Machine Learning in Physics

A. Sznajder

UERJ
Instituto de Fisica

August - 2023

Outline

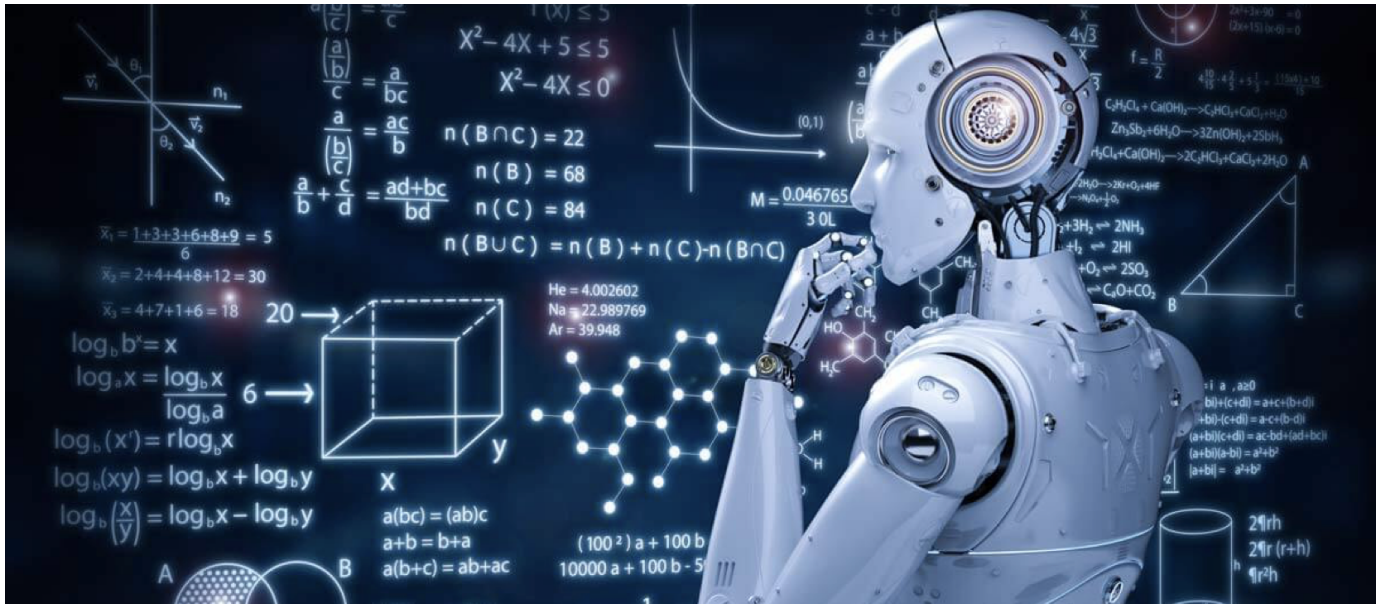
Support Material

- **Textbook:**
  Aurélien Géron - Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow
- **Jupyter Notebooks :**
  `https://github.com/sznajder/USP-Minicurso-de-ML-2023`
- **Google Colab :**
  `https://colab.google`

What is Machine Learning ?

Machine Learning (ML)

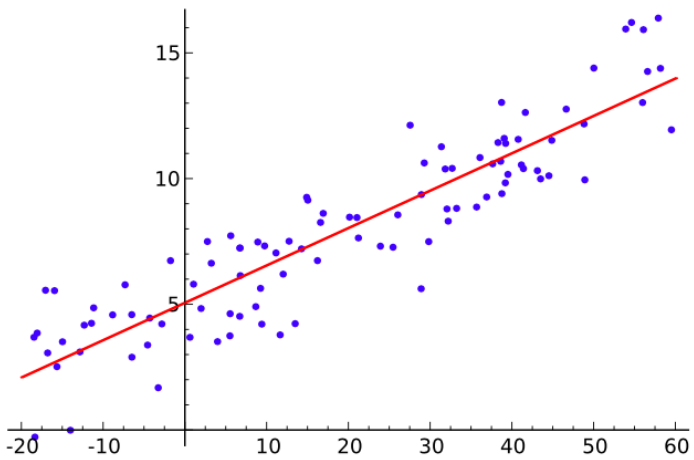
Machine learning (ML) is the study of computer algorithms capable of building a mathematical model out of a data sample (learn from examples). The algorithm builds a predictive model without being explicitly programmed (sequence of instructions) to do so



Centuries Old Machine Learning [1]

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

The Centuries Old Machine Learning Algorithm



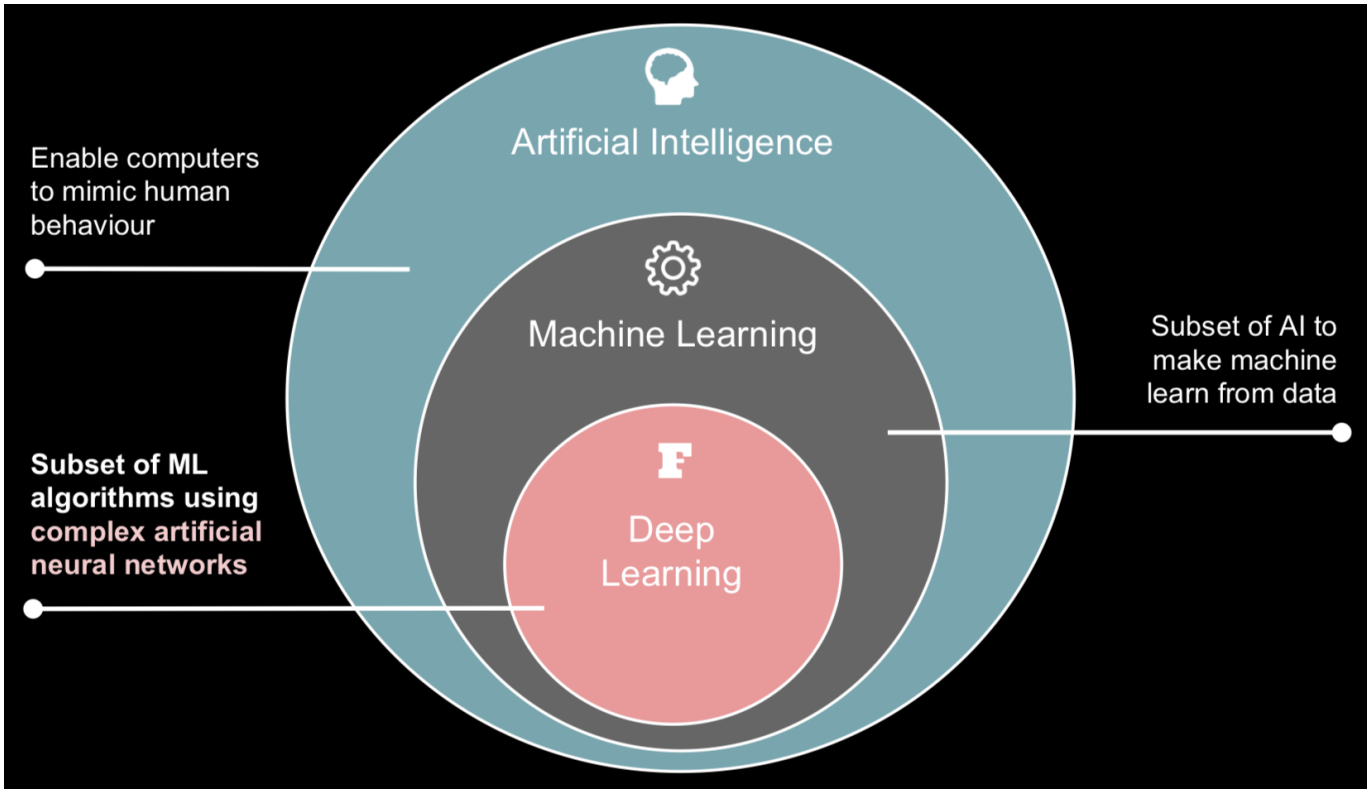
Linear regression (Source)

Linear regression is a bit too wimpy a technique to solve the problems of image , speech or text recognition, but what it does is essentially what supervised ML is all about (learn a function from a data samples) !

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

Artificial Inteligence

"Inteligence can be understood as the ability to process current information to inform future decisions"



Introduction to Machine Learning

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

1) Tasks:

- Classification
- Regression
- Density Estimation and Data Generation

2) Data Types

- Images
- Sequences
- Graphs

3) Learning Paradigms

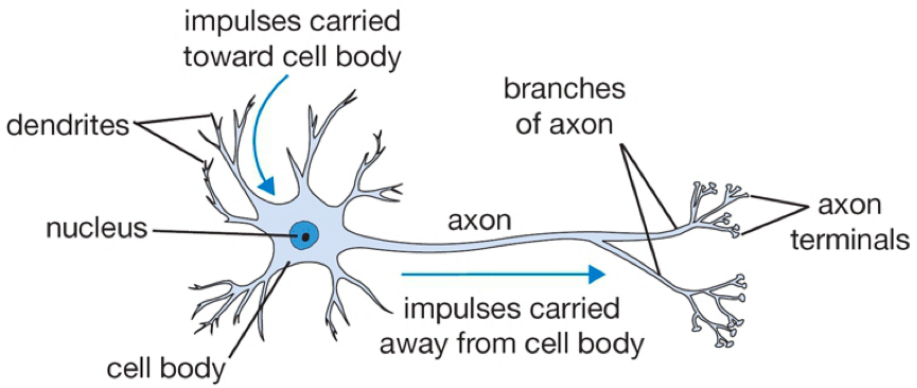
- Supervised Learning (labeled)
- Self and Unsupervised Learning (unlabeled)
- Reinforcement Learning (reward)

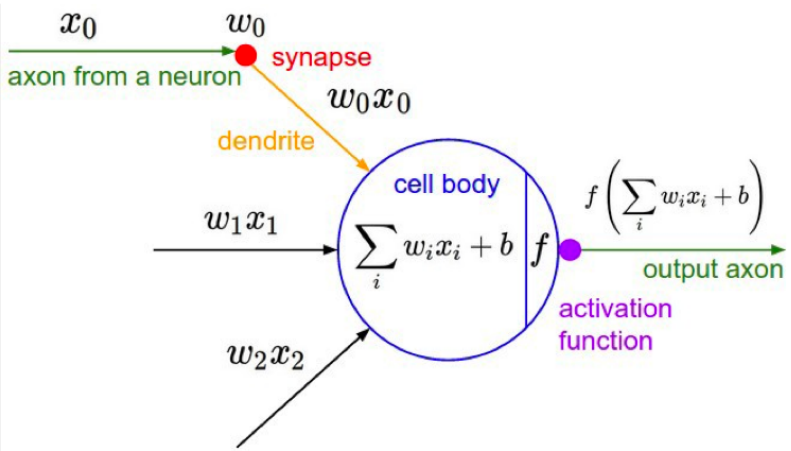
3) Neural Networks Architectures

- Multilayer Perceptron (MLP)
- Recurrent Networks (RNN,LSTM,GRU)
- Transformer Networks (TN)
- Convolutional Networks (CNN)
- Graph Networks (GCN,GAT,DGCN,IN)
- Autoencoder (AE,DAE,VAE)
- Generative Adversarial Network (GAN)
- Normalizing Flows (NF)
- Diffusion Models (DM,SDM)

Neural Networks

Artificial Neural Networks (NN) are computational models vaguely inspired [2] 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, ...

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

Artificial Neuron

Artificial Neuron Model

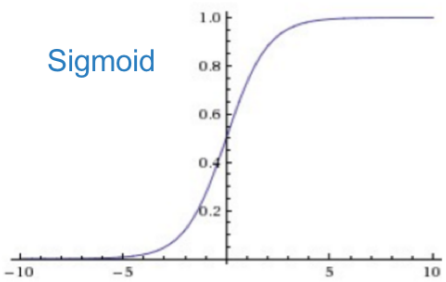
Each node of a NN receives inputs $\vec{x} = \{x_1, \ldots, x_n\}$ from other nodes or an external source and computes an output y according to the expression

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

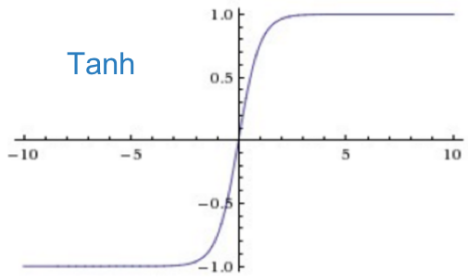
, where W_i are connection weights, B is the treshold and F the activation function [3]

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

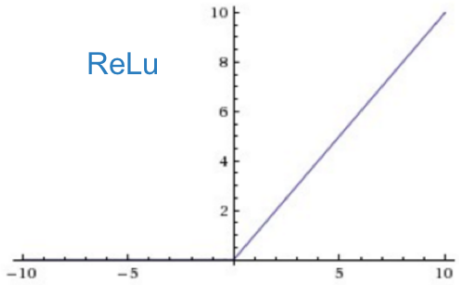
Sigmoid



Tanh



ReLu



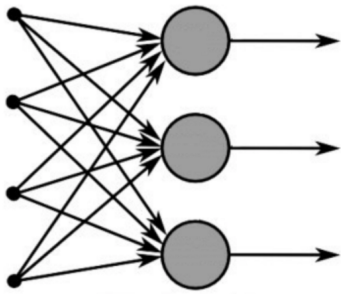
[3] Nonlinear activation is fundamental for nonlinear decision boundaries

Neural Network Topologies

Neural Networks can be classified according to the type of neuron interconections and the flow of information

Feed Forward Networks

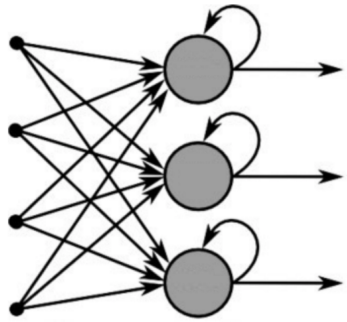
A feedforward NN is an neural network wherein connections between the nodes do not form a cycle. The information always moves one direction, from input to output, and it never goes backwards.



Feed-Forward Neural Network

Recurrent Neural Network

A Recurrent Neural Network (RNN) is a neural network that allows connections between nodes in the same layer, with themselves or with previous layers. RNNs can use their internal state (memory) to process sequential data.



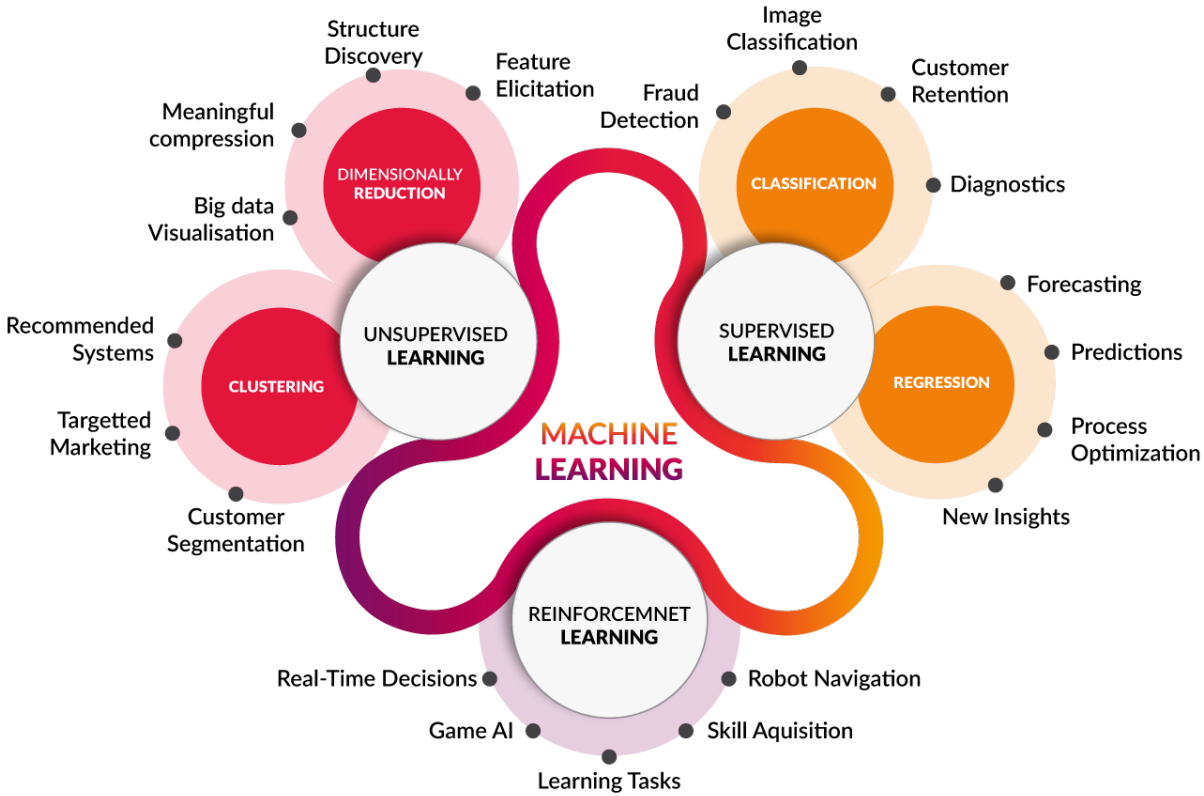
Recurrent Neural Network

A NN layer is called a dense layer to indicate that it's fully connected.

Learning Paradigms

Learning Paradigms

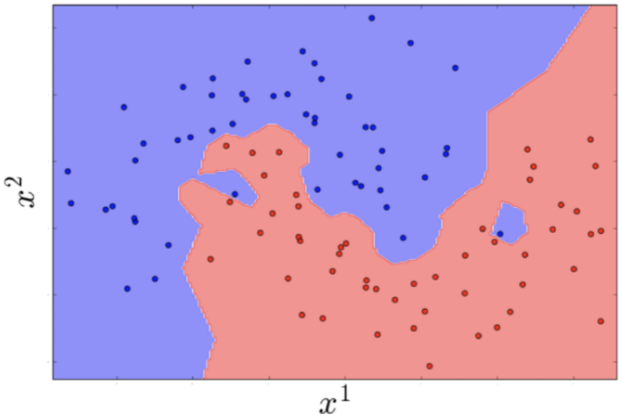
Depending on the problem and available dataset we can have different learning approaches

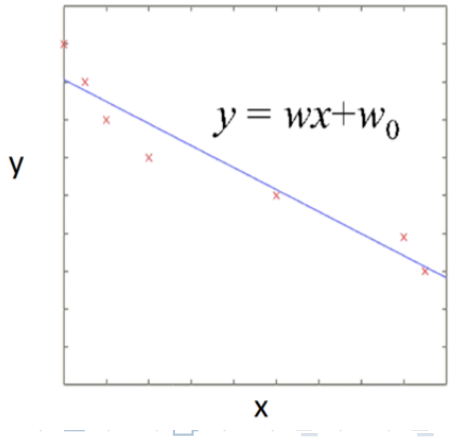


Supervised Learning

Supervised Learning

- During training the network learns how to map inputs to target outputs.
- A learning algorithm uses the error between targets and network outputs (scores) to adjust network weights and biases.
- Given some labeled data $D = \{(\vec{x}_1, \vec{t}_1), \ldots, (\vec{x}_n, \vec{t}_n)\}$ with features $\{\vec{x}_i\}$ and targets $\{\vec{t}_i\}$, the algorithm finds an approximation to the mapping $\vec{t}_i = F(\vec{x}_i)$
- **Classification:** $\{\vec{t}_1, \ldots, \vec{t}_n\}$ (finite set of labels)
- **Regression:** $\vec{t}_i \in \mathbb{R}^n$





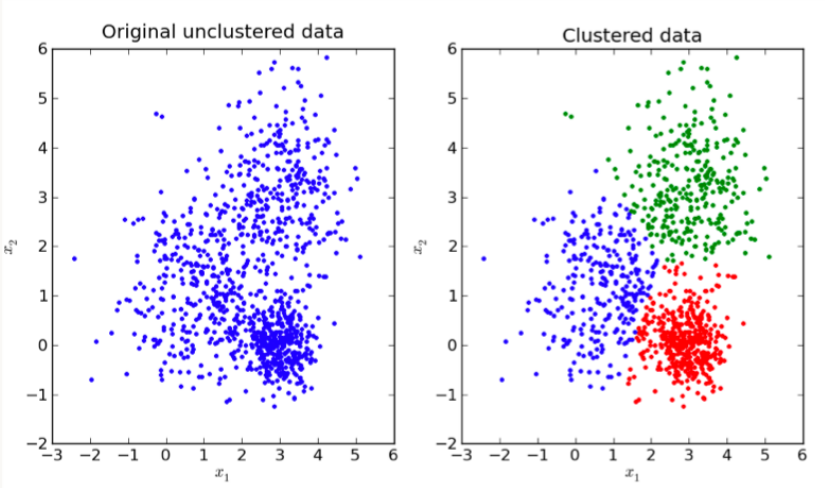
Unsupervised Learning

Unsupervised Learning

Given the unlabeled data $D = \{\vec{x}_1, \ldots, \vec{x}_n\}$ it finds underlying structures (patterns) in data

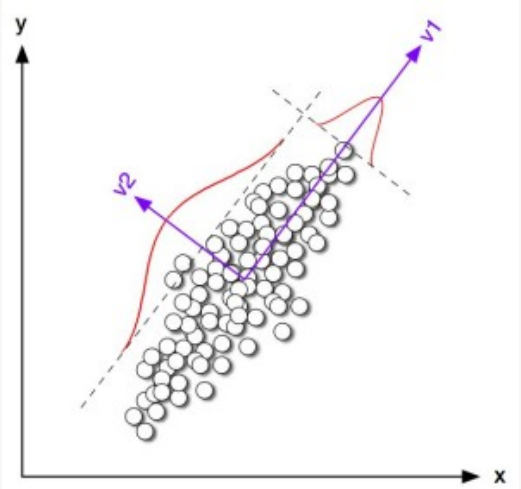
Clustering

Finds underlying partition the data into sub-groups $D = \{D_1 \cup D_2 \cup \ldots . D_k\}$



Dimensional Reduction

Find a lower dimensional representation of the data with a mapping $\vec{y} = F(\vec{x})$, where $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ with $n >> m$

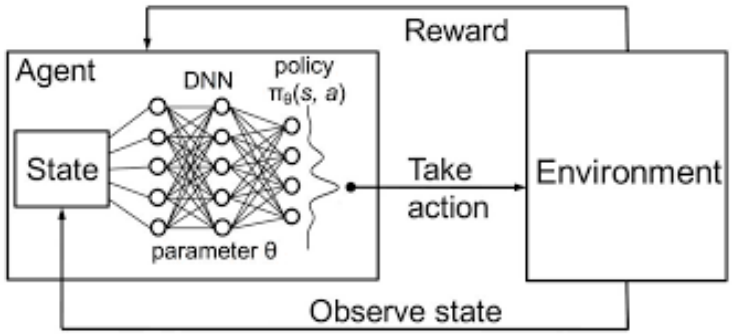


Reinforcement Learning

Reinforcement Learning

Instead of using labeled data, one maximizes some notion of reward.

- Inspired by behavioristic psychology and strongly related with how learning works in nature
- Uses trial-and-error approach to maximize a cumulative reward
- Algorithm learns the best sequence of decisions (actions) to achieve a given goal



Requires lots of data, so applicable in domains where simulated data is readily available: robotics, self-driving vehicles, gameplay ...

