# ML em HEP II

Escola INCT 2024

A. Sznajder

UERJ
Instituto de Fisica

Novembro - 2024

# Outline

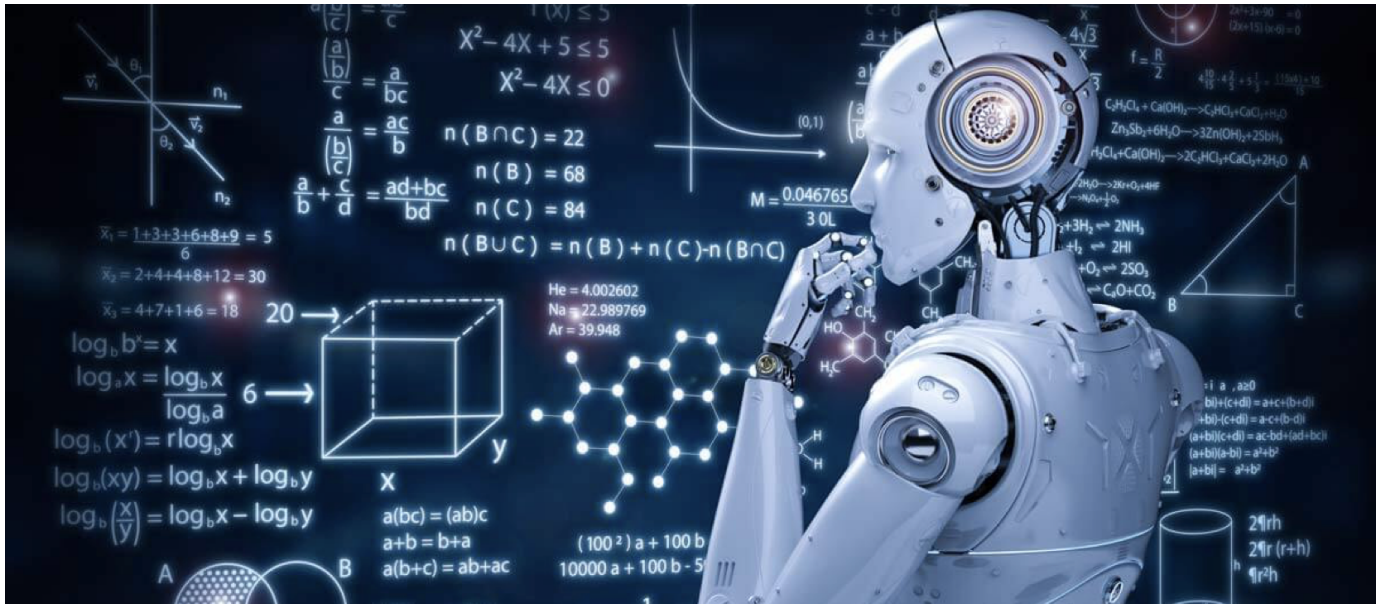
# Support Material

- **Textbook:**
  Aurélien Géron - Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow
- **Jupyter Notebooks :**
  `https://github.com/INCT-CERN-Brasil/Escola2024`
- **Google Colab :**
  `https://colab.google`

# What is Machine Learning ?

**Machine Learning (ML)**

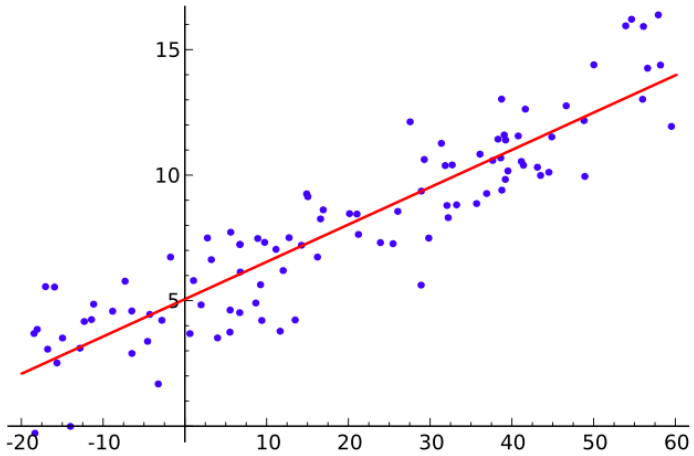
Machine learning (ML) is the study of computer algorithms capable of building a predictive model out of data samples ( learn from examples ), without being explicitly programmed ( sequence of instructions ).



# Centuries Old Machine Learning [1]

Take some points on a 2D graph, and fit a function to them. What you have just done is learn from a few $(x, y)$ pairs (examples), a general function that can map any input $x$ to an output $y$

**The Centuries Old Machine Learning Algorithm**



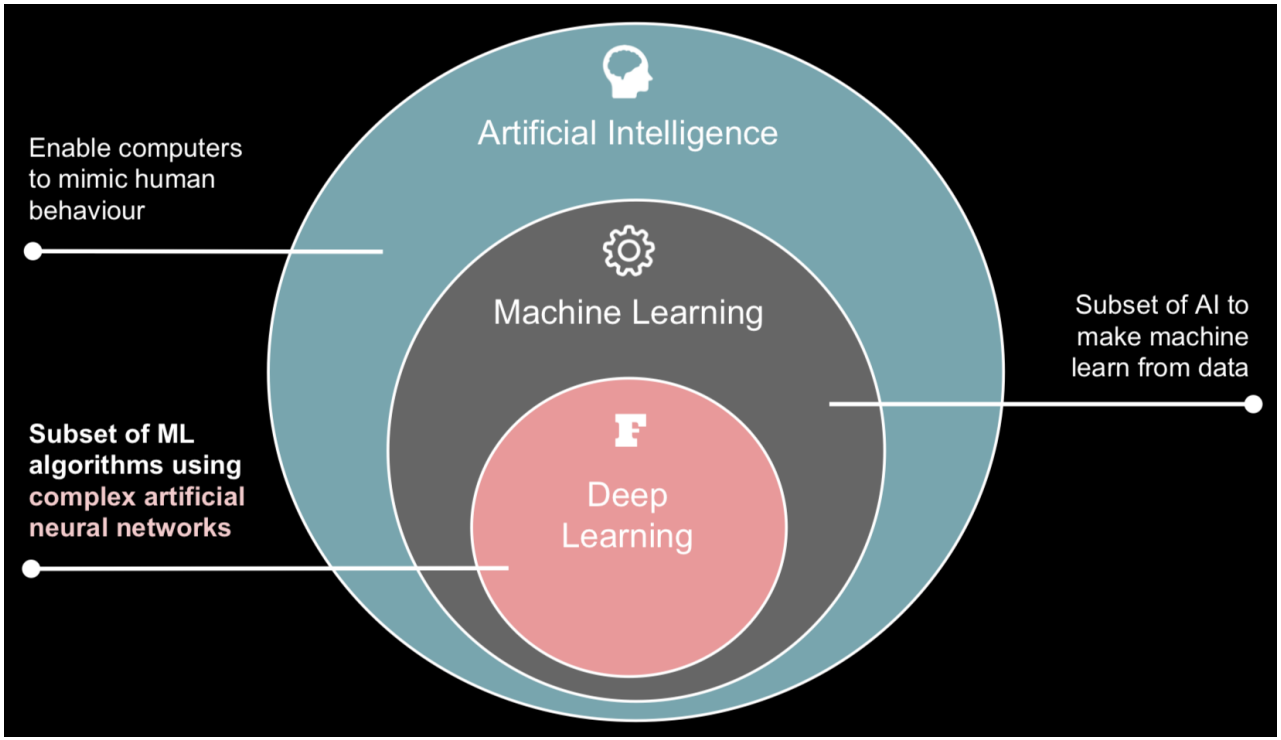

Linear regression (Source)

Linear regression is too wimpy to solve the complex problems (ex: image, speech, text, ...) , but it's essentially supervised ML ( glorified regression )

[1] http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learnin

# Artificial Inteligence x Machine Learning

”Inteligence can be understood as the ability to process current information to make informed future decisions”




None of current ML systems we have are real AI and the brain learns so efficiently that no ML method can match it [2]

[2] Yann LeCun , Epistemology of Deep Learning : https://www.youtube.com/watch?v=gG5NCkMerHU&t=944s , https://www.youtube.com/watch?v=cWzi38-vDbE&t=768s

# Introduction to Machine Learning

Machine Learning(ML) can be approached from many different angles:

**1) Tasks:**

- Classification
- Regression
- Data Generation or Simulation

**2) Data Structure**

- Tabular
- Image
- Sequence
- Graph
- Sets

**3) Learning Paradigms**

- Supervised Learning (labeled)
- Un(Self)supervised Learning (unlabeled)
- Reinforcement Learning (reward)

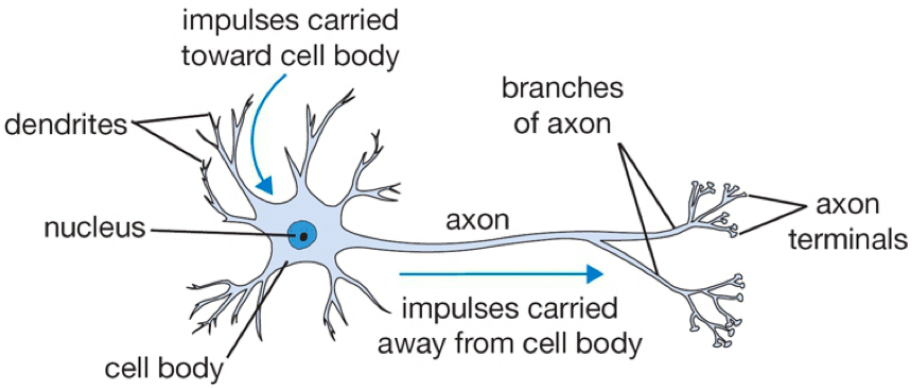
**3) Neural Networks Architectures**

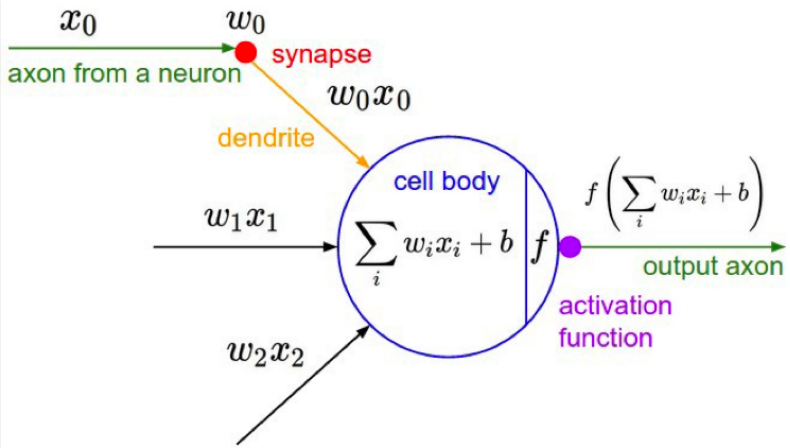
- Multilayer Perceptron (MLP)
- Recurrent Networks (RNN,LSTM,GRU)
- Transformer Networks (TN)
- Convolutional Networks (CNN)
- Graph Networks (GCN,GAT,DGCN,IN)
- Point Nets and Deep Sets (PN,DS)
- Autoencoders (AE,DAE,VAE)
- Generative Adversarial Network (GAN)
- Normalizing Flows (NF)
- Diffusion Models (DM,SDM)

Obs: ML can be implemented by different algorithms (ex: SVM, BDT, PCA ...) but we will discuss only Neural Networks

# Neural Networks

Artificial Neural Networks (NN) are computational models vaguely inspired [3] by biological neural networks. A NN is formed by a network of basic elements called neurons, which receive an input, change their state according to the input and produce an output




Original goal of NN approach was to solve problems like a human brain. However, focus moved to performing specific tasks, deviating from biology. Nowadays NN are used on a variety of tasks: image and speech recognition, translation, filtering, playing games, medical diagnosis, autonomous vehicles, ...

[3] Design of airplanes was inspired by birds, but airplanes don't flap wings to fly !

# Artificial Neuron

## Artificial Neuron Model

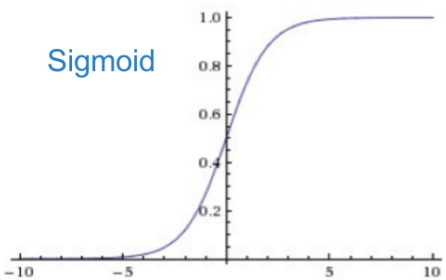
Each node of a NN receives inputs $\vec{x} = \{x_1, \ldots, x_n\}$ from other nodes or an external source and computes an output $y$ according to the expression

$$y = F\left(\sum_{i=1}^{n} W_i x_i + B\right) = F\left(\vec{W}.\vec{x} + B\right) \quad (1)$$

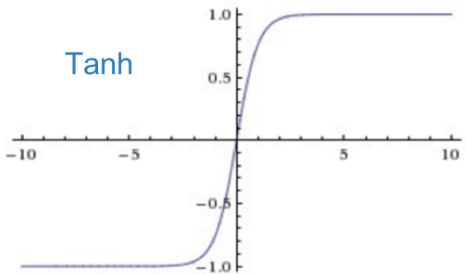
, where $W_i$ are connection weights, $B$ is the treshold and $F$ the activation function [4]

There are a variety of possible activation function and the most common ones are

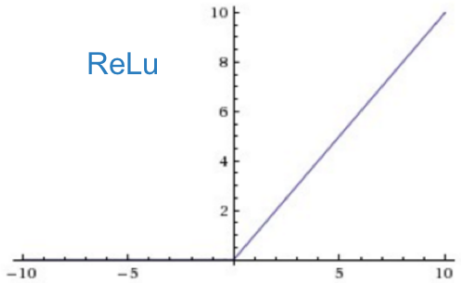
Sigmoid




Tanh




ReLu



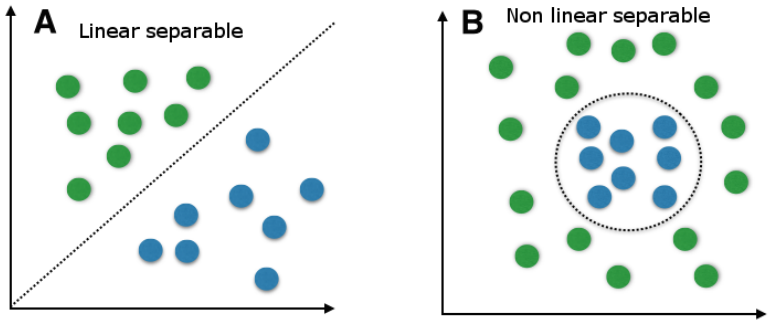

[4] Nonlinear activation is fundamental for nonlinear decision boundaries

# The Perceptron

The perceptron [5] algorithm is a binary linear classifier invented in 1957 by F.Rosenblatt. It's formed by a single neuron that takes input $\vec{x} = (x_1, \ldots, x_n)$ and outputs $y = 0, 1$ according to

**Perceptron Model** [6]

$$y = \begin{cases} 1, \textit{ if } (\vec{W}.\vec{x} + B) > 0 \\ 0, \textit{ otherwise} \end{cases}$$




To simplify notation define $W_0 = B$ , $\vec{x} = (1, x_1, \ldots, x_n)$ and call $\theta$ the Heaviside step function

**Perceptron Learning Algorithm**

1. Calculate the ouput error: $y_j = \theta(\vec{W}.\vec{x}_j)$ and $Error = 1/m \sum_{j=1}^{m} |y_j - t_j|$
2. Modify(update) the weights to minimize the error: $\delta W_i = r.(y_j - t_j).X_i$ , where $r$ is the learning rate
3. Return to step 1 until output error is acceptable

[5] https://medium.com/towards-data-science/perceptrons-the-first-neural-network-model-8b3ee

[6] Equation of a plane in $\mathbb{R}^n$ is $\vec{W}.\vec{x} + B = 0$

# The Perceptron as a Universal Function Approximator

## Universal Approximation Theorem

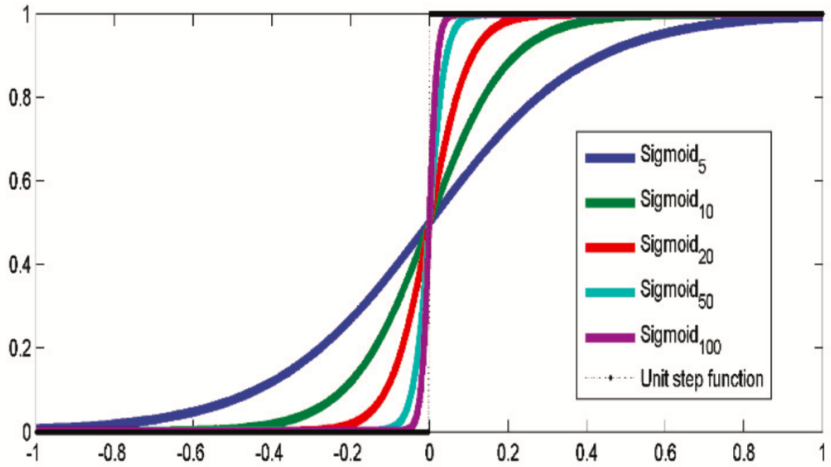
A single hidden layer feed forward neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden neurons [7]

The theorem doesn't tell us how many neurons or how much data is needed !

