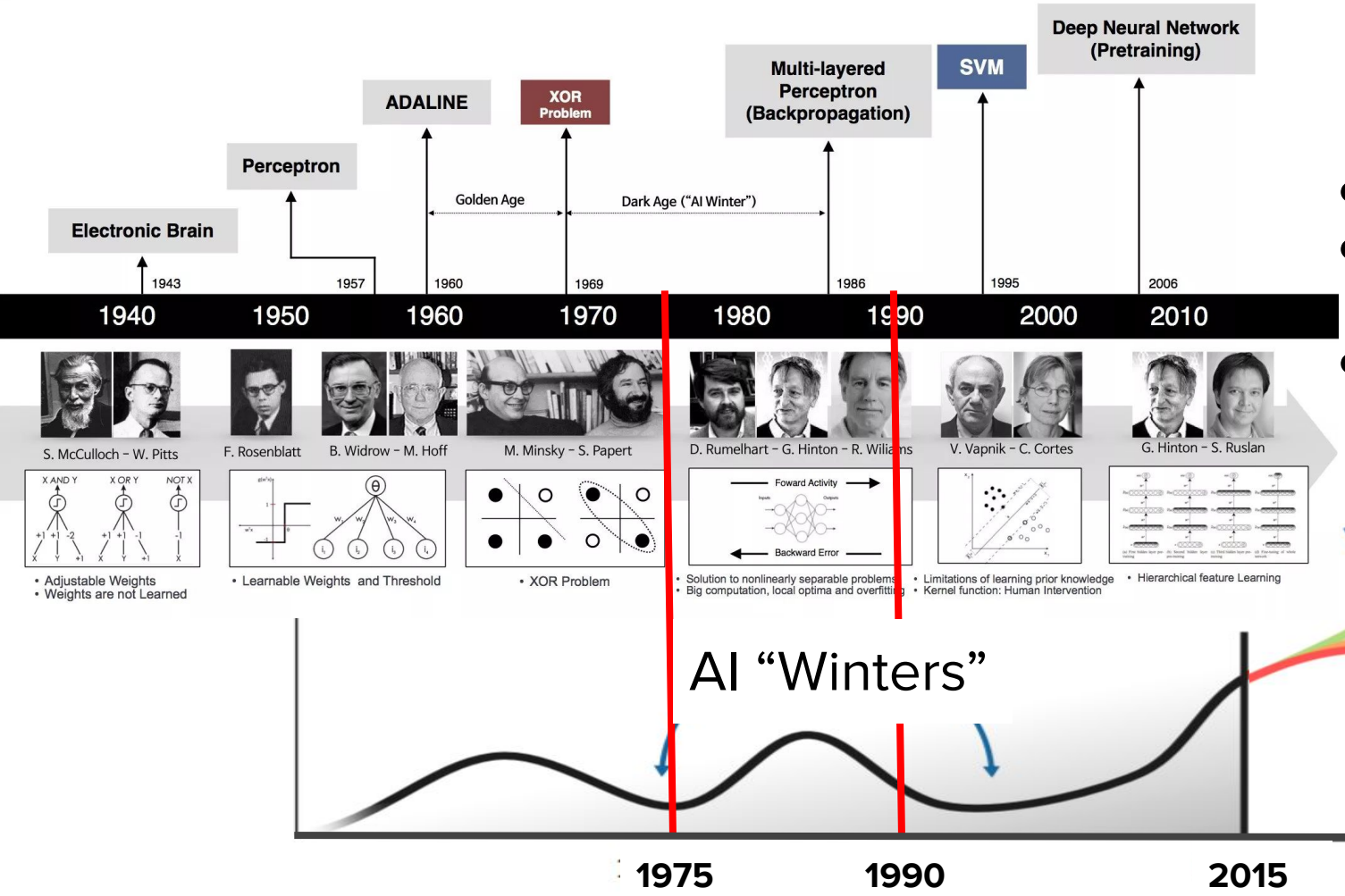
Graph Neural Network (GNN)

- 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: Artificial intelligence Colorectal Cancer Detection Technique Better than Pathologist



- explainable AI
- information bottleneck
- ethics

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]