Reinforcement Learning

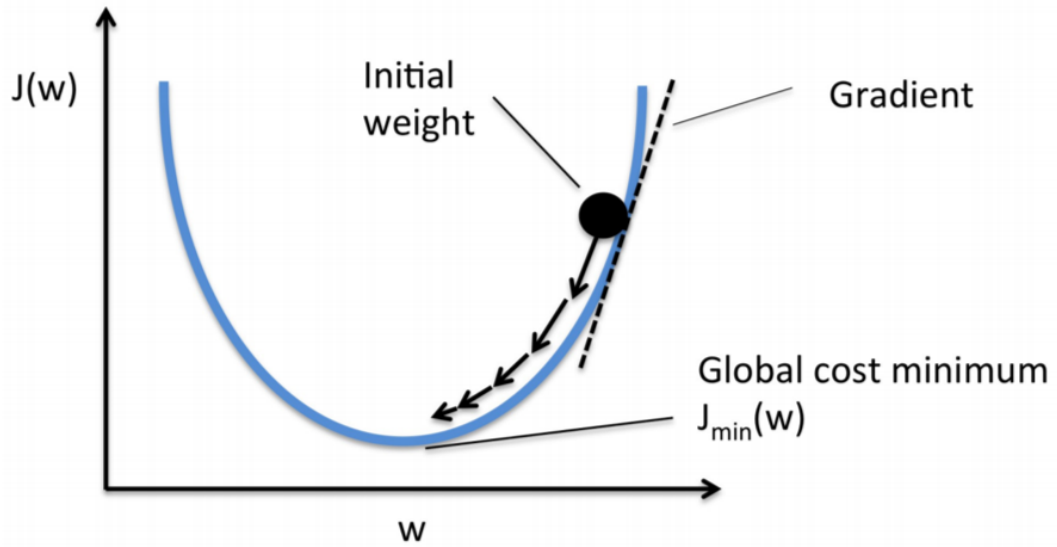
Google Deepmind neural network playing Atari Breakout game

(click on the figure bellow)

Supervised Learning - Training Process

Learning as an Error Minimization Problem

1. Random initialisation of weights and biases (learnable parameters)
2. Choose an objective (Loss Function), differentiable with respect to weights and biases
3. Use training data to adjust parameters (gradient descent + back propagation) and minimize the loss
4. Repeat until loss(accuracy) stabilizes or falls below(above) a treshold



Supervised Learning - Objective and Loss Functions

The Loss function quantifies the error between the NN output $\vec{y} = F(\vec{x})$ and the desired target output $\vec{t}$.

Squared Error Loss (Regression)

If we have a target $t \in \mathcal{R}$ and a real NN output y

$$L = \left(y - \vec{t}\right)^2$$

Cross Entropy Loss [4] (Classification)

If we have m classes with binary targets $t \in \{0, 1\}$ and output probabilities y:

$$L = -\sum_{i=1}^{m} t_i log(y_i)$$

Objective Function

The Objective(Cost) function is the mean Loss over a data sample $\{\vec{x}_i\}$

$$\overline{L} = \frac{1}{n}\sum_{i=1}^{n} L(\vec{x}_i)$$

The activation function type used in the output layer is directly related to loss used for the problem !

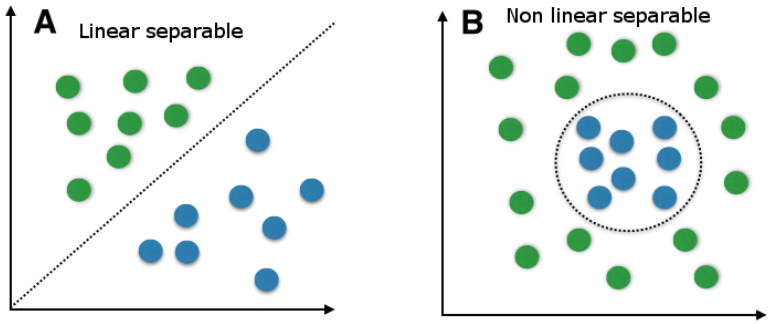
[4] For $m = 2$ we have the Binary Cross Entropy $L = -t \log y - (1 - t) \log (1 - y)$

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 θ 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

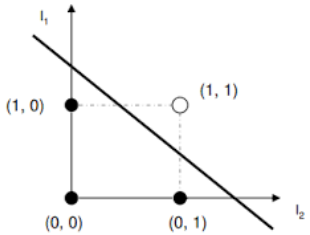
[6] Equation of a plane in $\mathbb{R}^n$ is $\vec{W}.\vec{x} + B = 0$

Perceptron and XOR Problem

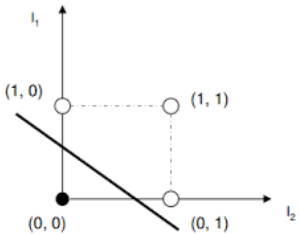
Problem:

Perceptrons are limited to linearly separable problems => unable to learn the *XOR* boolean function [7]

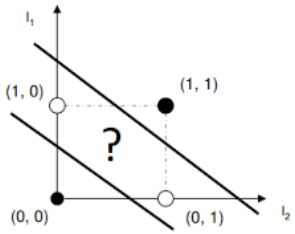
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



Solution:

Need two neurons (layer) to solve the *XOR* problem in two-stages

[7] R.Yehoshua, MLPs Explained and Illustrated - https://medium.com/p/8d76972afa2b

Multilayer Perceptron (MLP)

The Multilayer Perceptron(MLP) is a fully connected NN with at least 1 hidden layer and nonlinear activation function F [8] . It is the simplest feed forward NN. [9]

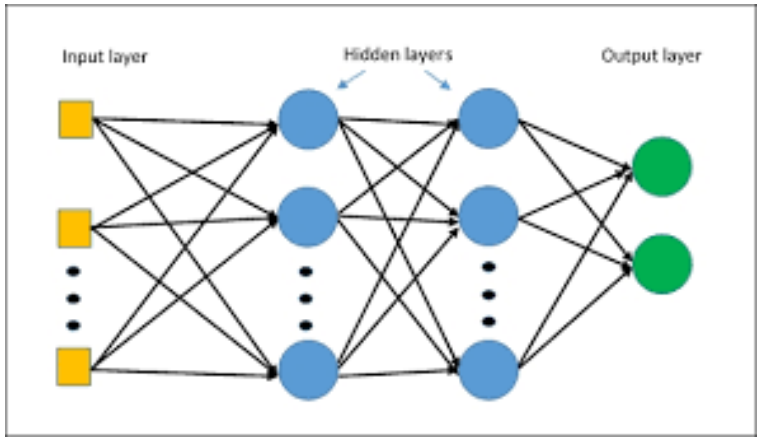
Multilayer Perceptron Model

For a MLP 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 $\vec{H}$ we get

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



A MLP can be seen as a parametrized composite mapping $F_{w,b} : \mathbb{R}^n \to \mathbb{R}^m$

[8] A MLP with m layers using linear activation functions can be reduced to a single layer !

[9] The tresholds $\vec{B}$ are represented as weights by redefining $\vec{W} = (B, W_1, \ldots, W_n)$ and $\vec{x} = (1, x_1, \ldots, x_n)$ (bias is equivalent to a weight on an extra input of activation=1)

Multilayer Perceptron as a Universal Function Approximator

Universal Approximation Theorem

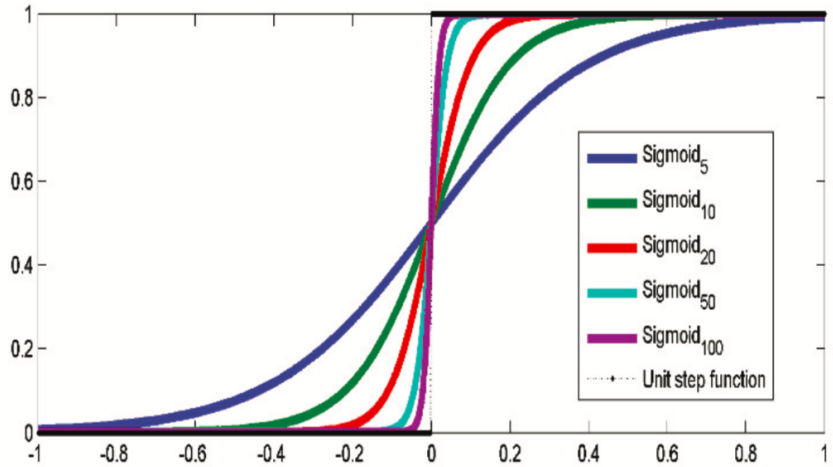
A single hidden layer feed forward neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden neurons [10]

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

$$y = \frac{1}{1 + e^{-(w.x+b)}} \tag{3}$$



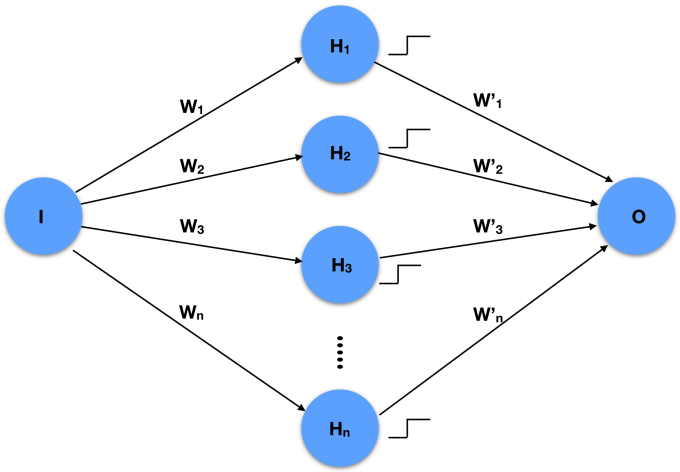
[10] Cybenko,G.(1989) Approximations by superpositions of sigmoidal functions, Math.ofCtrl.,Sig.,andSyst.,2(4),303
Hornik,K.(1991) Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks, 4(2), 251

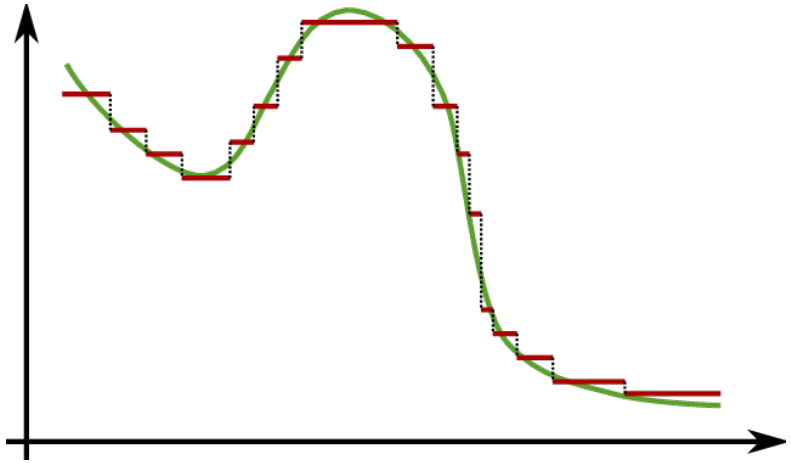
Multilayer Perceptron as a Universal Function Approximator

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 aproximation. [11]

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 !





One can always tune weights such that any activation behaves aproximately as a step function !

[11] M.Nielsen , http://neuralnetworksanddeeplearning.com/chap4.html

Learning as a Minimization Problem (Gradient Descent and Backpropagation)

Gradient Descent Method (Loss Minimization)

The learning process is a loss $L(\vec{W})$ minimization problem, where weights are adjusted to achieve a minimum output error. This minimization is usually implemented by a the Gradient Descent method

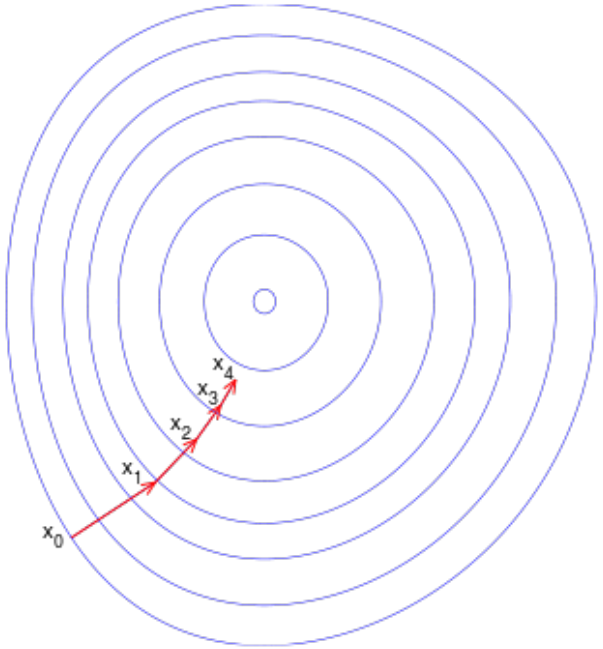
Gradient Descent

A multi-variable function $F(\vec{x})$ decreases fastest in the direction of its negative gradient $-\nabla F(\vec{x})$ [12].

From an initial point $\vec{x}_0$, a recursion relation gives a sequence of points $\{\vec{x}_1, \ldots, \vec{x}_n\}$ leading to a minimum

$$\vec{x}_{n+1} = \vec{x}_n - \lambda \nabla F(\vec{x}_n) \quad \text{, where } \lambda \text{ is the step}$$

The monotonic sequence $F(\vec{x}_0) \geq F(\vec{x}_1) \geq \ldots \geq F(\vec{x}_n)$ converges to a local minimum !

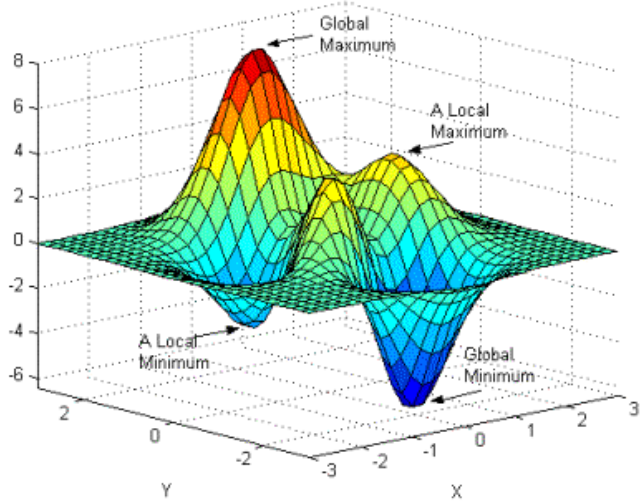


Gradient Descent uses 1^o derivatives, which is efficiently and simply calculated by backpropagation. Newton method is 2^o and involves inverting Hessian matrix, which is computationally costly and memory inefficient.

[12] The directional derivative of $F(\vec{x})$ in the $\vec{u}$ direction is $D_{\vec{u}} = \hat{u}.\nabla F$

Stochastic Gradient Descent (SGD)

NN usually have a non-convex loss function, with a large number of local minima (permutations of neurons of a hidden layer leads to same loss !)



- GD can get stuck in local minima
- Convergence issues
- Should use SGD and adaptive variants

Stochastic Gradient Descent(SGD) [13]

SGD [14] selectes data points randomly, instead of the order they appear in the training set. This allows the algorithm to try a different ”minimization path”at each epoch

- It can also average gradient with respect to a set of events (minibatch)
- Noisy estimates average out and allows “jumping” out of bad critical points
- Scales well with dataset and model size

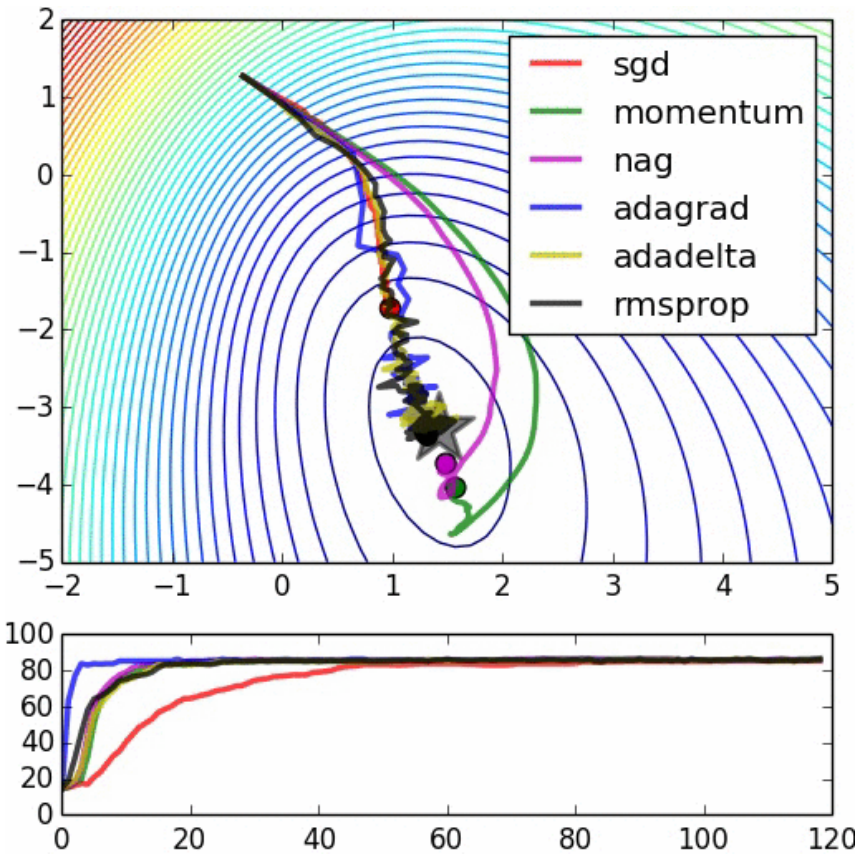
[13] https://www.youtube.com/watch?v=NE88eqLngkg
[14] https://deepnotes.io/sgd-momentum-adaptive

SGD Algorithm Improvements [16]

SGD Variants [15] :

- **Vanilla SGD**
- **Momentum SGD :** uses update Δw of last iteration for next update in linear combination with the gradient
- **Annealing SGD :** step, exponential or 1/t decay
- **Adagrad :** adapts learning rate to updates of parameters depending on importance
- **Adadelta :** robust extension of Adagrad that adapts learning rates based on a moving window of gradient update
- **Rmsprop :** rescale gradient by a running average of its recent magnitude
- **Adam :** rescale gradient averages of both the gradients and the second moments of the gradients

(Click on the figure bellow)



[15] http://danielnouri.org/notes/category/deep-learning
[16] http://ruder.io/optimizing-gradient-descent

Backpropagation

Backpropagation is a technique to apply gradient descent to multilayer networks. An error at the output is propagated backwards through the layers using the chain rule [17]

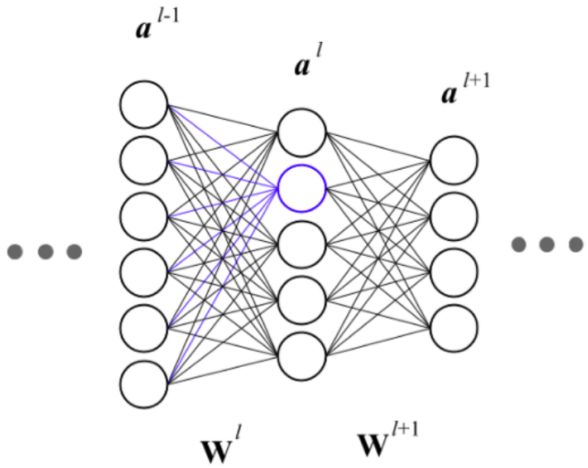
MLP Loss Function

Consider a MLP with n layers (not counting input layer) and a quadratic loss.

Defining the activation $a_i^{(l)} = F(W_{ij}^{(l)} a_j^{(l-1)})$ as the neuron output we have

$$L(\vec{W}) = \frac{1}{2}\left[a_i^{(n)} - t_i\right]^2 = \frac{1}{2}\left[F(W_{ij}^{(n)} \cdots F(W_{rs}^{(1)} a_s^{(0)}) \cdots) - t_i\right]^2$$

The loss is a n-composite function of the weights, where $W_{ij}^{(l)}$ connects the neuron j in layer $(l-1)$, to neuron i in layer l.



[17] http://neuralnetworksanddeeplearning.com/chap2.html

Backpropagation

Let's define $z_i^{(l)} = W_{ij}^{(l)} a_j^{(l-1)}$, such that $a_i^{(l)} = F(z_i^{(l)})$. The loss gradients in l-layer are

$$\frac{\partial L}{\partial W_{kj}^{(l)}} = \left[\frac{\partial L}{\partial z_k^{(l)}}\right] \underbrace{\frac{\partial z_k^{(l)}}{\partial W_{kj}^{(l)}}}_{a_j^{(l-1)}} = \left[\left(\sum_m \frac{\partial L}{\partial z_m^{(l+1)}} \underbrace{\frac{\partial z_m^{(l+1)}}{\partial a_k^{(l)}}}_{W_{mk}^{(l+1)}}\right) \underbrace{\frac{\partial a_k^{(l)}}{\partial z_k^{(l)}}}_{F'}\right] \underbrace{\frac{\partial z_k^{(l)}}{\partial W_{kj}^{(l)}}}_{a_j^{(l-1)}}$$

$$\Rightarrow \frac{\partial L}{\partial W_{kj}^{(l)}} = \left[\left(\sum_m \frac{\partial L}{\partial z_m^{(l+1)}} W_{mk}^{(l+1)}\right) F'(z_k^{(l)})\right] a_k^{(l-1)}$$

Backpropagation Formulas

The backpropagation master formulas for the gradients of the loss function in layer-l are given by

$$\boxed{\frac{\partial L}{\partial W_{kj}^{(l)}} = \delta_k^{(l)} a_k^{(l-1)}} \quad \text{and} \quad \boxed{\delta_k^{(l)} = \left(\sum_m \delta_m^{(l+1)} W_{mk}^{(l+1)}\right) F'(z_k^{(l)})}$$

, where the 'errors' in each layer are defined as $\delta_k^{(l)} = \frac{\partial L}{z_k^{(l)}}$

$\Rightarrow$ The only derivative is F', so the problem is reduced to matrix multiplication !

Backpropagation Algorithm

Given an input we run a *forward pass* to compute activations throughout the network layers and get the output "error". Then *backpropagation* determines each weight and bias corrections.