## Sigmoid $\rightarrow$ Step Function

For large weight $W$ the sigmoid turns into a step function, while $B$ gives its offset

$$y = \frac{1}{1 + e^{-(w.x+b)}} \tag{2}$$



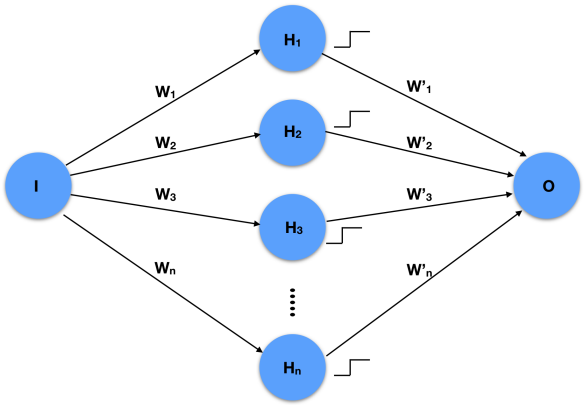

[7] Cybenko,G.(1989) Approximations by superpositions of sigmoidal functions, Math.ofCtrl.,Sig.,andSyst.,2(4),303
Hornik,K.(1991) Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks,4(2), 251

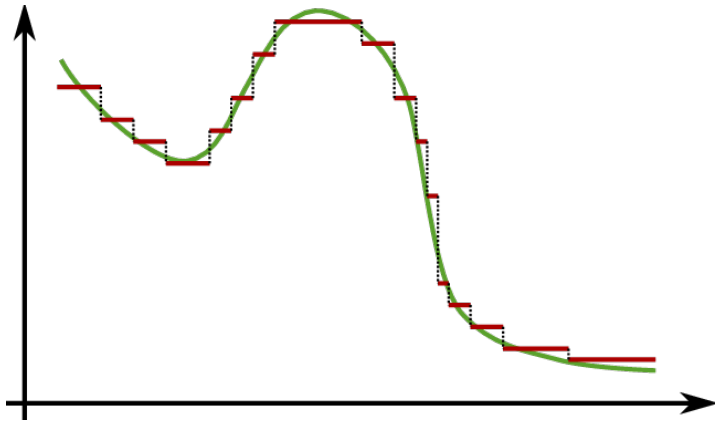
# The Perceptron as a Universal Function Approximator

**Approximating $F(x)$ by Sum of Steps**

A continuous function can be aproximated by a finite sum of step functions. The larger the number of steps(nodes), the better the approximation. [8]

Consider a NN composed of a single input , *n* hidden nodes and a single output. Tune the weights such that the activations approximate steps functions with appropriate treshold and add them together !



Obs: for other activations like RELU a similar argument applies in terms of approximation by linear segments !

[8] M.Nielsen , http://neuralnetworksanddeeplearning.com/chap4.html
S.Prince, Understanding Deep Learning 2023

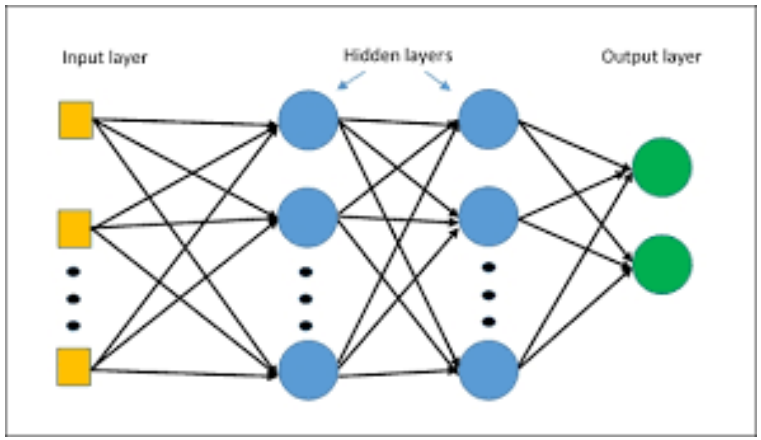
# Multilayer Perceptron (MLP)

**Multilayer Perceptron Model (MLP)**

The Multilayer Perceptron(MLP) is a fully connected NN with more than one hidden layer and nonlinear activation function $F$ [9] . A MLP can be seen as a parametrized composite mapping $F_{w,b} : \mathbb{R}^n \rightarrow \mathbb{R}^m$

For a MLP [10] with inputs nodes $\vec{x}^{(0)}$, one hidden layer of nodes $\vec{x}^{(1)}$ and output layer of nodes $\vec{x}^{(2)}$ , we have

$$\begin{cases} \vec{x}^{(1)} = \vec{F}^{(1)} \left( \vec{W}^{(1)} . \vec{x}^{(0)} \right) \\ \vec{x}^{(2)} = \vec{F}^{(2)} \left( \vec{W}^{(2)} . \vec{x}^{(1)} \right) \end{cases}$$




Eliminating the hidden layer variables $\vec{x^{(1)}}$ we get

$$\vec{x}^{(2)} = \vec{F}^{(2)} \left( \vec{W}^{(2)} . \vec{F}^{(1)} \left( \vec{W}^{(1)} . \vec{x}^{(0)} \right) \right) \quad (3)$$

Obs: MLP is the simplest deep NN with a compositional ”inductive bias” !

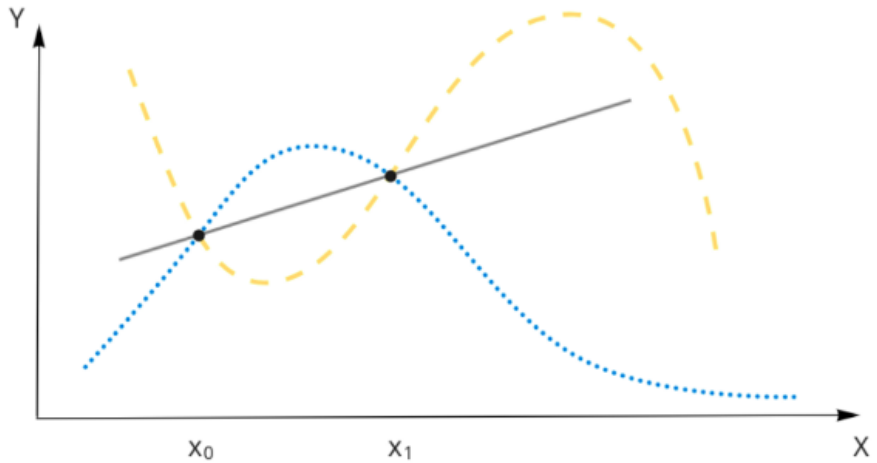
[9] A MLP with $m$ layers and linear activations can always be reduced to a single layer

[10] Biases can be represented as weights by redefining $\vec{x} = (1, x_1, \ldots, x_n)$ and $\vec{W} = (B, W_1, \ldots, W_n)$

# Inductive Bias

## Inductive Bias

Learning involves searching the space of solutions for one that provides the best data explanation. In case of multiple appropriate solutions an inductive bias allows a learning algorithm to prioritize one solution over another, independently of the data



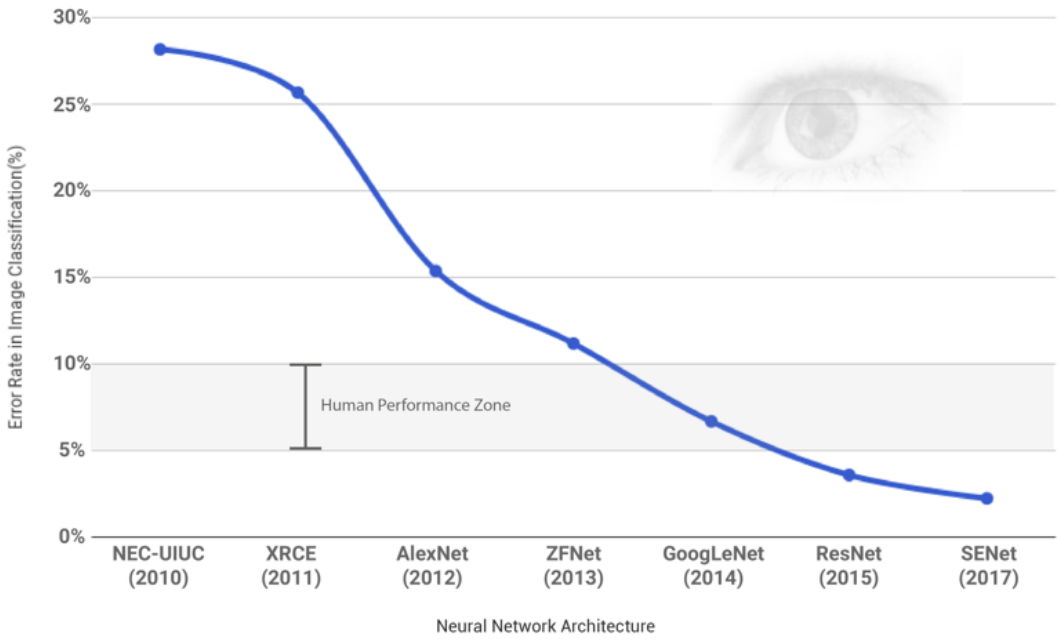

In the above fit there's an infinite number of functions that could pass through the data points [11]

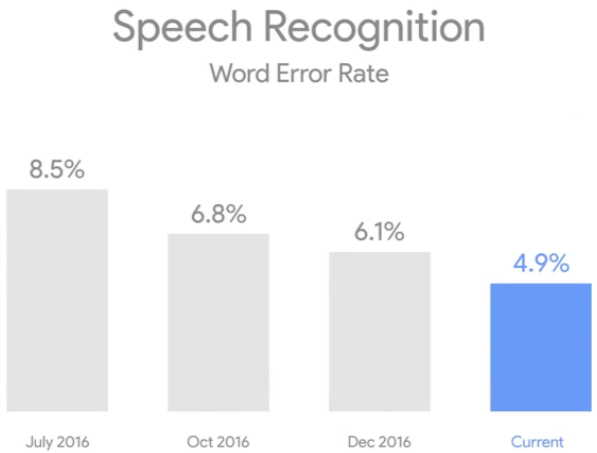
[11] https://towardsdatascience.com/the-inductive-bias-of-ml-models-and-why-you-should-care-about-it-979fe02a1

# Why Deep Learning ?
# and
# Why Now ?

# Why Deep Learning ?

## Image and Speech Recognition perfomance ( DNN versus Humans )

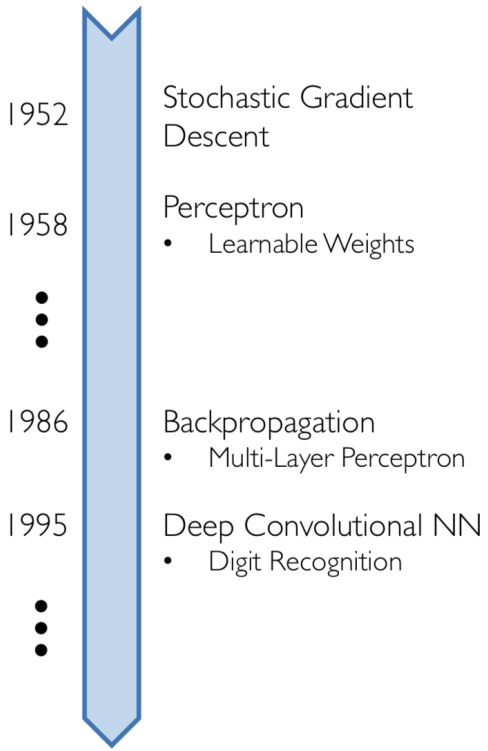



https://arxiv.org/pdf/1409.0575.pdf

# Why Now ?

Neural networks date back decades , so why the current resurgence ?

