Machine Learning in HEP

A. Sznajder

UERJ
Instituto de Fisica

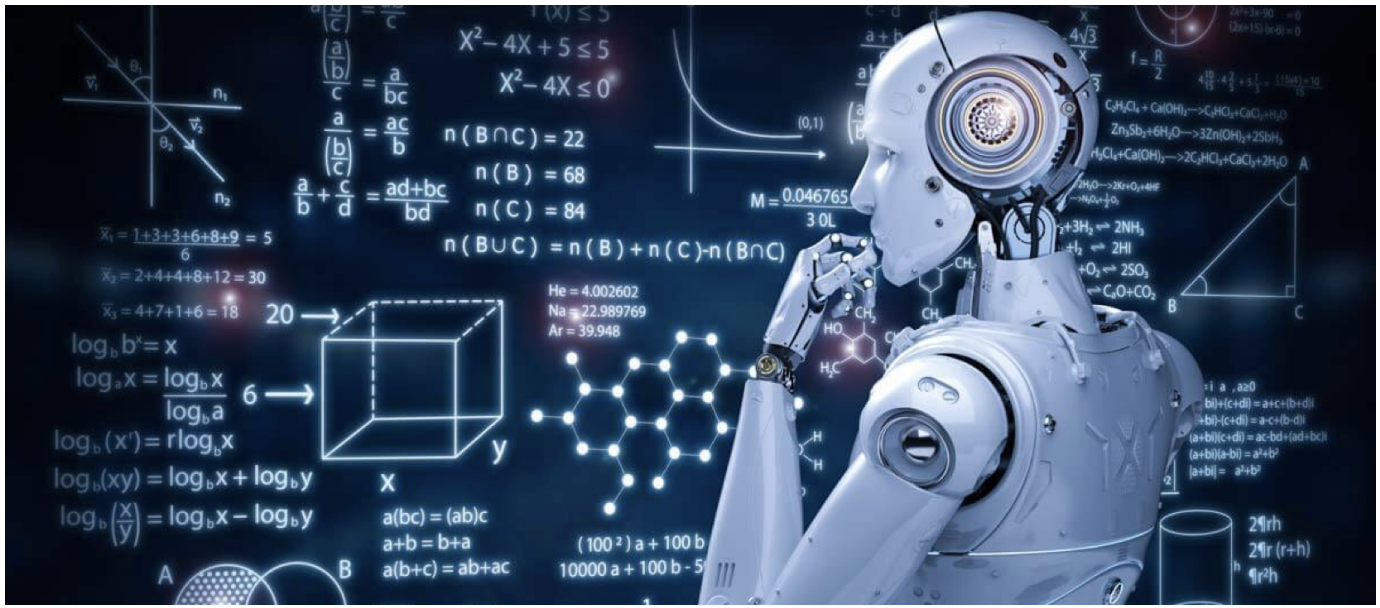
September - 2022

Outline

What is Machine Learning ?

Machine Learning (ML)

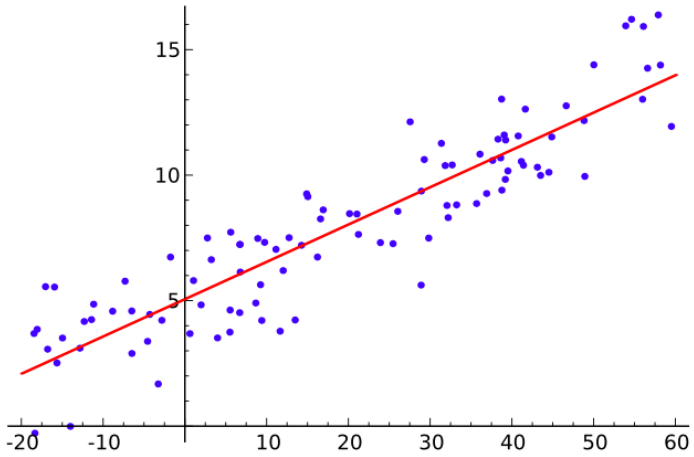
Machine learning (ML) is the study of computer algorithms capable of building a mathematical model out of a data sample, by learning from examples. The algorithm builds a predictive model without being explicitly programmed (sequence of instructions) to do so



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



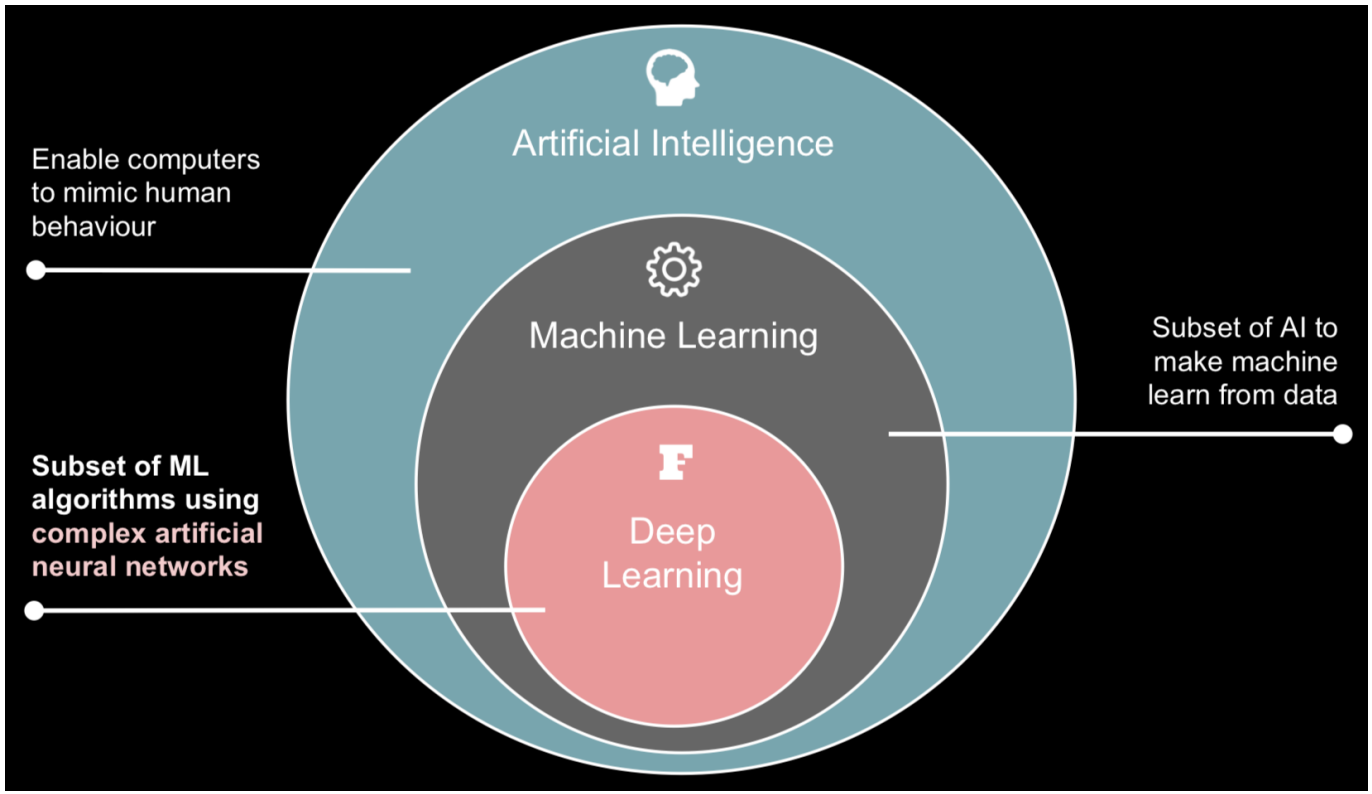
Linear regression (Source)

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: learning a function from a data sample

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

Artificial Inteligence

Inteligence is the ability to process current information to inform future decisions



Artificial Inteligence

None of the systems we have nowadays are real AI. The brain learns so efficiently that no ML method can match it

AI versus Brain

- Brain has 10^{14} parameters and we live only $10^9 s$ (lot more parameters than data)
- So, it must do lots of unsupervised learning and must predict what we observe !
- Supervised (reinforcement) learning requires milions of examples(trials)
- Still missing a learning paradigm that builds predictive models through observation and action

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

Introduction to Machine Learning

Machine learning can be implemented by many different algorithms (ex: SVM, BDT, Bayesian Net , Genetic Algo ...) , but we will discuss only Neural Networks(NN)

1) Neural Network Topologies

- Feed Forward NN
- Recurrent NN

2) Learning Paradigms

- Supervised Learning (Labeled data)
- Unsupervised Learning (Unlabeled data)
- Reinforcement Learning (Maximize reward)

2) Applications

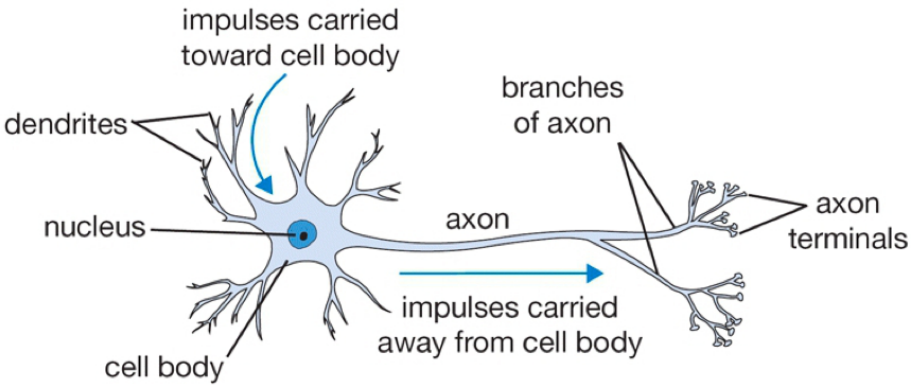
- Classification
- Regression
- Data Generation

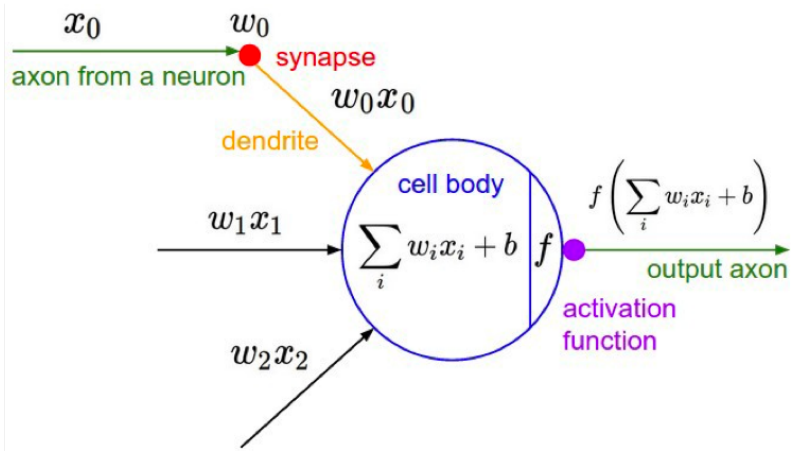
3) Neural Networks Architectures

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

Neural Networks

Artificial Neural Networks (NN) are computational models vaguely inspired [2] by biological neural networks. A Neural Network (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 (Node)

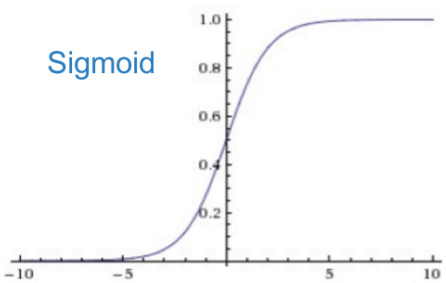
Each node of a NN receives inputs $\vec{x} = \{x_1, \ldots, x_n\}$ from other nodes or an external source and computes an output y according to the expression

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

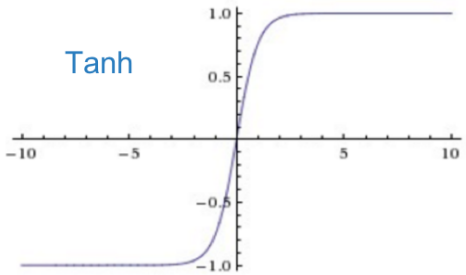
, where W_i are connection weights, B is the treshold and F the activation function [3]

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

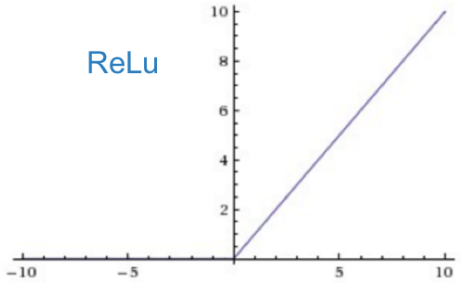
Sigmoid



Tanh



ReLu



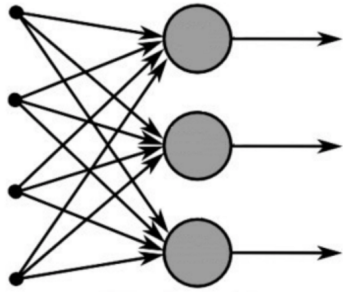
[3] Nonlinear activation is fundamental for nonlinear decision boundaries

Neural Network Topologies

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

Feed Forward Networks

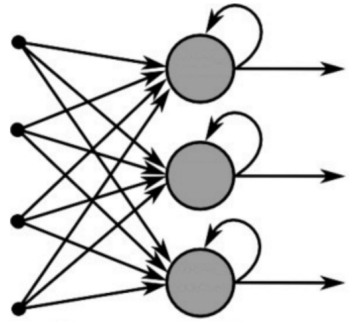
A feedforward NN is an neural network wherein connections between the nodes do not form a cycle. The information always moves one direction, from input to output, and it never goes backwards.



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.



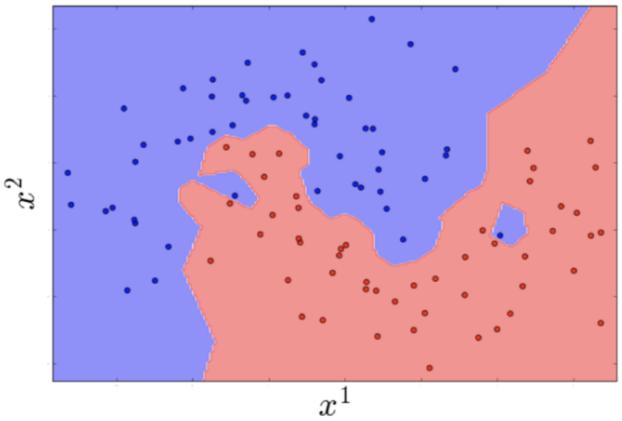
Recurrent Neural Network

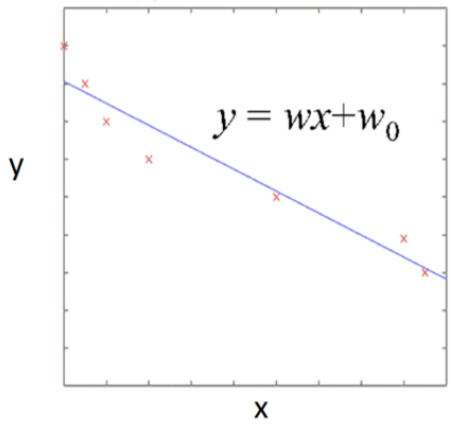
A NN layer is called a dense layer to indicate that it's fully connected.