Backpropagation Algorithm

1. Initialize weights and biases $W_{kj}^{(l)}$ randomly
2. Perform a feedforward pass, computing the arguments $z_k^{(l)}$ and activations $a_k^{(l)}$ for all layers
3. Determine the network output error: $\delta_i^{(n)} = \left[a_i^{(n)} - t_i\right] F'(z_i^{(n)})$
4. Backpropagate the output error: $\delta_k^{(l)} = \left(\sum_m \delta_m^{(l+1)} W_{mk}^{(l+1)}\right) F'(z_k^{(l)})$
5. Compute the loss gradients: $\dfrac{\partial L}{\partial W_{kj}^{(l)}} = \delta_k^{(l)} a_k^{(l-1)}$
6. Correct the weights and biases: $\Delta W_{kj}^{(l)} = -\lambda \dfrac{\partial L}{\partial W_{kj}^{(l)}}$

This algorithm can be applied to any NN architecture

Evaluation of Learning Process

Split dataset into 3 statistically independent samples, one for each learning phase

Training

Train(fit) the NN model by iterating through the whole training dataset (an epoch)

- High learning rate will quickly decay the loss faster but can get stuck or bounce around chaotically
- Low learning rate gives very low convergence speed

Validation

Check performance on independent validation dataset for every epoch

- Evaluate the loss over the validation dataset after each training epoch
- Examine for overtraining(overfitting), and determine when to stop training

Test

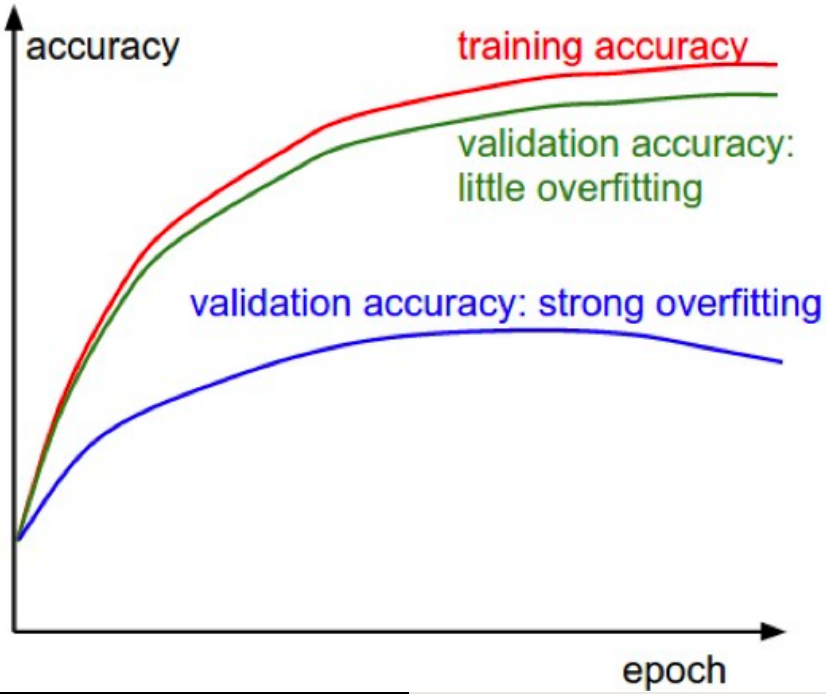
Final performance evaluation after finished training and hyper-parameters are fixed. Use the test dataset for an independent evaluation of performance obtaining a ROC curve

Overtrainging(Overfitting)

Overtraining(Overfitting)

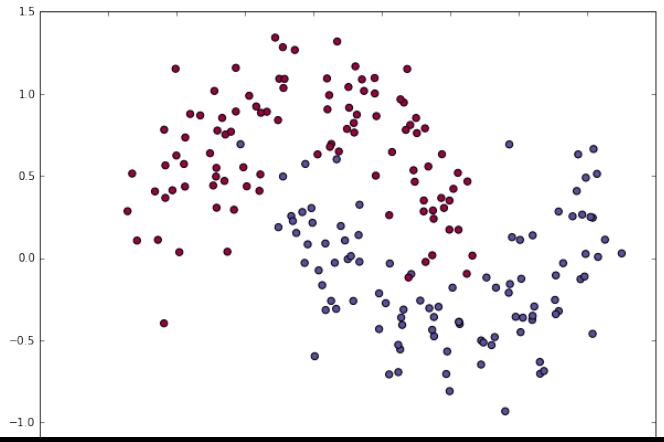
Gap between training and validation accuracy indicates the amount of overfitting

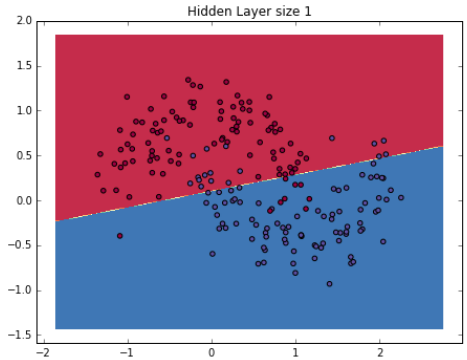
- If validation error curve shows small accuracy compared to training it indicates overfitting $\Rightarrow$ add regularization or use more data or reduce the model size
- If validation accuracy tracks the training accuracy well, the model capacity is not high enough $\Rightarrow$ increase model size

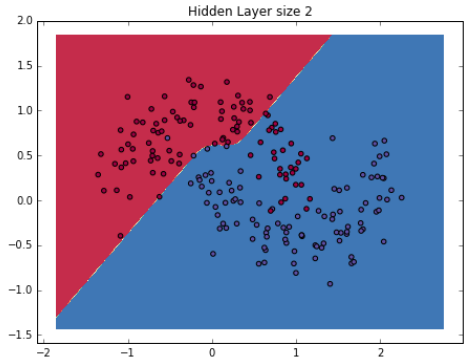


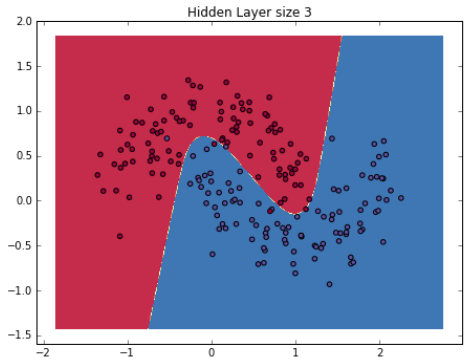
Overfitting - Hidden Layer Size X Decision Boundary [19]

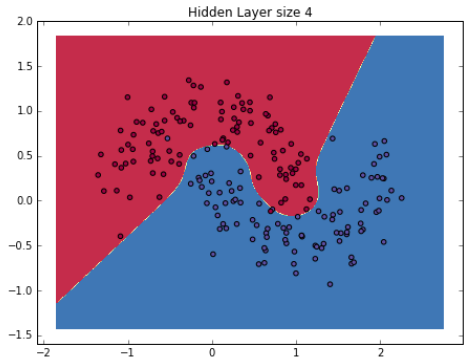
- Hidden layer of low dimensionality nicely captures the general trend of data.
- Higher dimensionalities are prone to overfitting (“memorizing” data) as opposed to fitting the general shape
- If evaluated on independent dataset (and you should !), the smaller hidden layer generalizes better
- Can counteract overfitting with regularization, but picking correct size for hidden layer is much simpler

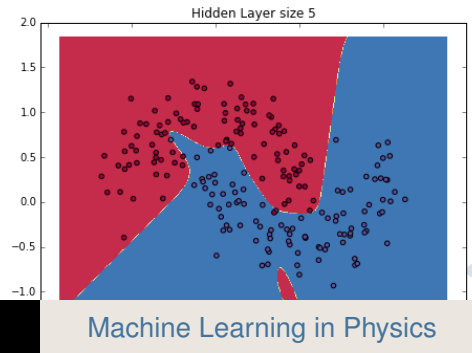


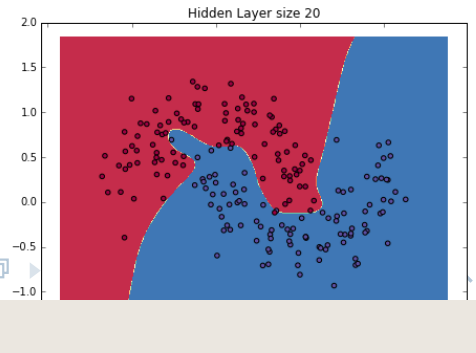












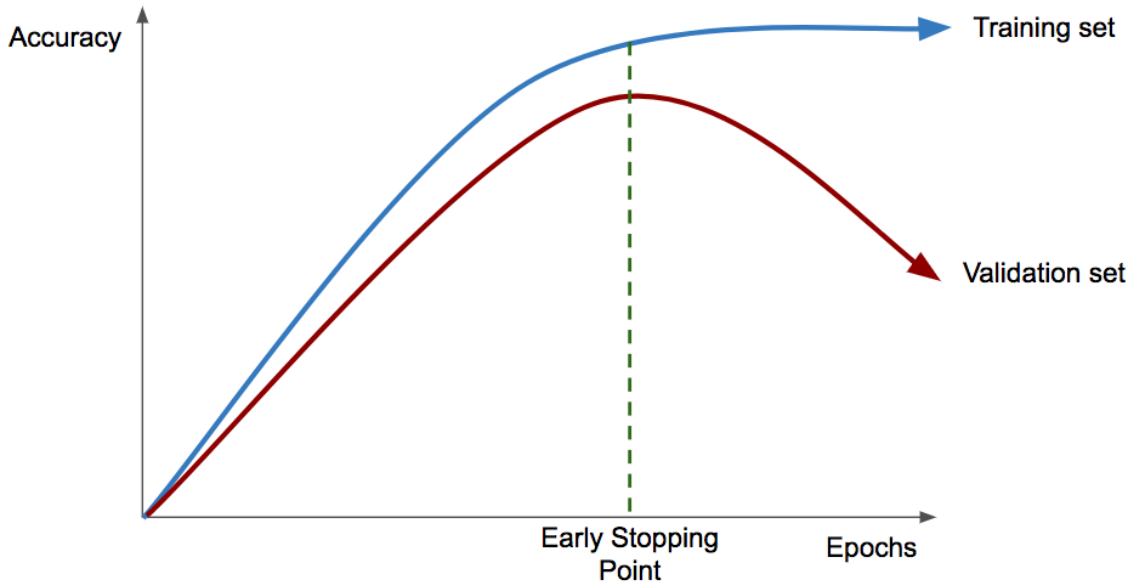
Regularization

Regularization techniques prevents a neural network from overfitting

Early Stopping

Early stopping can be viewed as regularization in time. Gradient descent will tend to learn more and more the dataset complexities as the number of iterations increases.

Early stopping is implemented by training just until performance on the validation set no longer improves or attained a satisfactory level. Improving the model fit to the training data comes at the expense of increased generalization error.

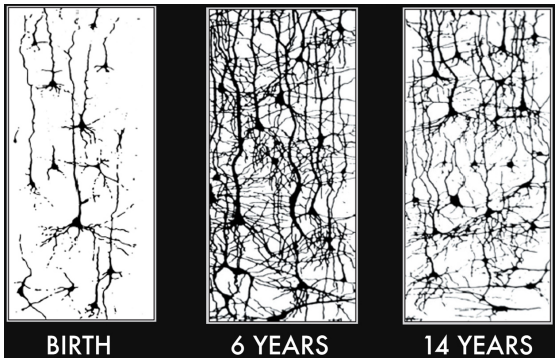


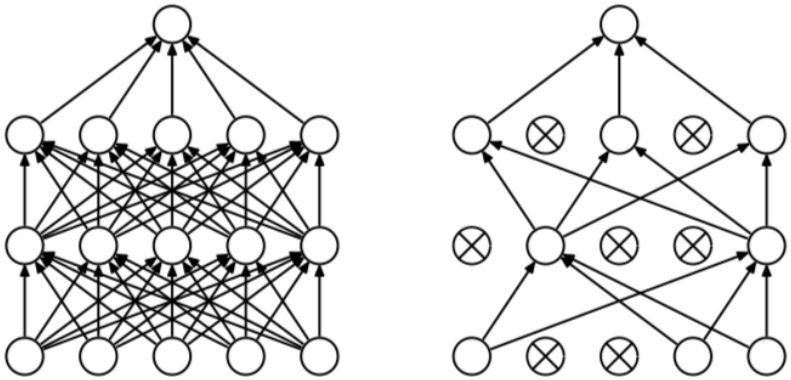
Dropout Regularization

Dropout Regularization

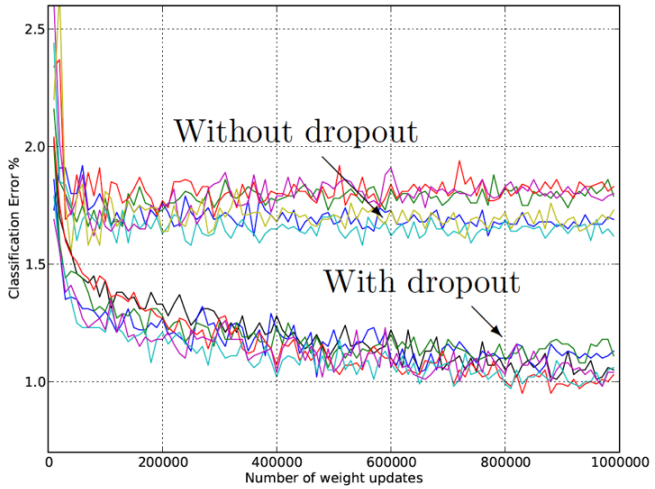
Regularization inside network that remove nodes randomly during training.

- It's ON in training and OFF in validation and testing
- Avoid co-adaptation on training data
- Essentially a large model averaging procedure





(a) Standard Neural Net (b) After applying dropout.



Usually worsens the results during training, but improves validation and testing results !

L1 & L2 Regularization

Between two models with the same predictive power, the 'simpler' one is to be preferred (NN Occam's razor)

L1 and L2 regularizations add a term to the loss function that tames overfitting

$$L'(\vec{w}) = L(\vec{w}) + \alpha\Omega(\vec{w})$$

L1 Regularization

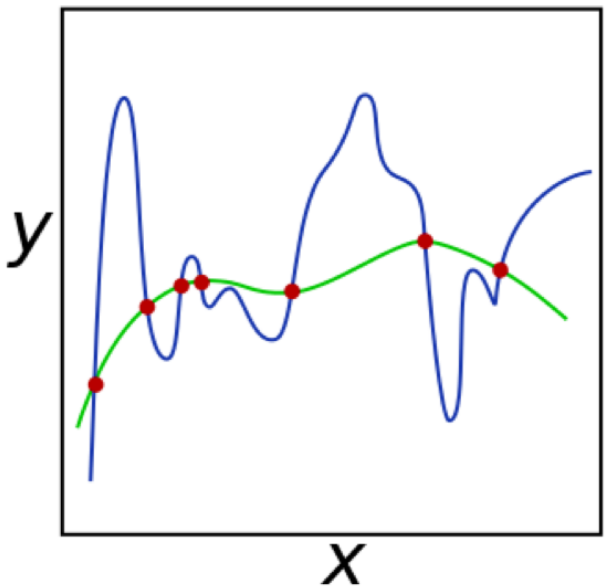
Works by keeping weights sparse

$$\Omega(\vec{w}) = |\vec{w}|$$

L2 Regularization

Works by penalising large weights

$$\Omega(\vec{w}) = |\vec{w}|^2$$

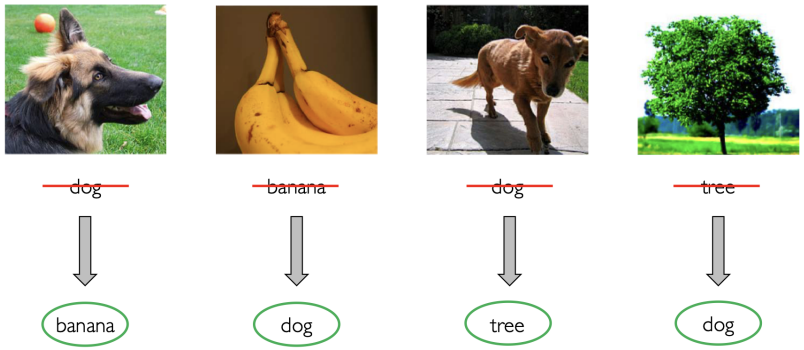


The combined $L1 + L2$ regularization is called Elastic

Neural Network Generalization and Datasets

Rethinking Generalization

"Understanding Deep Neural Networks Requires Rethinking Generalization"



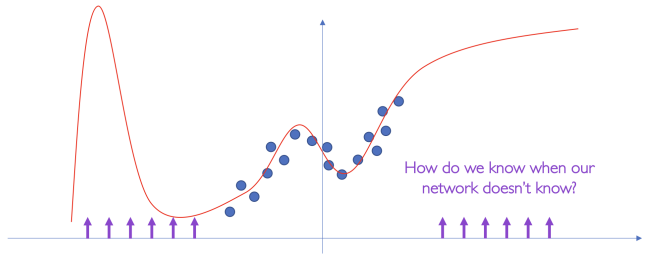
Zhang et al. *ICLR.* (2017)

Capacity of Deep Neural Networks

Neural networks can fit even random data ! [20]

Neural Networks as Function Approximators

Neural networks are **excellent** function approximators
…when they have training data



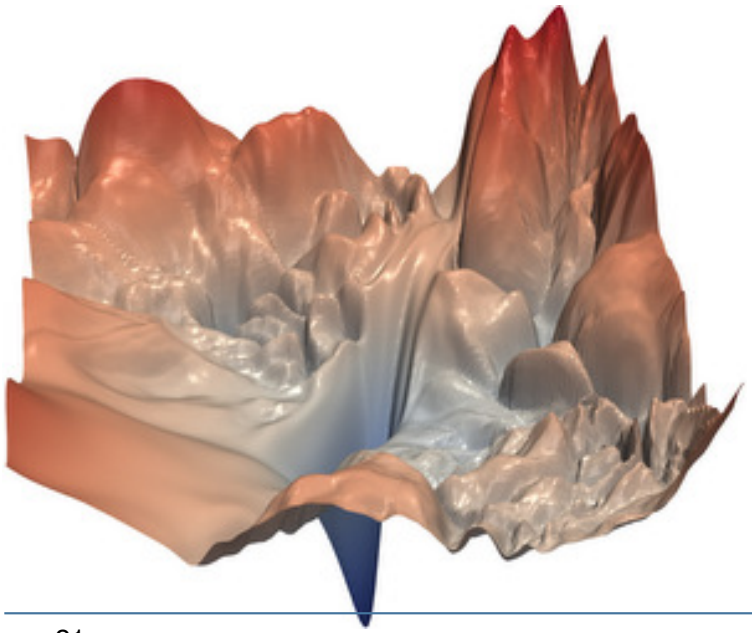
Can't expect a neural network to extrapolate outside training data domain !

[20] A.Amini & all , http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L7.pdf

Loss Landscape and Local Minima

NN Training

Neural network training relies on our ability to find "good" minima of highly non-convex loss functions. [21]



- If you permute the neurons in the hidden layer and corresponding weights the loss doesn't change. Hence if there is a global minimum it can't be unique since each permutation gives another minimum.
- For large networks, most local minima are equivalent and yield similar performance. [22]

[21] Hao Li & All, Visualizing the Loss Landscape of Neural Nets , https://arxiv.org/abs/1712.09913

[22] A.Choromanska, Y.LeCun & All, The Loss Surfaces of Multilayer Networks , https://arxiv.org/abs/1412.0233

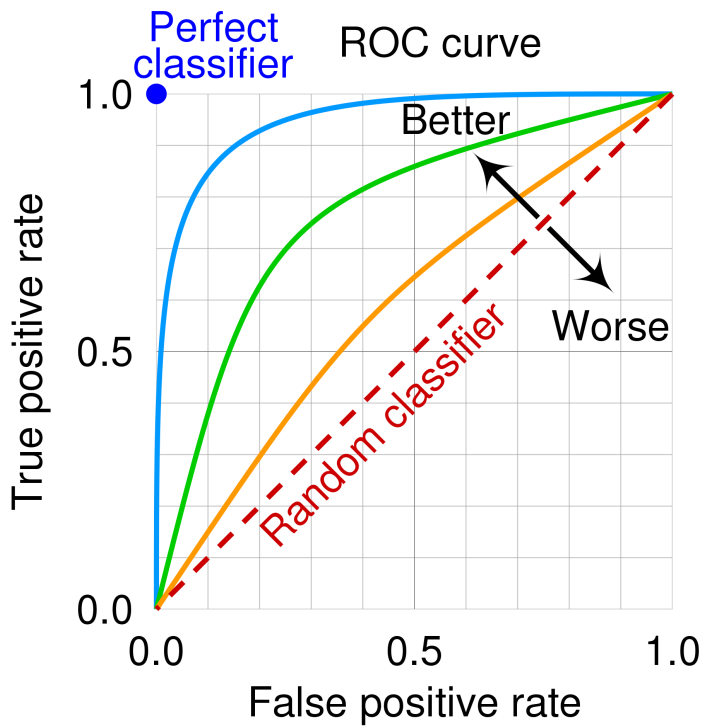
Classifier Performance - Confusion Matrix

		Predicted condition	
	Total population = P + N	**Positive (PP)**	**Negative (PN)**
Actual condition	**Positive (P)**	**True positive (TP),** hit	**False negative (FN),** type II error, miss, underestimation
	Negative (N)	**False positive (FP),** type I error, false alarm, overestimation	**True negative (TN),** correct rejection

Classifier Performance - ROC Curve

ROC Curve

A Receiver Operating Characteristic (ROC) curve illustrates a binary classifier performance



The Area Under the Curve (AUC) measures the classifier performance, but can be misleading with large imbalance between positives(P) and negatives(N) !

Classifier Performance - Figures of Merit

sensitivity, recall, hit rate, or true positive rate (TPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

specificity, selectivity or true negative rate (TNR)

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 1 - \mathrm{FPR}$$

precision or positive predictive value (PPV)

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} = 1 - \mathrm{FDR}$$

negative predictive value (NPV)

$$\mathrm{NPV} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FN}} = 1 - \mathrm{FOR}$$

miss rate or false negative rate (FNR)

Text

$$\mathrm{FNR} = \frac{\mathrm{FN}}{\mathrm{P}} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TP}} = 1 - \mathrm{TPR}$$

fall-out or false positive rate (FPR)

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} = 1 - \mathrm{TNR}$$

false discovery rate (FDR)

$$\mathrm{FDR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TP}} = 1 - \mathrm{PPV}$$

false omission rate (FOR)

$$\mathrm{FOR} = \frac{\mathrm{FN}}{\mathrm{FN} + \mathrm{TN}} = 1 - \mathrm{NPV}$$

Positive likelihood ratio (LR+)

$$\mathrm{LR+} = \frac{\mathrm{TPR}}{\mathrm{FPR}}$$

Negative likelihood ratio (LR-)

$$\mathrm{LR-} = \frac{\mathrm{FNR}}{\mathrm{TNR}}$$

prevalence threshold (PT)

$$\mathrm{PT} = \frac{\sqrt{\mathrm{FPR}}}{\sqrt{\mathrm{TPR}} + \sqrt{\mathrm{FPR}}}$$

threat score (TS) or critical success index (CSI)

$$\mathrm{TS} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN} + \mathrm{FP}}$$

Prevalence

$$\frac{\mathrm{P}}{\mathrm{P} + \mathrm{N}}$$

accuracy (ACC)

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

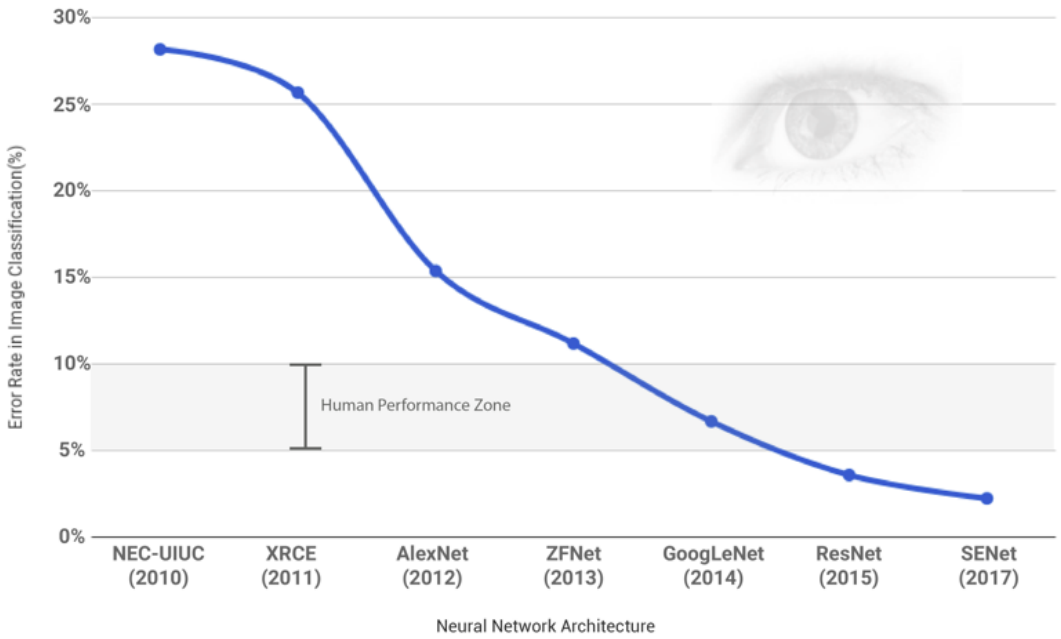
balanced accuracy (BA)

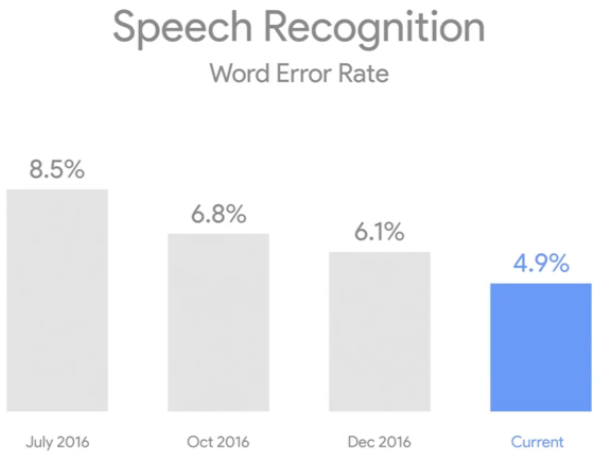
$$\mathrm{BA} = \frac{TPR + TNR}{2}$$

Why Deep Learning ?
and
Why Now ?