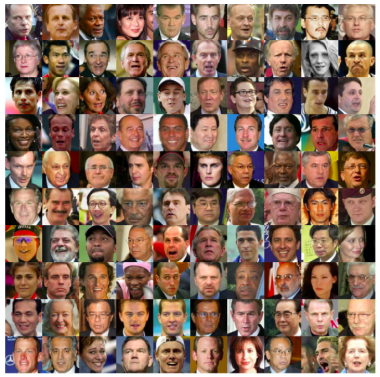


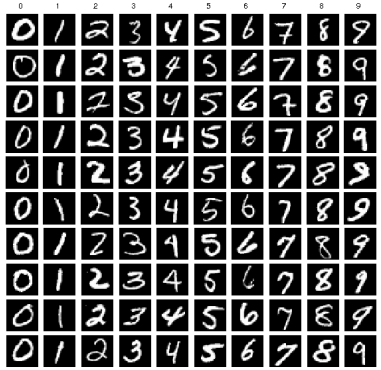

The main catalysts for the current Deep Learning revolution have been:

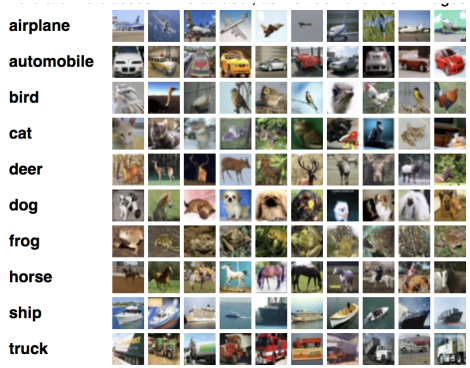
- **Software:**
  TensorFlow, PyTorch, Keras and Scikit-Learn
- **Hardware:**
  GPU, TPU and FPGA
- **Large Datasets:**
  MNIST

# Training Datasets

Large and new open source datasets for machine learning research [12]





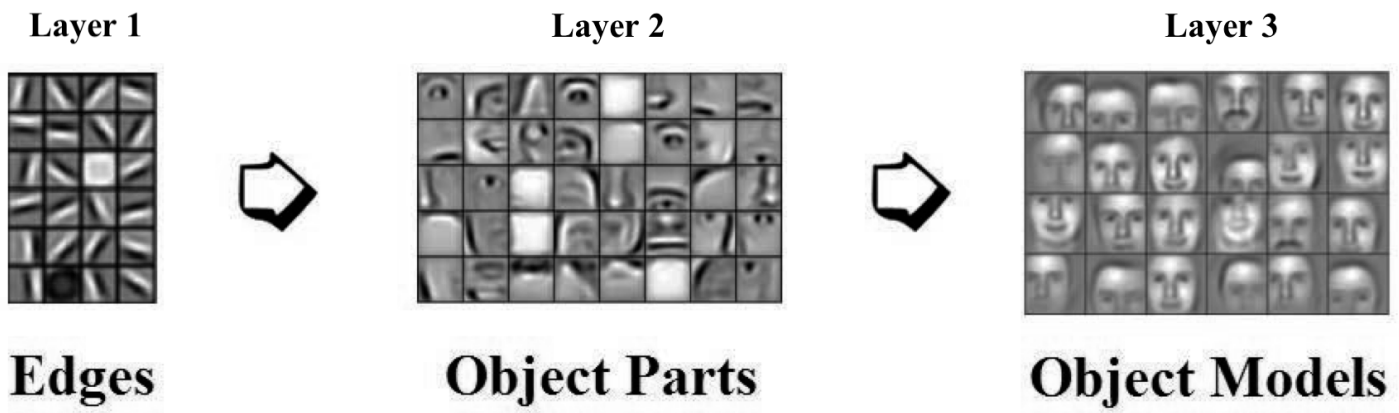



[12] https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research

# Deep Learning - Need for Depth

**Deep Neural Networks (DNN)**

Depth allows DNN to factorize data features, distributing its representation hierarchically across the layers. Deep architectures can learn relevant data features from raw input data without need for engineered input features




Obs: Depth explores the compositional character of nature as an inductive bias ! [13]

[13] Deep Learning , Y.LeCunn, J.Bengio, G.Hinton , Nature , vol. 521, pg. 436 , May 2015

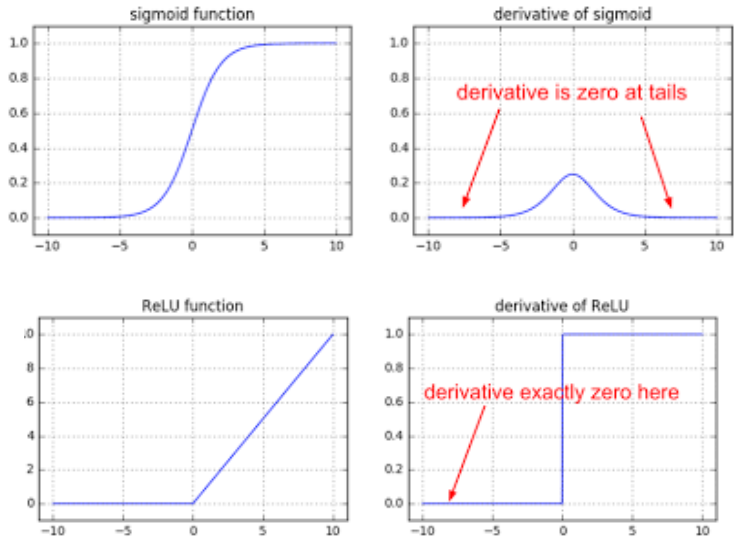
# Deep Learning - Vanishing Gradient Problem

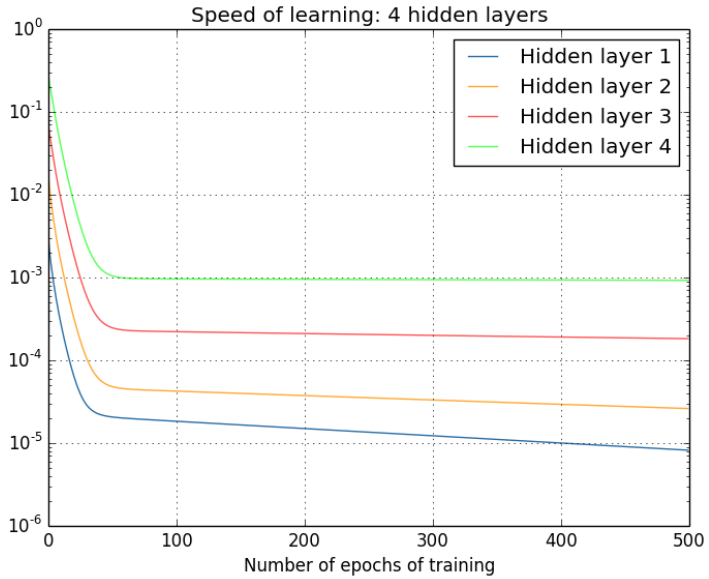
## Vanishing Gradient Problem [14].

Backpropagation computes gradients iteratively by multiplying the activation function derivate $F'$ through $n$ layers.

$$\delta_k^{(l)} = \left( \sum_m \delta_m^{(l+1)} W_{mk}^{(l+1)} \right) F'(z_k^{(l)})$$

For $\sigma(x)$ and $Tanh(x)$ the derivate $F'$ is asymptoticaly zero, so weights updates gets vanishing small when backpropagated $\Rightarrow$ Then, earlier layers learns much slower than later layers !!!



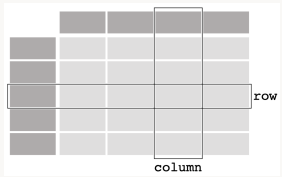

[14] http://neuralnetworksanddeeplearning.com/chap5.html

# Deep Architectures and Applications

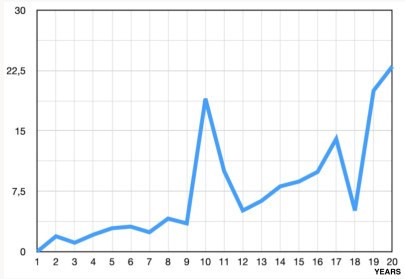
# Data Structures

Data comes structured in a variety of formats.

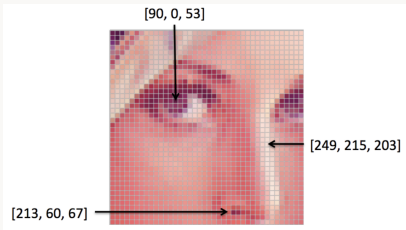
## Tabular Data



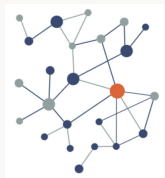

## Sequential Data




## Image Data



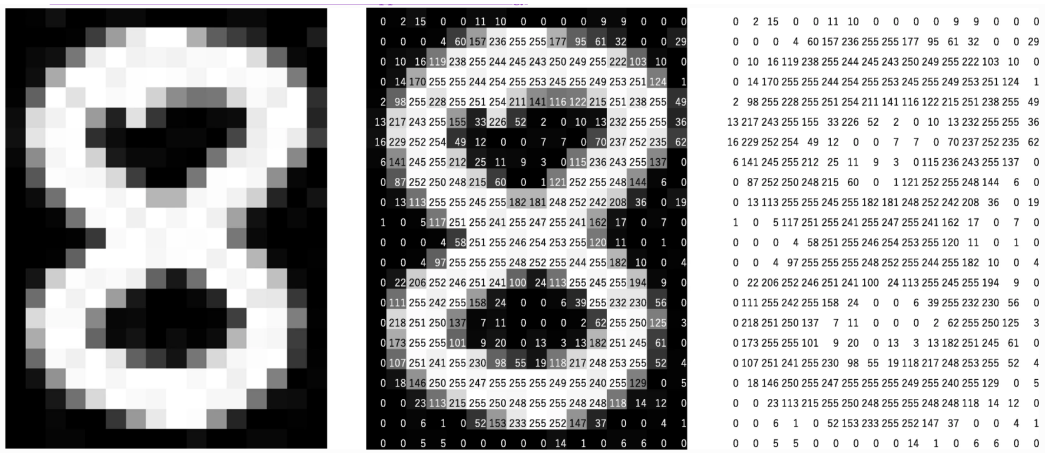

## Graph & Point Sets Data



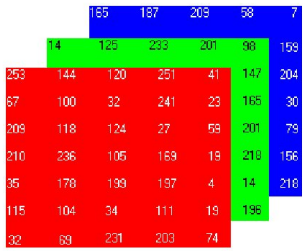
# Image Structured Data

## Image Representation

A computer sees an image as matrix, where each element is associated to a given pixel and its value represents the brightness.



The image color RGB can be represented by expanding the depth of the representation, where each matrix represent a fundamental color intensity

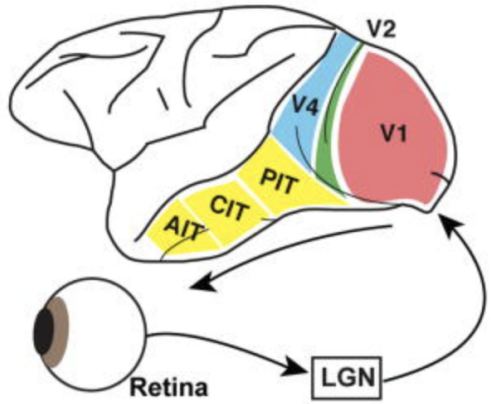


# Convolutional Neural Network

Full connectivity between neurons in a MLP makes it computationally too expensive to deal with high resolution images. A $1000x1000$ pixels image leads to $O(10^6)$ weights per neuron !

**Convolutional Neural Network (CNN)**

CNN is inspired by the visual cortex [15], where neurons responds to stimuli only in a restricted region of the visual field. This mitigates the challenges of high dimensional inputs by restricting the connections between the input and hidden neurons, connecting only small contiguous region and exploiting features.




CNN uses **local connectivity** as inductive bias built into the network layers, reducing the number of learnable parameters !

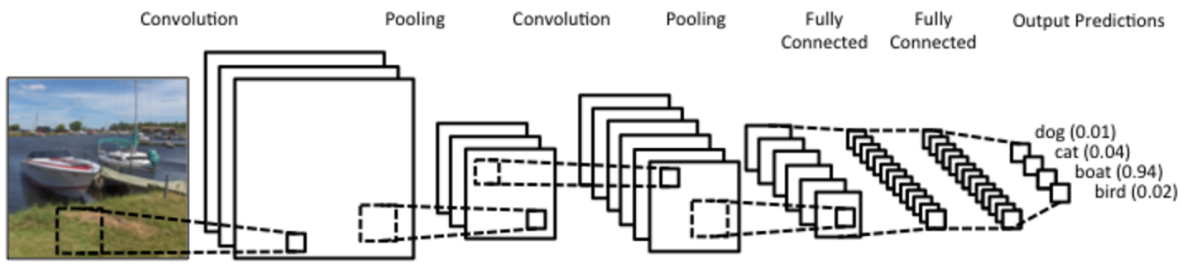
[15] How does the brain solve visual object recognition? , https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3306444

# Convolutional Neural Network

## Convolutional Neural Network (CNN)

A typical CNN architecture is composed by a stack of distinct and specialized layers:

1. Convolutional ( learnable feature maps ) [16]
2. Pooling (downsampling to reduce size)
3. Fully connected ( image classification )




[16] http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution

# CNN - Convolutional Layer

## Convolutional Layer

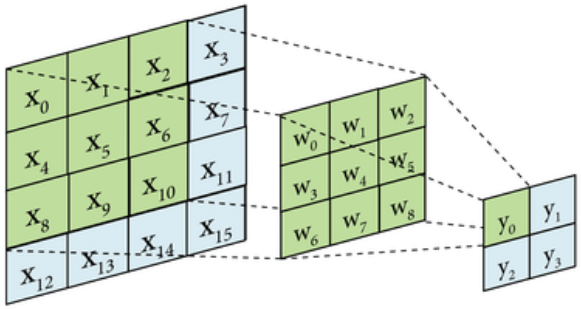
The convolutional layer [17] is the CNN core building block. A convolution can be seen as a sliding window transformation that applies a filter to extract local image features.

## Discrete Convolution

The convolution has a set of learnable filters weights that are shared across the image

$$y_{ij}^{(l+1)} = \sum_{a=0}^{n-1} \sum_{b=0}^{m-1} w_{ab}^{(l)} x_{(i+a)(j+b)}^{(l)}$$

(Click on the figure)

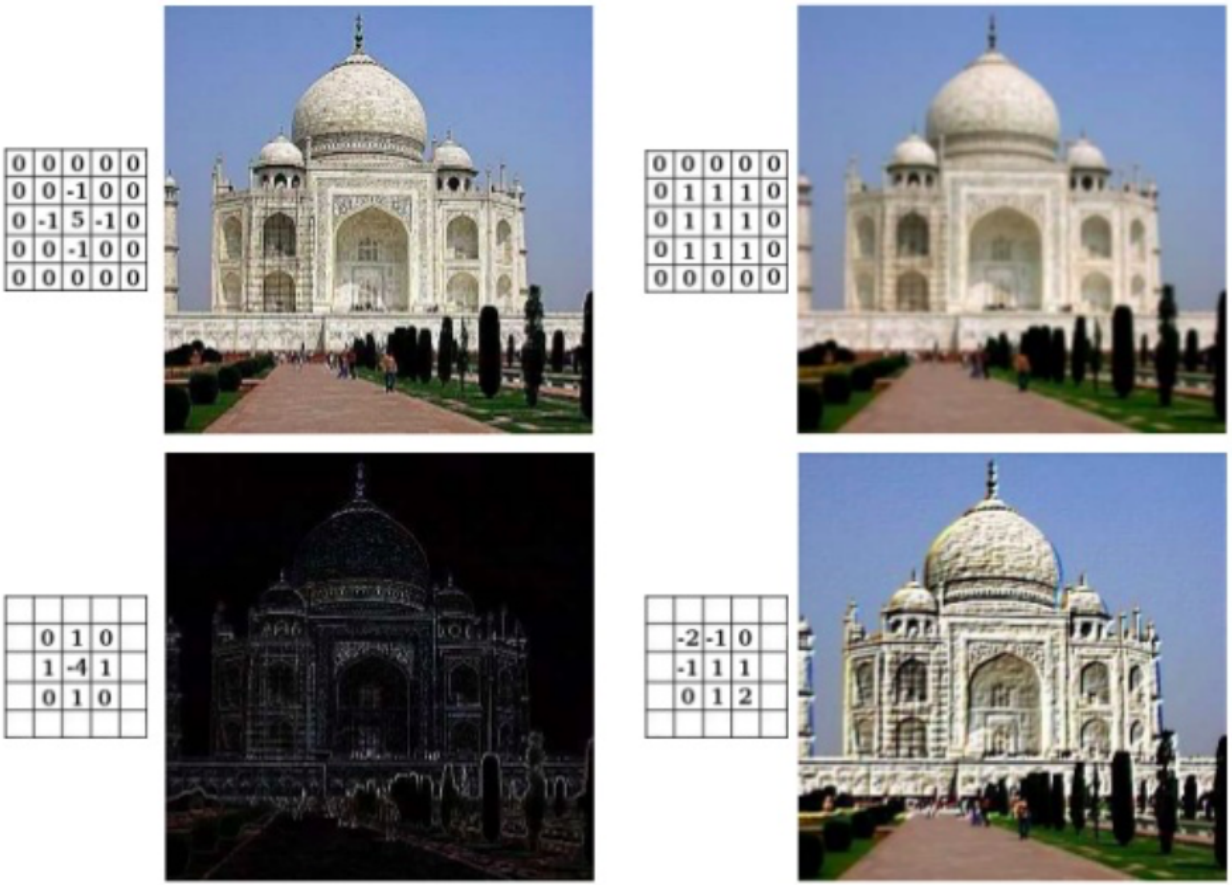



The same set of weights of a given filter are applied all across the image, reducing the number of learnable parameters ( shared weights )