Supervised Learning

Supervised Learning

- During training a learning algorithm adjust the network's weights to values that allows the NN to map the input to the correct output.
- It calculates the error between the target output and a given NN output and use error to correct the weights.
- 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 a mapping $\vec{t}_i = F(\vec{x}_i)$
- **Classification:** $\{\vec{t}_1, \ldots, \vec{t}_n\}$ (finite set of labels)
- **Regression:** $\vec{t}_i \in \mathbb{R}^n$





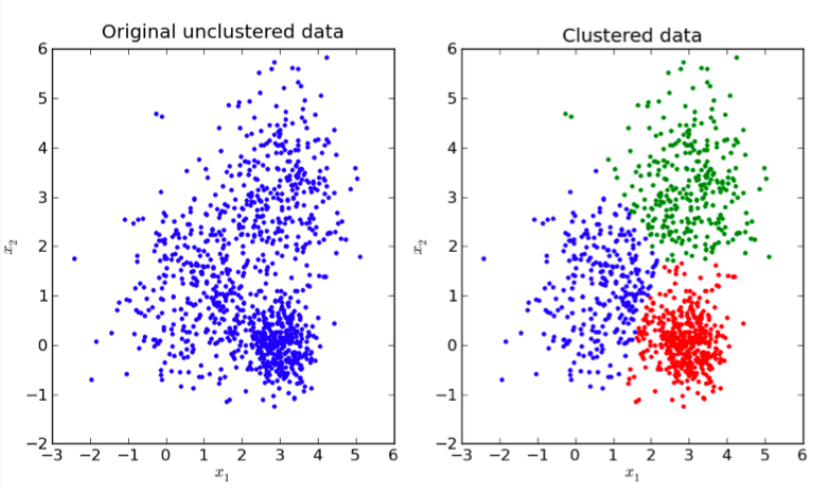
Unsupervised Learning

Unsupervised Learning

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

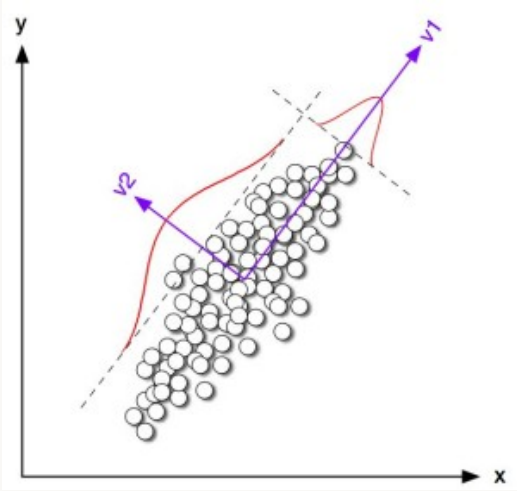
Clustering

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 $n >> m$

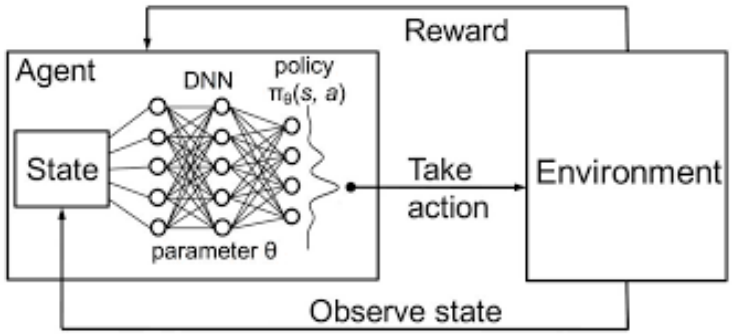


Reinforcement Learning

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
- Maximize the cumulative reward the system receives through a trial-and-error approach
- Algorithm learns the best sequence of decisions (actions) to achieve a given goal



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

Reinforcement Learning

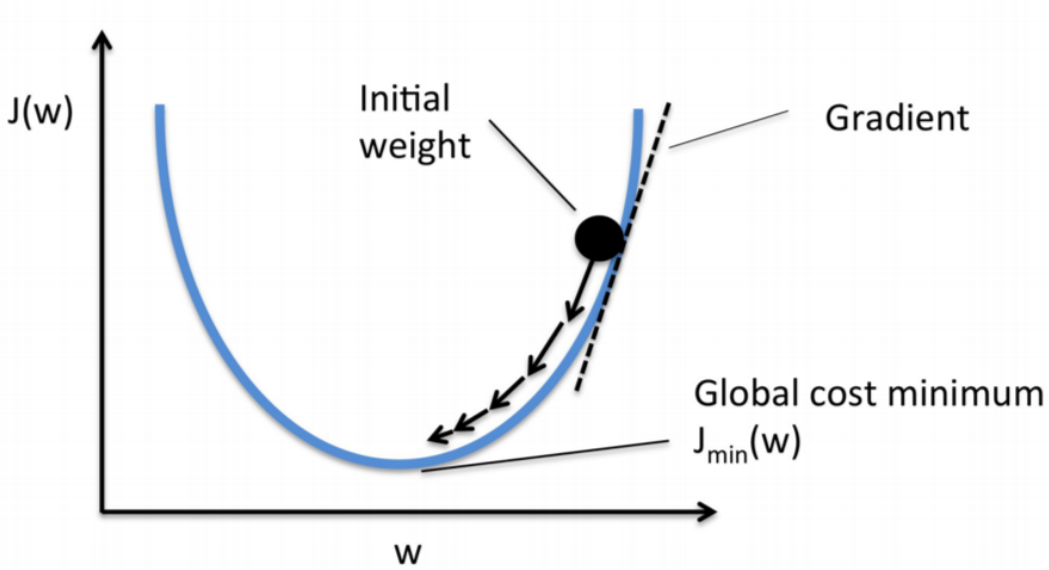
Google Deepmind neural network playing Atari Breakout game

(click on the figure bellow)

Supervised Learning - Training Process

Learning as an Error Minimization Problem

1. Random NN parameters (weights and bias) initialisation
2. Choose an Objective(Loss) Function , differentiable with respect to model parameters
3. Use training data to adjust parameters (gradient descent + back propagation) that minimize the loss
4. Repeat until parameters values stabilize or loss is below a chosen 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 !

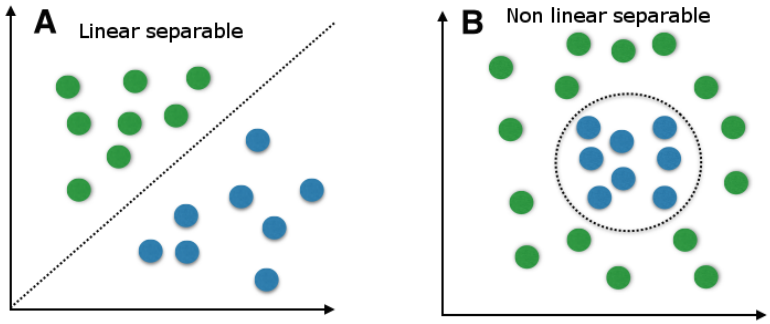
[4] For $m = 2$ we have the Binary Cross Entropy $L = -t \log y - (1 - t) \log (1 - y)$

The Perceptron

The perceptron 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 [5]

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



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

Initialize the weights and for each example $(\vec{x_j}, t_j)$ in training dataset D, perform the following steps:

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

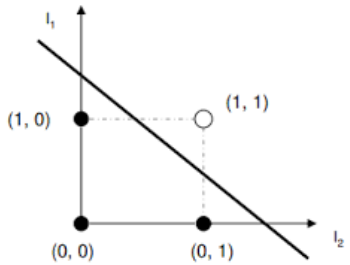
[5] Equation of a plane in $\mathbb{R}^n$ is $\vec{W}.\vec{x} + B = 0$

Perceptron and XOR Problem

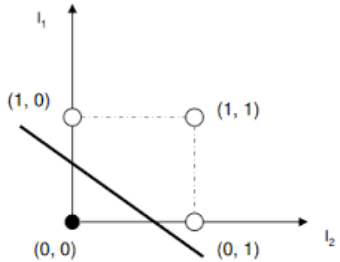
Problem:

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

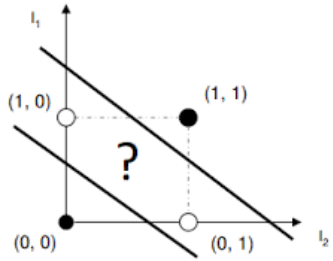
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



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

Multilayer Perceptron (MLP)

The Multilayer Perceptron(MLP) is a fully connected NN with at least 1 hidden layer and nonlinear activation function F [6] . It is the simplest feed forward NN. [7]

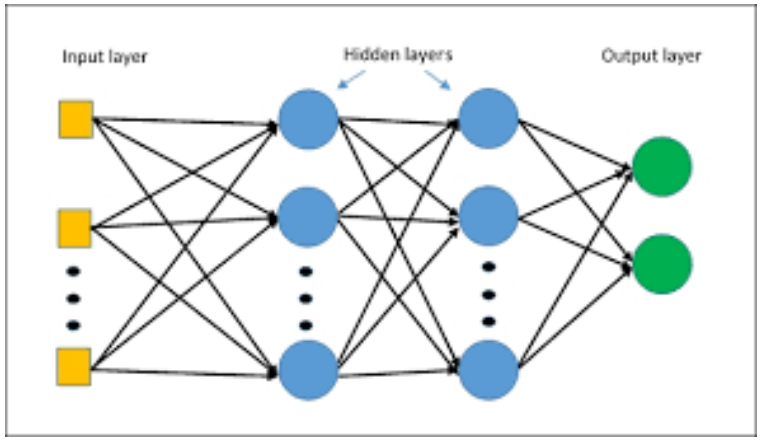
Multilayer Perceptron Model

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 $F_{w,b} : \mathbb{R}^n \rightarrow \mathbb{R}^m$

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

[7] 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 Approximator

Universal Approximation Theorem

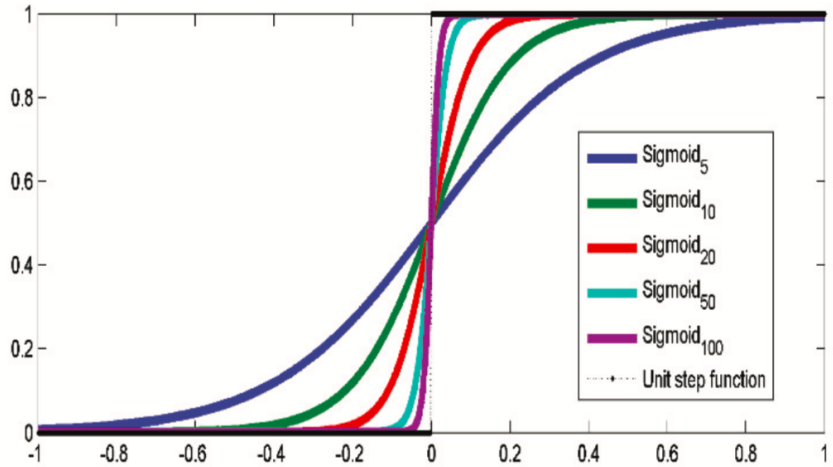
A single hidden layer feed forward neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden neurons [8]

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

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



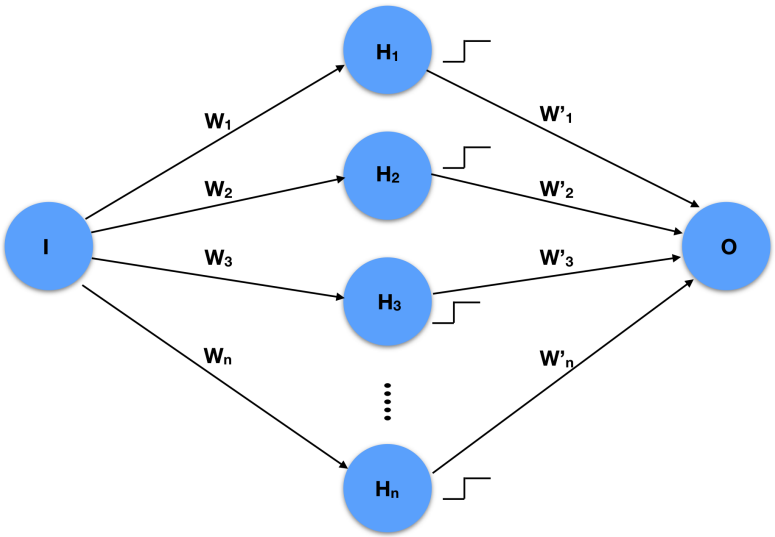
[8] Cybenko,G.(1989) Approximations by superpositions of sigmoidal functions, Math.ofCtrl.,Sig.,andSyst.,2(4),303
Hornik,K.(1991) Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks, 4(2), 251

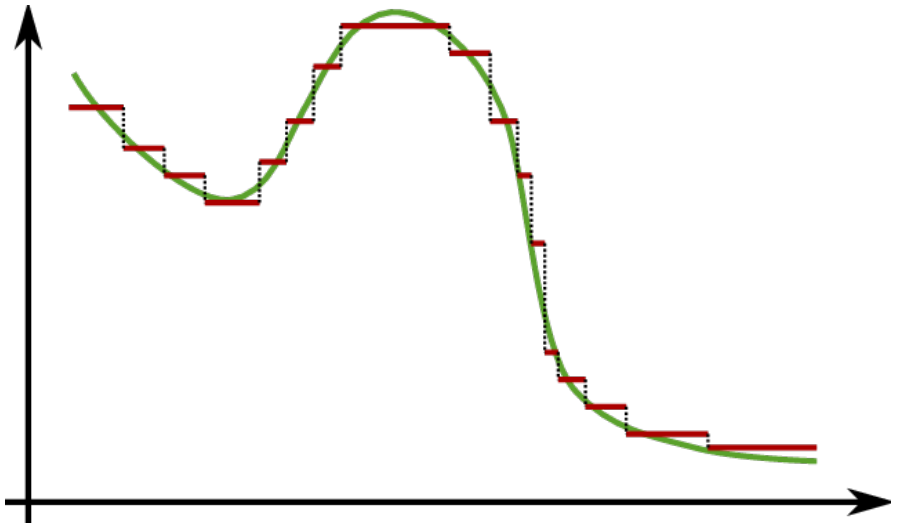
Multilayer Perceptron as a Universal Approximator

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

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





The same works for any activation function $f(x)$, limited when $x \to \pm\infty$. One can always tune weights W and tresholds B such that it behaves like a step function !

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

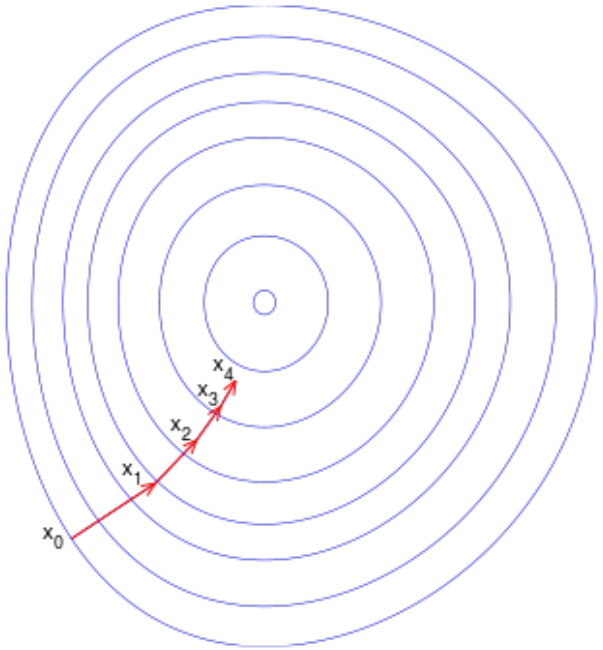
Gradient Descent

A multi-variable function $F(\vec{x})$ decreases fastest in the direction of its negative gradient $-\nabla F(\vec{x})$ [9].

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 !

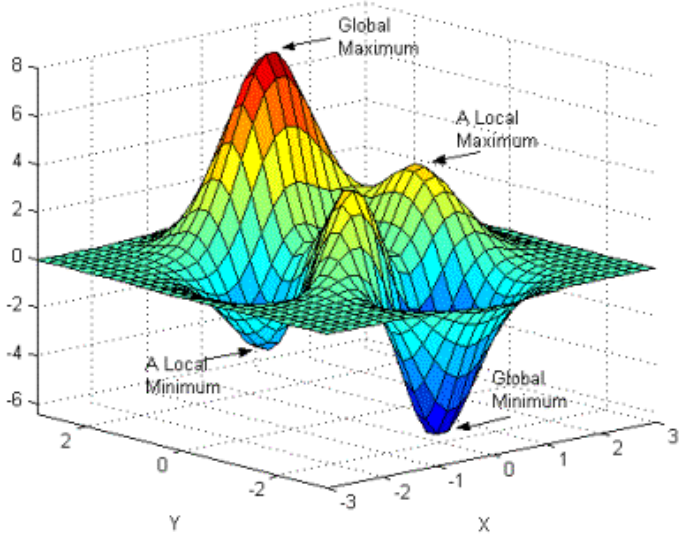


Gradient Descent uses 1^o derivatives, which is efficiently and simply calculated by backpropagation. Methods like Newton uses 2^o derivative(Hessian), which is computationally costly and memory inefficient.