Why Deep Learning ?

Image and Speech Recognition perfomance (DNN versus Humans)

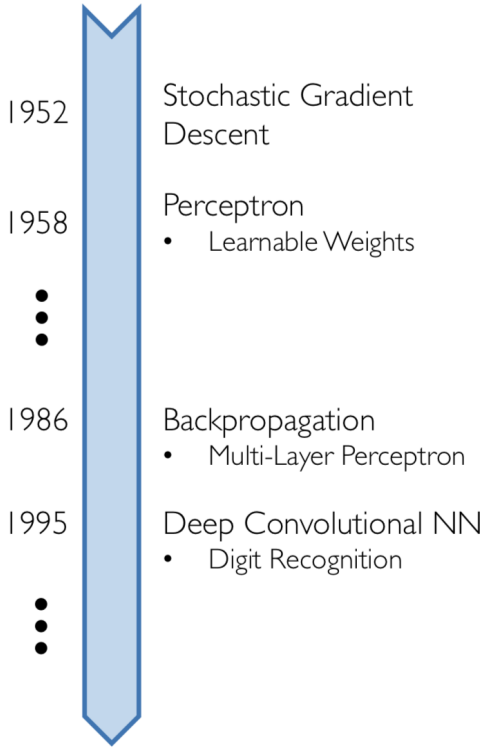




https://arxiv.org/pdf/1409.0575.pdf

Why Now ?

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

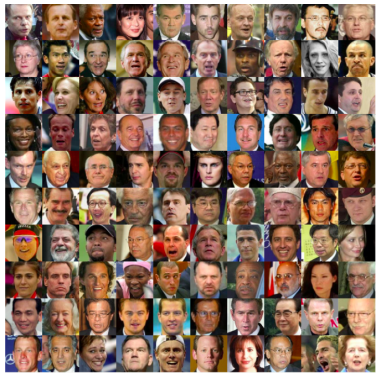


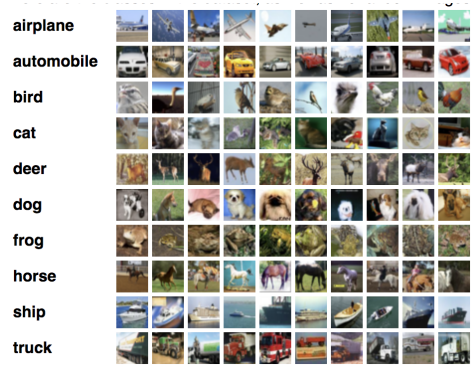
The main catalysts for the current Deep Learning revolution have been:

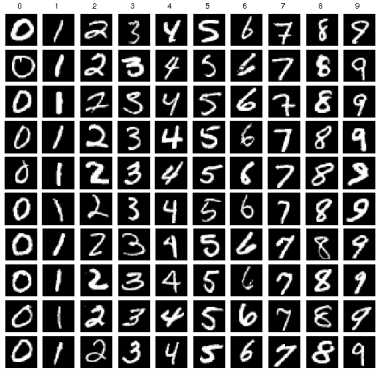
- **Software:** TensorFlow, PyTorch, Keras and Scikit-Learn
- **Hardware:** GPU, TPU and FPGA
- **Large Datasets:** MNIST

Training Datasets

Large and new open source datasets for machine learning research [23]





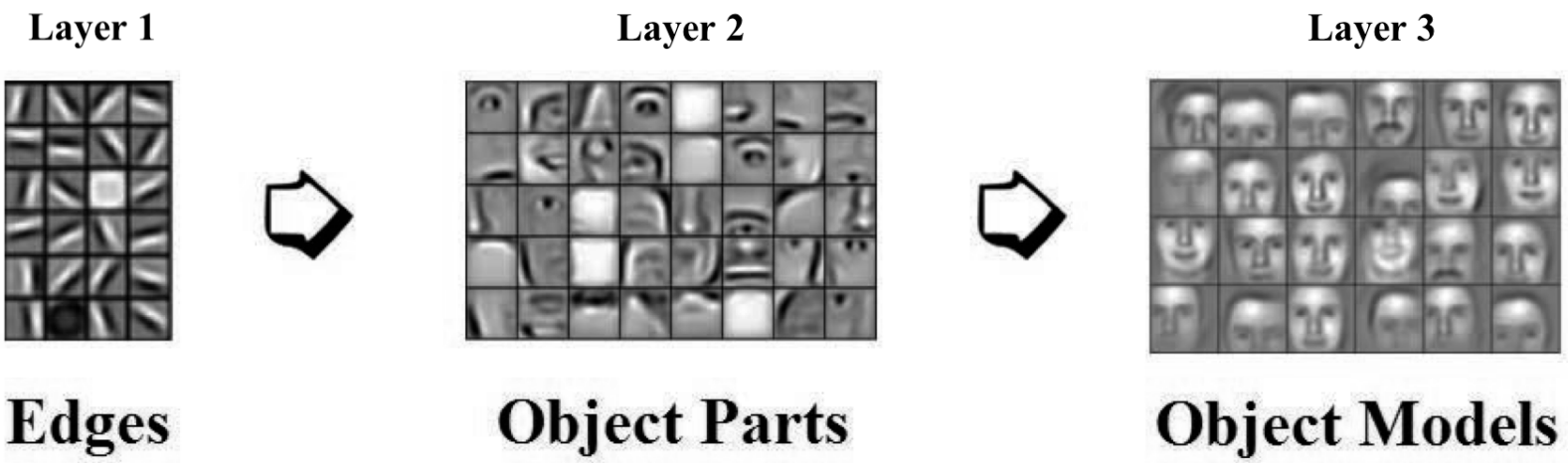


[23] https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research

Deep Learning - Need for Depth

Deep Neural Networks(DNN)

Depth allows the NN to factorize the data features, distributing its representation hierarchically across the layers



$\Rightarrow$ DNN explores the compositional character of nature [24]

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

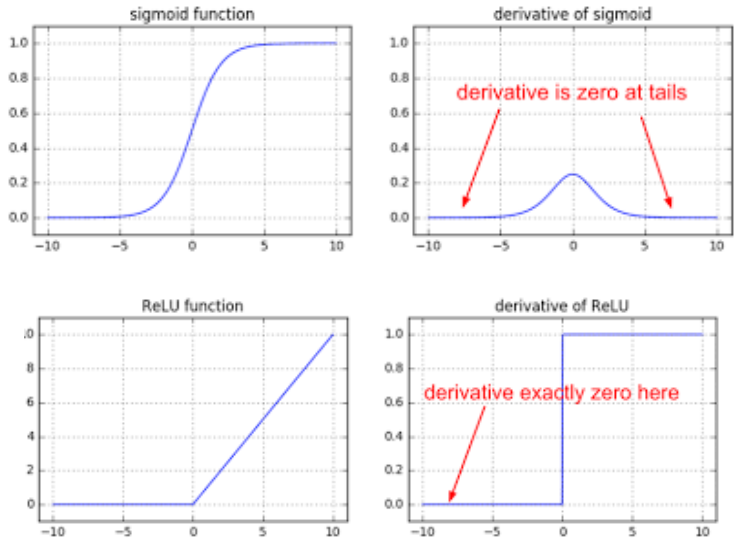
Deep Learning - Vanishing Gradient Problem

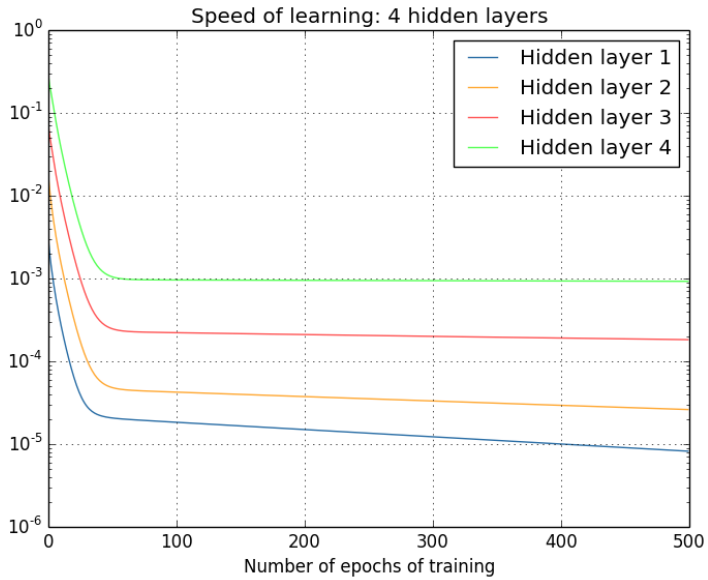
Vanishing Gradient Problem [25]

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 !!!



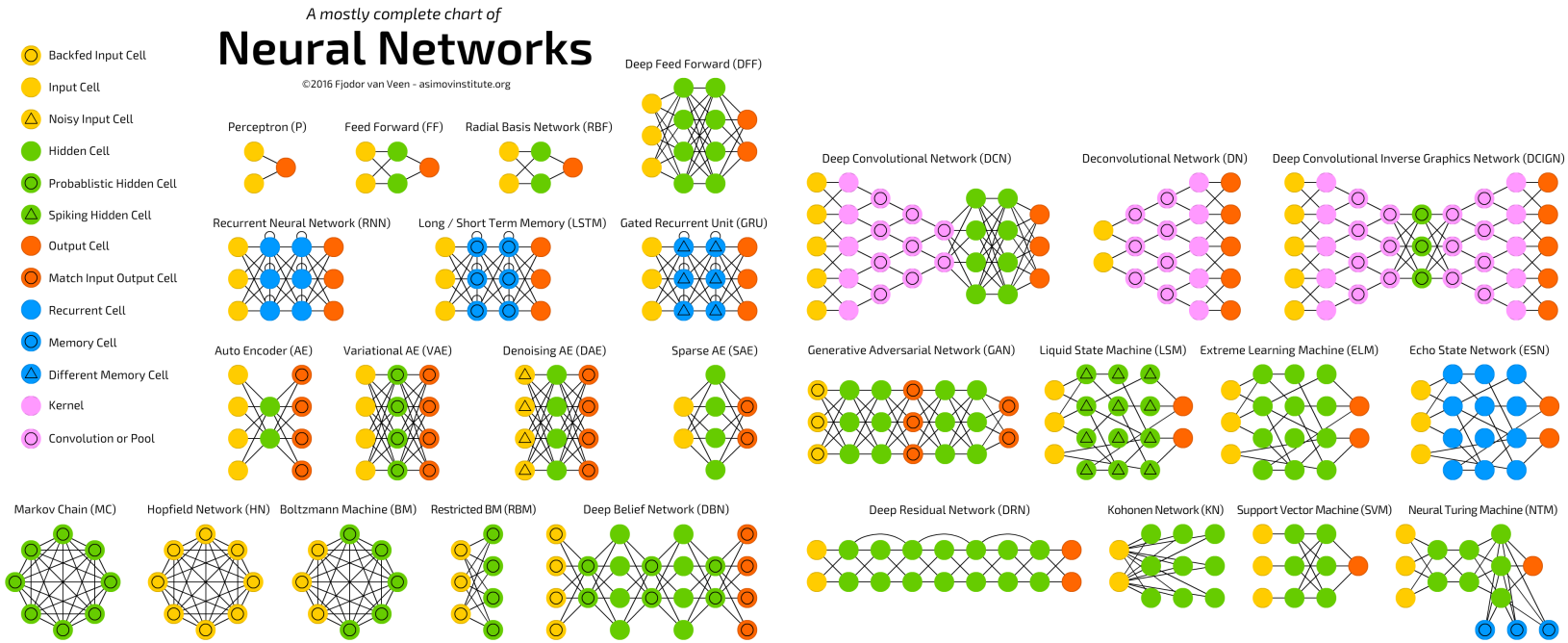


[25] http://neuralnetworksanddeeplearning.com/chap5.html

Deep Architectures and Applications

Neural Network Zoo

NN architecture and nodes connectivity can be adapted for the problem at hand [26]

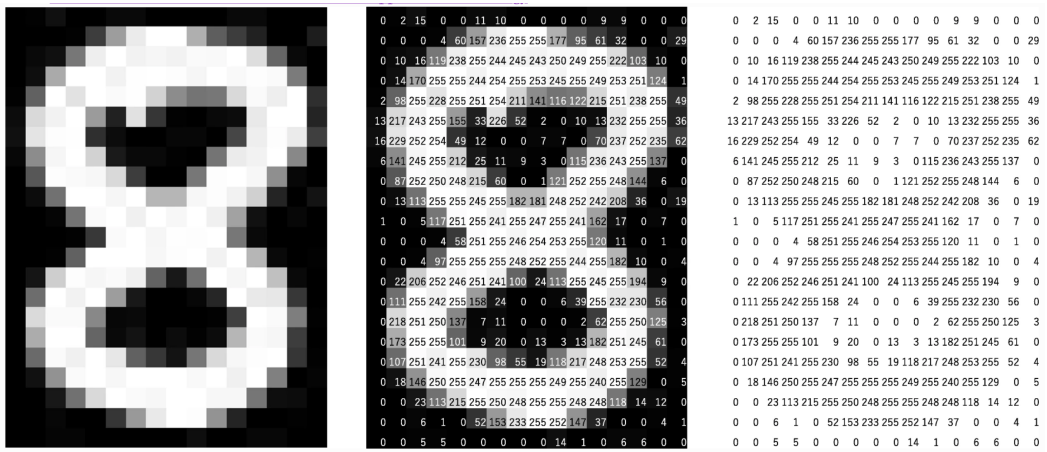


[26] http://www.asimovinstitute.org/neural-network-zoo

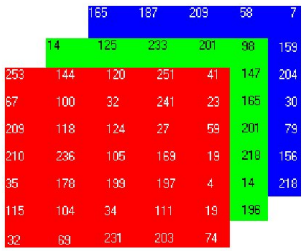
Image Structured Data

Image Representation

A computer sees an image as an array of numbers. The image is associated to a matrix containing numbers between 0 and 255, corresponding to each pixel brightness.



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

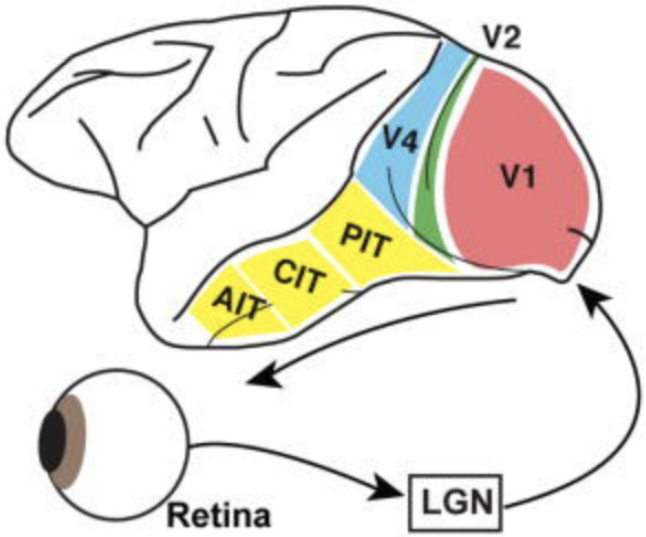


Convolutional Neural Network

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)

Inspired by the visual cortex [27] , where neurons respond to stimuli only in a restricted region of the visual field, a CNN mitigates the challenges of high dimensional inputs by restricting the connections between the input and hidden neurons. It connects only a small contiguous region, exploiting local image features.



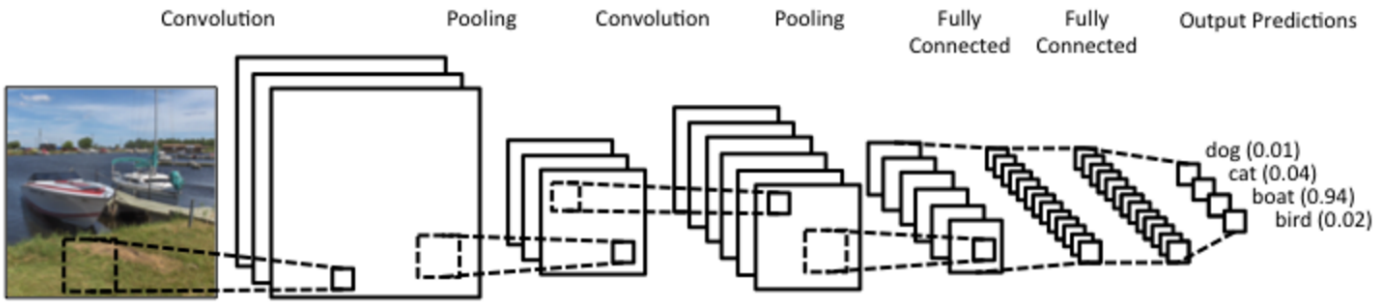
[27] How does the brain solve visual object recognition? , https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3306444

Convolutional Neural Network

Convolutional Neural Network (CNN)

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

1. Convolutional (extract image feature maps) [28]
2. Pooling (downsampling to reduce size)
3. Fully connected (MLP for image classification)



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

CNN - Convolutional Layer

Convolutional Layer

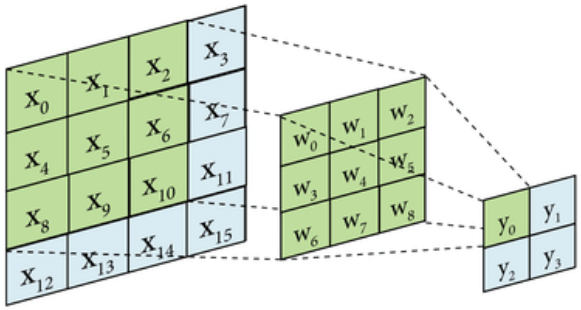
The convolutional layer [29] is the CNN core building block. A convolution can be seen as a sliding window transformation that applies a filter to extract local image features.

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 filter(channel) set of weights are shared all across the image $\Rightarrow$ number of weights reduction

[29] 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) [30]

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

	0	1	0	
	1	-4	1	
	0	1	0	

	-2	-1	0	
	-1	1	1	
	0	1	2	

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

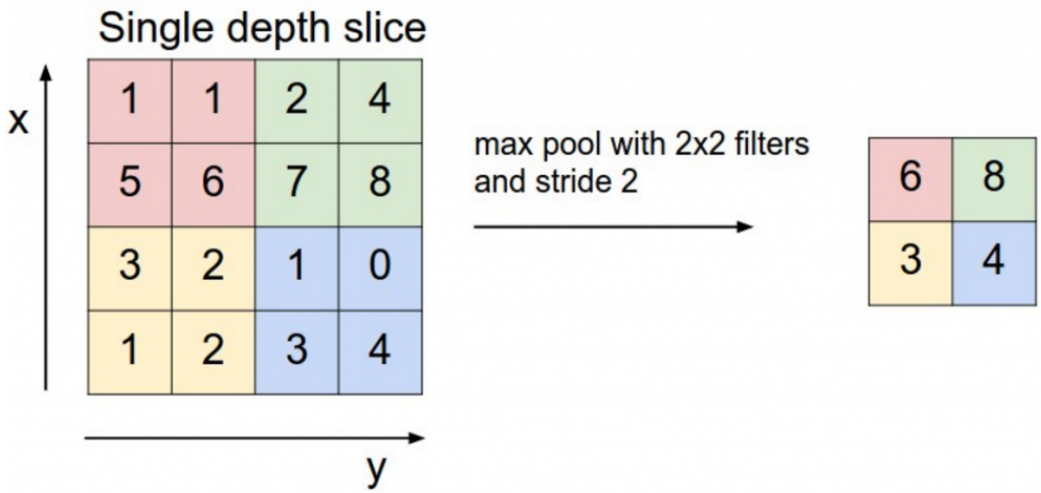
CNN - Pooling Layer

Pooling(Downsampling) Layer

The pooling (downsampling) layer [31] 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)



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

CNN - Fully Connected Layers

Fully Connected Layer

CNN chains together convolution (extract features) , pooling (downsample) and then fully connected (classification) layers.

- After processing with convolutions and pooling, use fully connected layers (MLP) for classification
- Architecture allows capturing local structure in convolutions, and long range structure in later stage convolutions and fully connected layers

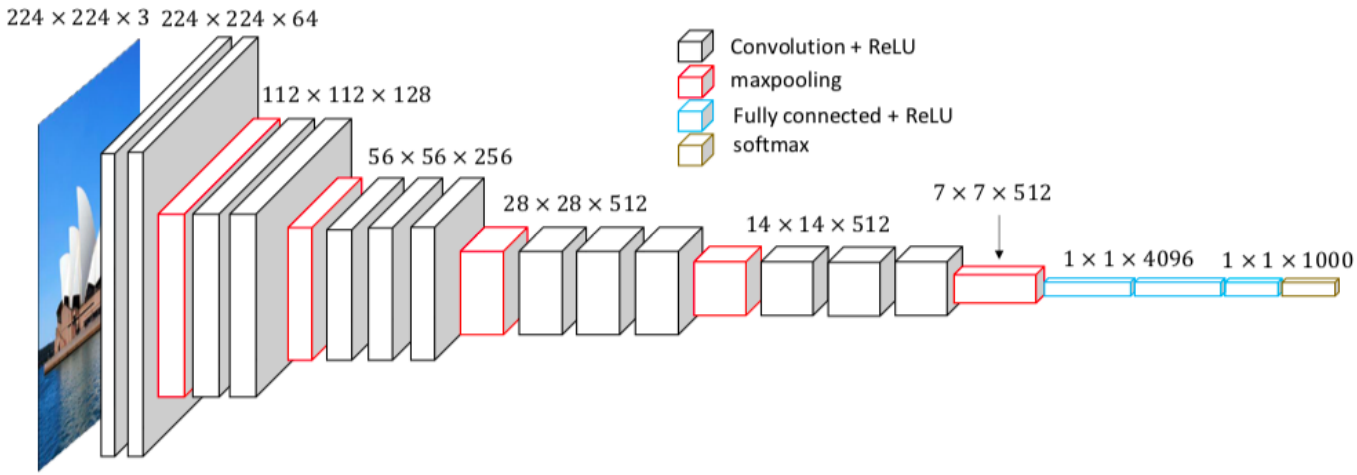
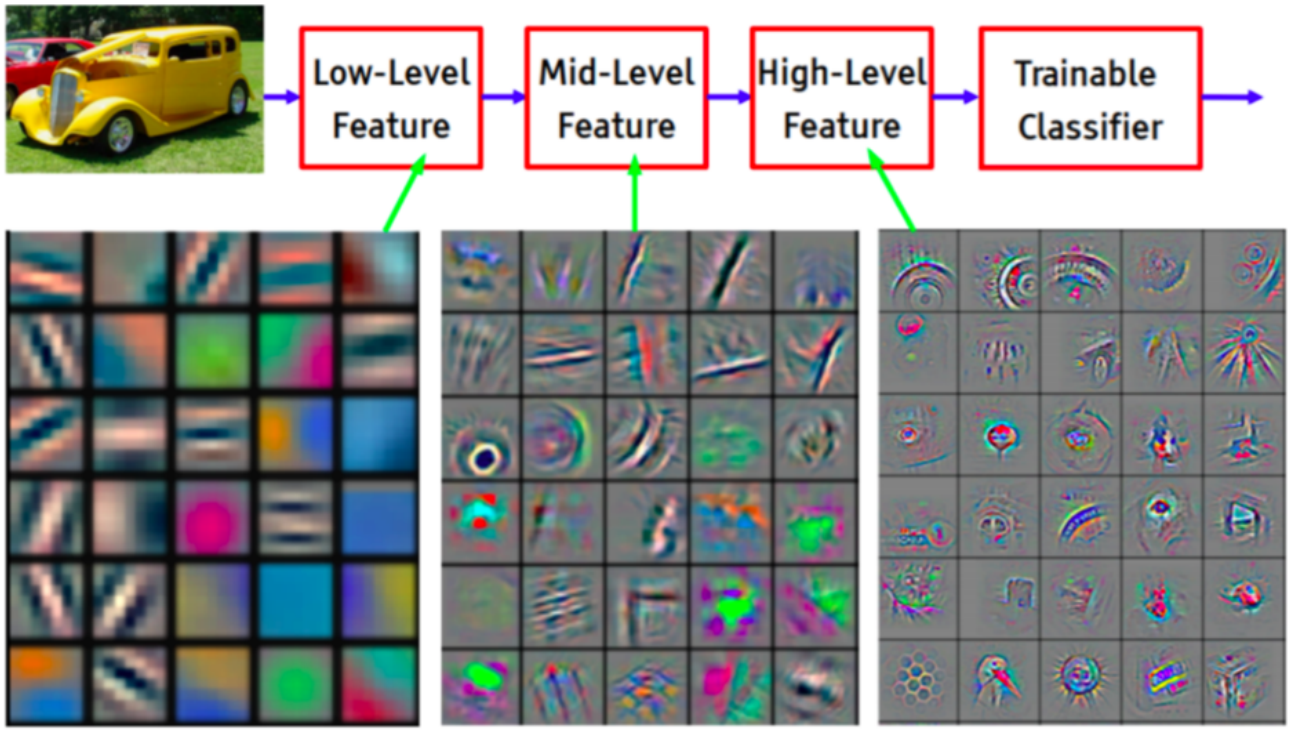


Figure 2: The architecture of VGG16 model .