[17] http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution
https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks

# Convolutional Filters

A convolution filter can apply an effect ( sharpen, blurr ), as well as extract features ( edges, texture ) [18]




[18] https://docs.gimp.org/2.6/en/plug-in-convmatrix.html

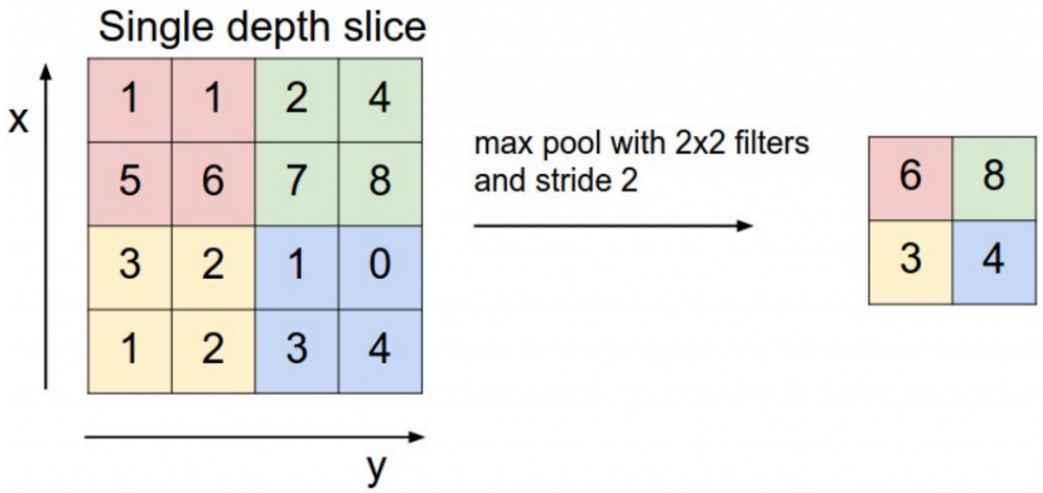
# CNN - Pooling Layer

## Pooling(Downsampling) Layer

The pooling (downsampling) layer [19] has no learning capabilities and serves a dual purpose:

- Decrease the representation size $\Rightarrow$ reduce computation
- Make the representation approximately invariant to small input translations and rotations

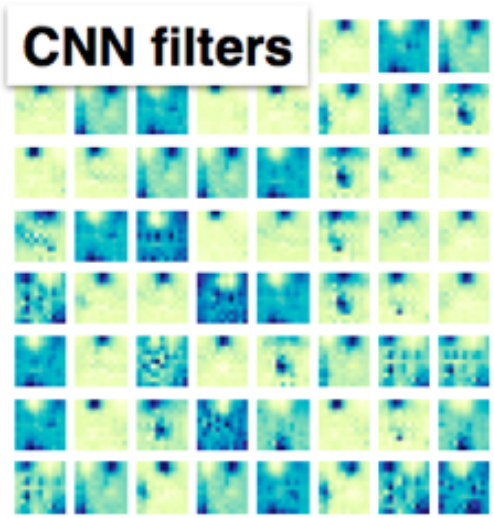
Pooling layer partitions the input in non-overlapping regions and, for each sub-region, it outputs a single value (ex: max pooling, mean pooling)

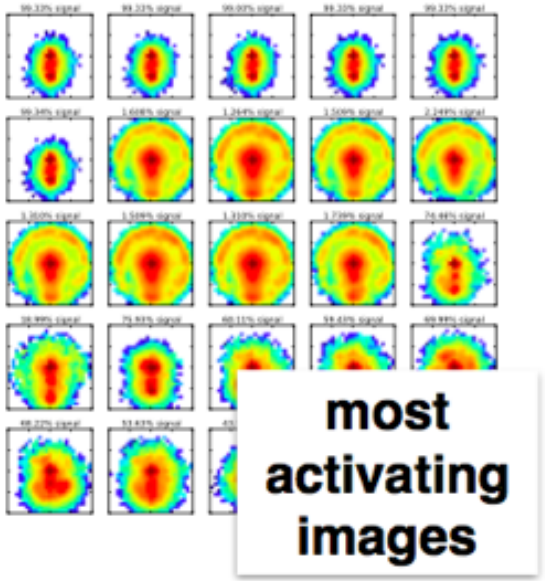


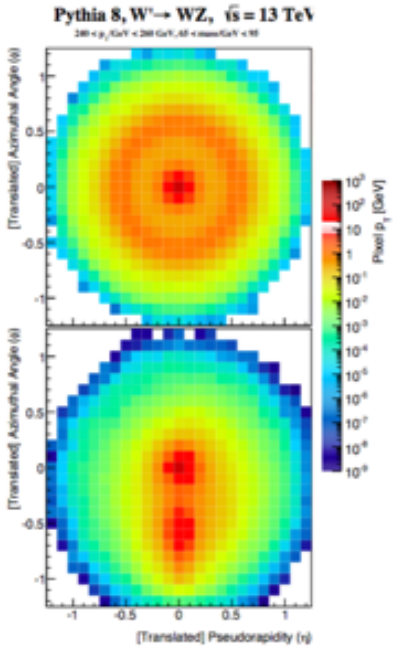

[19] http://ufldl.stanford.edu/tutorial/supervised/Pooling

# CNN Application in HEP: Jet ID

## Jet images with convolutional nets



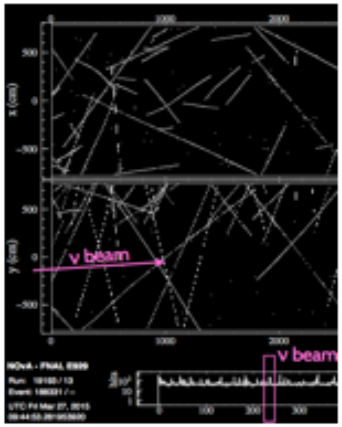



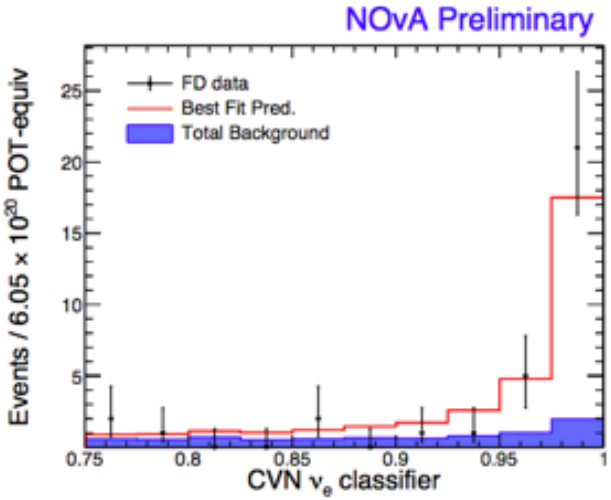




L. de Oliveira et al., 2015

# CNN Application HEP: Neutrino ID

## Neutrinos with convolutional nets

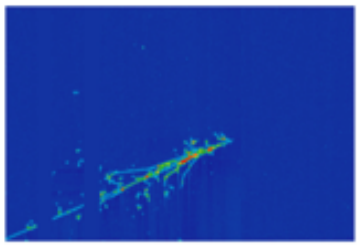


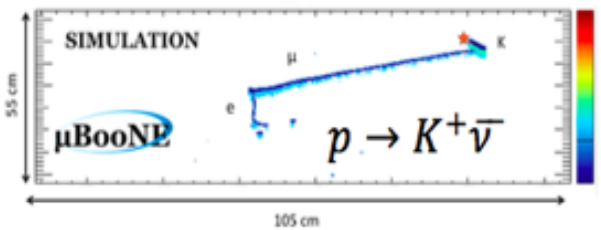



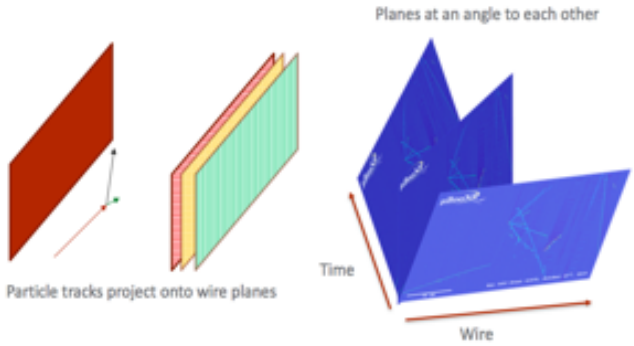

76% Purit
73% Effici

An equivalent increased exposure of 30%

Aurisiano et al. 2016






μBooNE

# Sequential Data

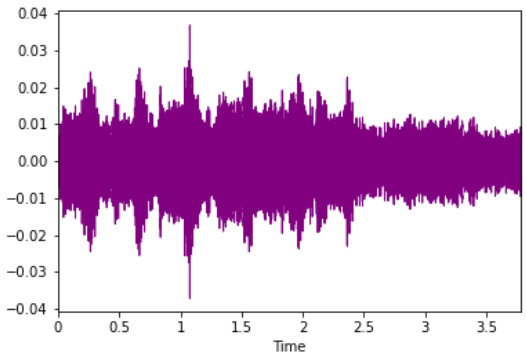
**Sequential Data**

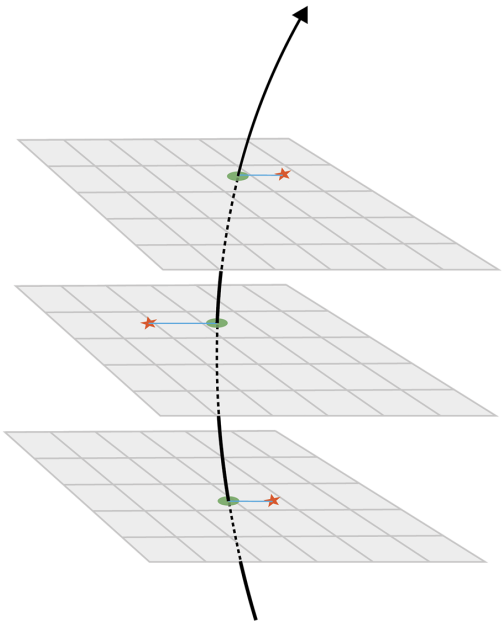
Sequential data is an interdependent data stream where data ordering contains relevant information

The food was good, not bad at all.

vs.

The food was bad, not good at all.



⇒ brain memorizes sequences for alphabet, words, phone numbers and not just symbols !

# Recurrent Neural Network(RNN)

Feed forward networks can't learn correlation between previous and current input !

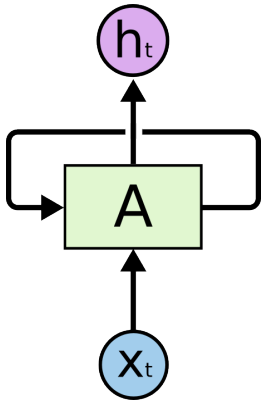
## Recurrent Neural Networks (RNN)

RNNs are networks that use feedback loops to process sequential data [20]. These feedbacks allows information to persist, which is often described as **memory**.

## RNN Cell ( Neuron [21] )

The hidden state depends not only on the current input, but also on the entire history of past inputs

1. **Hidden State:** $h^{[t]} = F(W_{xh}x^{[t]} + W_{hh}h^{[t-1]})$
2. **Output:** $y^{[t]} = W_{hy}h^{[t]}$



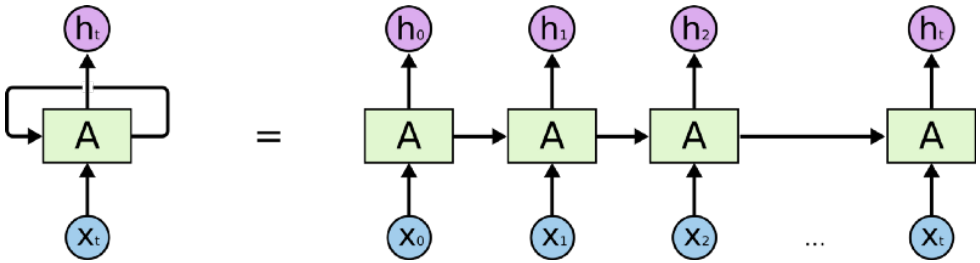

[20] https://eli.thegreenplace.net/2018/understanding-how-to-implement-a-character-based-rnn-
[21] https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks

# Recurrent Neural Network(RNN)

RNNs process each input sequence element at a time, maintaining in their hidden units a 'state vector' that contains information about all the past elements history. [22]

## RNN Unrolling

A RNN can be thought of as multiple copies of the same network, each passing a message to a successor. Unrolling is a visualization tool which views a RNN as a sequence of unit cells.




## Backpropagation Through Time (BPTT)

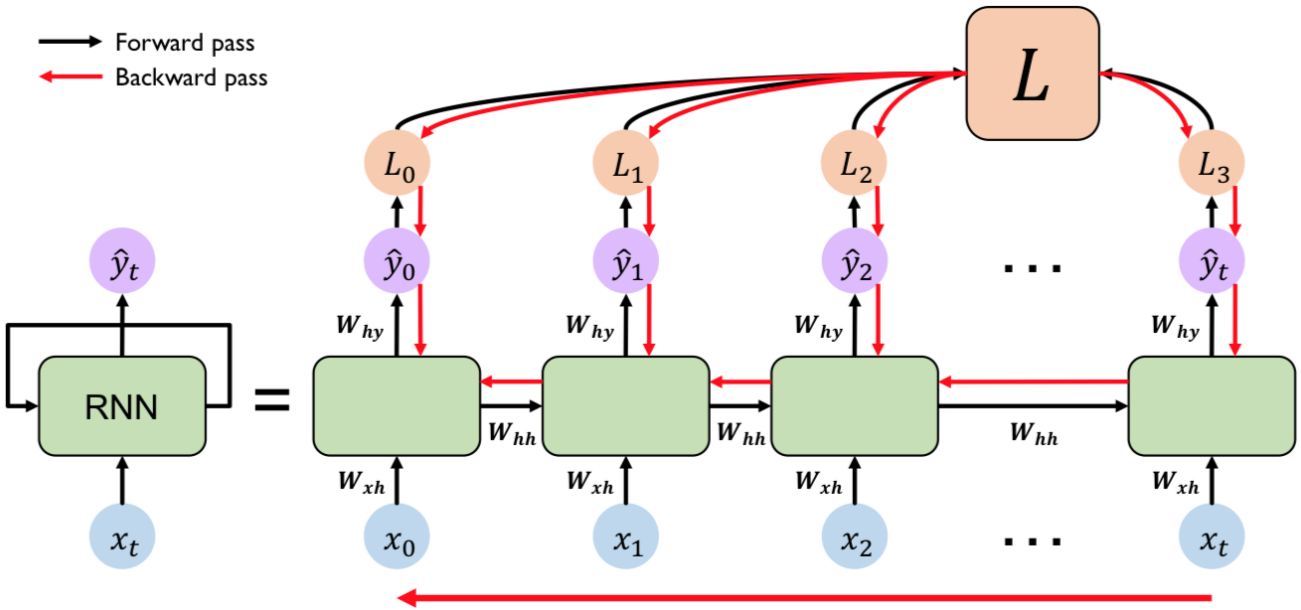
Backpropagation through time is just a fancy buzz word for backpropagation on an unrolled RNN

Unrolled RNN can lead to very deep networks $\Rightarrow$ tendency to capture only short term dependencies !