[9] 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 in a layer leads to same loss !)



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

Stochastic Gradient Descent(SGD)

SGD [10] selectes data points randomly, instead of the order they appear in the training set. This allows the algorithm to try different ”minimization paths”at each training 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

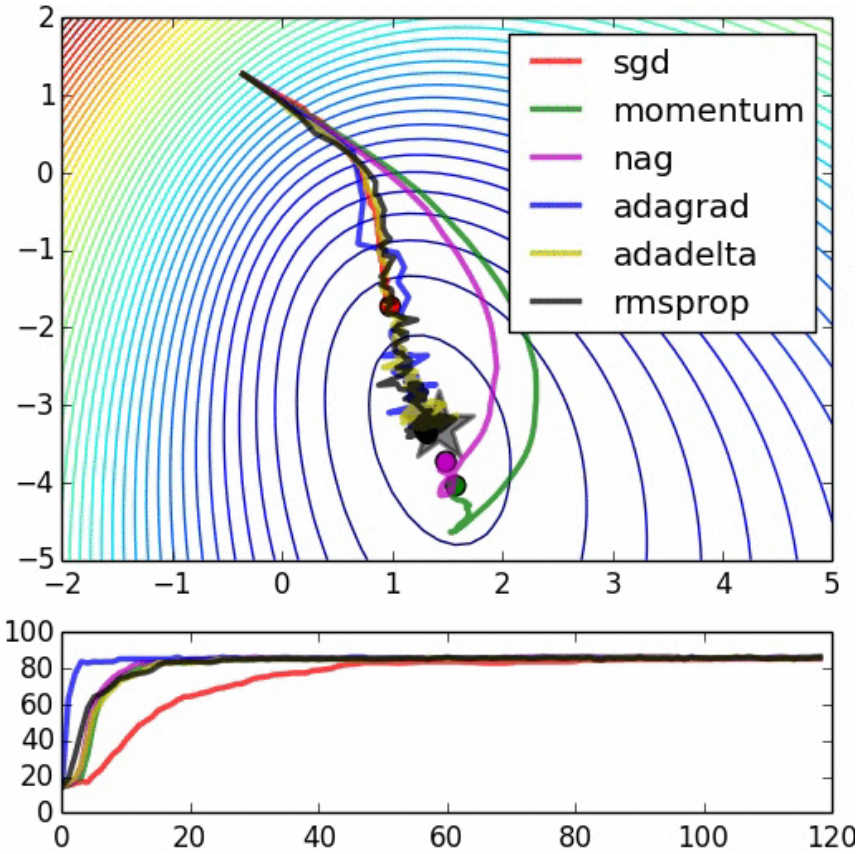
[10] https://deepnotes.io/sgd-momentum-adaptive

SGD Algorithm Improvements [12]

SGD Variants [11] :

- **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)



[11] http://danielnouri.org/notes/category/deep-learning
[12] 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 [13]

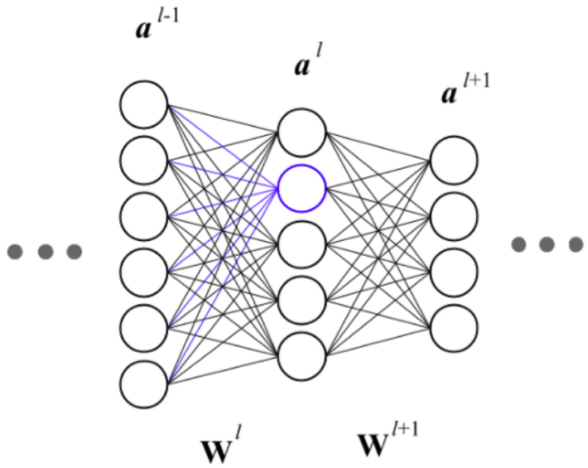
MLP Loss Function

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.



[13] 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 $\delta_k^{(l)} = \frac{\partial L}{z_k^{(l)}}$

→ The only derivative one needs to calculate is F' !

Backpropagation Algorithm

Given a training dataset $D = \{(x_i, t_i)\}$ we first run a *forward pass* to compute all the activations throughout the network, up to the the output layer. Then, one computes the network output "error" and backpropagates it to determine each neuron error contribution $\delta_k^{(l)}$

Backpropagation Algorithm

1. Initialize the weights $W_{kj}^{(l)}$ 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 using the neurons error and activation: $\frac{\partial L}{\partial W_{kj}^{(l)}} = \delta_k^{(l)} a_k^{(l-1)}$
6. Update the weights according to the gradient descent: $\Delta W_{kj}^{(l)} = -\lambda \frac{\partial L}{\partial W_{kj}^{(l)}}$

This algorithm can be applied to other NN architectures (different connectivity patterns)

Evaluation of Learning Process

Split dataset into 3 independent parts , 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

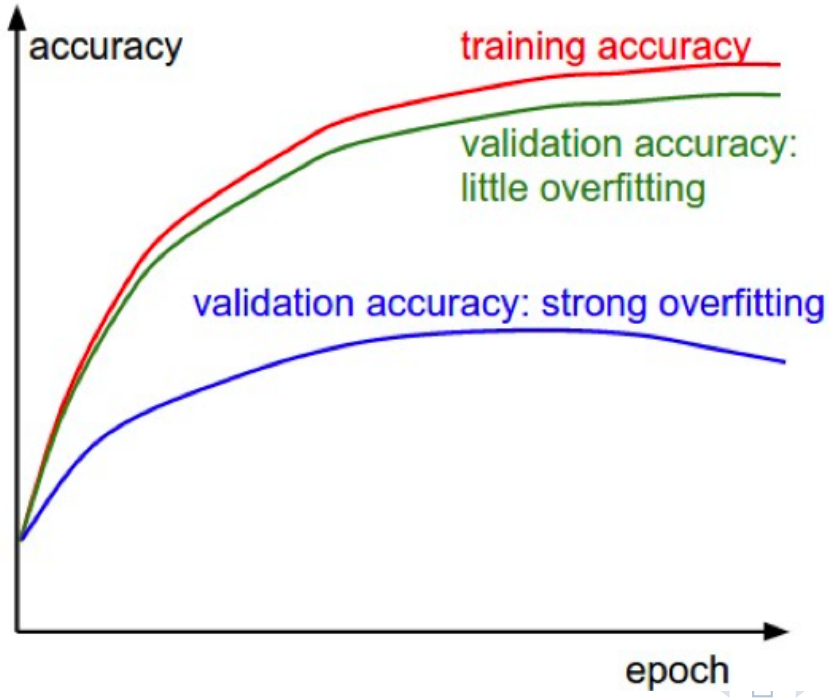
Final performance evaluation after finished training and hyper-parameters are fixed. Use the test dataset for an independent evaluation of performance obtaining a ROC curve

Overtrainging(Overfitting)

Overtraining(Overfitting)

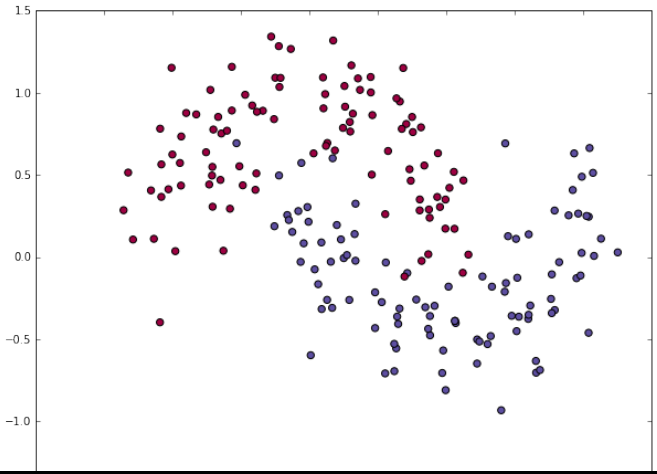
Gap between training and validation accuracy indicates the amount of overfitting

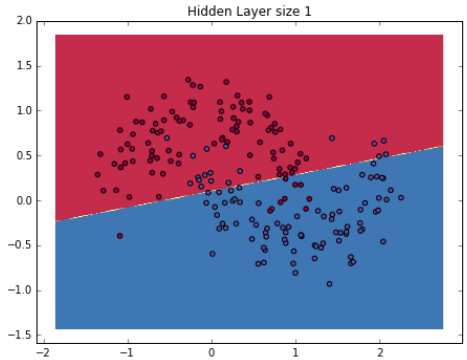
- If validation error curve shows small accuracy compared to training it indicates overfitting $\Rightarrow$ add regularization or use more data.
- If validation accuracy tracks the training accuracy well, the model capacity is not high enough $\Rightarrow$ use larger model

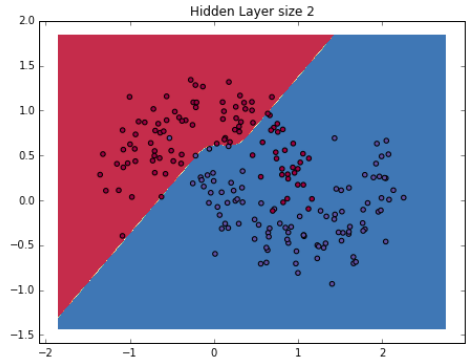


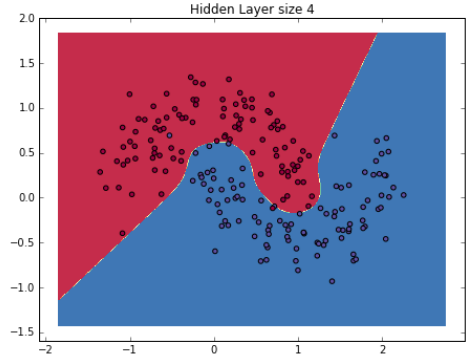
Overfitting - Hidden Layer Size X Decision Boundary [15]

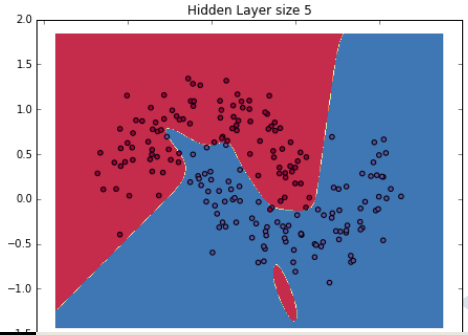
- Hidden layer of low dimensionality nicely captures the general trend of data.
- Higher dimensionalities are prone to overfitting (“memorizing” data) as opposed to fitting the general shape
- If evaluated on independent dataset (and you should !), the smaller hidden layer generalizes better
- Can counteract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

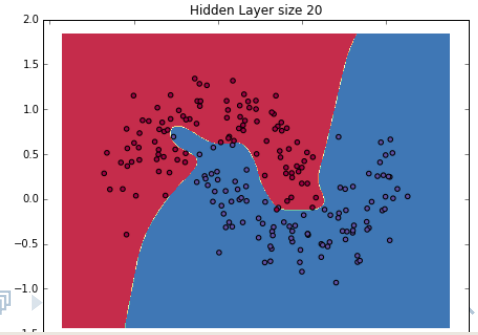












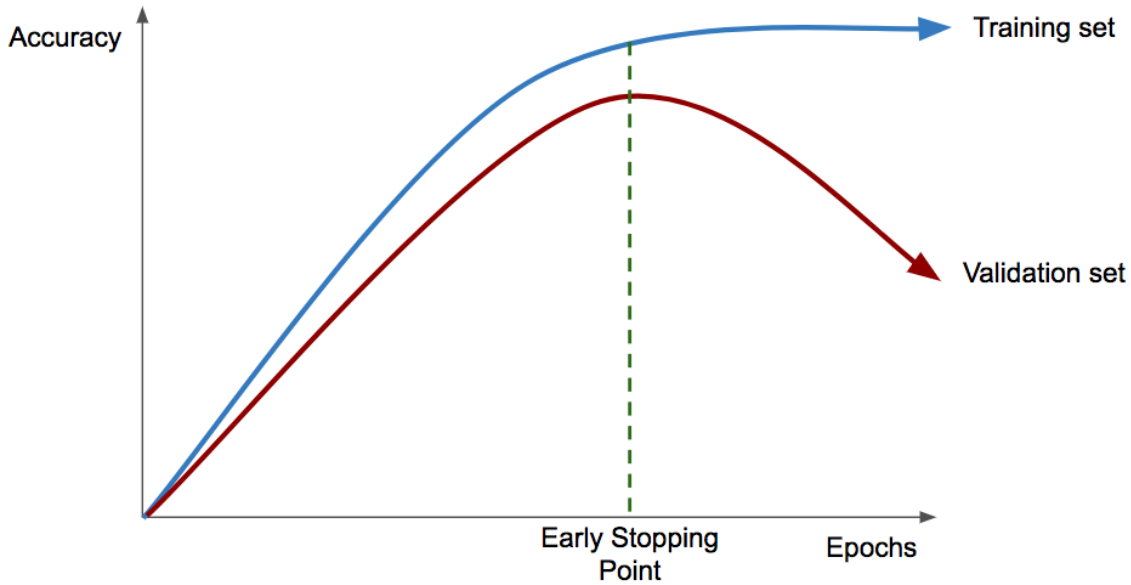
Regularization

Regularization techniques prevents the 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.

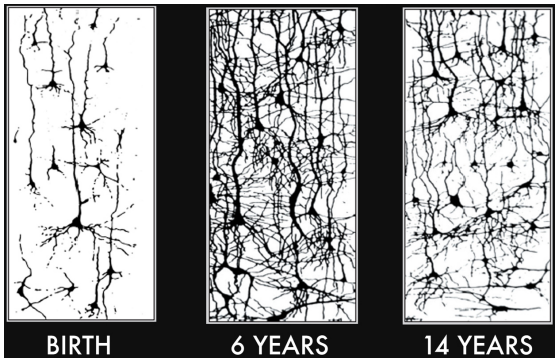


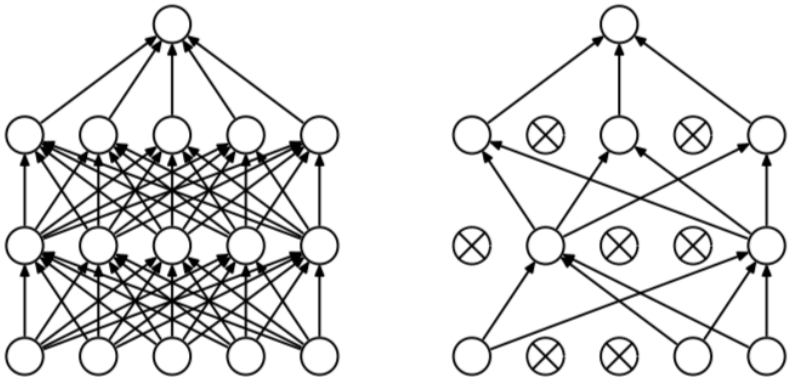
Dropout Regularization

Dropout Regularization

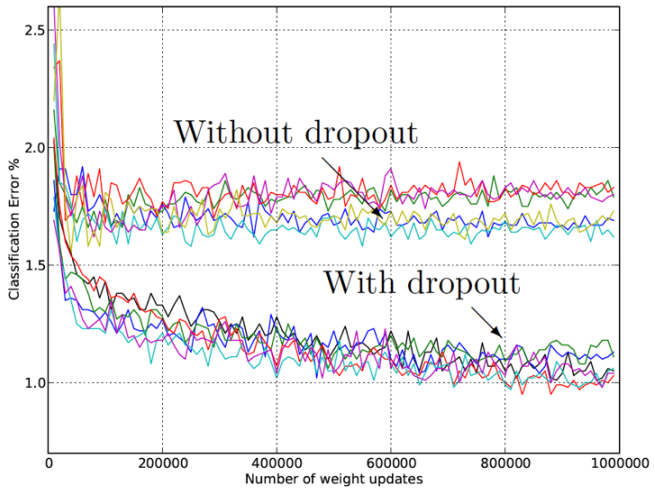
Regularization inside network that remove nodes randomly during training.