CNN Feature Visualization

Each CNN layer is responsible for capturing a different level of features as can be seen from ImagiNet [32]

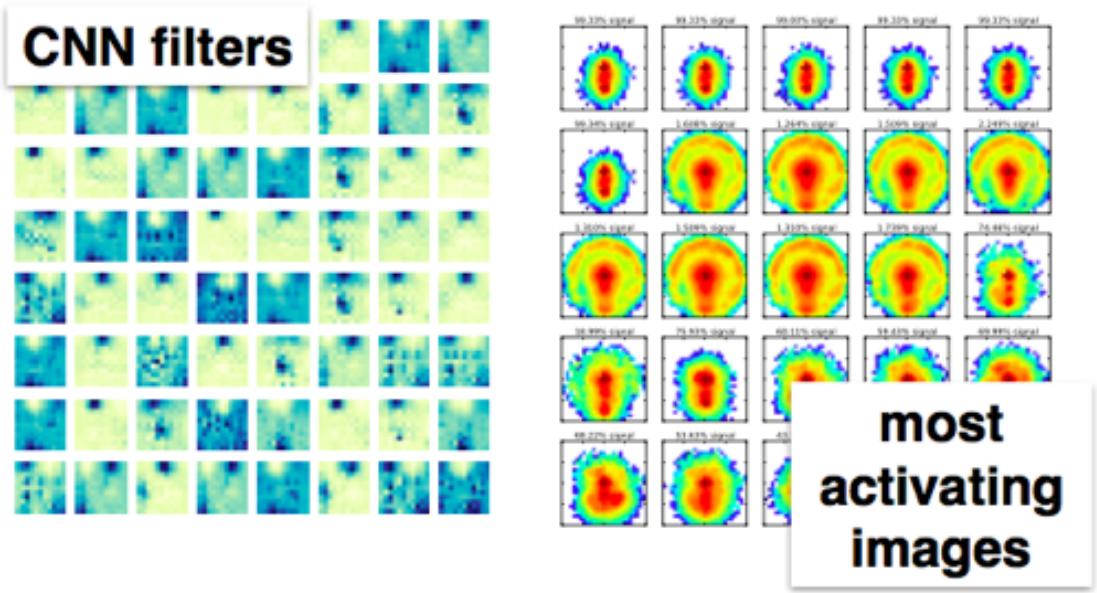


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[32] https://arxiv.org/pdf/1311.2901

CNN Application in HEP: Jet ID

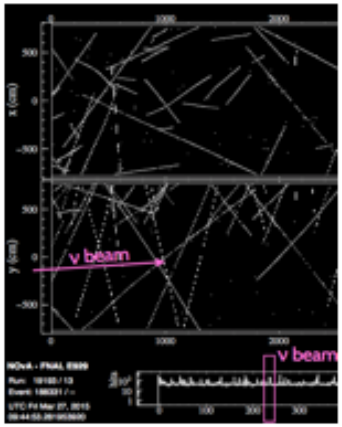
Jet images with convolutional nets

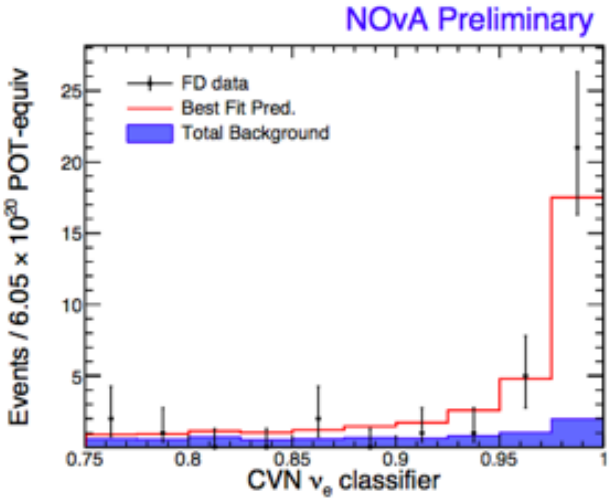


L. de Oliveira et al., 2015

CNN Application HEP: Neutrino ID

Neutrinos with convolutional nets

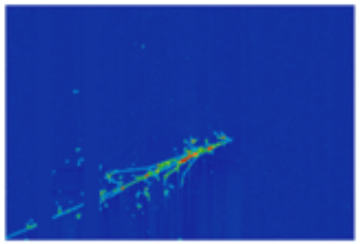


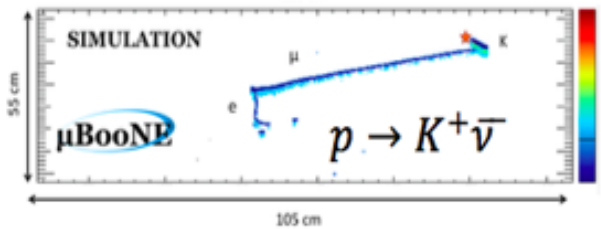


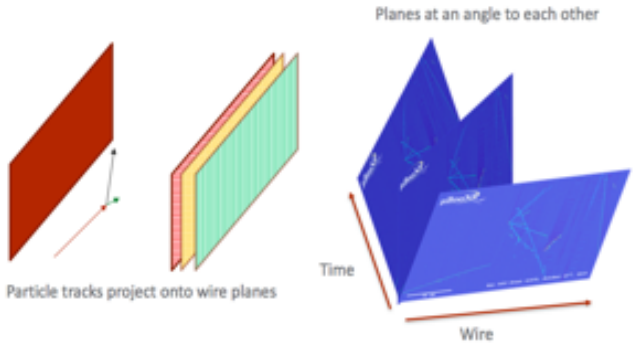
76% Purit
73% Effici

An equivalent increased exposure of 30%

Aurisiano et al. 2016







μBooNE

Sequential Data

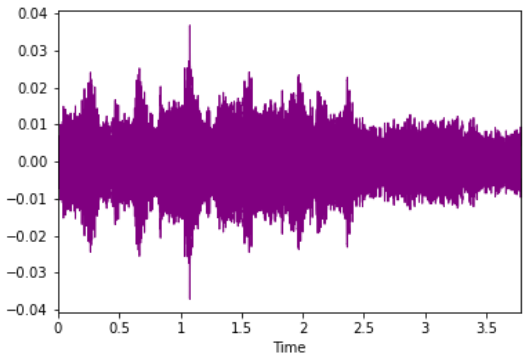
Sequential Data

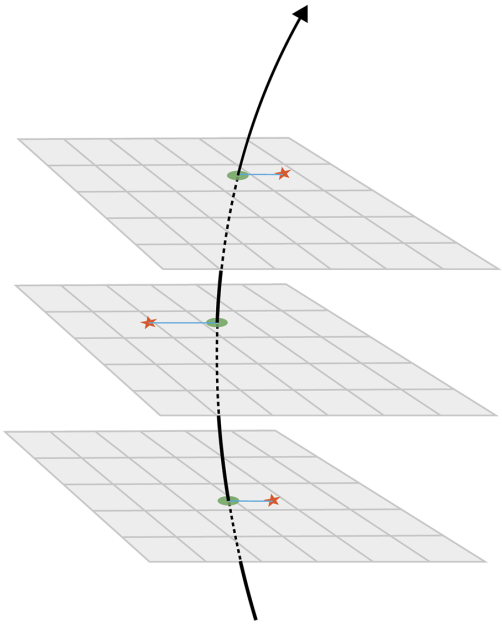
Sequential data is an interdependent data stream where data ordering contains relevant information

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 !

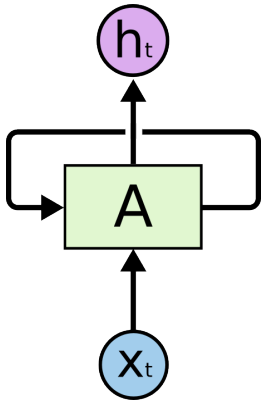
Recurrent Neural Networks (RNN)

RNNs are networks that use feedback loops to process sequential data [33]. These feedbacks allows information to persist, which is an effect often described as **memory**.

RNN Cell (Neuron [34])

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:** $y^{[t]} = W_{hy}h^{[t]}$



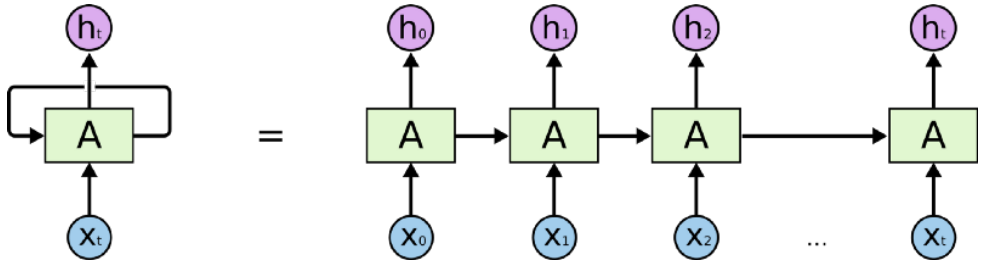
[33] https://eli.thegreenplace.net/2018/understanding-how-to-implement-a-character-based-rnn-
[34] https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks

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. [35]

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)

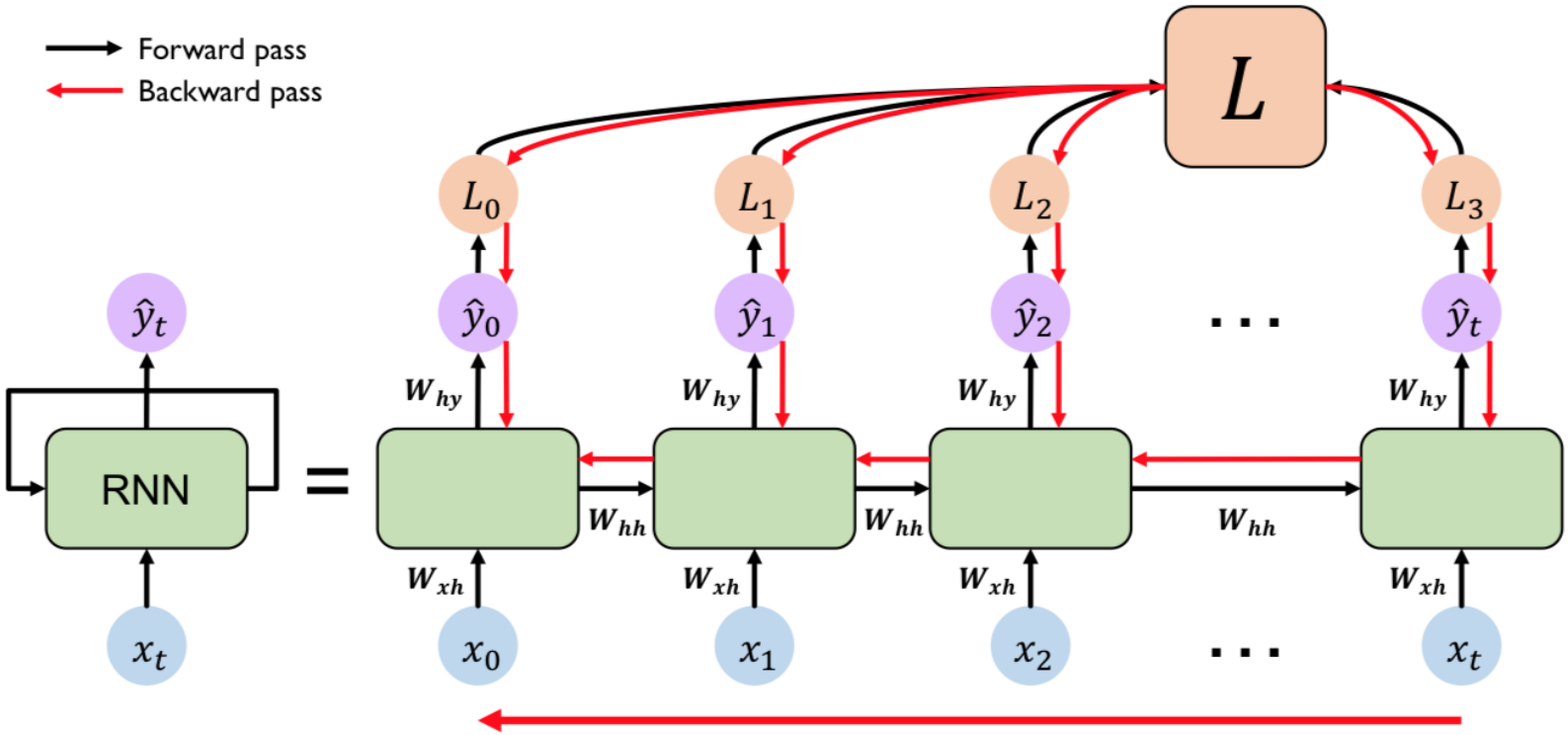
Backpropagation through time is just a fancy buzz word for backpropagation on an unrolled RNN

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

[35] 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



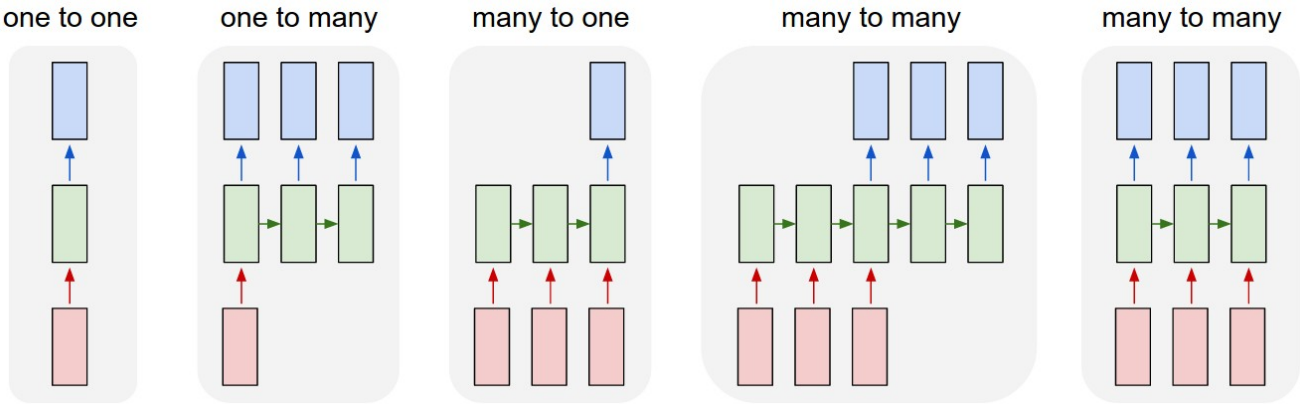
The same set of weights are used to compute the hidden state and output for all time steps (shared weights)

Recurrent Neural Network(RNN)

RNN Properties

- It can take as input variable size data sequences
- RNN allows one to operate over sequences of vectors in the input, the output or both

Bellow, input vectors are in red, output vectors are in blue and green vectors hold the RNN's state [36]



There are no constraints on the sequences lengths because the recurrent transformation can be applied as many times as necessary

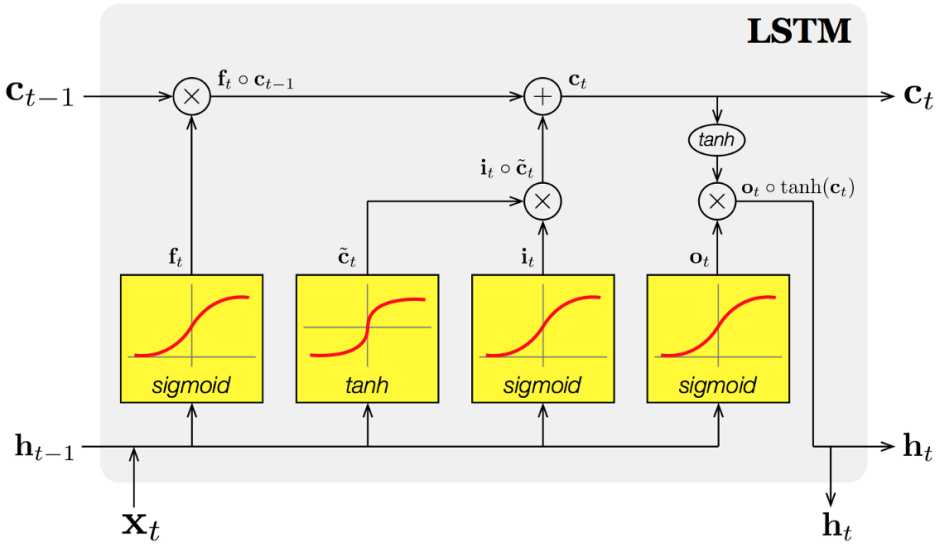
[36] http://karpathy.github.io/2015/05/21/rnn-effectiveness

Long Short Term Memory(LSTM)

Long Short-Term Memory (LSTM) Network

LSTM [37] is a gated RNNs capable of learning long-term dependencies. It categorize data into **short** and **long** term, deciding its importance and what to remember or forget.

The LSTM unit cell has an **input** and a **forget** gate. The **input** defines how much of the newly computed state for the current input is accepted, while the **forget** defines how much of the previous state is accepted.



Gating variables

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_t\right)$$
$$\mathbf{i}_t = \sigma\left(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i\right)$$
$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o\right)$$

Candidate (memory) cell state

$$\tilde{\mathbf{c}}_t = \tanh\left(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c\right)$$

Cell & Hidden state

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t)$$

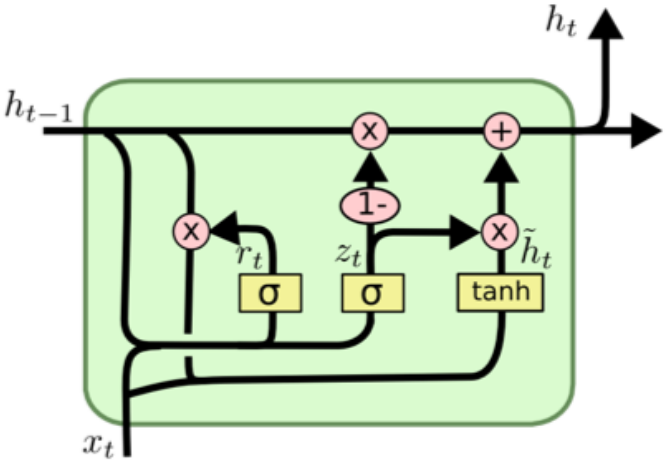
[37] http://colah.github.io/posts/2015-08-Understanding-LSTMs
https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-reme

Gated Recurrent Network (GRU)

Gated Recurrent Network (GRU)

GRU [38] networks have been proposed as a simplified version of LSTM, which also avoids the vanishing gradient problem and is even easier to train.

In a GRU the **reset** gate determines how to combine the new input with the previous memory, and the **update** gate defines how much of the previous memory is keept



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

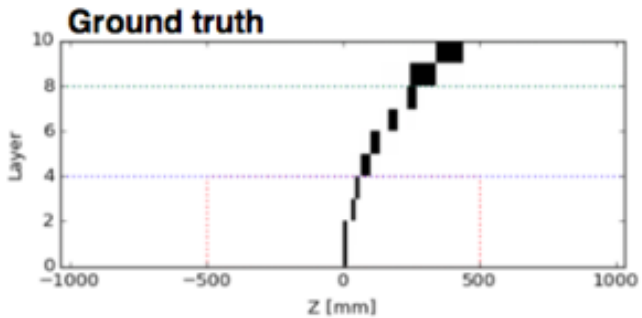
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

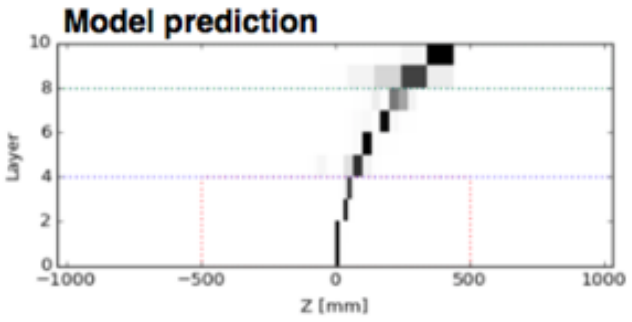
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

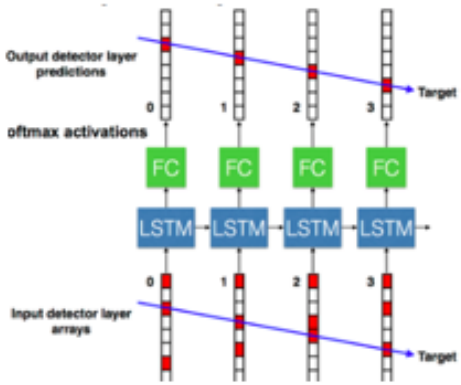
[38] https://isaacchanghau.github.io/post/lstm-gru-formula

LSTM Application in HEP: Tracking

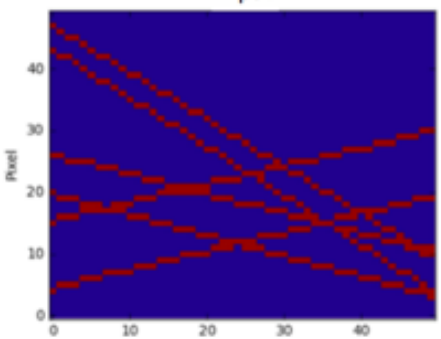
Tracking with recurrent neural networks (LSTM) [39]

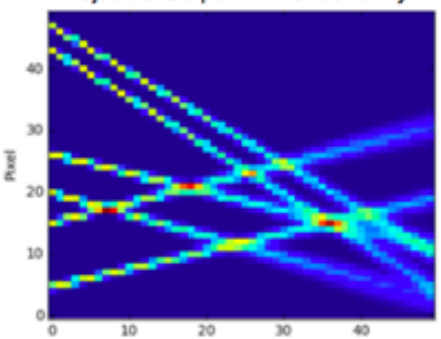






Time dimension (state memory)

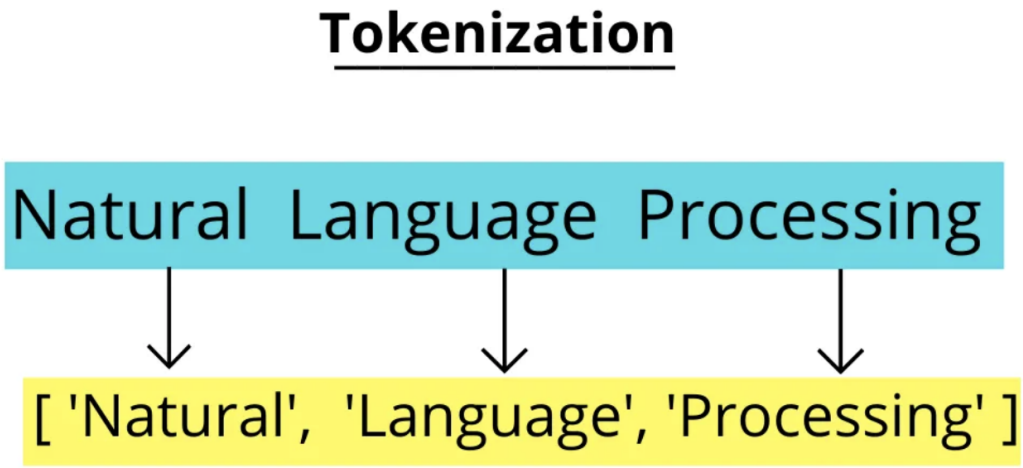




[39] https://heptrkx.github.io

Natural Language Processing(NLP)

To input a text (sequence of words) into a neural network we need to represent the text in a mathematical way. Tokenization is the process of splitting a text into a list of tokens (words, characters, etc.)