[22] http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-
https://towardsdatascience.com/rnn-recurrent-neural-networks-how-to-successfully-model-seque

# Recurrent Neural Network(RNN)

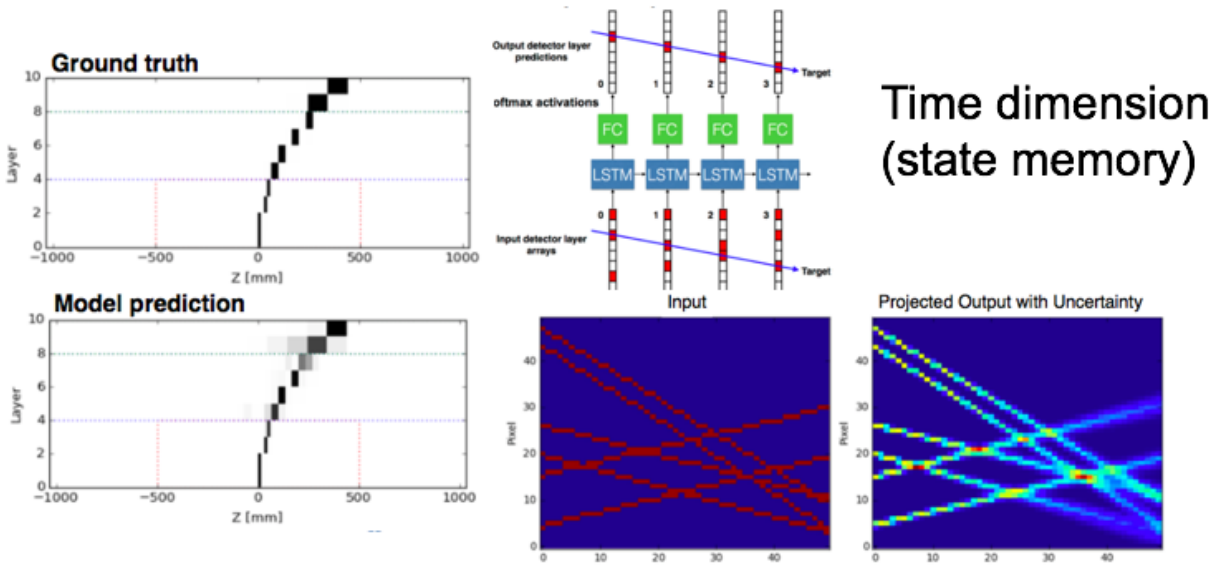
The RNN computational graph and information flow




Obs: the same set of **shared weights** are used at all time steps !

# LSTM Application in HEP: Tracking

Tracking with recurrent neural networks ( LSTM ) [23]




[23] https://heptrkx.github.io

# RNN Difficulties

## RNN Difficulties

- Long inputs leads to very deep RNN networks:
  - vanishing or exploding gradient problem
  - memory limitations
  - hard to train
- Process data in a sequential way $\Rightarrow$ no parallel processing (GPU,TPU)

A solution to these problems is the attention mechanism , which allows to grab a whole data sequence in a parallel way and infer distant correlations. [24]

## Attention Mechanism

Attention is motivated by how we pay visual attention to different regions of an image, correlate words in setences or focus on a voice ignoring background noise.

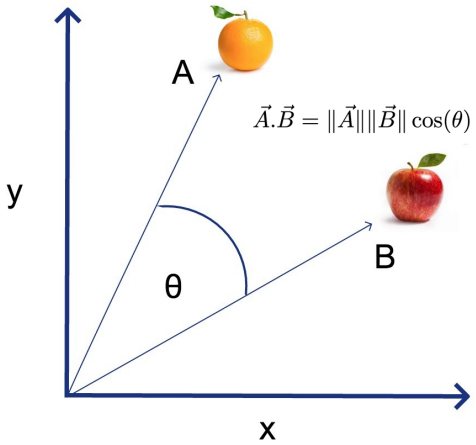
[24] https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

# Attention Mechanism

Geometric similarity between vectors (points) can be estimated by the "dot product"

**Attention Mechanism**

Attention mecanism generalizes geometric similarity by learning a metric to quantify similarity between arbitrary objects (words, images, ...) [25]




Euclidean dot product metric

$$\vec{A}.\vec{B} = \begin{pmatrix} A_1 & A_2 & A_3 \end{pmatrix} \begin{pmatrix} g_{00} & g_{01} & g_{02} \\ g_{10} & g_{11} & g_{12} \\ g_{20} & g_{21} & g_{22} \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ B_3 \end{pmatrix}$$

, where $g_{00} = g_{11} = g_{22} = 1$ and $g_{ij} = 0$ if $i \neq j$

**Attention** associates weights to pixels in an image or to words in a text, focusing on relevant part of the data and ignoring others. [26]

[25] A.Karpathy, Introduction to Transformers - https://www.youtube.com/watch?v=XfpMkf4rD6E

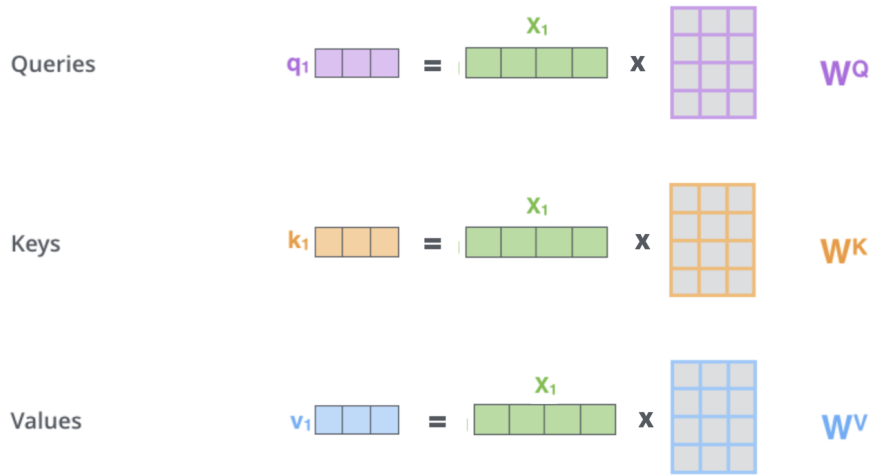
[26] C.Olah, Attention and Augmented RNN - https://distill.pub/2016/augmented-rnns/

# Transformers : Attention is All you Need

The Transformer architecture uses attention based MLPs and they outperform RNNs. [27]

## Self-Attention

Self-attention is calculated from three vectors (Query, Key, Value) obtained by multiplying for each input vector embedding X , by three learnable weight matrices $W^Q$, $W^K$ and $W^V$ [28]




Obs: the same set of weights $W^Q$, $W^K$ and $W^V$ are shared among all inputs $X_i$

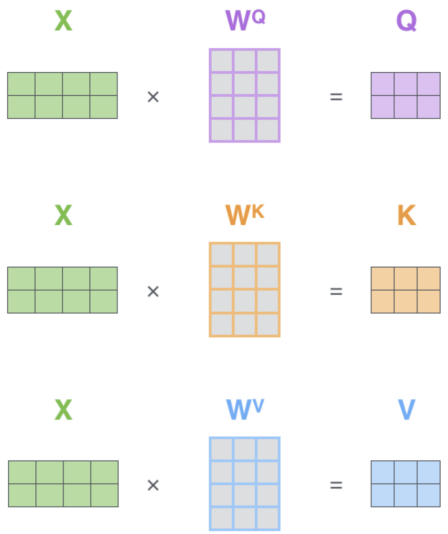
[27] A.Vaswani & all , Attention Is All you Need (2017) , https://arxiv.org/abs/1706.03762

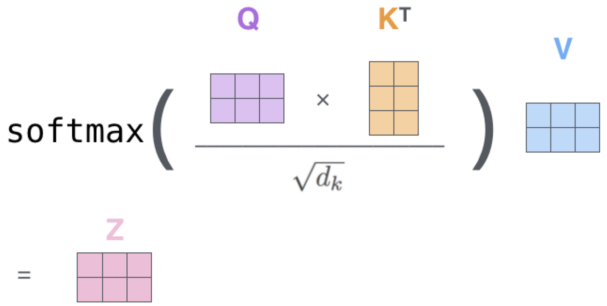
[28] https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-m

# Transformer Networks

## Matrix Calculation of Self-Attention

Query, Key, and Value matrices are defined by packing the input embeddings into a matrix X and multiplying by weight matrices $W^Q$, $W^K$ and $W^V$. The self-attention layer output is then softmaxed



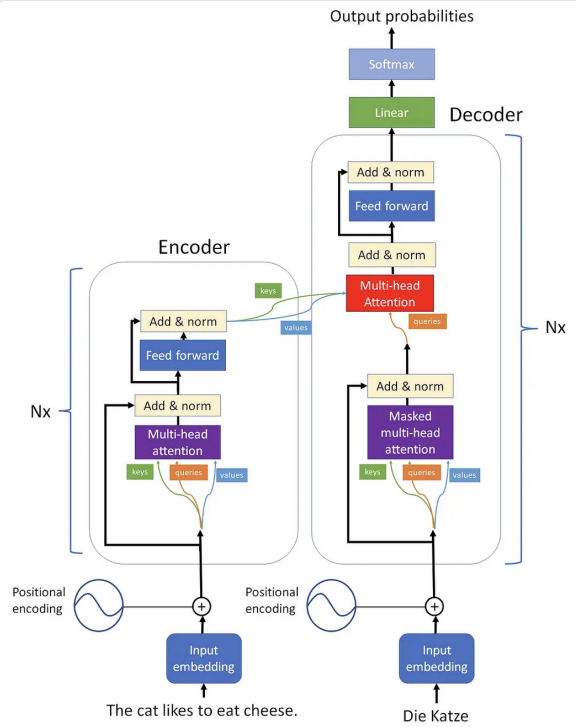




Obs: Transformers uses multi-headed attention , where each head has it's own set of weigth matrices $W^Q$, $W^K$ and $W^V$ ( like CNN channels ). Each head processes the inputs in parallel and independently, focusing on different aspects.

# Transformer Networks

## Transformer Architecture

The transformer architecture [29] has an encoder decoder struture. It uses attention and positional encoding to weight the importance of each part of the input data.



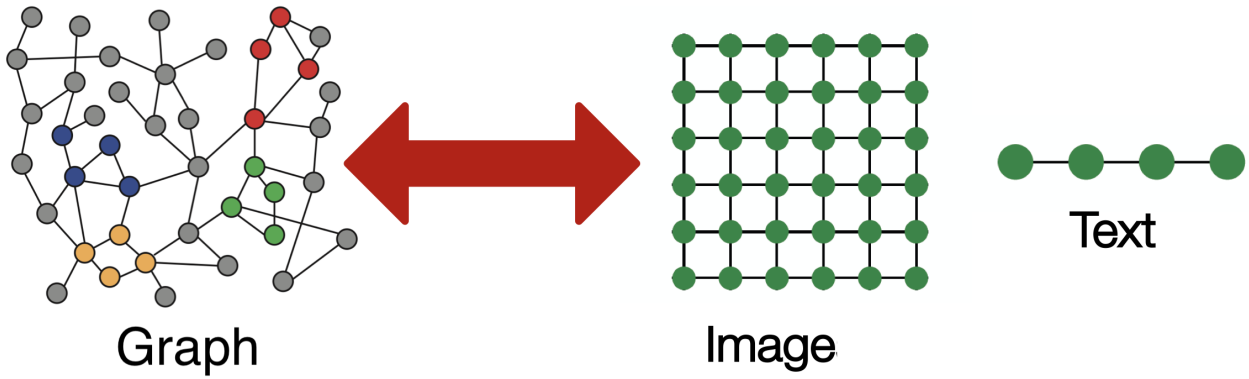

- The network has an encoder and decoder structure with independent weights.
- Self attention is used to weight the importance of each input token
- Cross attention combines keys and values from encoder (input sequence) with query from the decoder to predict the next output sequence token [30]
- Transformers can also deal with images by splitting it into patches (”image tokens”)

[29] https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functi
[30] https://vaclavkosar.com/ml/cross-attention-in-transformer-architecture

# Graph Structured Data

**Graph**

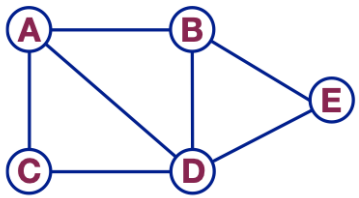
A graph is an abstract mathematical structure defined by its nodes(vertices) and edges(links): $G(N, E)$. The nodes list is unordered and can have variable length, while edges represent nodes relations. Each node and edge can have features ( ex: node position, edge lengh, ... ).




Sequences ( linear graph ) and images ( regular grid graph ) can be seen as special types of graph structured data !

# Graph Representation

For a NN to operate on a given dataset it's necessary to encode it in a mathematical representation. A naive approach would be to represent graph structured data as an augmented adjacency matrix with node features information and use it as an MLP input. [31]




| | A | B | C | D | E | Feat | |
|---|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| B | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| C | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| D | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| E | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

**Problems:**

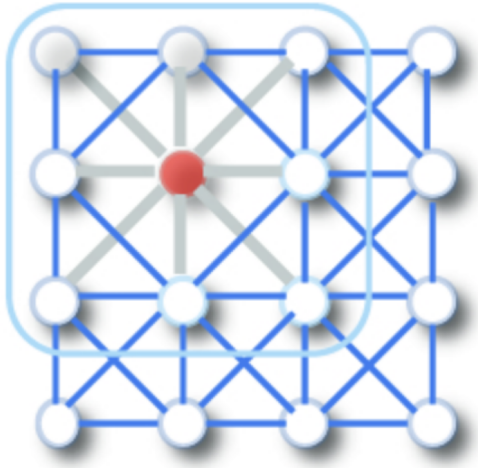
- Adjacency matrix depends on arbitrary node labeling ( not permutation invariant )
- Adjacency matrix can became too large and sparse ( need compact representattion )
- Matrix size depend on number of nodes ( MLP takes fixed input size )

[31] https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutiona
https://tkipf.github.io/graph-convolutional-networks

# Graph Neural Network (GNN)

## From Convolution to Message Passing

Inspired by CNNs we define a graph convolution by aggregating information from the node neighbourhood. The neighbouring nodes features are collected in a process known as message passing and then it's aggregated to the given node features [32].