- 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

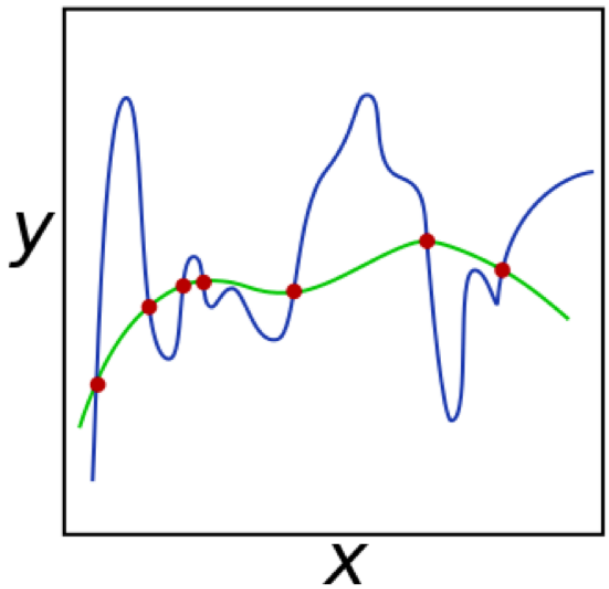
Works by keeping weights sparse

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

L2 Regularization

Works by penalising large weights

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

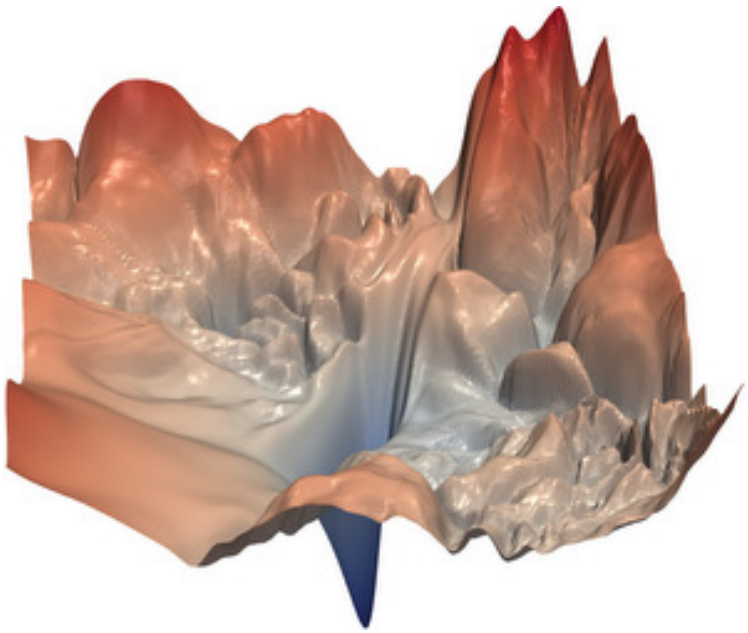


The combined $L1 + L2$ regularization is called Elastic

Loss Landcape and Local Minima

NN Training

Neural network training relies on our ability to find “good” minima of highly non-convex loss functions. [16]



- 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. [17]

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

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

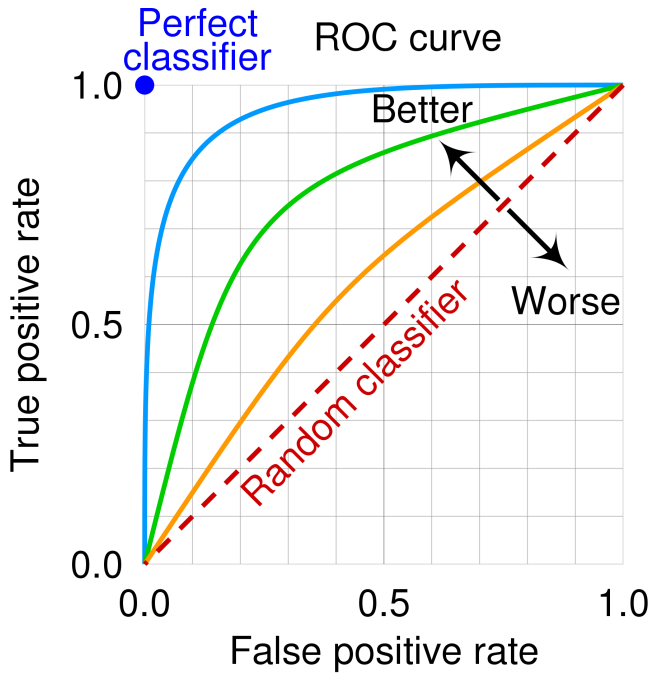
Classifier Performance - Confusion Matrix

		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 curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier



Obs: the Area Under the Curve (AUC) is often used as measure of the classifier performance, but it can be misleading if you have a 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)

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

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)

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

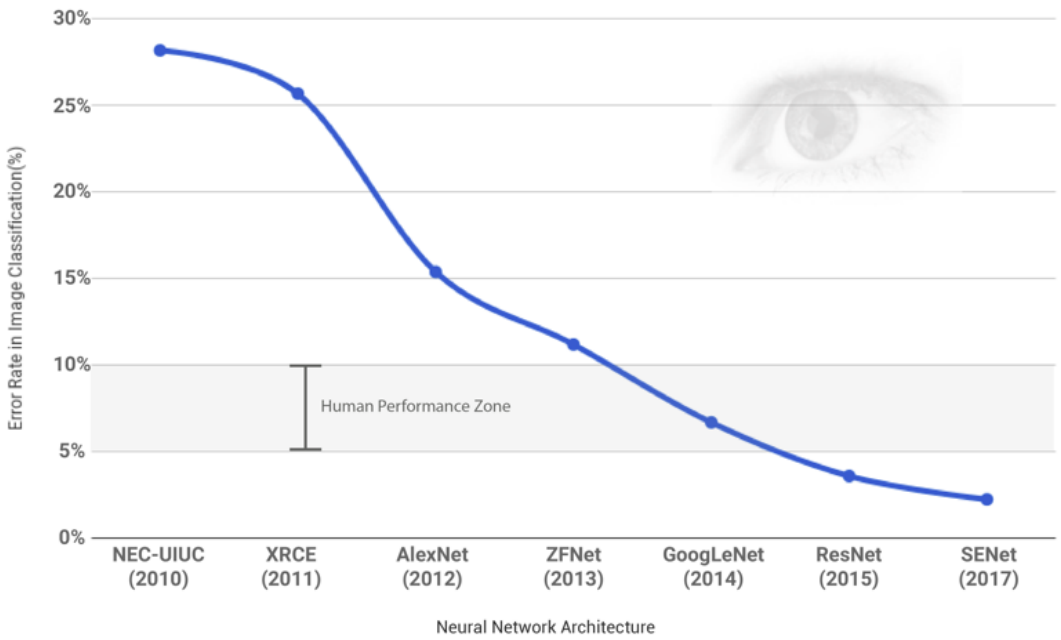
balanced accuracy (BA)

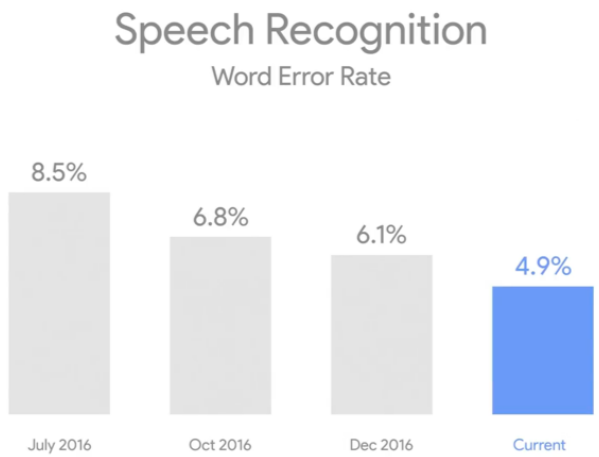
$$\mathrm{BA} = \frac{TPR+TNR}{2}$$

Why Deep Learning ?
and
Why Now ?

Why Deep Learning ?

Image and Speech Recognition perfomance (DNN versus Humans)

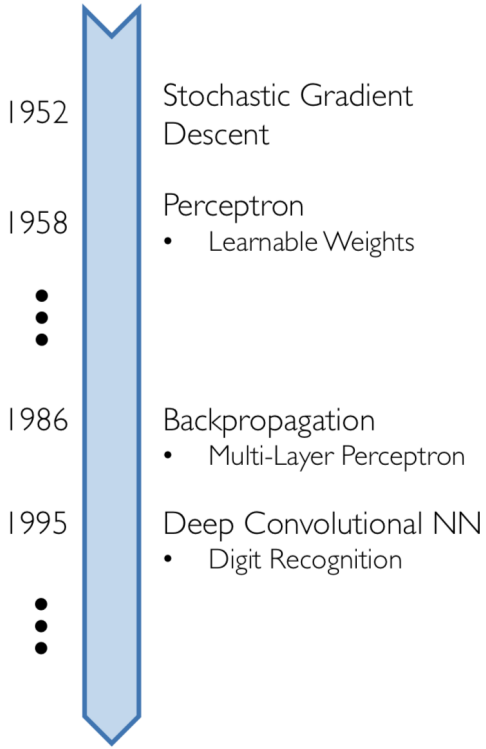




https://arxiv.org/pdf/1409.0575.pdf

Why Now ?

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

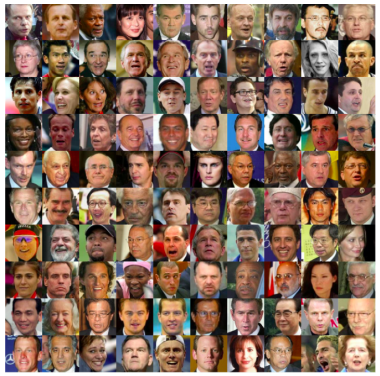


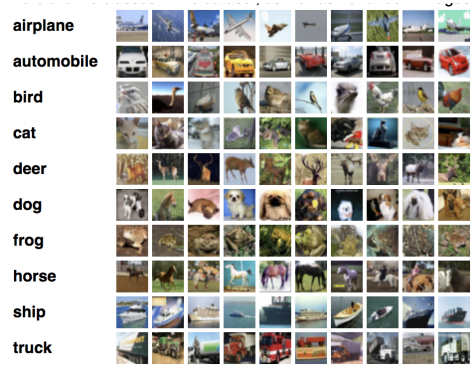
The main catalysts for the current Deep Learning revolution have been:

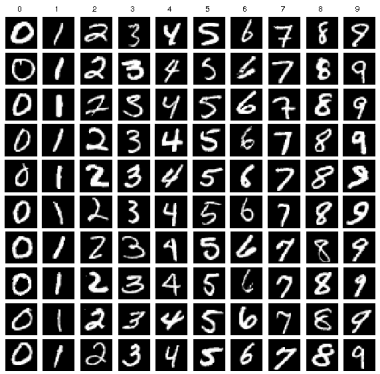
- **Software:** TensorFlow, PyTorch, Keras and Scikit-Learn
- **Hardware:** GPU, TPU and FPGA
- **Large Datasets:** MNIST

Training Datasets

Large and new open source datasets for machine learning research [18]





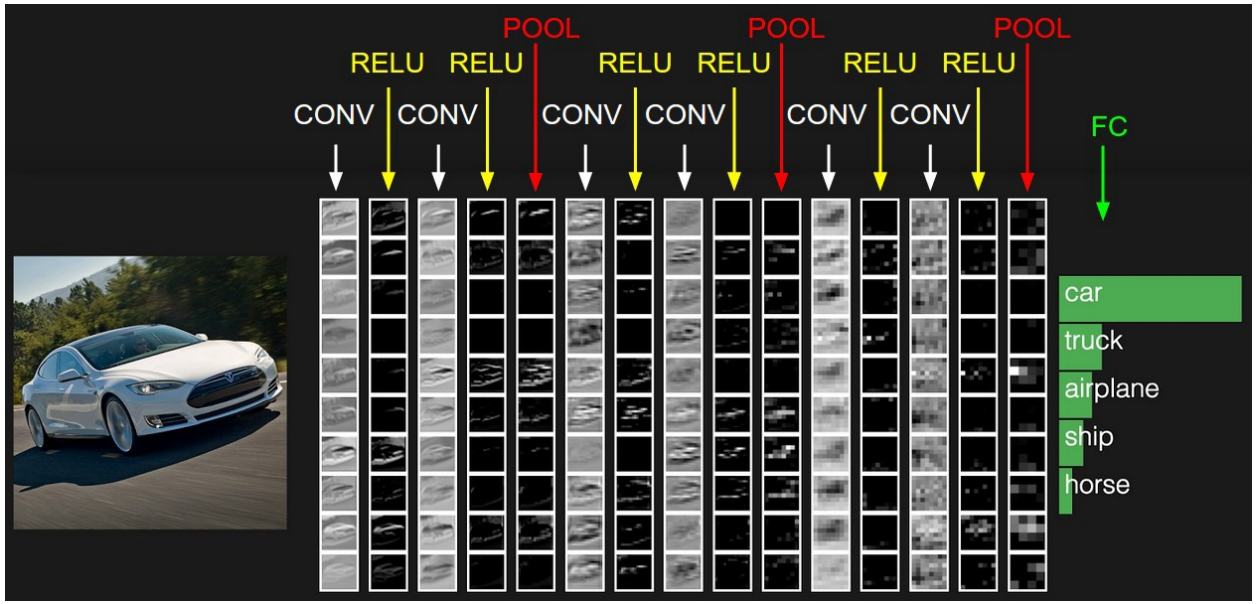


[18] https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research

Deep Learning - Need for Depth

Deep Neural Networks(DNN)

Depth allows the NN to factorize the data features, distributing its representation across the layers, exploring the compositional character of nature [19]



⇒ DNN allows a hierarchical representation of data features !

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

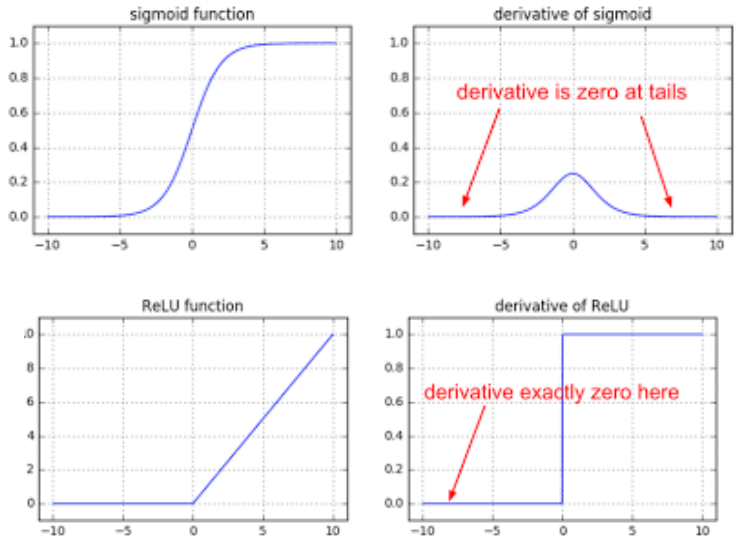
Deep Learning - Vanishing Gradient Problem

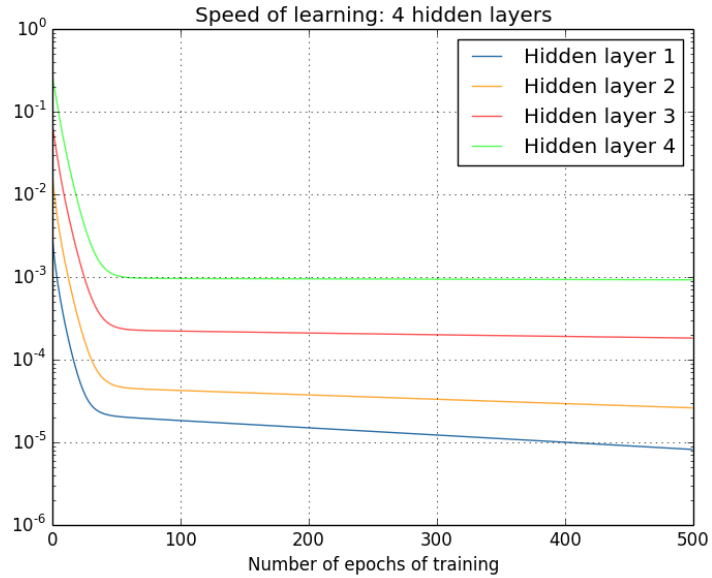
Vanishing Gradient Problem [20]

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 !!!