- **Bag-of-Words:** represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity
- **Word-2-Vec:** represents each word with a particular list of numbers (vector), chosen to capture the semantic and syntactic qualities of the words

RNN Difficulties

RNN Difficulties

- Large 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. [40]

Attention

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. It also allow us to pursue one thought at a time and remember an individual event rather that all events.

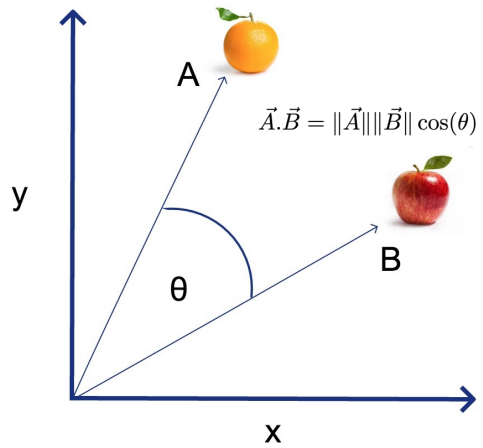
[40] https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Attention Mechanism

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

Attention

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



⇒ Attention can be interpreted as weights attributed to pixels in an image or words in a text, focusing on some part of the data while ignoring others. [41]

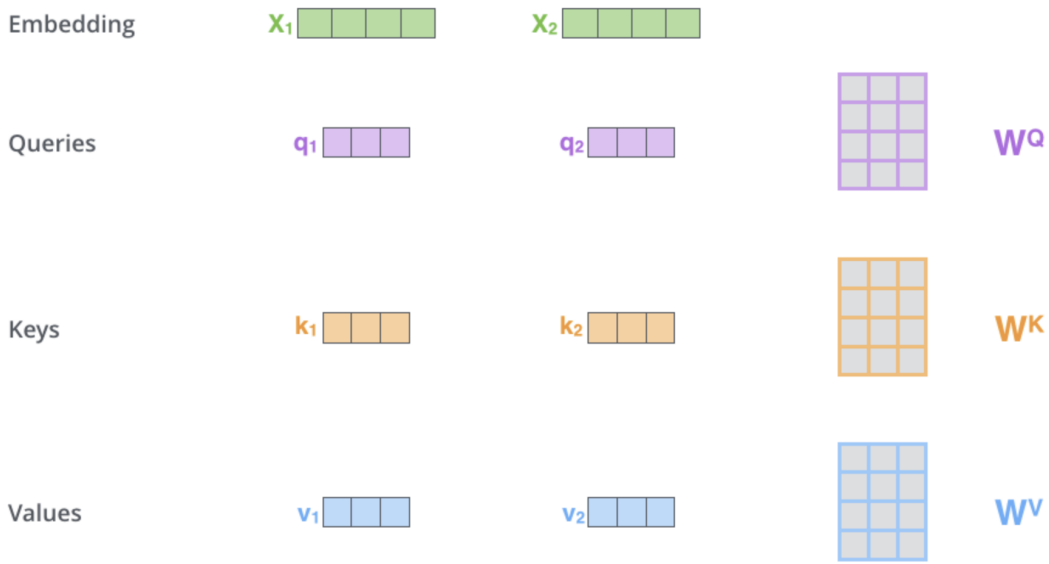
[41] https://distill.pub/2016/augmented-rnns/

Transformers : Attention is All you Need

The Transformer architecture is a sequential data model that uses attention based MLPs outperforming RNNs. [42]

Self-Attention

Self-attention is calculated from three vectors (Query, Key, Value) for each input vector (word embedding). They are defined by multiplying three weight matrices learned during training [43]



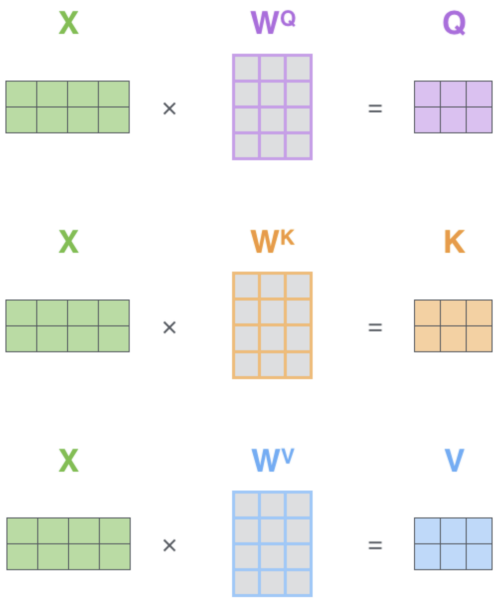
[42] A.Vaswani & all , Attention Is All you Need (2017) , https://arxiv.org/abs/1706.03762

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

Transformer Networks

Matrix Calculation of Self-Attention

The first step is to calculate the Query, Key, and Value matrices. We do that by packing our embeddings into a matrix X, and multiplying it by the weight matrices W^Q, W^K and W^V. The output of the self-attention layer is given by softmax normalized weights

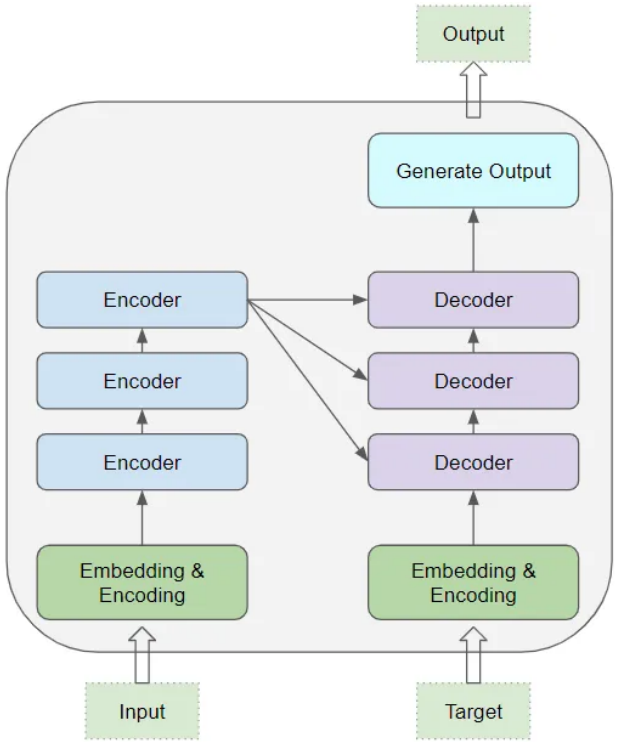


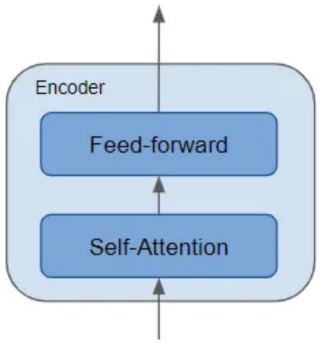
$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

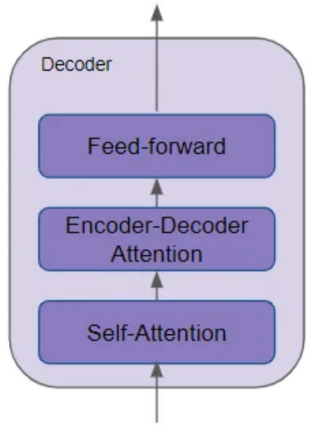
The Transformer uses a “multi-headed” attention where each ”head”has it’s own set of learned weigths (like feature maps in CNNs)

Transformer Networks

The transformer architecture [44] uses self-attention to weight the importance of each part of the input data. Transformers can be applied in tasks of translation, text generation (ChatGPT) and etc







The encoder and decoder have each their own set of weights

[44] https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functi

Transformer Networks

ChatGPT

ChatGPT is a generative pre-trained transformer (GPT) whose core function of a chatbot is to mimic a human conversationalist. ChatGPT is versatile and it can write and debug computer programs, compose music, fairy tales, poetry and student essays and it also answers test questions and play games.

`https://chat.openai.com/chat`

Vision Transformers (ViT)

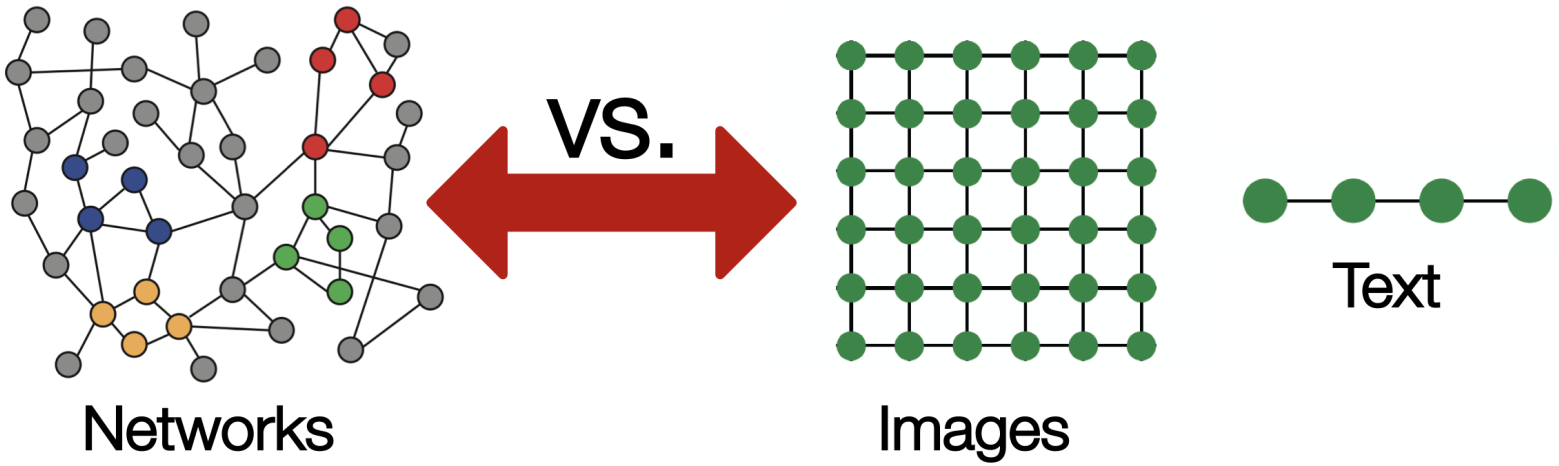
Computing relationships for every pixel pair in a typical image is computationally prohibitive. Vit splits an image into fixed size patches and flattens it to create lower-dimensional embeddings. After including positional embeddings the sequence is input to a Transformer. ViT attains state of the art performance in image classification, segmentation object detection and etc. ViT outperform the current state-of-the-art CNNs in terms of computational efficiency and accuracy.

`https://viso.ai/deep-learning/vision-transformer-vit`

Graph Structured Data

Graphs

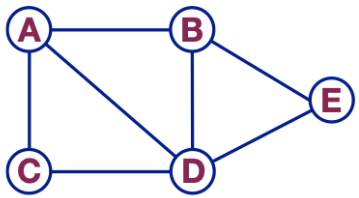
A graph is an abstract mathematical structure defined by its nodes(vertices) and edges(links): $G(N, E)$. The nodes list is unordered and can have variable length, while edges represent nodes relations. Each node and edge can have features (ex: node position, edge lengh, ...).



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. [46]



	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:

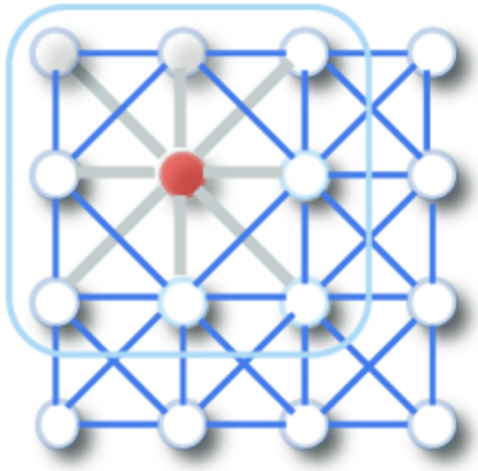
- Adjacency matrix depends on arbitrary node labeling (need permutation invariance)
- Adjacency matrix can became too large and sparse (need compact representattion)
- Matrix size depend on number of nodes (NN takes fixed input size)

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

Graph Neural Network

From Convolution to Message Passing

Inspired by convolution on images, which are grids of localized points, we can define a convolution on a graph where nodes have no spatial order. The convolution on graphs is often referred as a neighborhood **aggregation** or **message passing** [47].

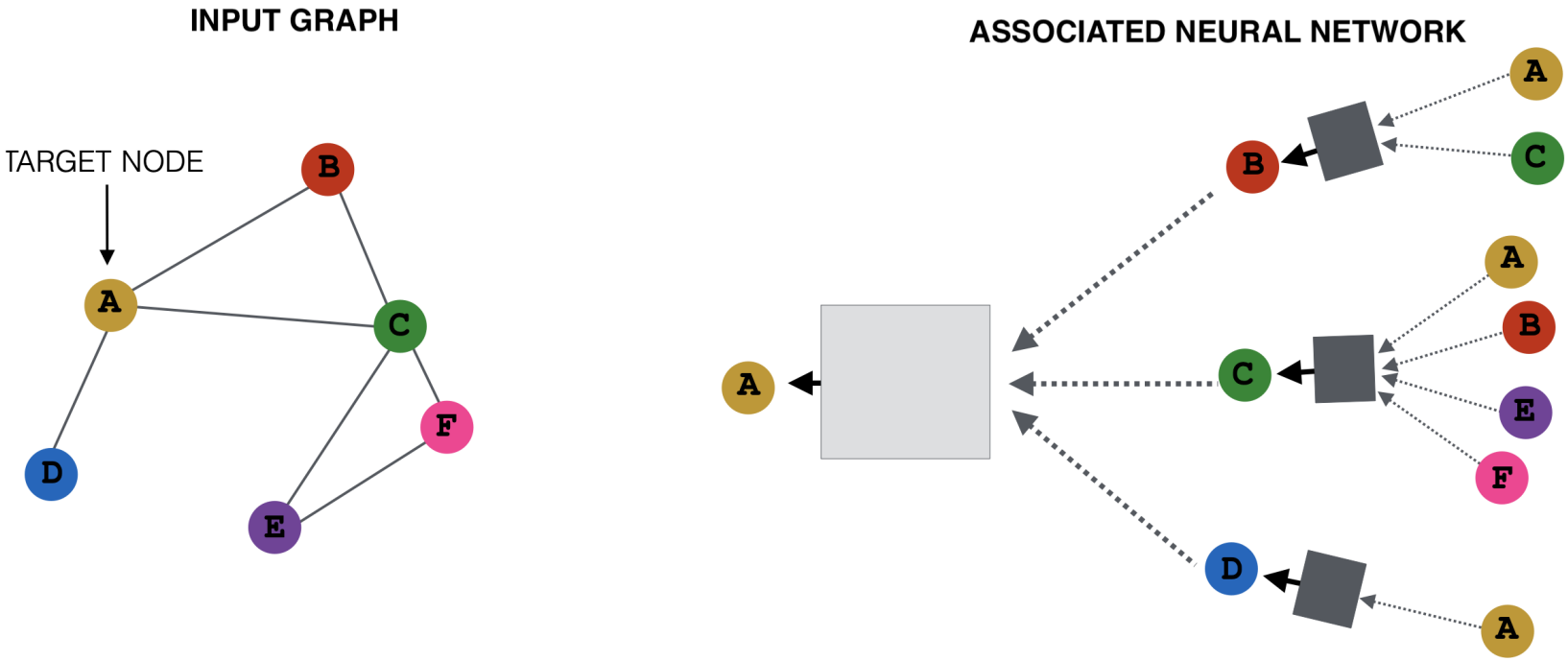


[47] J.Gilmer & all. , Neural Message Passing for Quantum Chemistry , https://arxiv.org/abs/1704.01212

Graph Neural Network

Node Neural Network

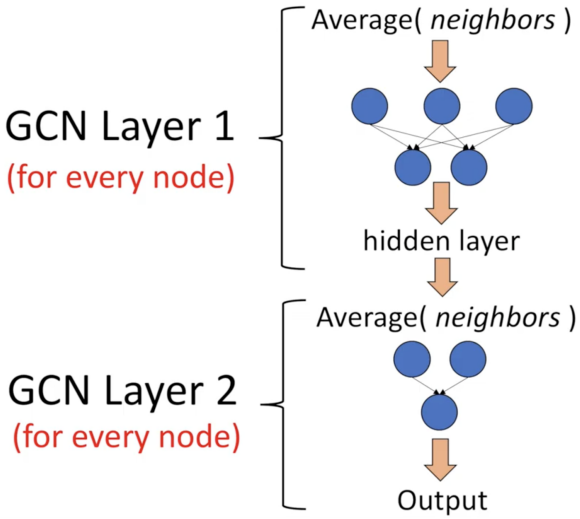
Each node [48] aggregates information from its neighbours defining its own neural network (MLP) . This aggregation function must be permutation invariant !



[48] A Comprehensive Survey on Graph Neural Networks https://arxiv.org/pdf/1901.00596.pdf

Graph Neural Network

Taking the node features as inputs the node network propagates [49] it through the hidden layers getting a latent representation (embedding) of the node features



Node Neural Network

$$h_v^0 = x_v \quad \text{(input features)}$$

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

$$z_v = h_v^n \quad \text{(output)}$$

, where h_v^k is the node embedding in layer-k

Obs: Message passing aggregation can be implemented by any permutation invariant function (ex: sum, average, ...)

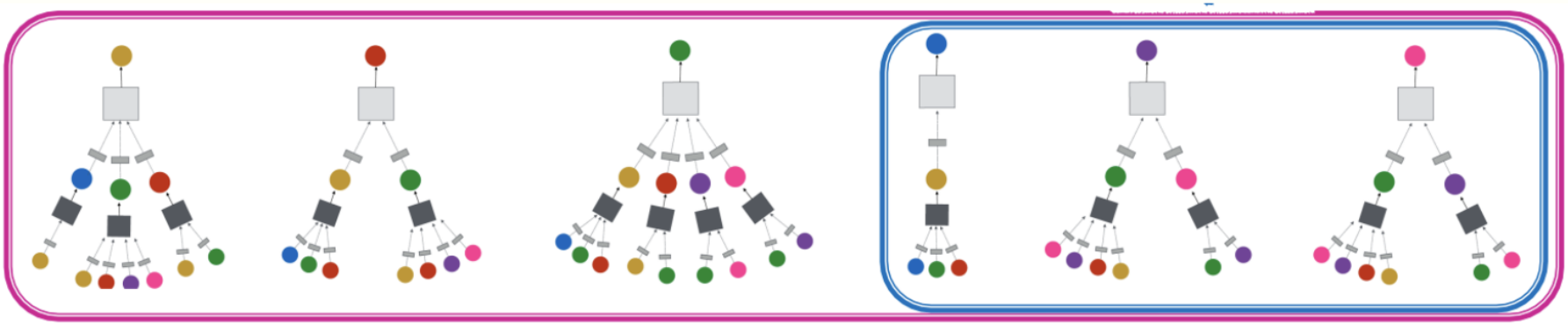
[49] https://towardsdatascience.com/hands-on-graph-neural-networks-with-pytorch-pytorch-geome
http://web.stanford.edu/class/cs224w/slides/08-GNN.pdf

Graph Neural Network

Inductive Capability

GNN can be applied to graphs with arbitrary number of nodes, since weights W_k are shared across all nodes of a given layer-k.

For an example , a model trained on graphs with nodes A,B,C , can also evaluate graphs with nodes D,E,F.



Transformer as a GNN

1) Transformers are GNNs :
https://towardsdatascience.com/transformers-are-graph-neural-networks-bca9f75412aa

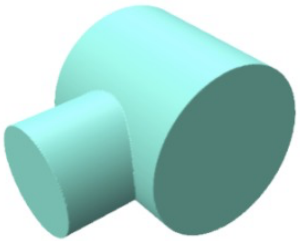
5) C.Joshi - Transformers are Graph Neural Networks:
https://towardsdatascience.com/transformers-are-graph-neural-networks-bca9f75412aa

From Graphs to Sets

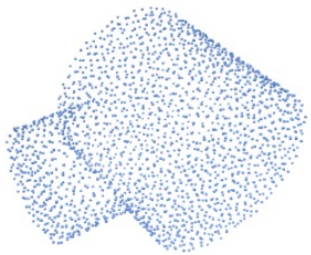
Sets

Abstracting even more on graphs we encounter sets , which are collections without intrinsic order or relations [52]

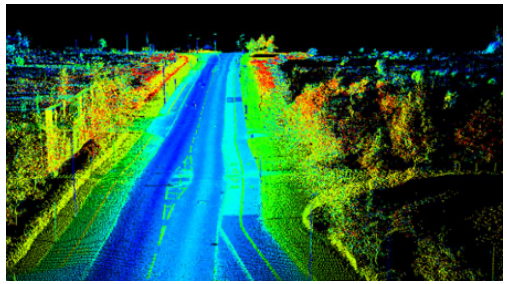
A MLP could in principle learn not to care about orderings, but training it for that would be a waste of computing resources. We can build a permutation invariant network by design where this symmetry is used as an inductive bias built into the network. DeepSets and PointNet are examples of networks capable of learning on set structured data.



(a) Surface view.



(b) Point cloud.