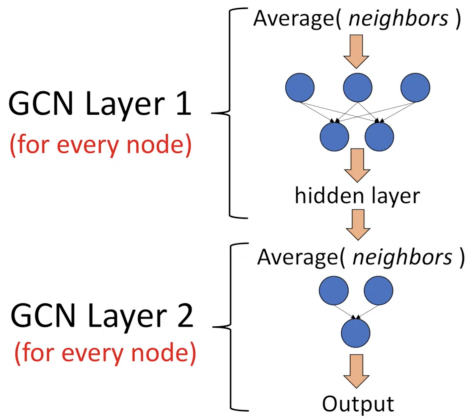


[32] J.Gilmer & all. , Neural Message Passing for Quantum Chemistry , https://arxiv.org/abs/1704.01212

# Graph Neural Network (GNN)

## Graph Convolution Network (GCN)

Taking the nodes features as inputs the GCN aggregates neighbouring nodes features by averaging and multiplying by shared weights, getting a latent representation (embedding) of the node features. [33]




$h_v^0 \rightarrow$ input features of node-v

$$h_v^k = \sigma\left(W_k' h_v^{k-1} + W_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|}\right)$$

$h_v^k \rightarrow$ latent node features in Layer-k

Obs: Weights $W_k$ are shared between nodes of the graphs and the node latent representation can have an arbitrary number of dimensions (like CNN filters)

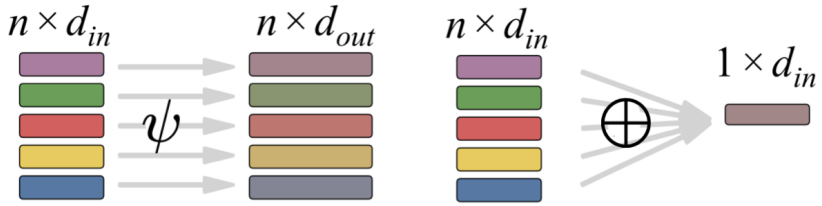
[33] https://towardsdatascience.com/hands-on-graph-neural-networks-with-pytorch-pytorch-geome
http://web.stanford.edu/class/cs224w/slides/08-GNN.pdf

# Graph Network (GN)

A general graph can have edge features as well as node features. [34]

**Graph Network (GN)**

The most general Graph Network(GN) can be modeled as a graph to graph mapping. An update (message passing) function $\psi$ acts on nodes features $x_i$ or edges features $e_{ij}$, followed by a permutation invariant aggregation $\bigoplus$ and an activation function $\phi$.




$$h_i = \phi\left(x_i, \bigoplus_{j \in \mathcal{N}(i)} \psi(x_i, x_j, e_{ij})\right)$$

The update function takes in a *fixed size* $d_{in}$ input and returns a *fixed size* $d_{out}$ output, while the aggregation function takes in *variable sized* $n$ inputs and returns a *fixed size* $d_{in}$ output.

Obs: GN formalism is not a specific model architecture and doesn't determine what the update and aggregate functions are.

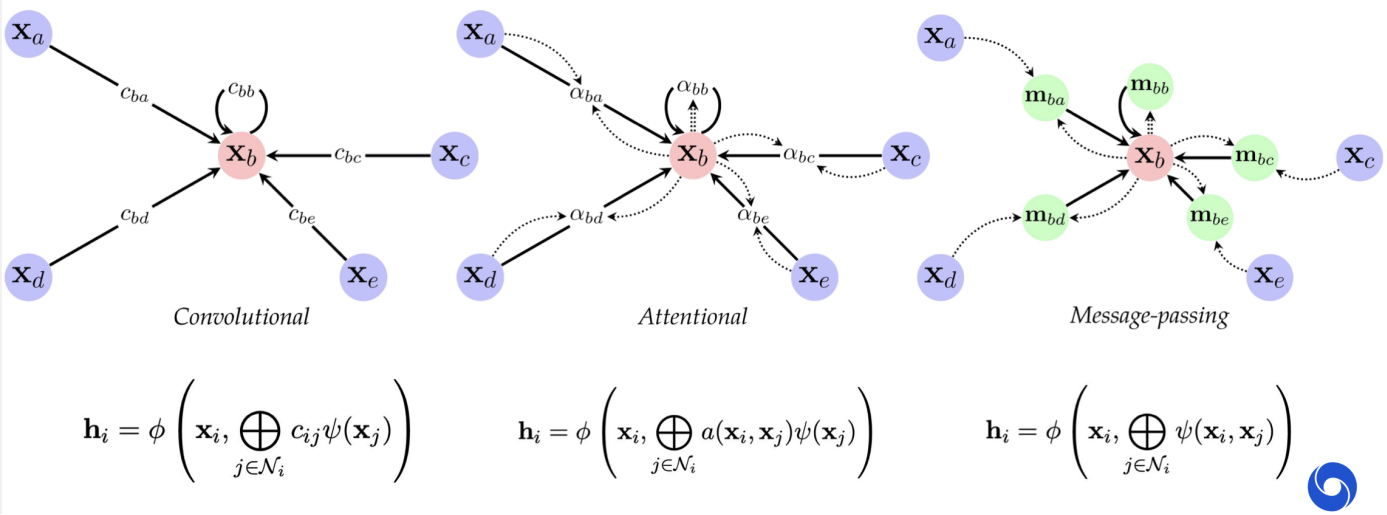
[34] https://github.com/deepmind/graph_nets
https://arxiv.org/pdf/2007.13681

# Graph Neural Network

## Three Particular Flavours of GNNs

GNNs can be further grouped according to the following flavours: [35]

- **Convolutional (GCN):** fixed weights coeficients $c_{ij}$
- **Attention Based (GAT):** learnable attention weights calculated as $\alpha_{ij} = a(x_i, x_j)$
- **Message Passing (GN):** general messages $m_{ij} = \psi(x_i, x_j)$ sent across edges



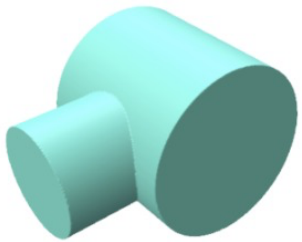

[35] https://petar-v.com/talks/GNN-Wednesday.pdf

# From Graphs to Sets

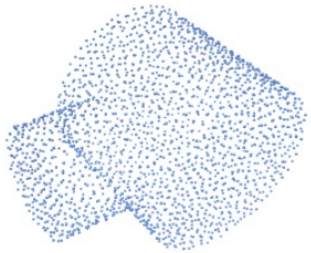
**Sets**

Abstracting even more on graphs we encounter sets, which are collections without intrinsic order or relations. A set can be seen as a graph without vertices. [36]

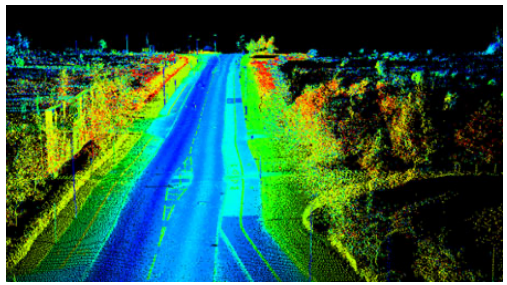
DeepSets and PointNet are examples of networks capable of learning on point set structured data. They have permutation invariance by design as inductive bias in the architecture. A MLP in principle could learn not to care about orderings, but training for that would be a waste of computing resources.



(a) Surface view.



(b) Point cloud.



Obs: data generated by 3D scanners and LIDAR sensors often come as point clouds with coordinates (x,y,z) as its features

[36] P.Velickovic, Graphs and Sets, https://www.youtube.com/watch?v=E_Wweuk5iqA&list=PLn2-dEmQeTfQ8YVuHBOvAhUlnIPYxkeu3&index=5&t=13s

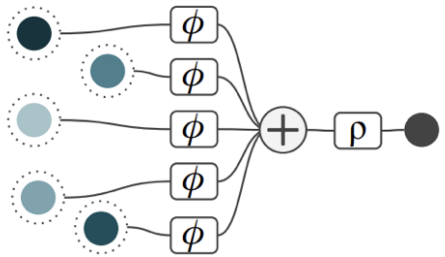
# DeepSets / PointNet Networks

Consider an input vector $\vec{x}$ with $N$ components and the set $S_N$ of all permutations of its components. The most general permutation invariant function is the sum over all permutations $\pi(\vec{x})$ of $\vec{x}$. [37]

**Janossy Pooling**

$$f(\vec{x}) = \frac{1}{|S_n|} \sum_{\pi \in S_N} \phi\left(\pi(\vec{x})\right) \qquad \Rightarrow \text{computationally expensive and scales as N!}$$

Complexity can be reduced using permutation invariants sum of $k-tuples$ , where $k < N$. For $k = 1$ we have the architectures known as DeepSets and PointNet

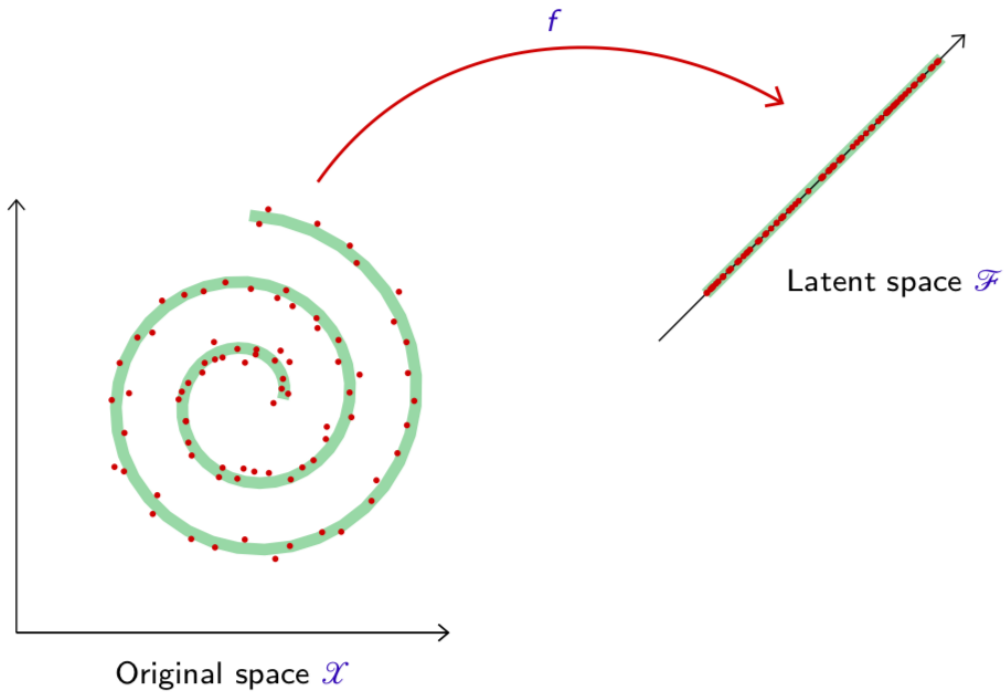



Obs: the sum aggregation of $\phi$ layers gives permutation invariance over inputs points

[37] https://fabianfuchsml.github.io/learningonsets

# Autoencoder

Many applications such as data compression, denoising and data generation require to go beyond classification and regression problems. This modeling usually consists of finding “meaningful degrees of freedom”, that can describe high dimensional data in terms of a smaller dimensional representation



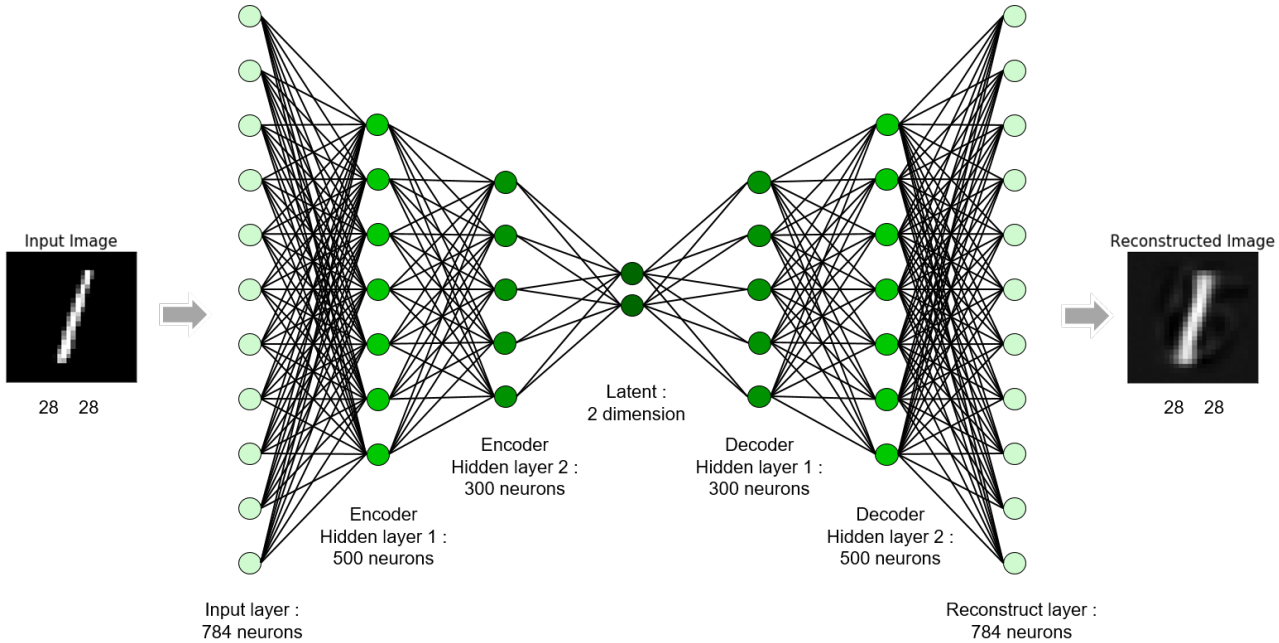

Traditionally, autoencoders were used for dimensionality reduction and denoising. Recently autoencoders are being used also in generative modeling

# Autoencoder(AE)

## Autoencoder(AE)

An AE is a neural network that is trained to attempt to copy its input to its output in an **self-supervised way**. In doing so, it learns a representation(encoding) of the data set features *l* in a low dimensional latent space.

It may be viewed as consisting of two parts: an encoder $l = f(x)$ and a decoder $y = g(l)$.