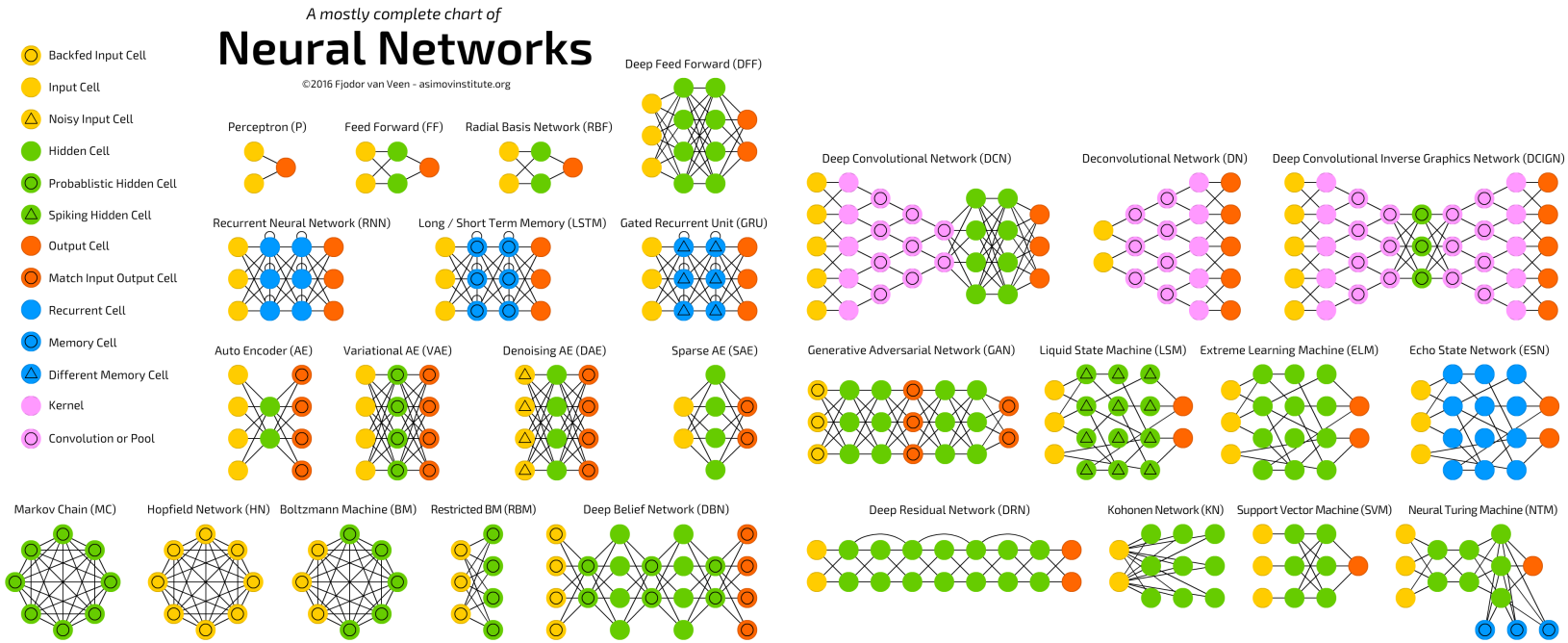


[20] http://neuralnetworksanddeeplearning.com/chap5.html

Deep Architectures and Applications

Neural Network Zoo

NN architecture and nodes connectivity can be adapted for the problem at hand [21]

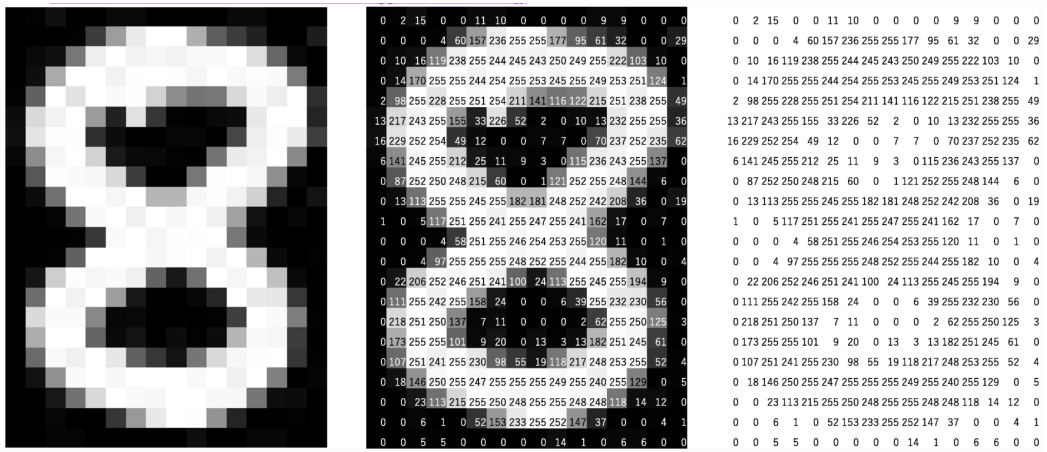


[21] http://www.asimovinstitute.org/neural-network-zoo

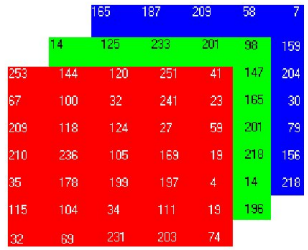
Image Data

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, each of which corresponds to a pixel brightness.



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

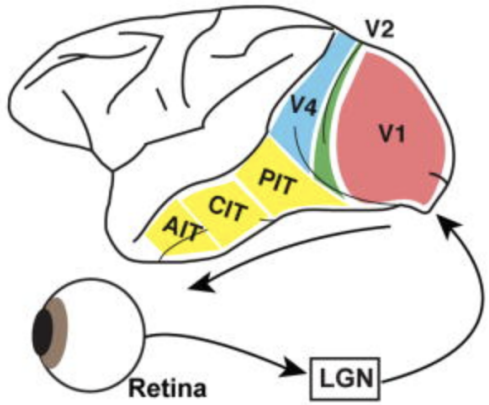


Convolutional Neural Network

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 [22] , 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.



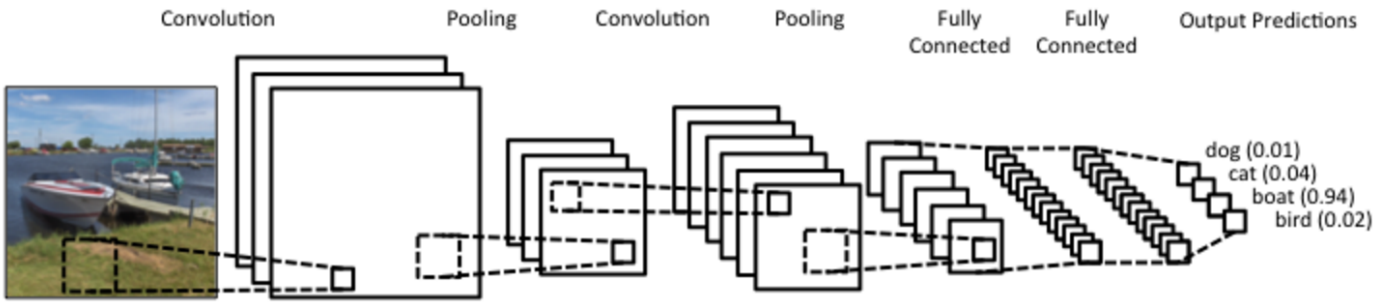
[22] How does the brain solve visual object recognition? , https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3306444

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) [23]
2. Pooling (downsampling to reduce size)
3. Fully connected (MLP for image classification)



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

CNN - Convolutional Layer

Convolutional Layer

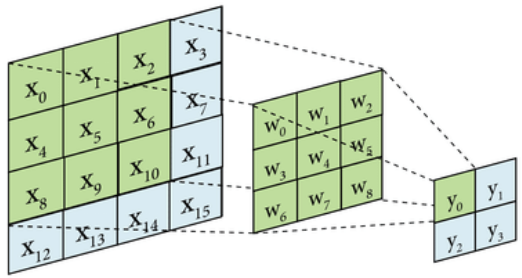
The convolutional layer [24] is the CNN core building block. A convolution can be seen as a sliding window transformation that applies a filter to extract local image features.

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)



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

Convolution Filters

An image convolution (filter) can apply an effect (sharpen, blurr), as well as extract features (edges, texture) [25]

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	

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

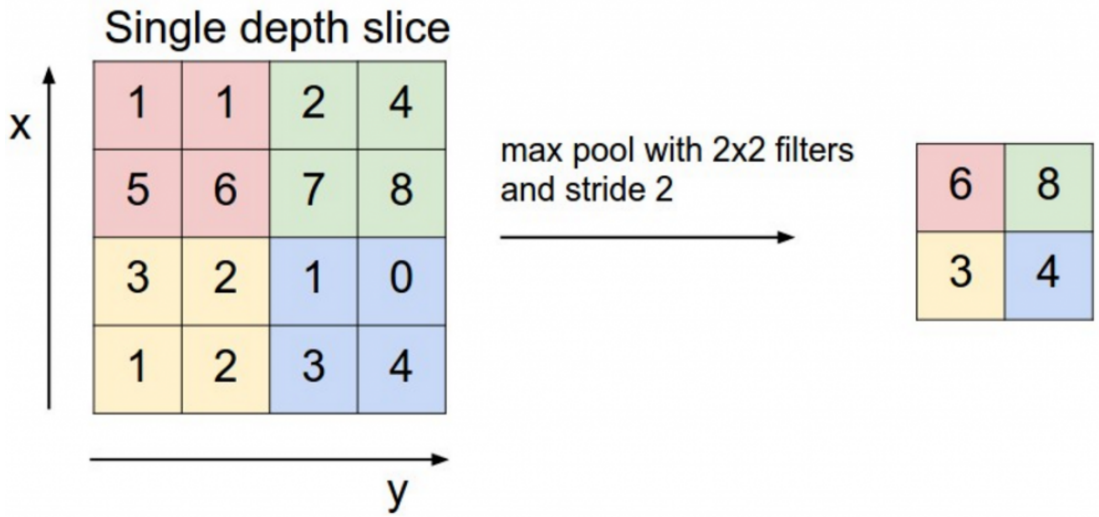
CNN - Pooling Layer

Pooling(Downsampling) Layer

The pooling (downsampling) layer [26] 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)



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

CNN - Fully Connected Layers

Fully Connected Layer

CNN chains together convolutional(filtering) , pooling (downsampling) and then fully connected(MLP) layers.

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

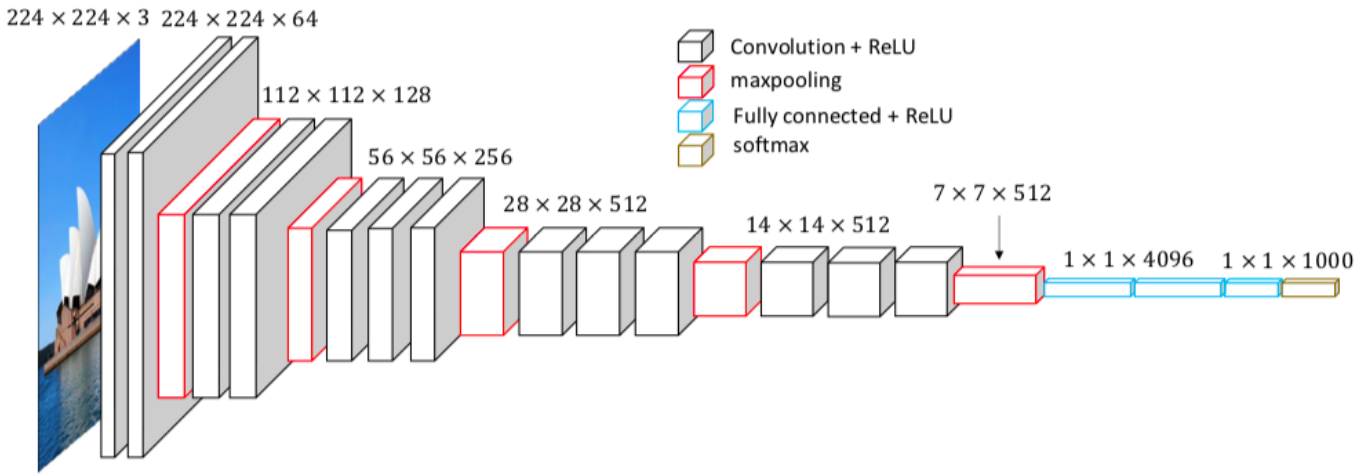
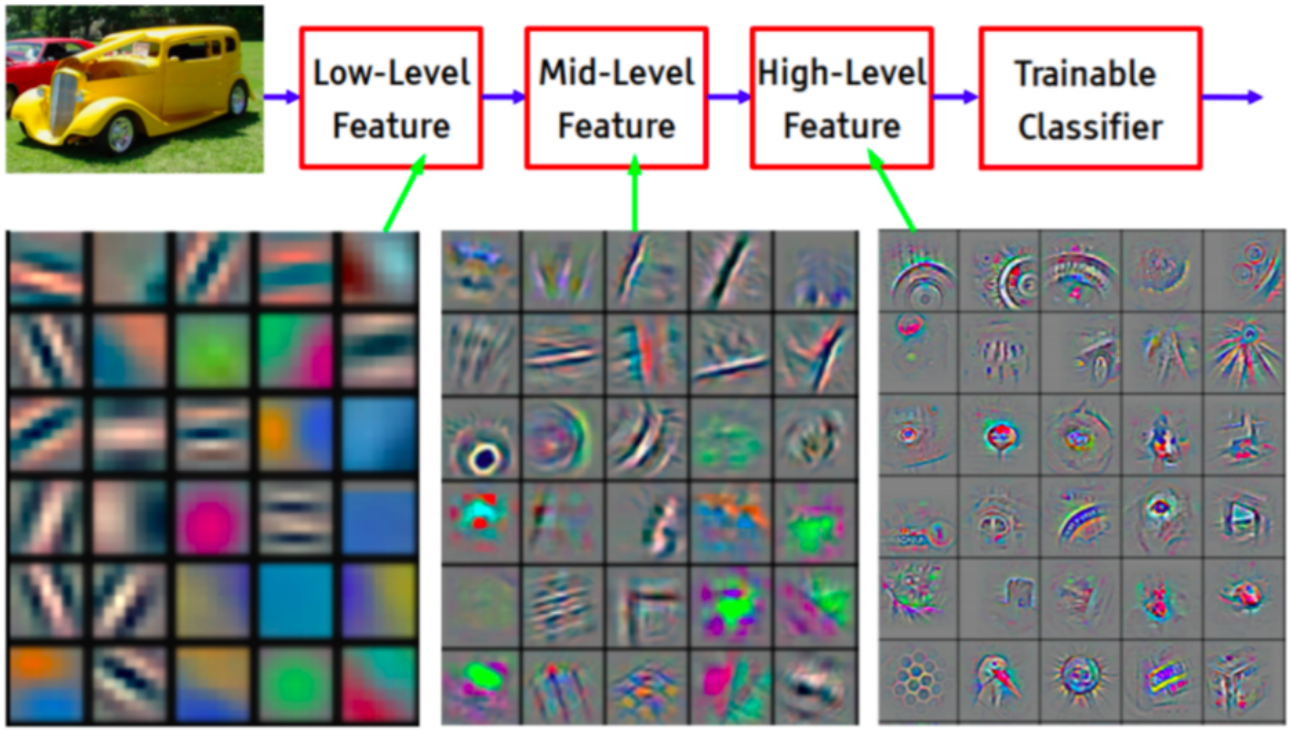


Figure 2: The architecture of VGG16 model .