[52] 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 the input components. To construct a permutation invariant function we sum over all permutations $\pi(\vec{x})$ of $\vec{x}$. [53]

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 e PointNet [54]

(a) Janossy pooling with $k = 1$ (*Deep Sets*)

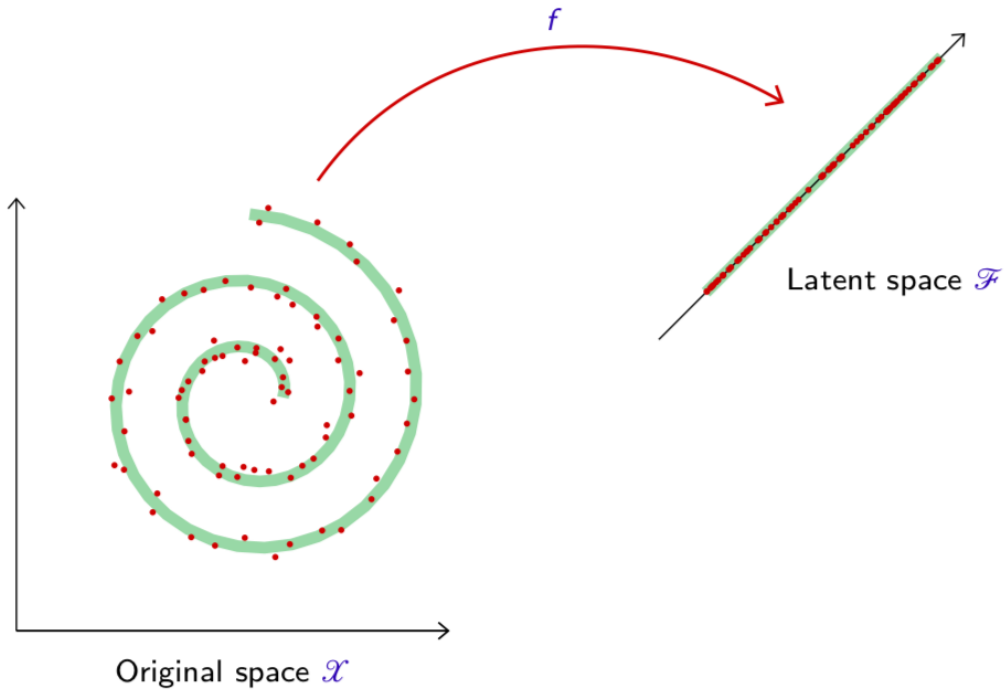
(b) Janossy pooling with $k = 2$

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

[54] F.Fuchs & all, Universal Aproximation of Functions on Sets , https://arxiv.org/abs/2107.01959

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



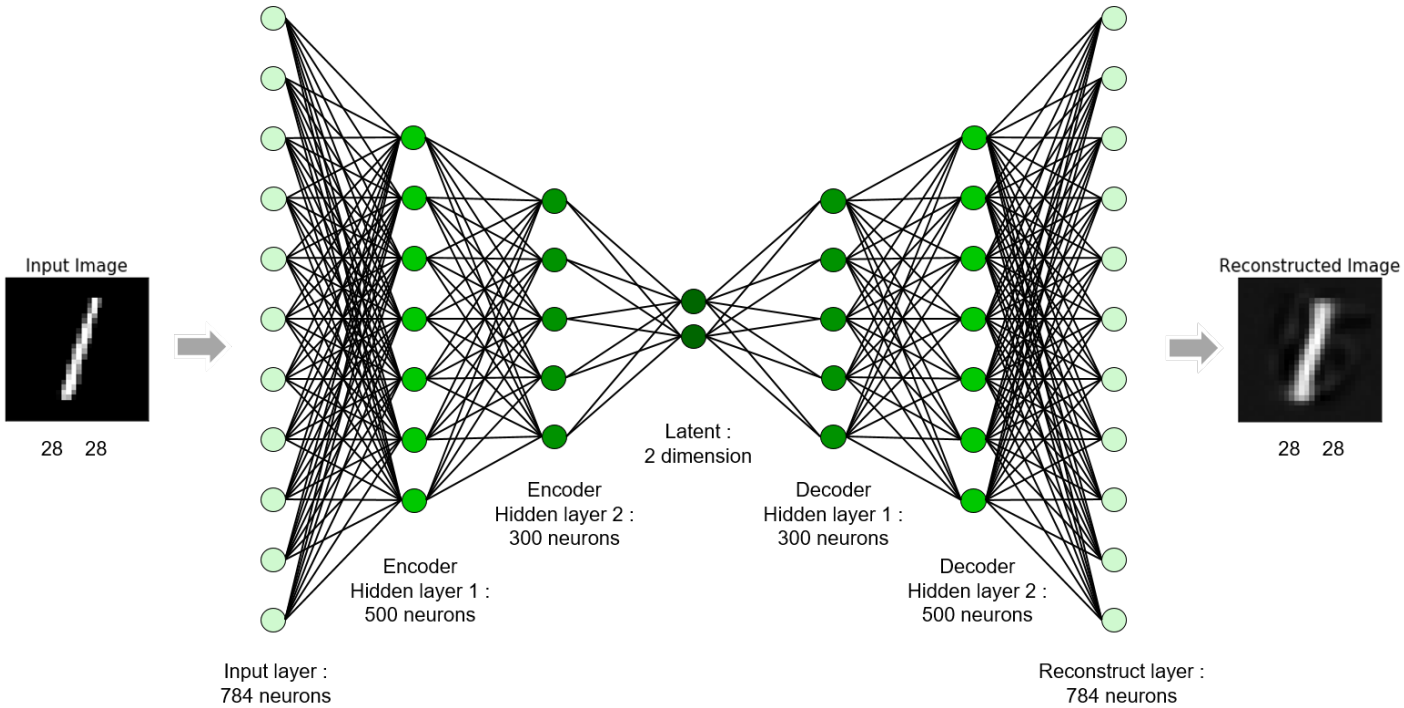
Traditionally, autoencoders were used for dimensionality reduction and denoising. Recently autoencoders are being used also in generative modeling

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 $y = g(l)$.



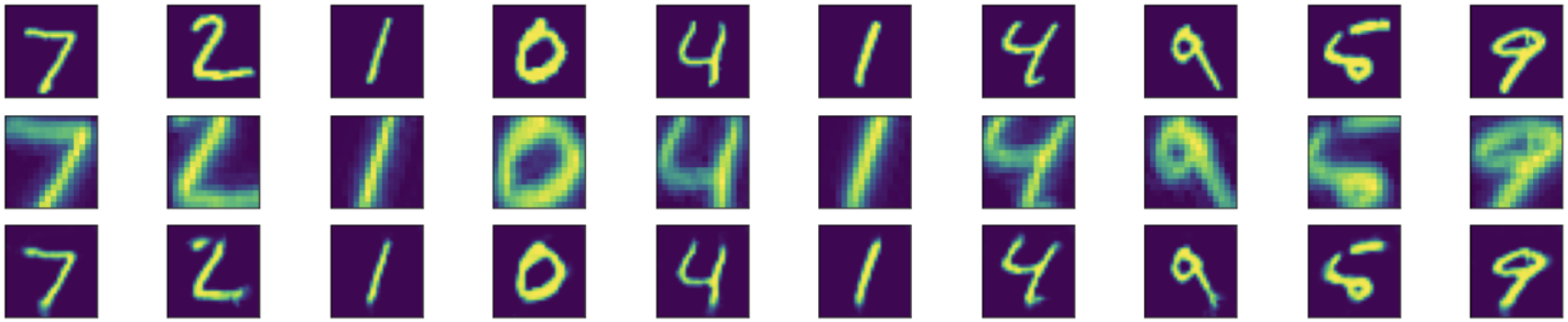
$\Rightarrow$ Understanding is data reduction

Convolutional Autoencoder(CAE)

Convolutional Autoencoder(CAE)

A CAE [55] 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)



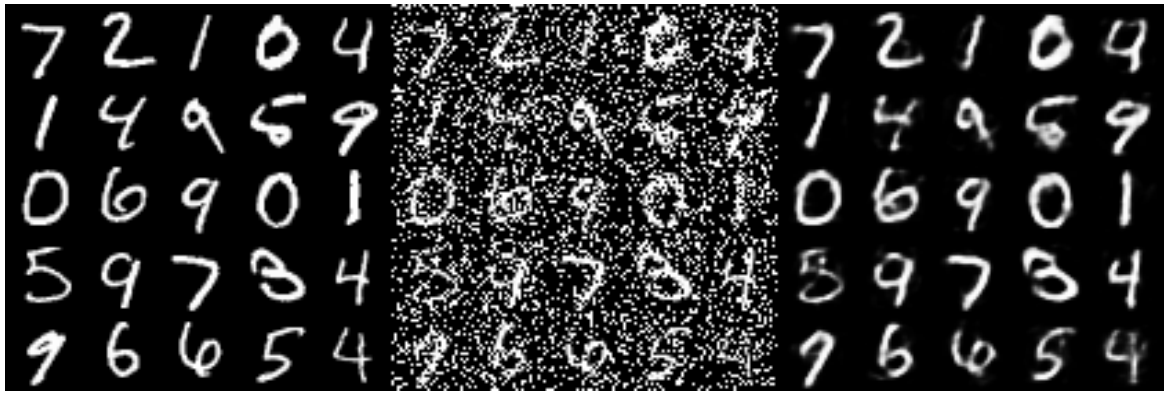
⇒ CAE learns that image borders have no information and chops the central part.

[55] 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

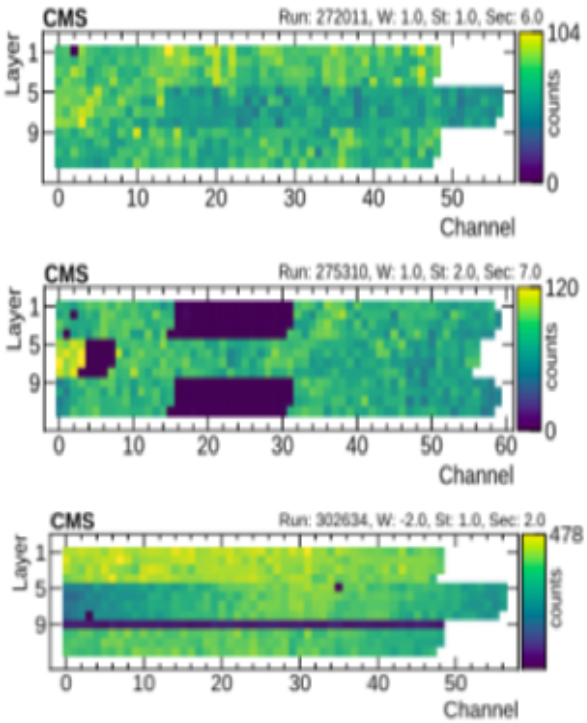


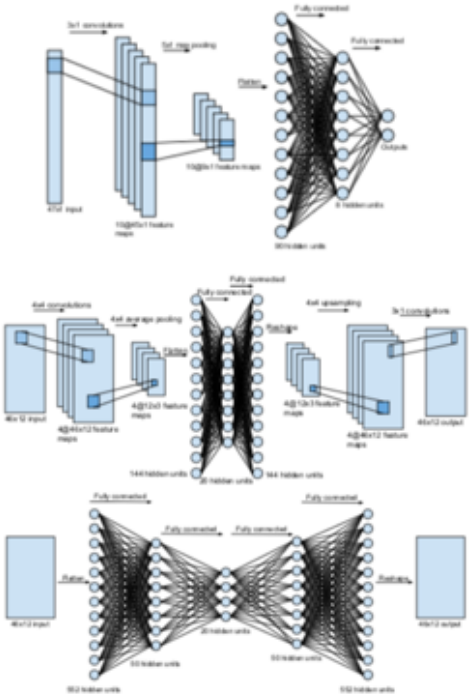
Autoencoder Application in HEP: DQM

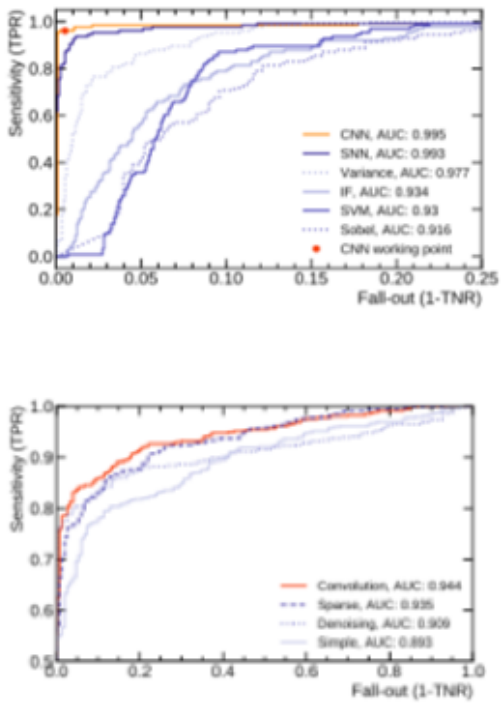
AE can be used for anomaly detection by training on a single class , so that every anomaly gives a large reconstruction error

Detector Quality Monitoring (DQM)

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





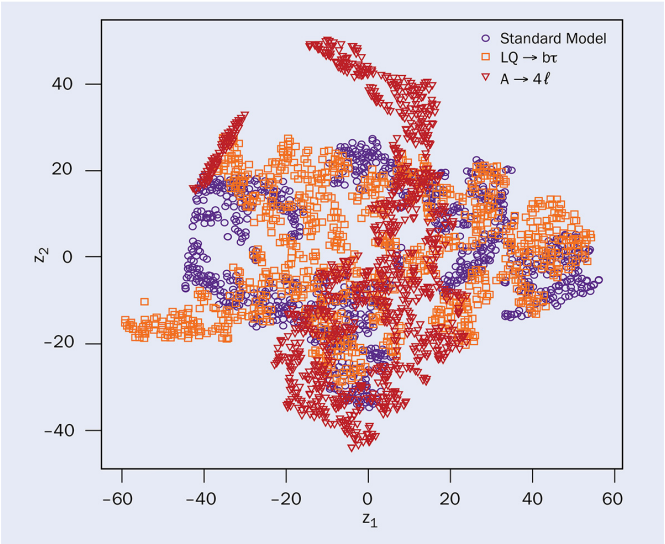


Autoencoder Application in HEP: Anomaly Detection at LHC (BSM)

AE can watch for signs of new physics at the LHC that have not yet been dreamt up by physicists [56]

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.

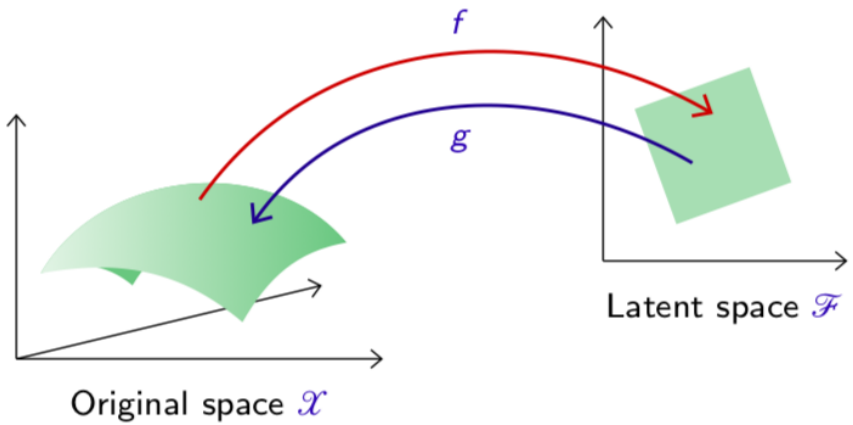


Outliers could go into a data stream of interesting events to be further scrutinised

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

Generative Autoencoder

An autoencoder combines an encoder f from the original space $\mathcal{X}$ to a latent space $\mathcal{F}$, and a decoder g to map back to $\mathcal{X}$, such that the composite map $g \circ f$ is close to the identity when evaluated on data.



Autoencoder Loss Function

$$L = \| X - g \circ f(X) \|^2$$

Autoencoder as a Generator

One can train an AE on images and save the encoded vector to reconstruct (generate) it later by passing it through the decoder. The problem is that two images of the same number (ex: 2 written by different people) could end up far away in latent space !

Variational Autoencoder(VAE)

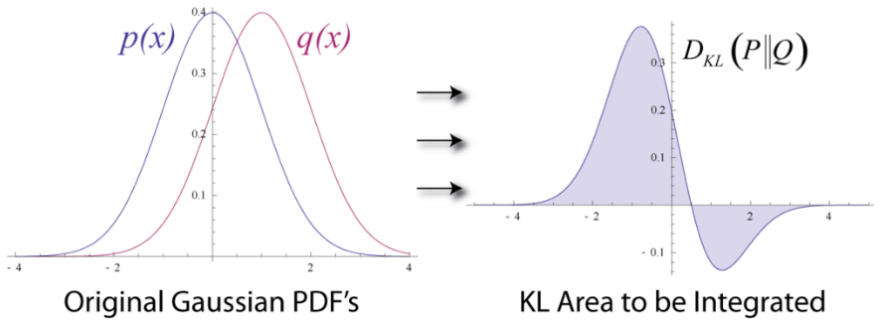
Variational Autoencoder(VAE)

A VAE [61] is an autoencoder with a loss function penalty Kullback–Leibler (KL)divergence [62] [63] that forces it to generate latent vectors that follows a unit gaussian 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)

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



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

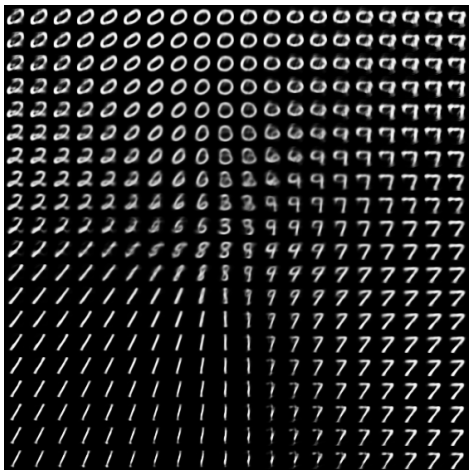
Variational Autoencoder(VAE)

VAE Loss

The VAE loss [64] 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



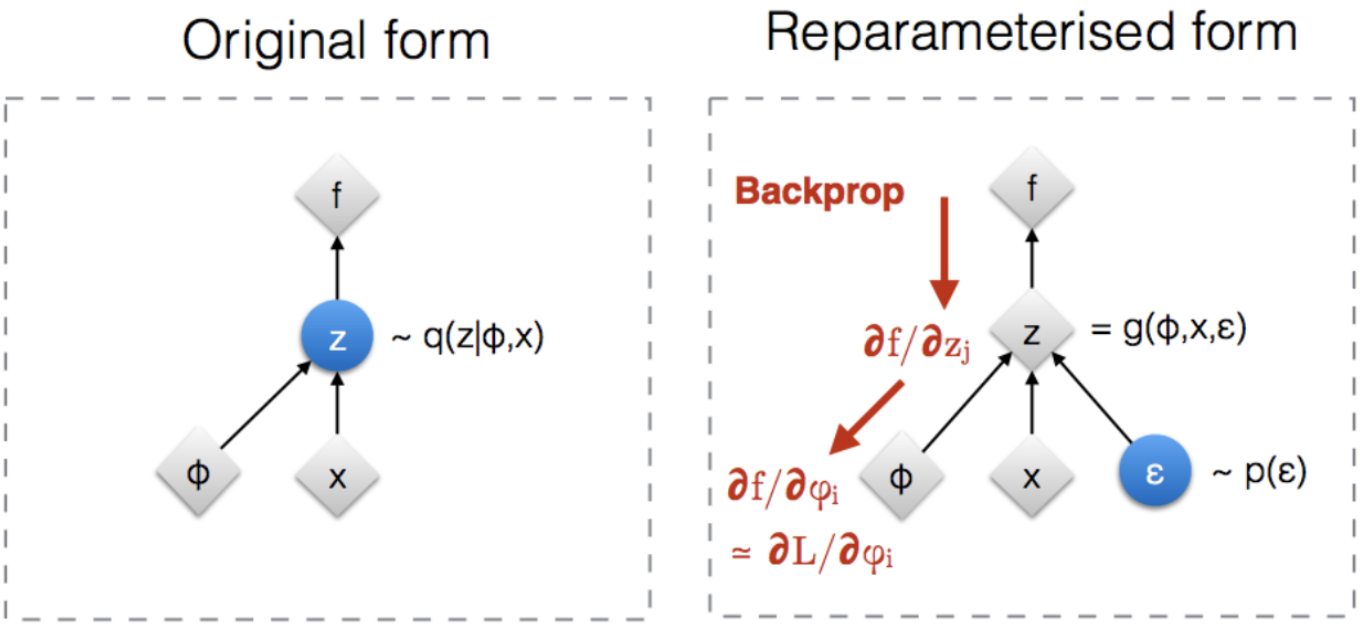
$\Rightarrow$ KL loss is a regularization term that helps learning "well formed"latent space

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

Variational Autoencoder(VAE)

VAE Reparametrization Trick

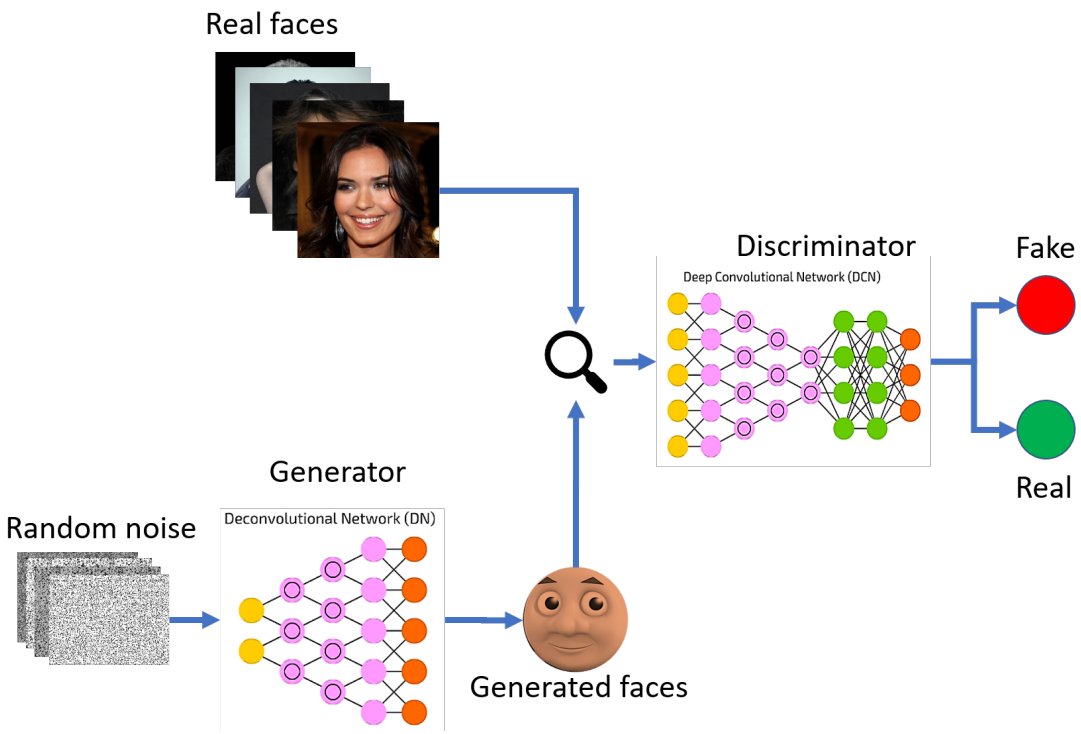
$$z = \mu + \sigma * z_c = \mu + \sigma * (\mu_c + \sigma_c * \epsilon)$$
$$= \underbrace{(\mu + \sigma * \mu_c)}_{\textit{VAE mean}} + \underbrace{(\sigma * \sigma_c)}_{\textit{VAE std}} * \epsilon$$



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.