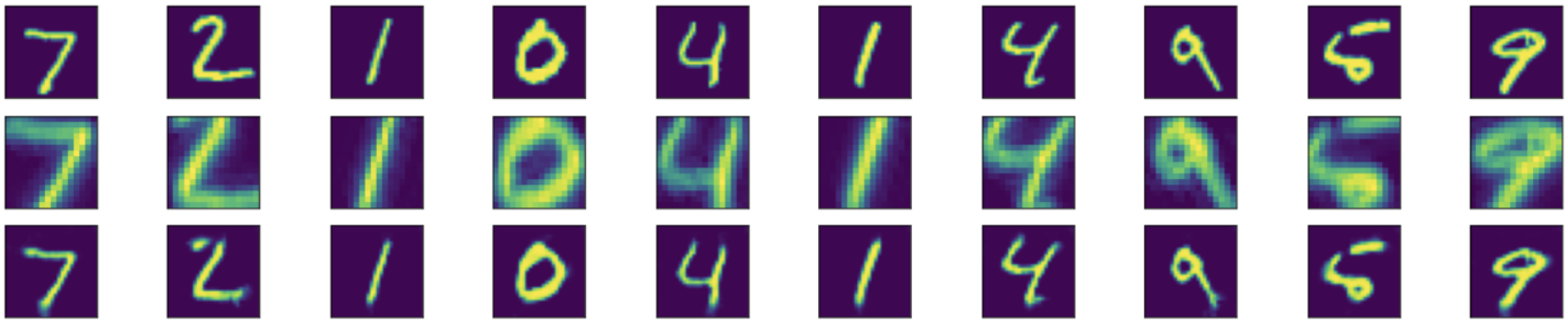

$\Rightarrow$ Understanding is data reduction

# Convolutional Autoencoder(CAE)

## Convolutional Autoencoder(CAE)

A CAE [38] is built out of convolutional layers assembled in a AE architecture and it's trained to attempt to reconstruct an output image from an input image after data compression. In doing so, it learns how to compress images.

Bellow we have MNIST digits ( 28x28 pixels ) used as input and the output images and the corresponding latent space compressed images ( 16x16 pixels )



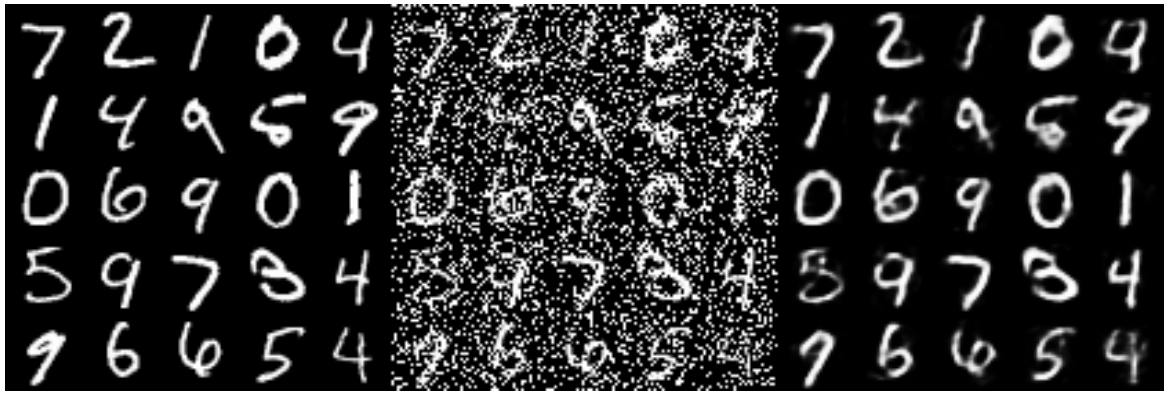
⇒ CAE learns that image borders have no information and chops the central part.

[38] https://towardsdatascience.com/introduction-to-autoencoders-b6fc3141f072

# Denoising Autoencoder (DAE)

## Denoising Autoencoder (DAE)

The DAE is an extension of a classical autoencoder where one corrupts the original image on purpose by adding random noise to it's input. The autoencoder is trained to reconstruct the input from a corrupted version of it and then used as a tool for noise extraction

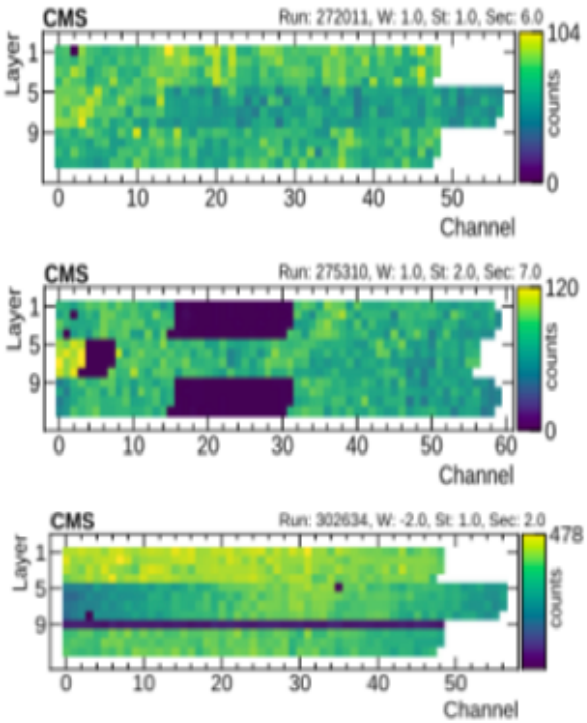


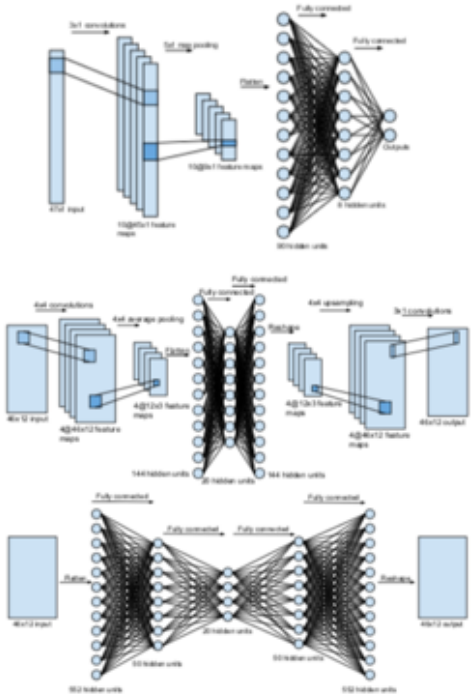
# Autoencoder Application in HEP: DQM

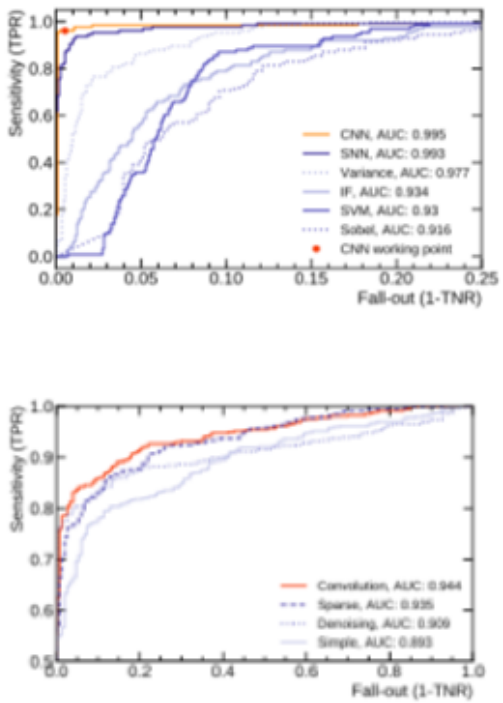
AE can be used for anomaly detection by training on a single class , so that every anomaly gives a large reconstruction error

## Detector Quality Monitoring (DQM)

Monitoring the CMS data taking to spot failures ( anomalies ) in the detector systems



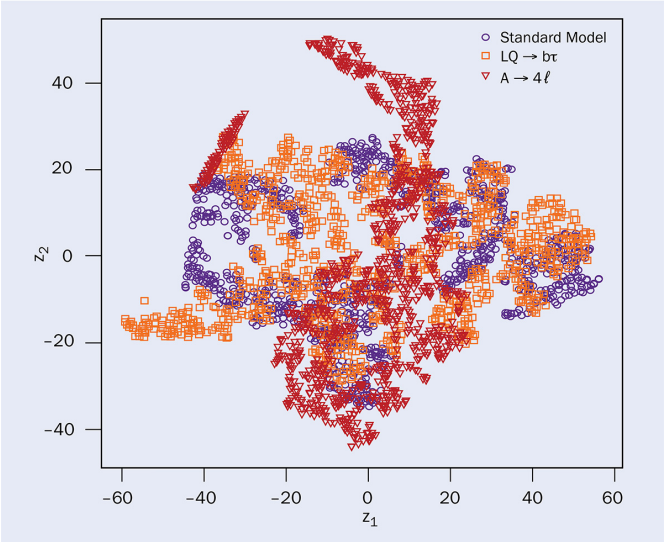


# Autoencoder Application in HEP: Anomaly Detection at LHC ( BSM )

AE can watch for signs of new physics at the LHC that have not yet been dreamt up by physicists [39]

**Anomaly Detection ( BSM Physics Search)**

AE trained on a SM data sample , so that an anomalous event gives a large reconstruction error. The LHC collisions are compressed by an AE to a two-dimensional representation (z1,z2). The most anomalous events populate the outlying regions.




Outliers could go into a data stream of interesting events to be further scrutinised

[39] https://cerncourier.com/a/hunting-anomalies-with-an-ai-trigger

# Variational Autoencoder(VAE)

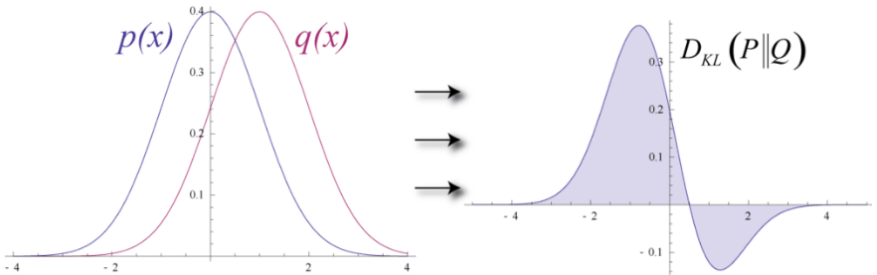
## Variational Autoencoder(VAE)

A VAE [40] is an autoencoder with a loss function penalty Kullback–Leibler (KL)divergence [41] [42] that enforces a latent space representation that follows a normal distribution. To generate images with a VAE one just samples a latent vector from a unit gaussian and pass it through the decoder.

The KL divergence, or relative entropy, is a measures of how one probability distribution differs from a second, reference distribution ( $\Rightarrow$ information loss )

## Kullback–Leibler Divergence ( Relative Entropy )

$$D_{KL}(p|q) = \int_{-\infty}^{+\infty} dx\ p(x)\ log\left(\frac{p(x)}{q(x)}\right)$$




Original Gaussian PDF's KL Area to be Integrated

[40] http://kvfrans.com/variational-autoencoders-explained
[41] https://en.wikipedia.org/wiki/Evidence_lower_bound
[42] https://www.youtube.com/watch?v=HxQ94L8n0vU

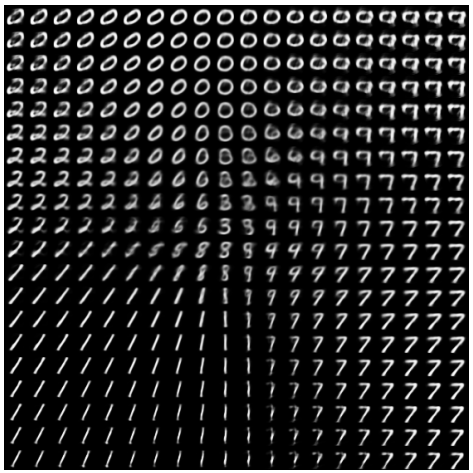
# Variational Autoencoder(VAE)

## VAE Loss

The VAE loss [43] is composed of a mean squared error term that measures the reconstruction accuracy and a KL divergence term that measures how close the latent variables match a gaussian.

$$L = \| x - f \circ g(x) \|^2 + D_{KL}(p(z|x)|q(z|x))$$

VAE generated numbers obtained by gaussian sampling a 2D latent space



$\Rightarrow$ KL loss is a regularization term that helps learning "well organized"latent space

[43] https://tiao.io/post/tutorial-on-variational-autoencoders-with-a-concise-keras-implement

# Variational Autoencoder(VAE)

## Feature Disentanglement(Factorization) in VAE

Feature disentanglement is isolating the source of variation in data [44]

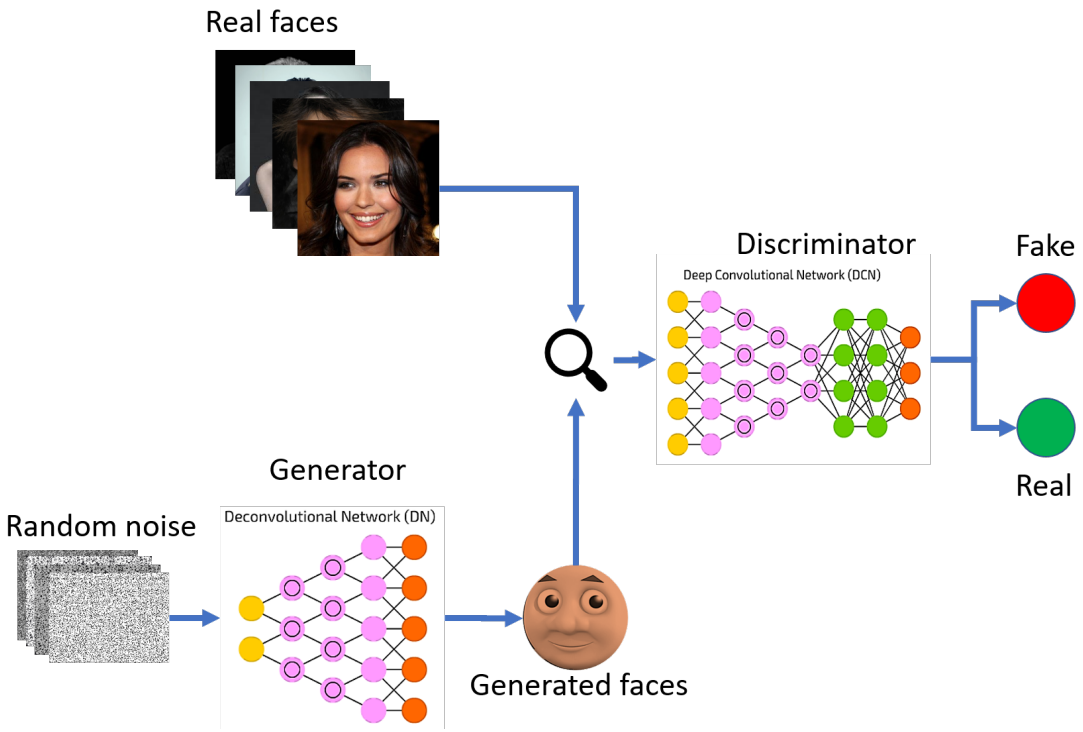

(a) Emotion (smile) (b) Gender (c) Azimuth

[44] http://proceedings.mlr.press/v97/mathieu19a/mathieu19a.pdf

# Generative Adversarial Network(GAN)

## Generative Adversarial Networks (GAN)

GANs are composed by two NN, where one generates candidates and the other classifies them. The generator learns a map from a latent space to data, while the classifier discriminates generated data from real data.