CNN Feature Visualization

Each CNN layer is responsible for capturing a different level of features as can be seen from ImagiNet [27]

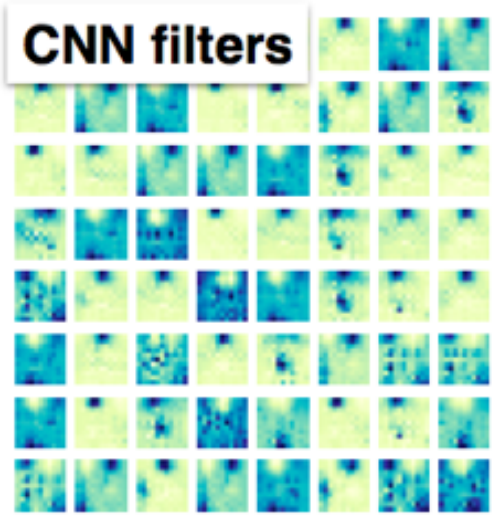


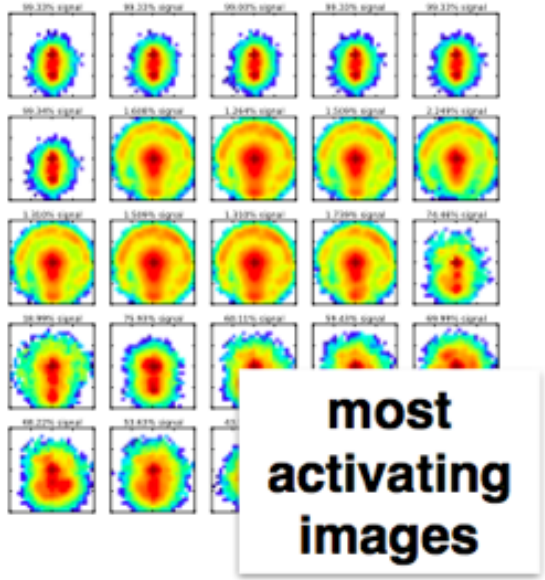
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

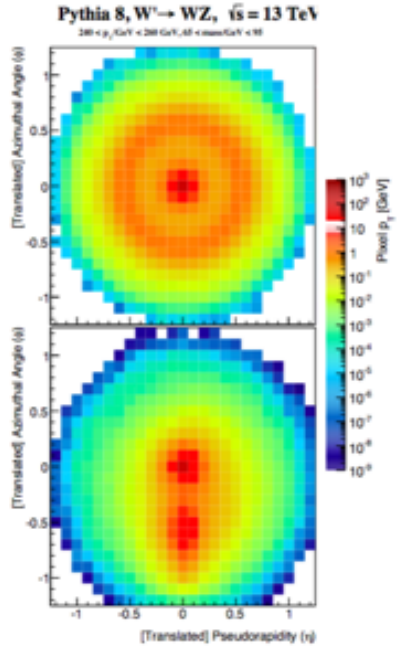
[27] https://arxiv.org/pdf/1311.2901

CNN Application in HEP: Jet ID

Jet images with convolutional nets



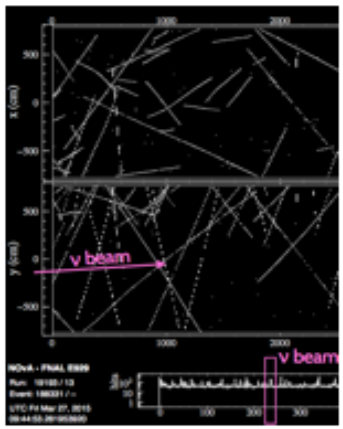


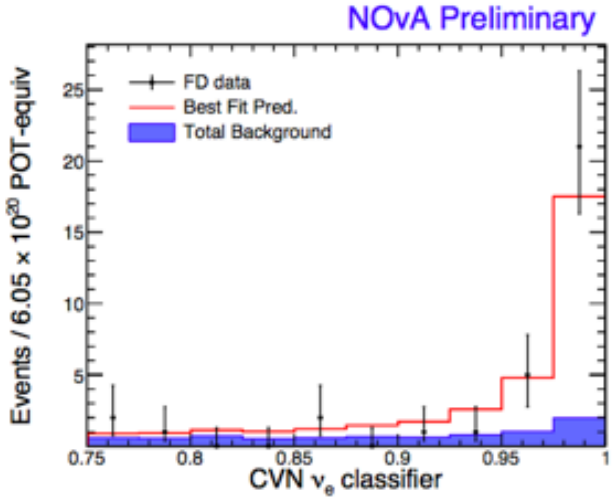


L. de Oliveira et al., 2015

CNN Application HEP: Neutrino ID

Neutrinos with convolutional nets

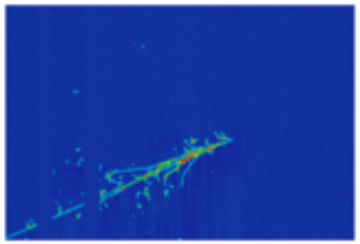


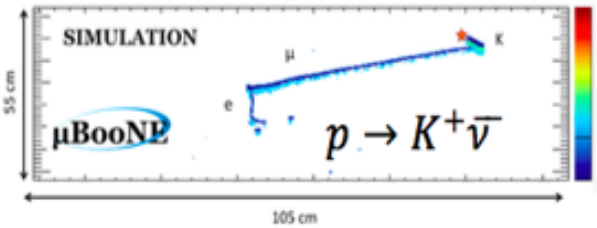


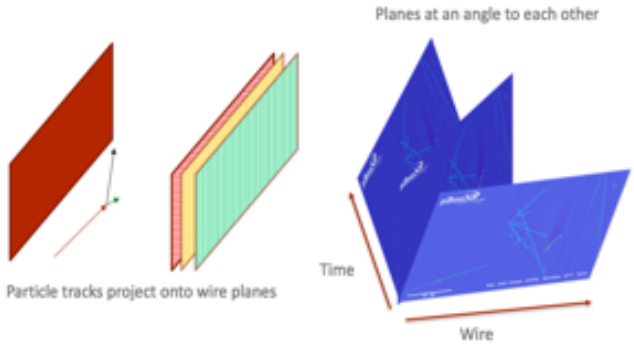
76% Purit
73% Effici

An equivalent increased exposure of 30%

Aurisiano et al. 2016



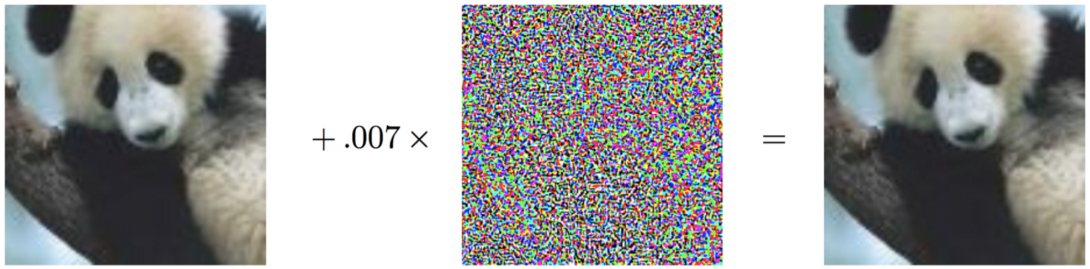




μBooNE

Adversarial Attacks on CNNs

In 2014 researchers at Google and NYU found that it was far too easy to fool a CNN !



This is like an optical illusion for the neural network. Our brain can clearly tell that both the images look like pandas !

Small perturbation in individual pixels can cause a dramatic change in the NN dot products, leading to a ’point’ in high dimensional input space that our networks have never seen before. This space is very sparse and data is concentrated in small regions. ReLu has a non-zero gradient everywhere to the right of 0, making NN more stable and faster to train. That also makes it possible to push the ReLu activation function to arbitrarily high values, leading to a trade-off between trainability and robustness to adversarial attacks.

Sequential Data

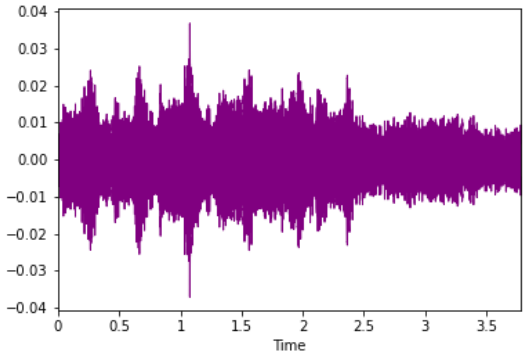
Sequential Data

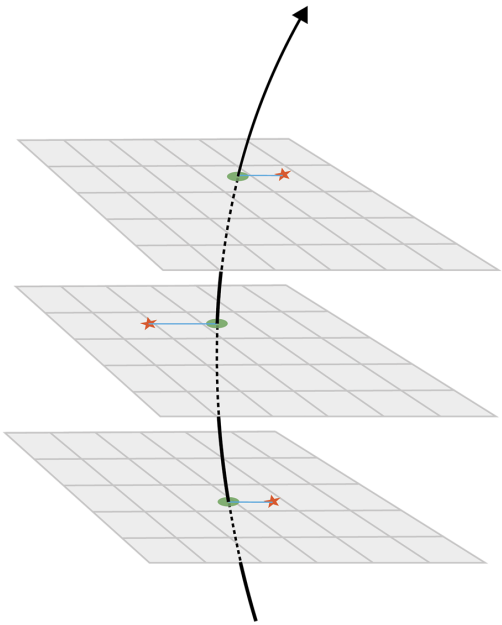
Sequential data is an interdependent data stream where data ordering contains relevant information

The food was good, not bad at all.

vs.

The food was bad, not good at all.





Obs: our 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 !

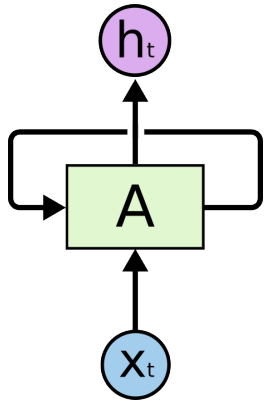
Recurrent Neural Networks (RNN)

RNNs are networks that use feedback loops to process sequential data [28]. These feedbacks allows information to persist, which is an effect often described as memory.

RNN Cell (Neuron [29])

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



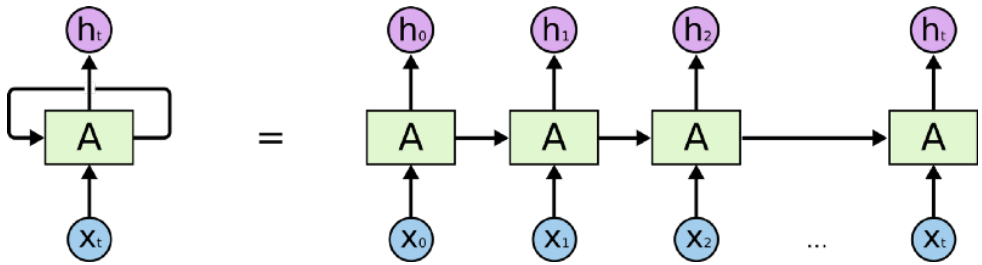
[28] https://eli.thegreenplace.net/2018/understanding-how-to-implement-a-character-based-rnn-
[29] https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks

Recurrent Neural Network(RNN)

RNNs [30] process an input sequence element at a time, maintaining in their hidden units a 'state vector' that contains information about all the past elements history.

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 !

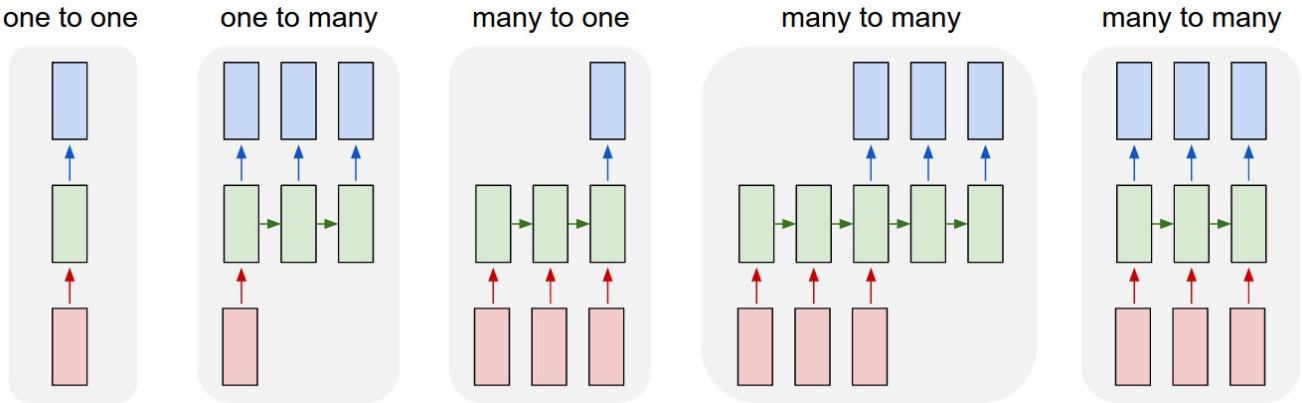
[30] http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-
https://towardsdatascience.com/rnn-recurrent-neural-networks-how-to-successfully-model-seque

Recurrent Neural Network(RNN)

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 [31]

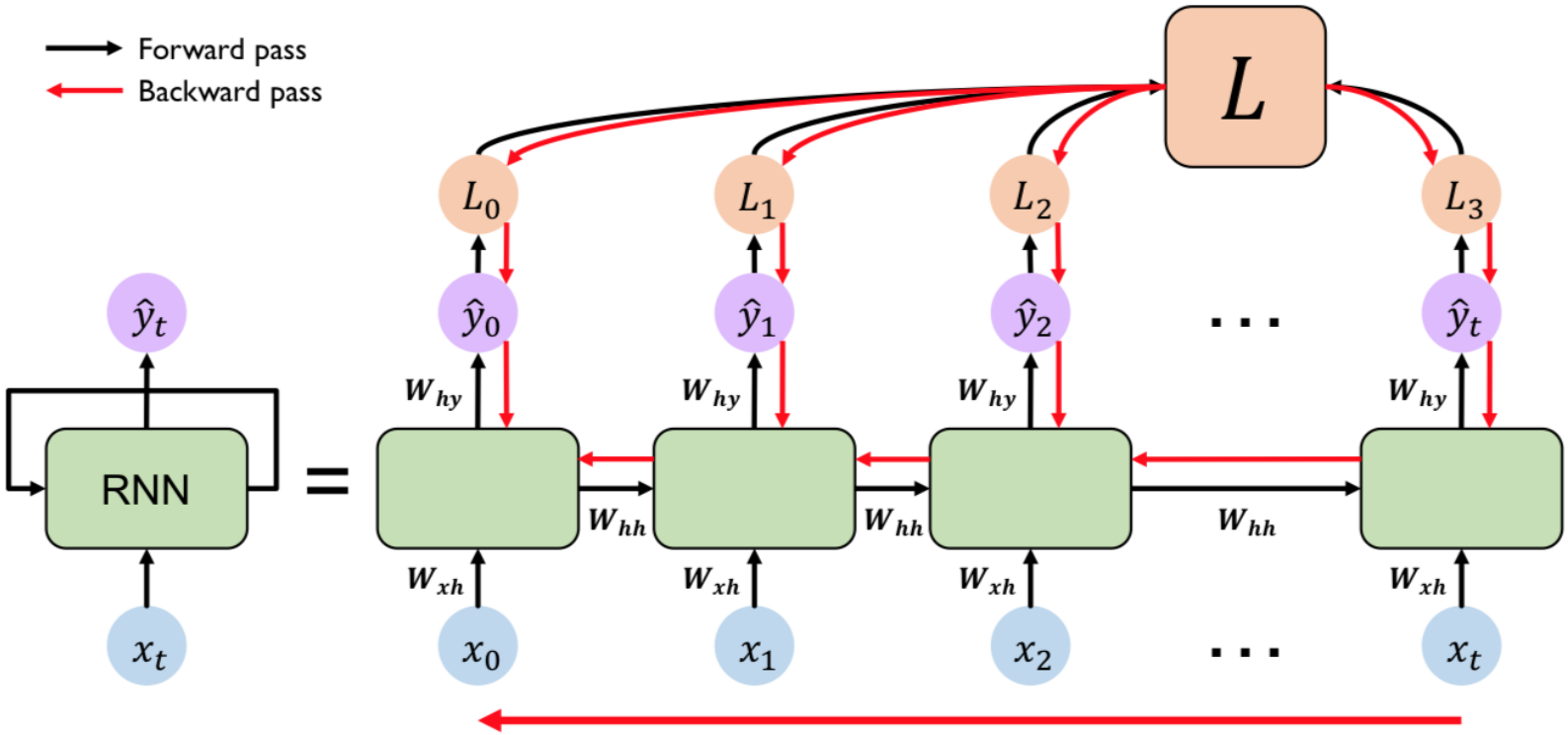


There are no constraints on the sequences lengths because the recurrent transformation can be applied as many times as necessary