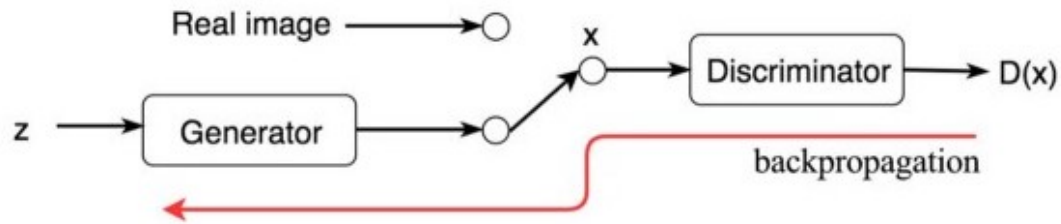


Generative Adversarial Network(GAN)

GAN adversarial training [66] 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

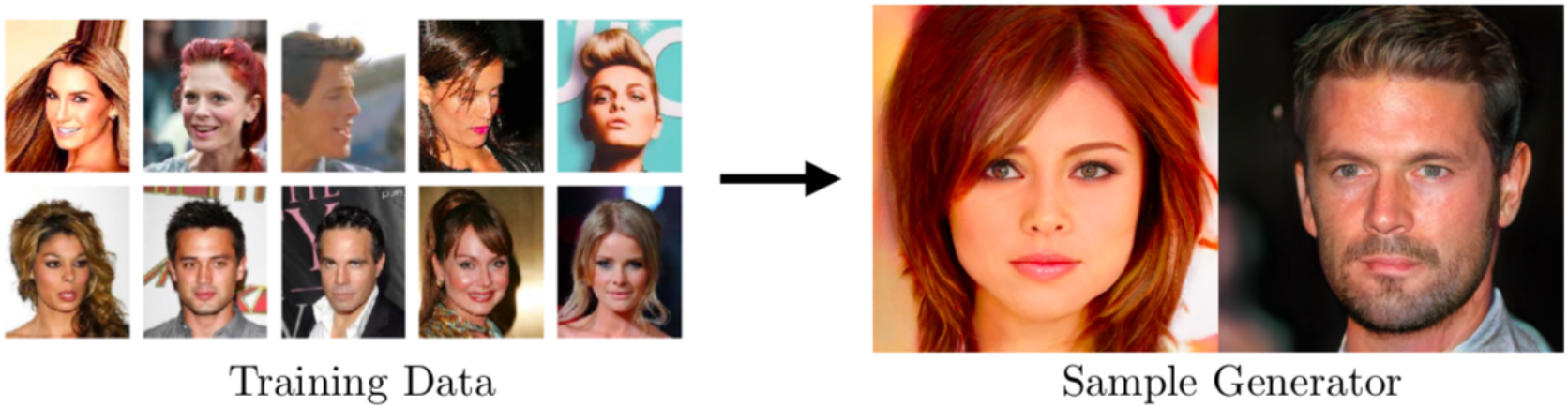
1. The discriminator receives as input samples synthesized by the generator and real data . It is trained just like a classifier, so if the input is real, we want output=1 and if it's generated, output=0
2. The generator is seeded with a randomized input that is sampled from a predefined latent space (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



[66] 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 [68] or Monet style painitings [69] !

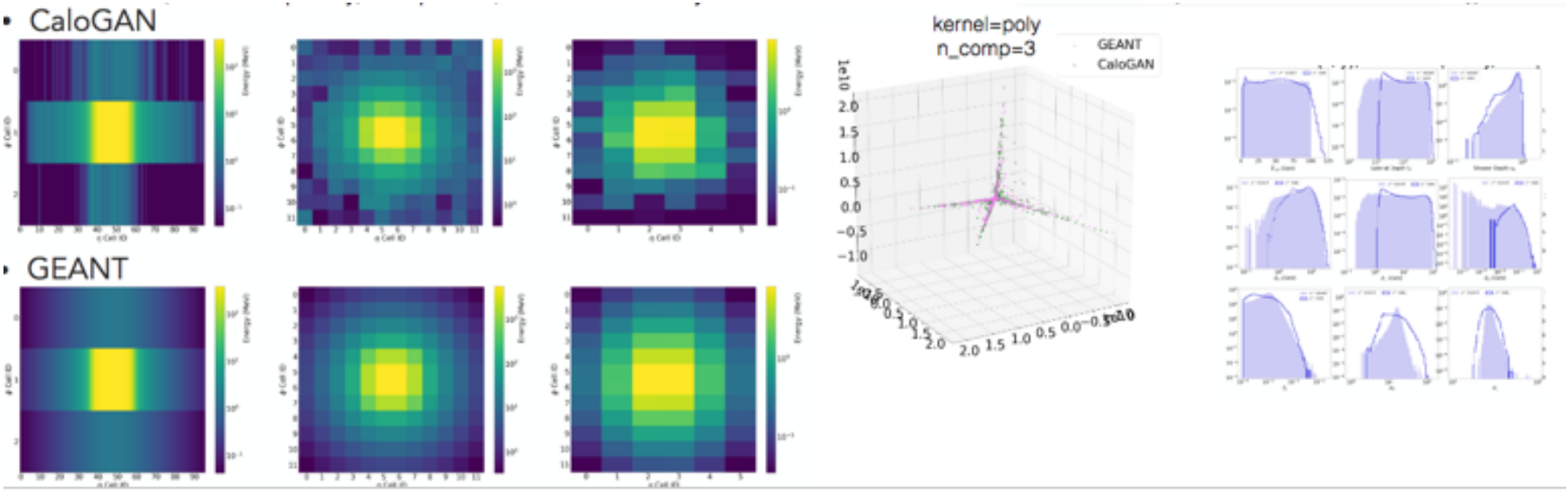


[68] https://research.nvidia.com/publication/2017-10_Progressive-Growing-of
[69] https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

GAN Application in HEP: MC Simulation

CaloGAN

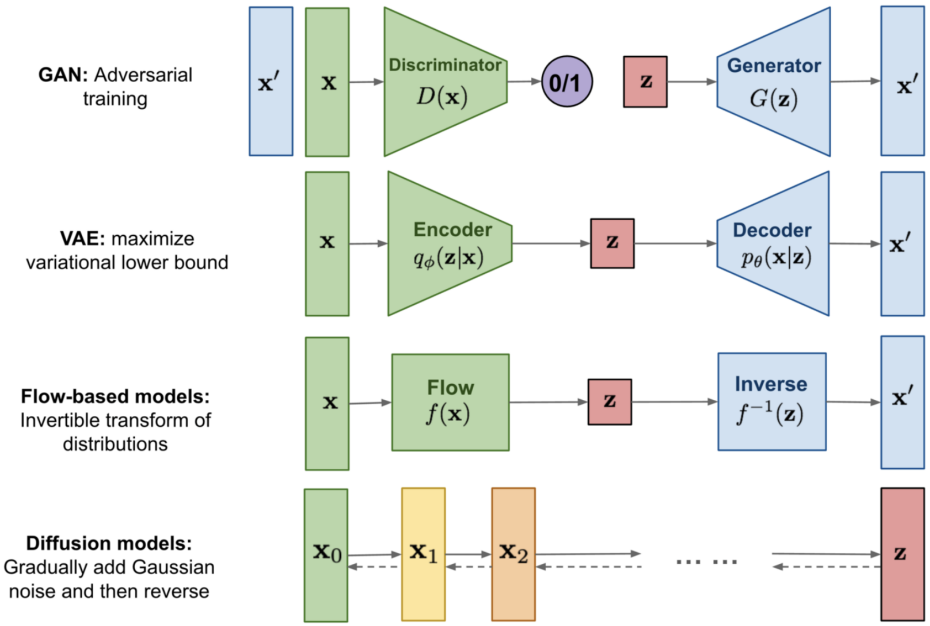
Simulating 3D high energy particle showers in multi-layer electromagnetic calorimeters with a GAN [70]



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

Generative Models And Density Estimation

There are 4 basic types of generative models: VAE , GAN , Normalizing Flows and Diffusion Models [71]



⇒ Neither GAN and VAE explicitly learns the data probability density function (implicit models) [72]

[71] https://www.youtube.com/watch?v=Erwz9LKY0n4&list=PL1TnhG8zyyNyLC7ypMdHWYiX0Y5NTd-8C&index=43&t=841s

[72] https://www.youtube.com/watch?v=2tVHbcUP9b8

Diffusion Model

Diffusion Model

Diffusion models are inspired by non-equilibrium thermodynamics. They 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. [75]

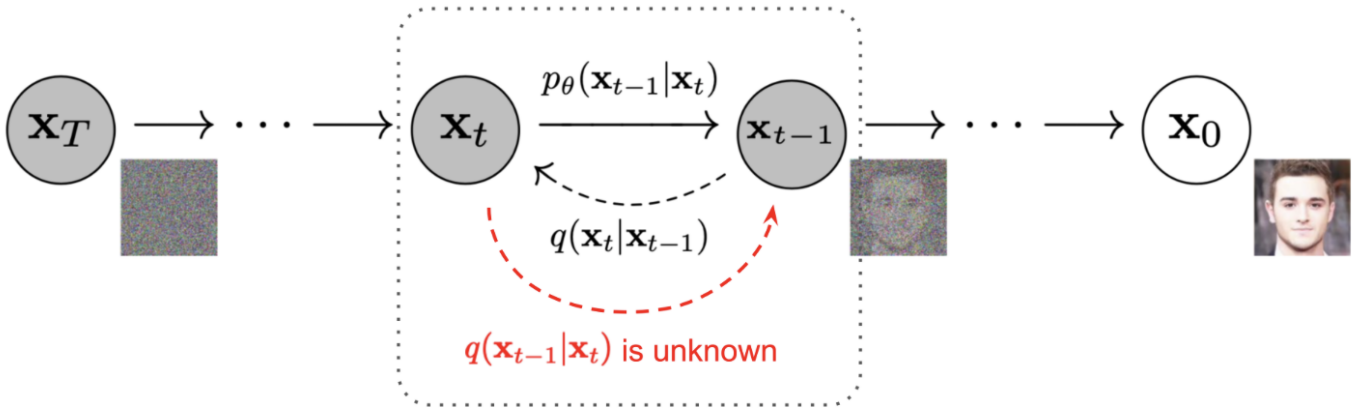


Fig. 2. The Markov chain of forward (reverse) diffusion process of generating a sample by slowly adding (removing) noise. (Image source: Ho et al. 2020 with a few additional annotations)

Obs: For images a diffusion model works as a neural network trained to denoise images blurred with Gaussian noise

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

Diffusion Model

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 β_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)

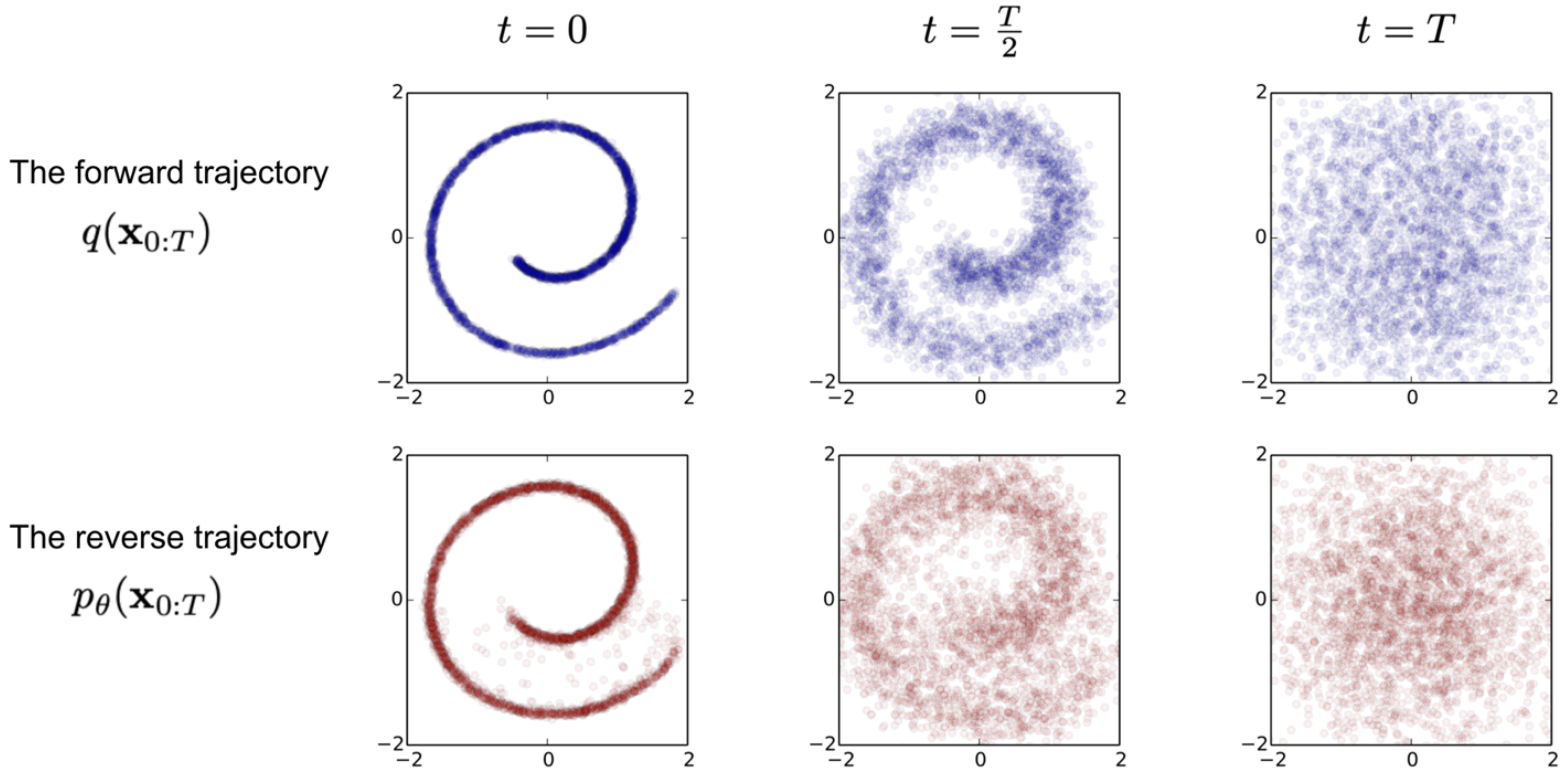
Reverse/Backward Process

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

Example of training a diffusion model for modeling a 2D Swiss roll data /footnote[frame]https://arxiv.org/abs/1503.03585



Diffusion Model

Diffusion Loss

Diffusion models minimize the cross entropy L_{CE} as the learning objective. This can be shown to be equivalent to minimize the variational lower bound L_{VLB} in the espression bellow

$$
\begin{aligned}
L_{\mathrm{VLB}} &= L_T + L_{T-1} + \cdots + L_0 \\
\text{where } L_T &= D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p_\theta(\mathbf{x}_T)) \\
L_t &= D_{\mathrm{KL}}(q(\mathbf{x}_t|\mathbf{x}_{t+1}, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})) \text{ for } 1 \le t \le T-1 \\
L_0 &= -\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)
\end{aligned}
$$

$\Rightarrow$ Every KL term in it (except for L_0) compares two Gaussian distributions that can be computed in closed form. L_T is constant (no learnable parameters) and x_T is a Gaussian noise.

Generative Diffusion Models

Stable Diffusion Playground

https://stablediffusionweb.com/#demo

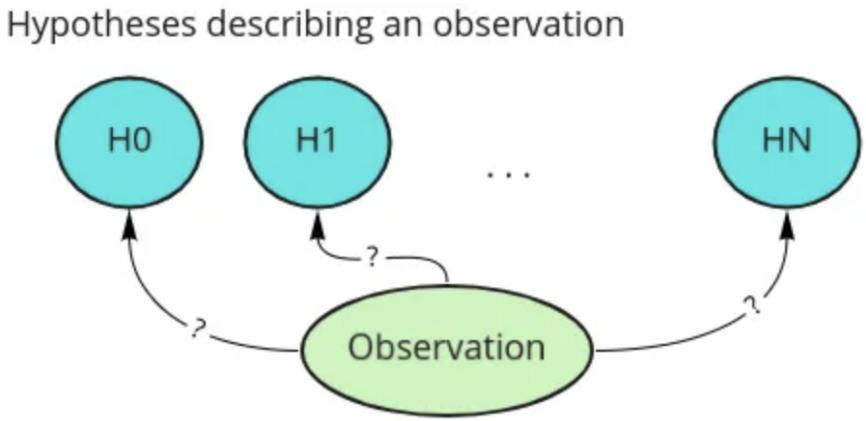
- https://en.wikipedia.org/wiki/Diffusion_model
- A Survey on Generative Difusion Models https://arxiv.org/abs/2209.02646
- Understanding Diffusion Models: A Unified Perspective https://arxiv.org/abs/2208.11970
- L.Weng, What are Diffusion Models? https://lilianweng.github.io/posts/2021-07-11-diffusion-models/
- Sing&Rath , Introduction to Diffusion Models for Image Generation – https://learnopencv.com/image-generation-using-diffusion-models/

Domain Knowledge and Inductive Bias

Every machine learning model has some type of inductive bias (ex: linear regression assumes linearly related variables).

Inductive Bias

Inductive bias is an assumption that allows a model to prioritize one solution over another or to learn more efficiently. [76]



⇒ A given set of observations can lead to different hypothesis depending on the inductive biases

[76] https://arxiv.org/pdf/1806.01261.pdf
https://sgfin.github.io/2020/06/22/Induction-Intro

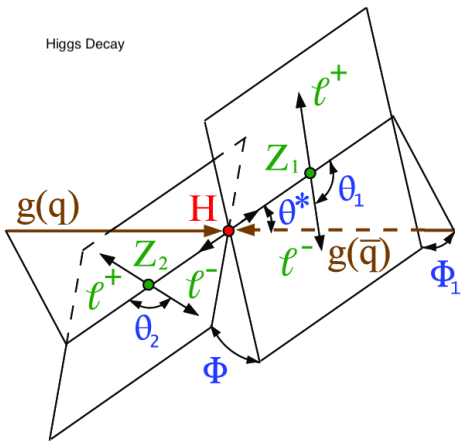
Domain Knowledge and Inductive Bias

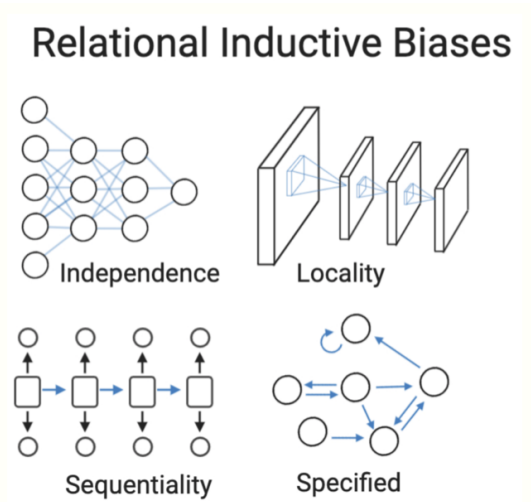
Domain knowledge can be infused into a model in different ways [77]

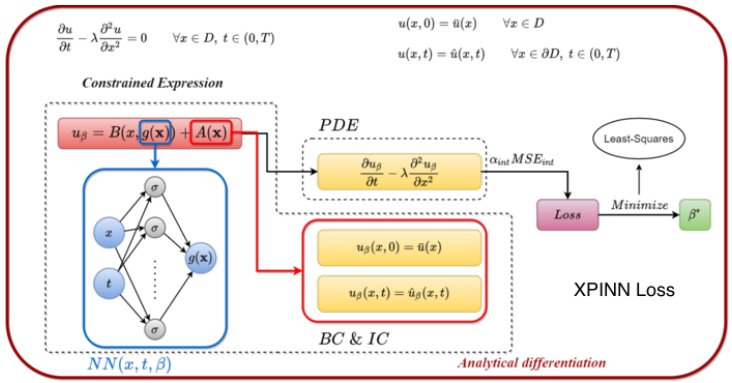
Domain Knowldge

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

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







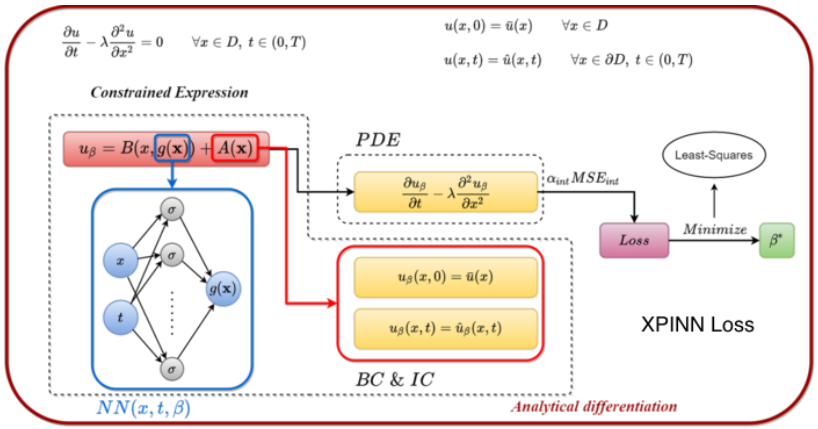
[77] https://sgfin.github.io/2020/06/22/Induction-Intro/
https://www.nature.com/articles/s41598-021-04590-0

Physics Informed Neural Network(PINN)

Neural networks can be used to approximate the solution of differential equations [78]

Physics Informed Neural Network(PINN)

Physics-informed neural networks (PINNs) are a type of universal function approximators that can embed the knowledge of any physical laws that govern a given dataset in the learning process, and can be described by partial differential equations (PDEs)



The boundary and initial conditions , together with the differential equation are added as additional terms to the loss function in order for the learning algorithm to capture the correct solution.

[78] https://benmoseley.blog/my-research/so-what-is-a-physics-informed-neural-network
https://physicsbaseddeeplearning.org

Machine Learning Software

- General ML library (Python):
  1. `https://scikit-learn.org/stable`
- Deep learning libraries:
  1. `https://www.tensorflow.org` (TensorFlow)
  2. `https://pytorch.org` (PyTorch)
  3. `https://www.microsoft.com/en-us/cognitive-toolkit` (CNTK)
- High level deep learning API:
  1. `https://keras.io` (Keras)
  2. `https://docs.fast.ai` (FastAI)

Online Lectures and Resources 1

- Neural Networks and Deep Learning, M.Nielsen : http://neuralnetworksanddeeplearning.com
- MIT Introduction do Deep Learning Course: http://introtodeeplearning.com
- P.Bloem , MACHINE LEARNING Lectures - Vrije Universiteit Amsterdam , https://mlvu.github.io
- P.Bloem , DEEP LEARNING Lectures - Vrije Universiteit Amsterdam , https://https://dlvu.github.io
- G.Louppe , Advanced Machine Learning, Univ. Liège (2023) , https://github.com/glouppe/info8010-deep-learning
- G. Louppe , Deep Learning, Univ. Liège (2023) , https://github.com/glouppe/info8004-advanced-machine-learning
- A.Canziani and Y.LeCun, Deep Learning Course , NYU https://atcold.github.io/didactics.html
- Stanford CS231 class, A.Karpathy : http://cs231n.stanford.edu
- Stanford CS229 class, A.Ng : http://cs229.stanford.edu
- Stanford CS224W - Machine Learning with Graphs (2021) , J.Leskovec : https://youtube.com/playlist?list=PLoROMvodv4rPLKxIpqhjhPgdQy7imNkDn
- M.Bronstein, P.Veličković, J.Bruna and T.Cohen, Geometric Deep Learning Course

Online Lectures and Resources 2

- Waterloo Univ. CS480 class (2020), P. Poupart: https://cs.uwaterloo.ca/~ppoupart/teaching/cs480-fall20/schedule.html
- Google DeepMind x UCL - Deep Learning Lectures 2018: https://deepmind.com/learning-resources/deep-learning-lectures-series-2018
- Google DeepMind x UCL - Deep Learning Lectures 2020: https://deepmind.com/learning-resources/deep-learning-lecture-series-2020
- Machine Learning Summer School (MLSS-2020) : https://www.youtube.com/channel/UCBOgpkDhQuYeVVjuzS5Wtxw/videos
- Variational Inference Tutorial, S.Mohamed (2016) : https://shakirm.com/neurips2016.html
- Graph Neural Networks Lectures (Google 2021) - P.Velickovic : https://www.youtube.com/channel/UC9bkKi8Us7yevvP1KIBQHog/featured
- Sebastian Raschka , Univ. Wisconsin-Madison Courses , https://sebastianraschka.com/teaching/
- J.Jensen , Machine Learning Basics, Univ. Copenhagen - https://sites.google.com/view/ml-basics/home
- Unsupervised Learning with Graph Neural Networks, T.Kipf : https://www.youtube.com/watch?v=9jSFBcptZ9A&t=2168s

Additional Topics

Solving Differential Equations with NN

The End !!!