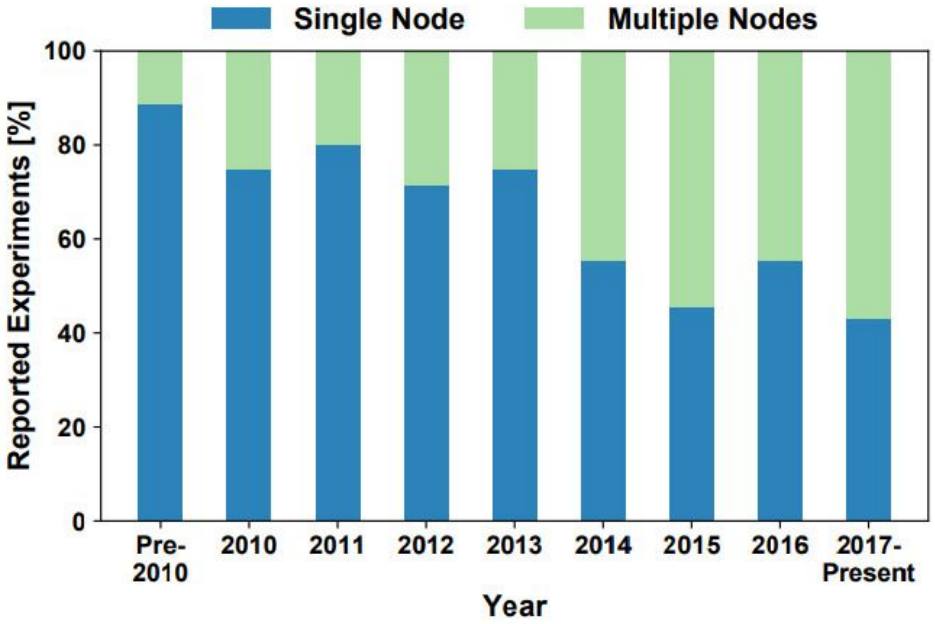
Consciousness Prior

- 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

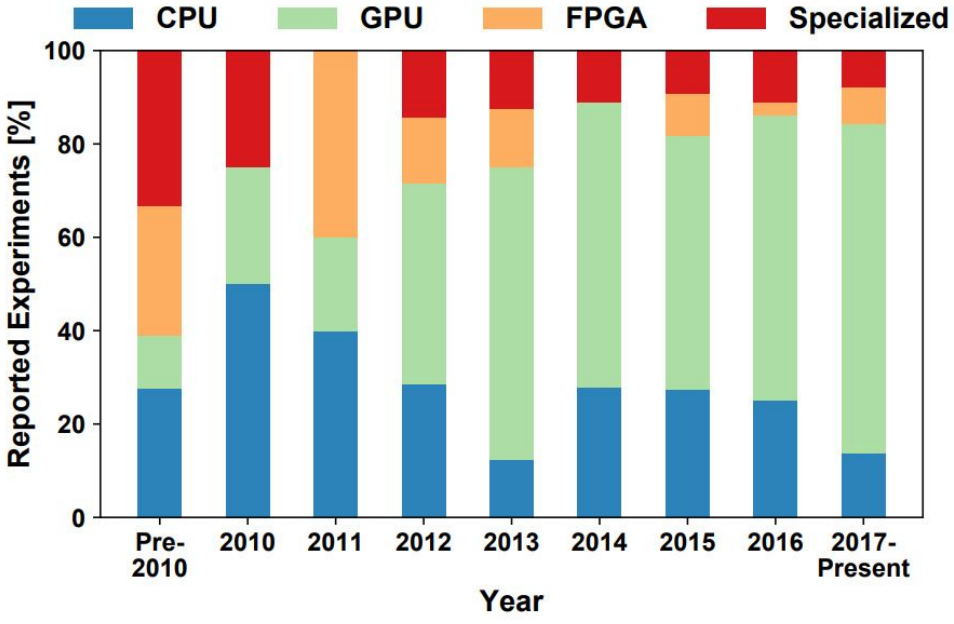
$$P(S) = \frac{\prod_j f_j(S_j)}{Z}$$

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

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

Heterogeneous programming: **Kokkos, Alpaka, SyCL**

Machine Learning on GPUs

- CNN need a lot of computing power (need high number of cores, FLOPS)
- LSTM and RNN are made of small matrix multiplication: memory bandwidth matters
- 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

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.

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 worker or to achieve faster training
  - the *forward* pass involves no communication among workers (all workers hold the full model)
  - the *backward* pass involves aggregating the gradients computed by each individual worker with respect to its separate part of the “global batch”
- **Synchronous update**:
  - all 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

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*.

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

Challenges with distributed systems

- Great computing power comes at a cost
  - **I/O contention** with large data sets, or data sets with many small examples, and the data has to be physically stored on shared storage facilities (for example in HPC systems)
    - Might be faster to copy data locally to worker node
  - **Communication** bottlenecks during aggregation of the results - one machine may be late, all others waiting