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

Recurrent Neural Network(RNN)

The RNN computational graph and information flow



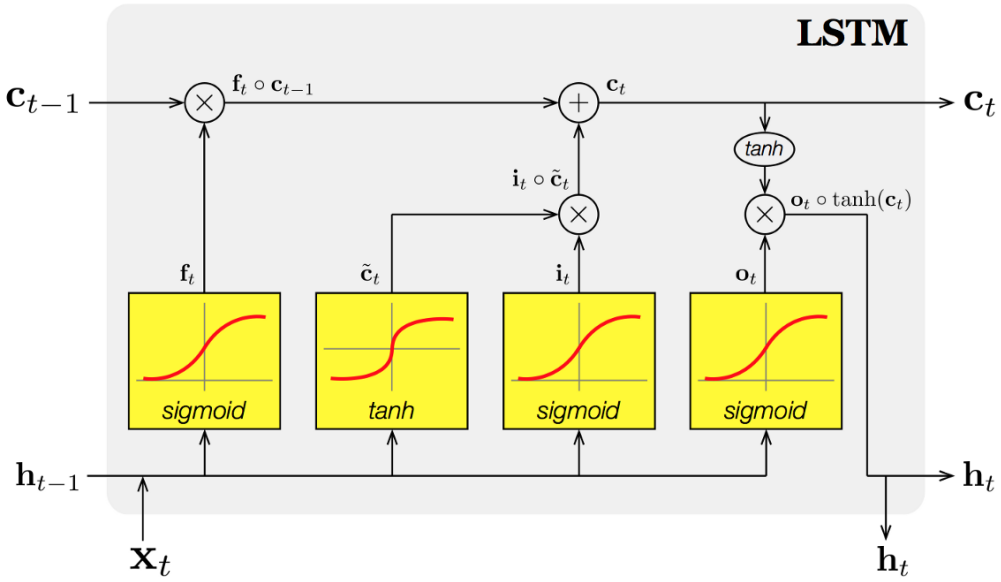
The same set of weights are used to compute the hidden state and output for all time steps (shared weights)

Long Short Term Memory(LSTM)

Long Short-Term Memory (LSTM) Network

LSTM [32] 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

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

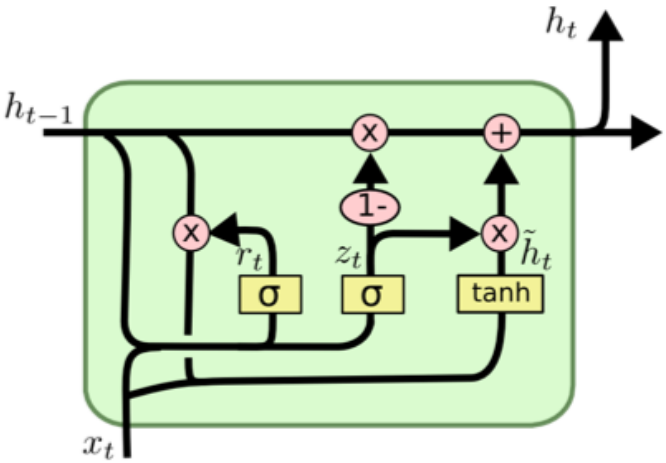
[32] http://colah.github.io/posts/2015-08-Understanding-LSTMs
https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-reme

Gated Recurrent Network (GRU)

Gated Recurrent Network (GRU)

GRU [33] 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



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

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

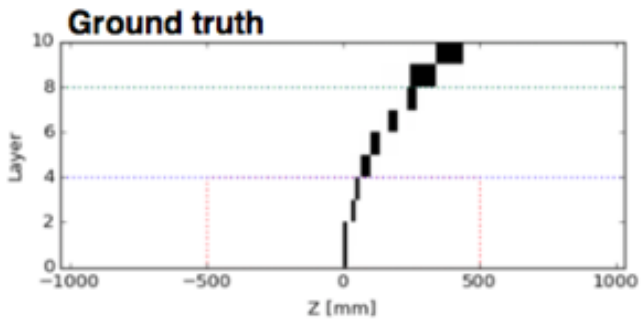
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

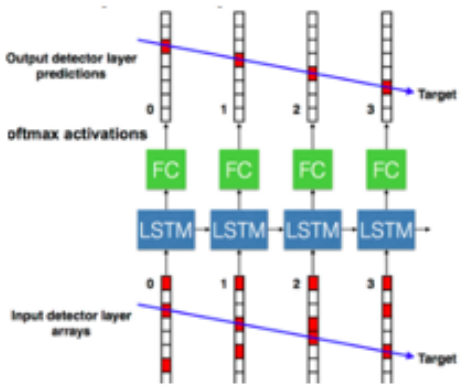
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

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

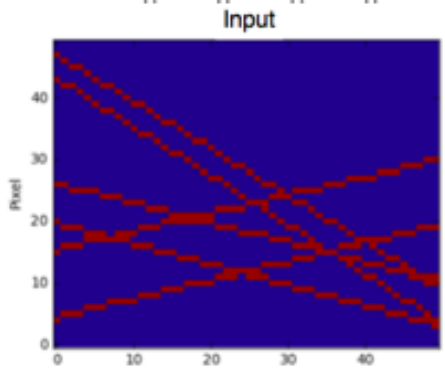
LSTM Application in HEP: Tracking

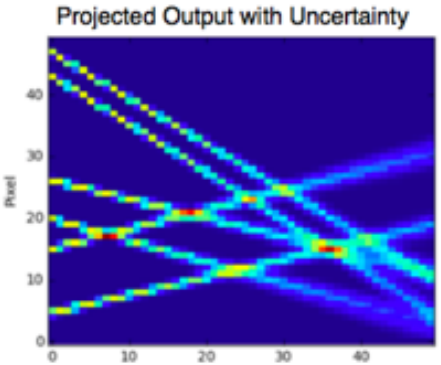
Tracking with recurrent neural networks (LSTM) [34]





Time dimension (state memory)



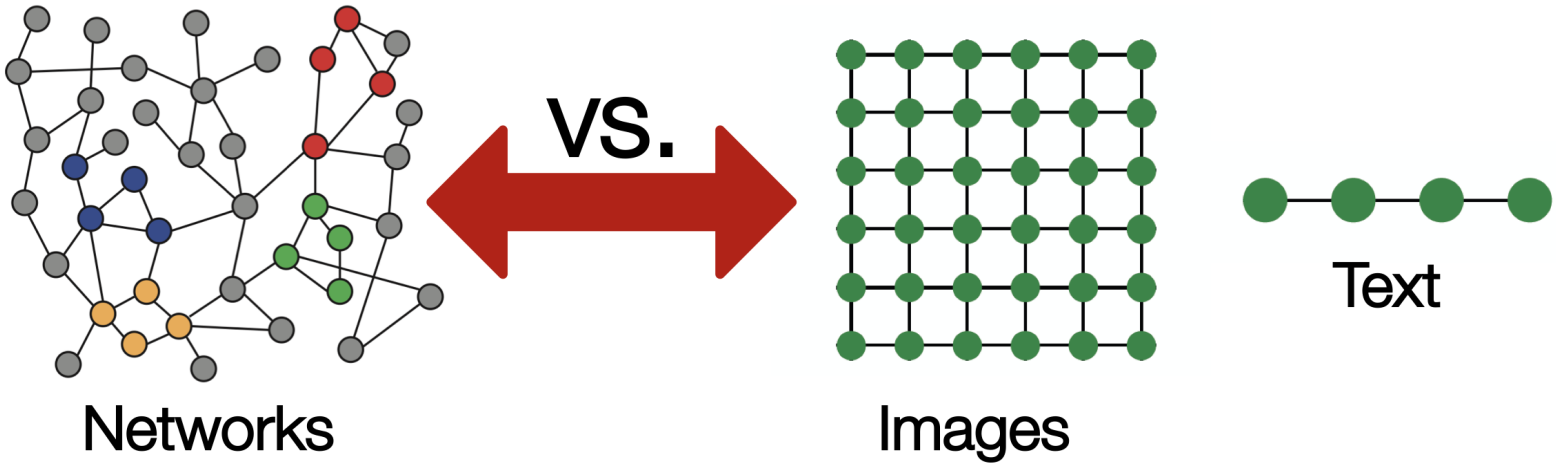


[34] https://heptrkx.github.io

Graph Data

Graph Structutred Data

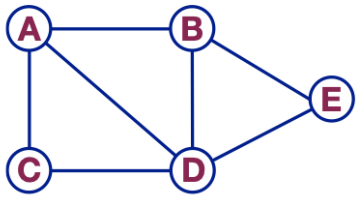
Sequences (linear graph) and images (regular grid graph) are special types of data structures called graphs(networks) !



Graph is defined by its nodes(vertices) and edges(links): $G(N, E)$

Graph Representation

To apply a NN to a given problem it's necessary to encode the DATA in a mathematical representation. A naive aproach would be to represent graph structured data as an augmented adjacency matrix with node features information and use it as an MLP input. [35]



	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:

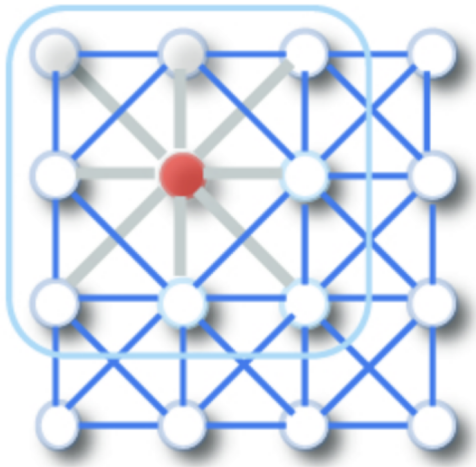
- Adjacency matrix depends on arbitrary node labeling (need permutation invariance)
- Adjacency matrix can became too large and sparse (need compact representattion)
- Matrix size depend on number of nodes (NN takes fixed input size)

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

Graph Neural Network

Convolution on Graphs

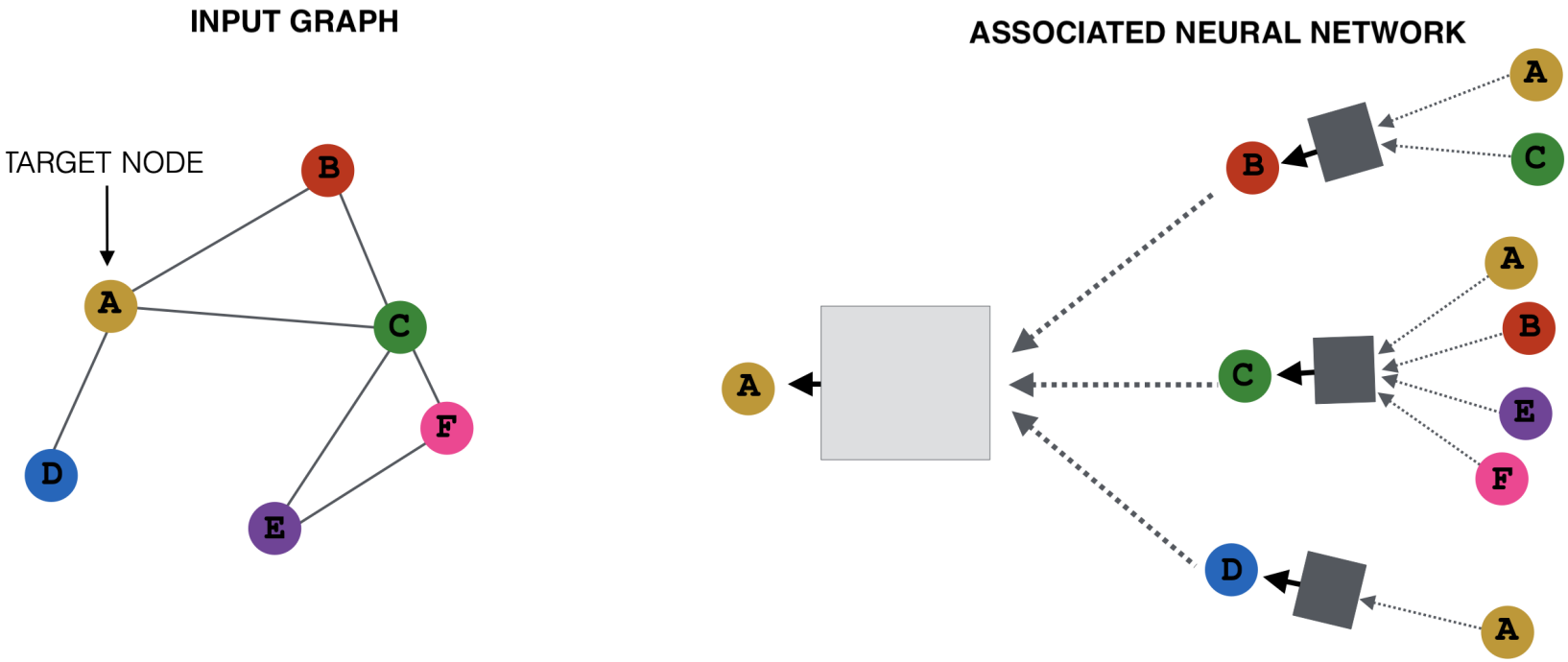
Inspired by convolution on images, which are grids of localized points, we can define a convolution on a graph where nodes have no spatial order. The convolution on graphs is often referred as a neighborhood **aggregation** or **message passing**.



Graph Neural Network

Node Neural Network

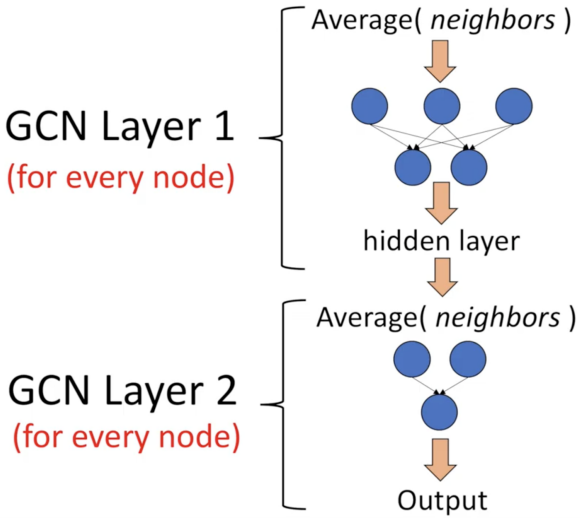
Each node [36] aggregates information from its neighbours defining its own neural network (MLP) . This aggregation function must be permutation invariant !



[36] 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 [37] 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

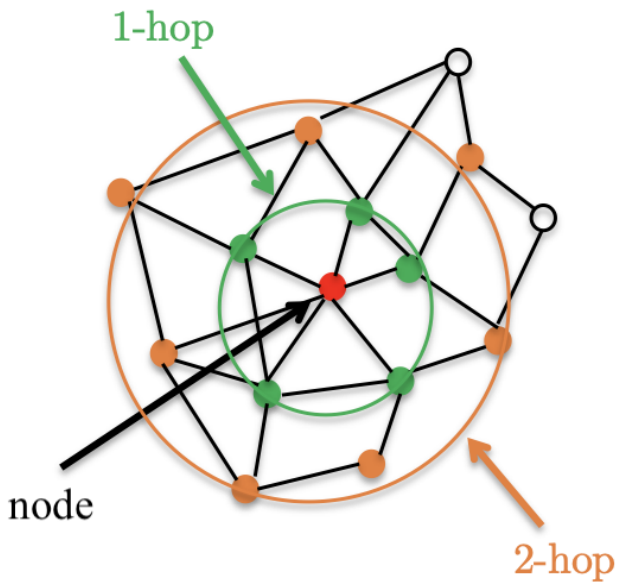
Obs: The MLP inputs are the aggregated node features, so it doesn't depend on number of graph nodes !