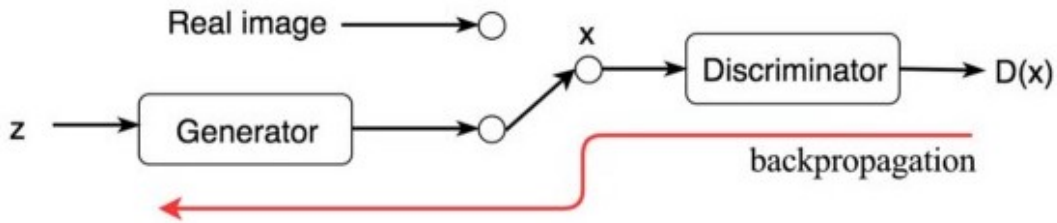


# Generative Adversarial Network(GAN)

GAN adversarial training [45] works by the two neural networks competing and training each other. The generator tries to ”fool” the discriminator, while the discriminator tries to uncover the fake data.

## GAN Training

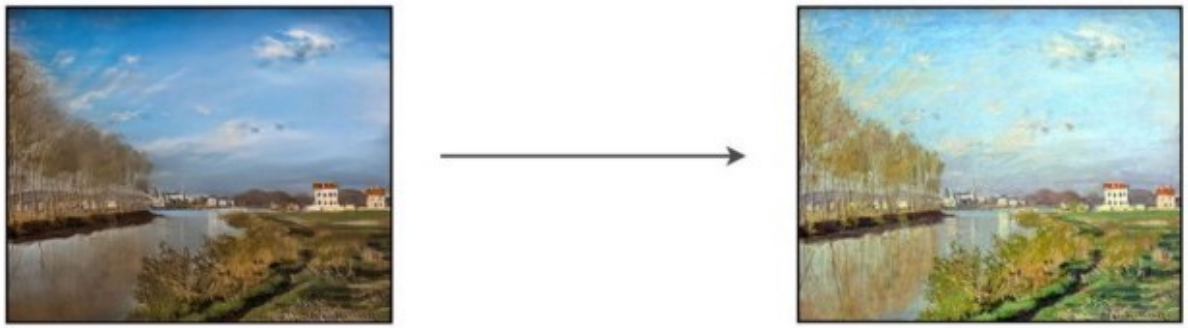
1. The discriminator receives as input samples synthesized by the generator and real data . It is trained just like a classifier, so if the input is real, we want output=1 and if it's generated, output=0
2. The generator is seeded with a randomized input that is sampled from a predefined latent space (ex: multivariate normal distribution)
3. We train the generator by backpropagating this target value all the way back to the generator
4. Both networks are trained in alternating steps and in competition




[45] https://medium.com/@jonathan_hui/gan-whats-generative-adversarial-networks-and-its-appli

# Generative Adversarial Network(GAN)

GANs are quite good on faking celebrities images [46] or Monet style painitings [47] !

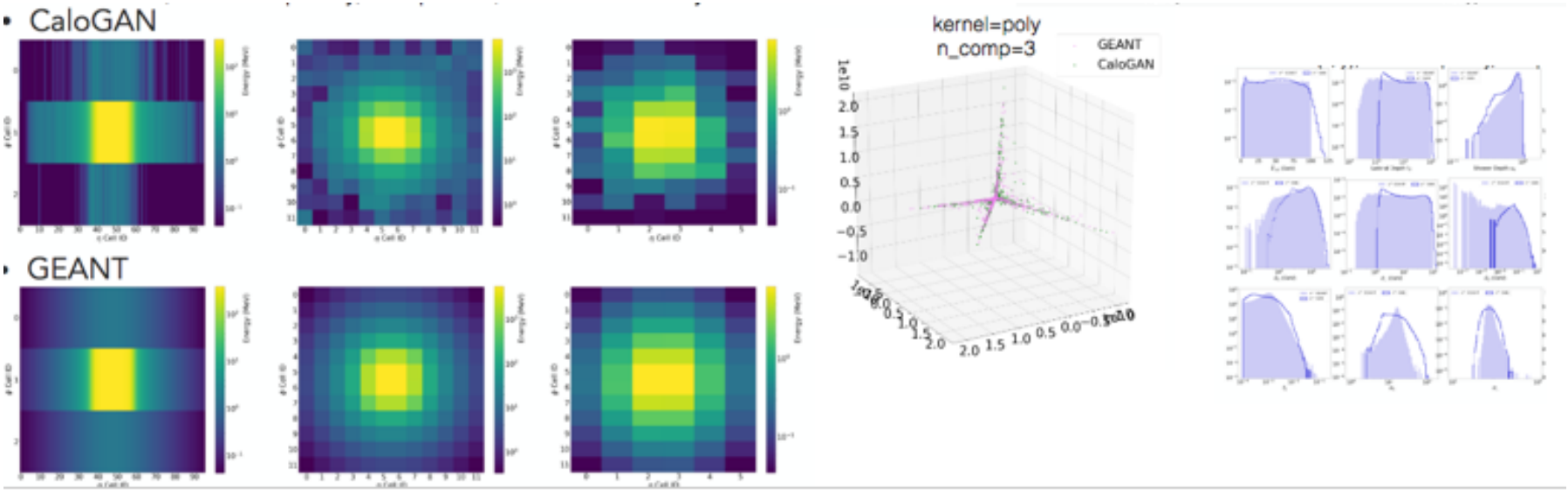



[46] https://research.nvidia.com/publication/2017-10_Progressive-Growing-of
[47] https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

# GAN Application in HEP: MC Simulation

## CaloGAN

Simulating 3D high energy particle showers in multi-layer electromagnetic calorimeters with a GAN [48]




[48] https://github.com/hep-lbdl/CaloGAN

# Diffusion Model (DM)

A diffusion model works as a Denoising Autoencoder(DAE) trained to denoise images blurred with Gaussian noise.


**Introducing Sora — OpenAI's text-to-video model**

# Diffusion Model (DM)

## Diffusion Model

Diffusion models define a Markov chain of diffusion steps by slowly adding random noise to data and then learn to reverse the diffusion process in order to reconstruct desired data samples from the noise. [49]

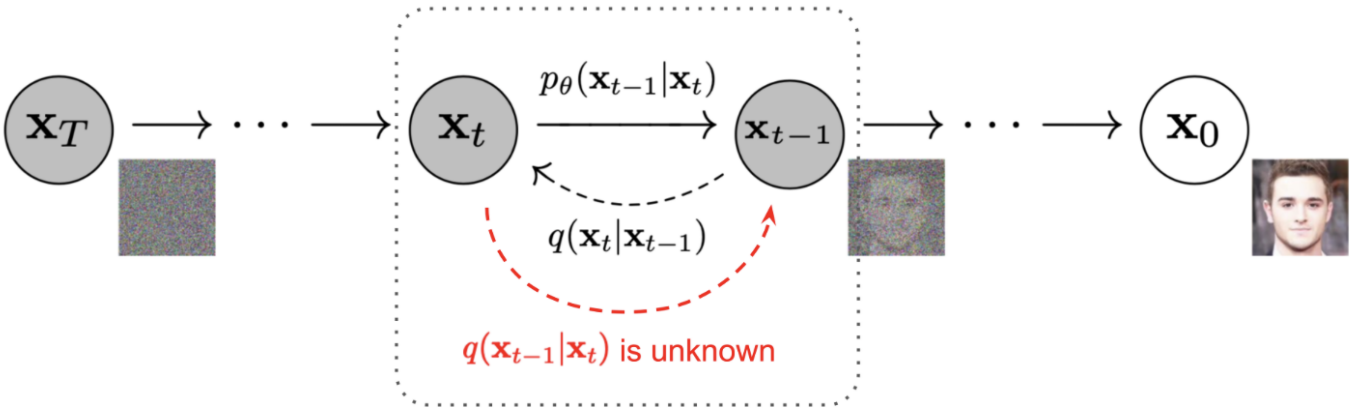



Fig. 2. The Markov chain of forward (reverse) diffusion process of generating a sample by slowly adding (removing) noise. (Image source: Ho et al. 2020 with a few additional annotations)

[49] https://lilianweng.github.io/posts/2021-07-11-diffusion-models

# Diffusion Model (DM)

## Forward Process

Given a data point $x_0$, the forward diffusion process adds small amount of Gaussian noise in steps (Markov Chain) , producing a sequence of noisy samples. The step sizes are controlled by $\beta_t$. At the end we are left with a noisy image represented by an isotropic Gaussian distribution ( same variance along all dimensions).

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

$\Rightarrow$ The model latent space has the same dimensionality as the original data ( high dimensionality )

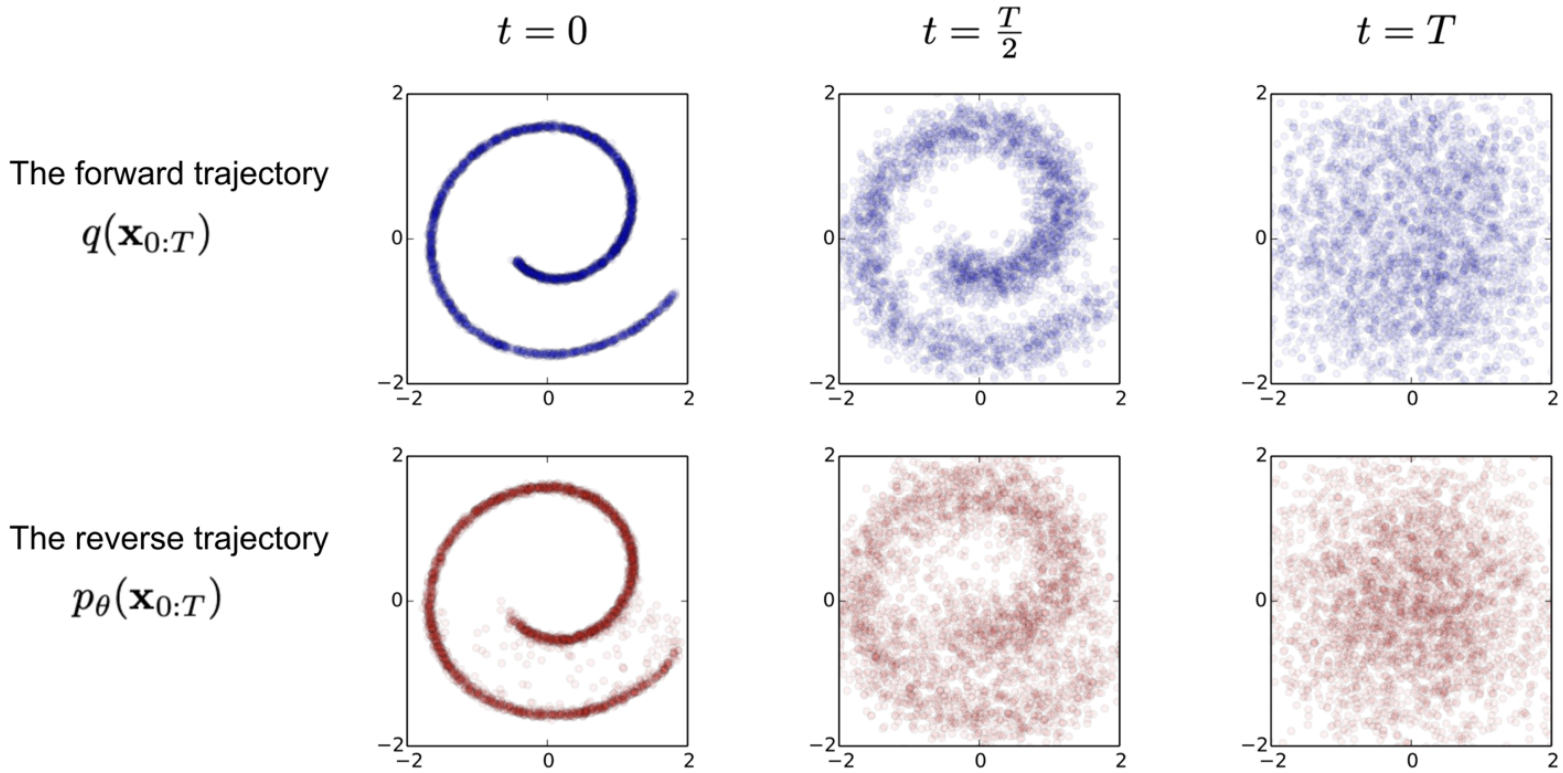
## Reverse/Backward Process

The task of removing the added noise in the forward process, again in an iterative fashion, is done using a neural network. If we can reverse the above process and sample from $q(x_{t-1}|x_t)$ , we can recreate the original data sample from a Gaussian noise input $X_T \simeq \mathcal{N}(0,1)$.

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \quad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

# Diffusion Model (DM)

Example of training a diffusion model for modeling a 2D Swiss roll data [50]




[50] https://arxiv.org/abs/1503.03585

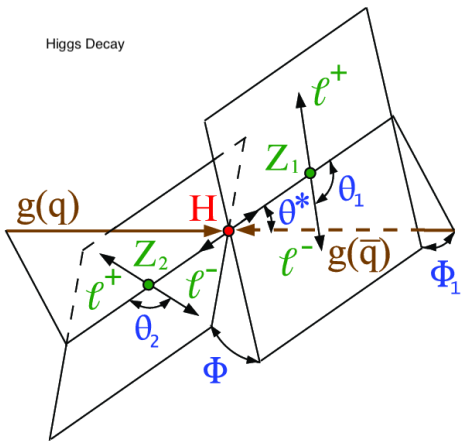
# Domain Knowledge and Inductive Biases

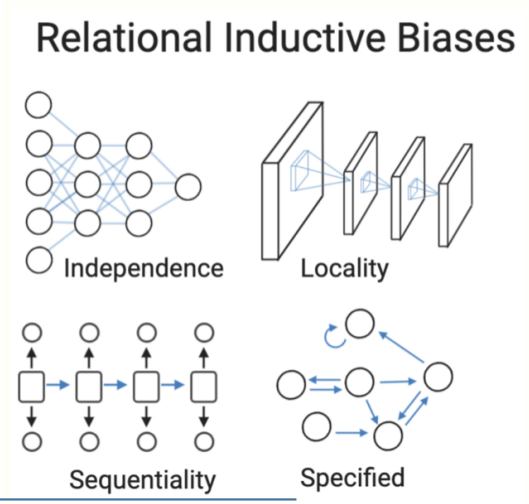
Domain knowledge can be infused into a model in different ways [51]

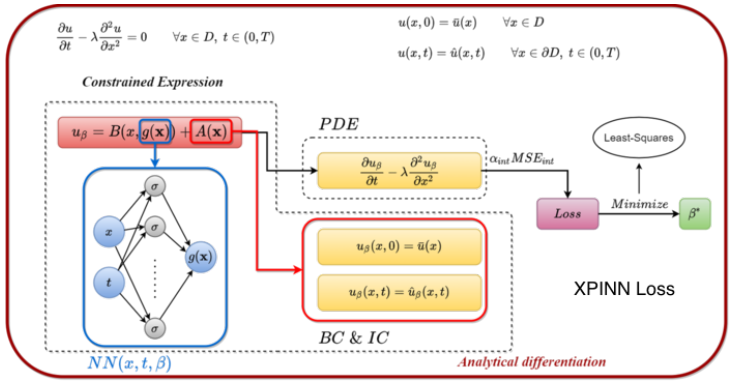
## Domain Knowldge

- Using engineered variables variables as input data
- Creating specialized architecture layers: CNN, RNN, GNN, DS
- Adding a penalty term to the loss function: PINN

⇒ It's also possible to use simultaneous combinations of the above methods




[51] https://sgfin.github.io/2020/06/22/Induction-Intro/
https://www.nature.com/articles/s41598-021-04590-0

# The End !!!