[37] 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

GNN Hidden Layers

The number of hidden GNN layers is related to the size of the node neighborhood. The larger the number of layers the farther away messages gets passed



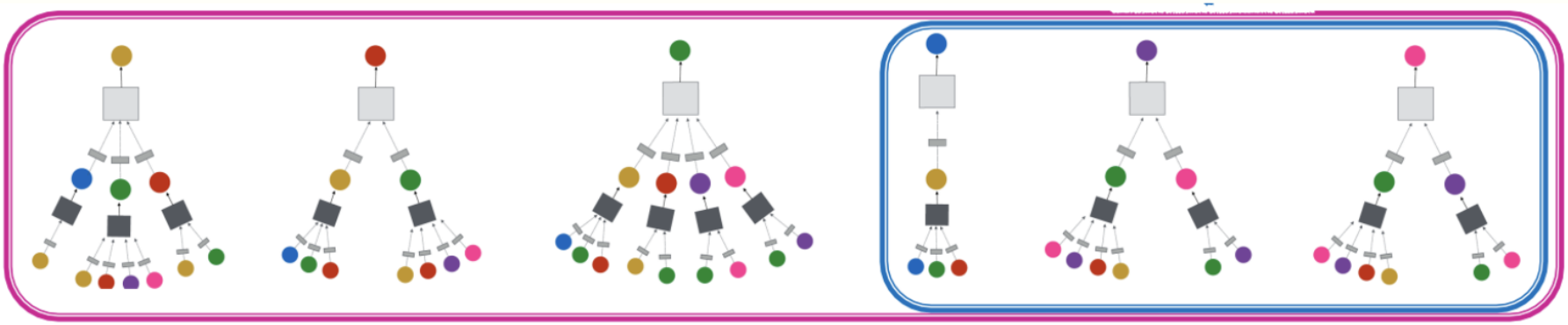
Obs: GNNs with few hidden layers are enough to exhaust small graphs with a few hops (node neighborhoods)

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.

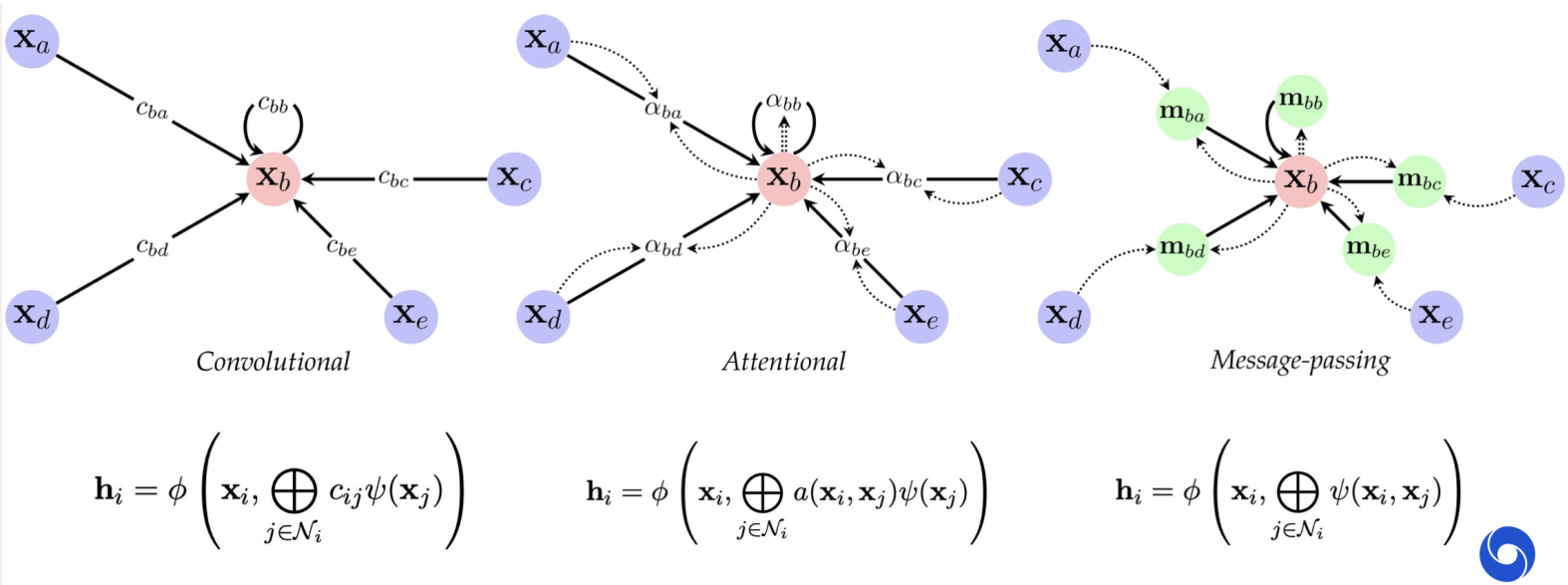


Graph Neural Network

Three Flavours of GNNs

GNNs can grouped [39] according to following flavours:

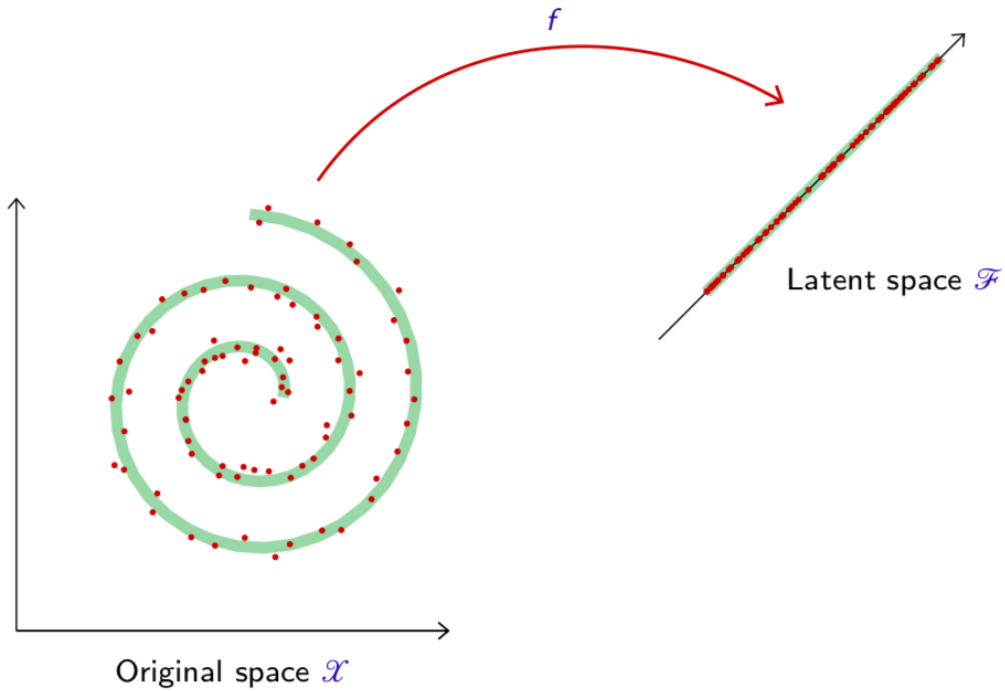
- **Convolutional:** fixed weights coeficients c_{ij}
- **Attention Based:** learnable attention weights calculated as $\alpha_{ij} = a(x_i, x_j)$
- **Message Passing:** general messages $m_{ij} = \psi(x_i, x_j)$ sent across edges



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

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



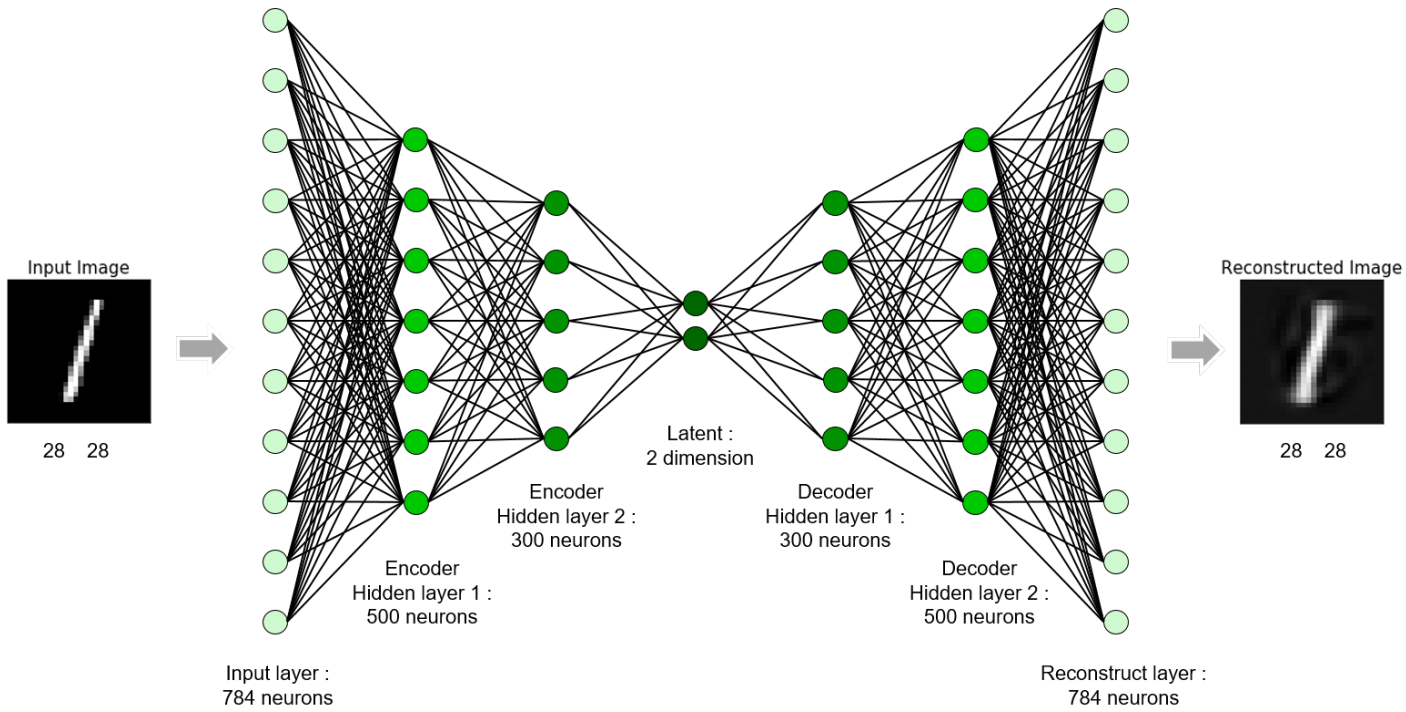
Traditionally, autoencoders were used for dimensionality reduction and denoising. Recently autoencoders are being used also in generative modeling

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 $y = g(l)$.



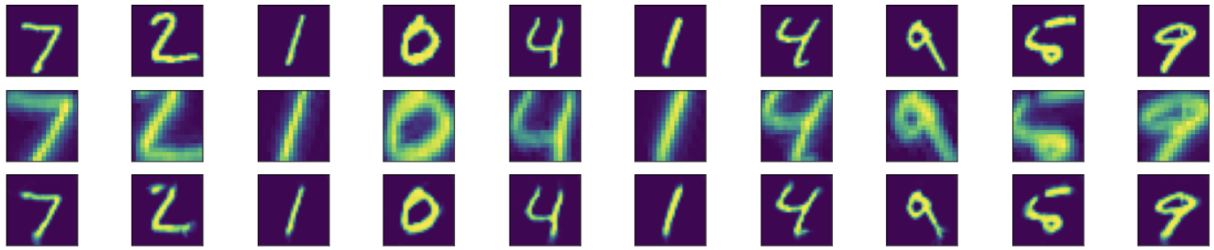
⇒ Understanding is data reduction

Convolutional Autoencoder(CAE)

Convolutional Autoencoder(CAE)

A CAE [42] 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)



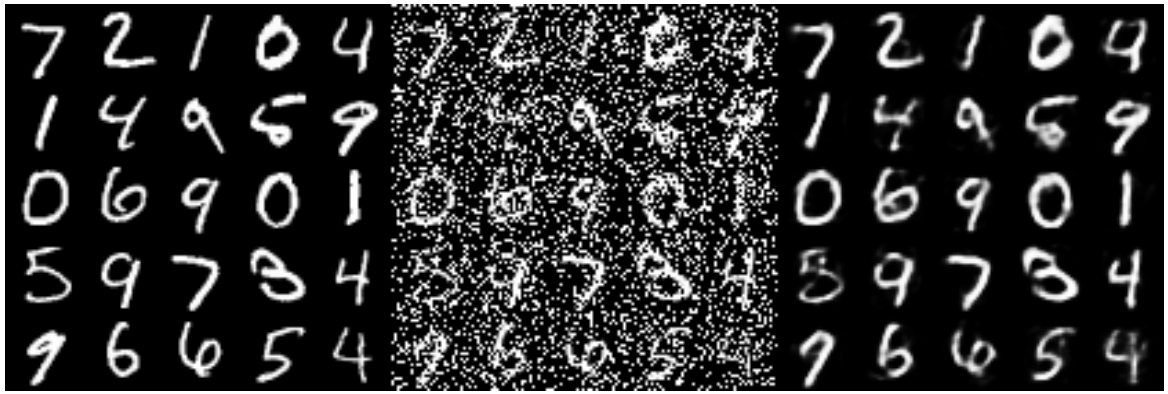
OBS: CAE learns the image borders have no information and chops the central part.

[42] 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

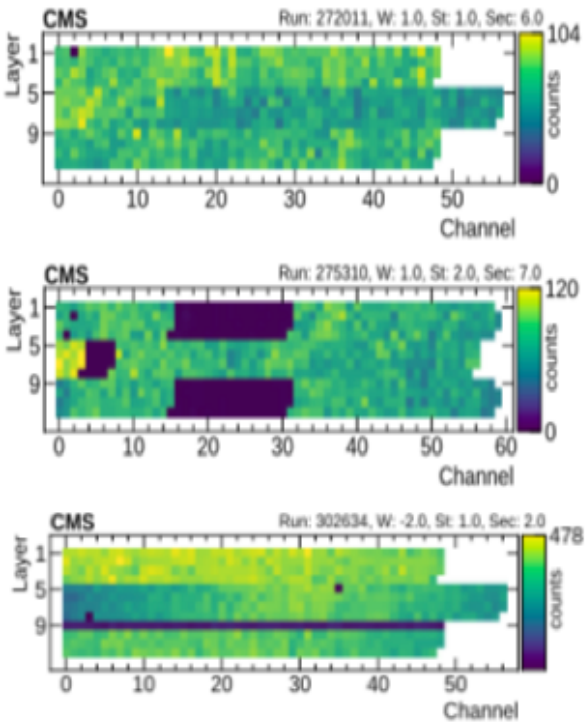


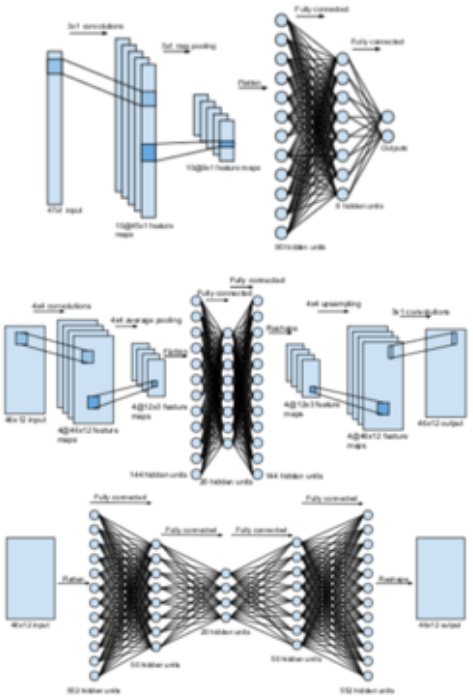
Autoencoder Application in HEP: DQM

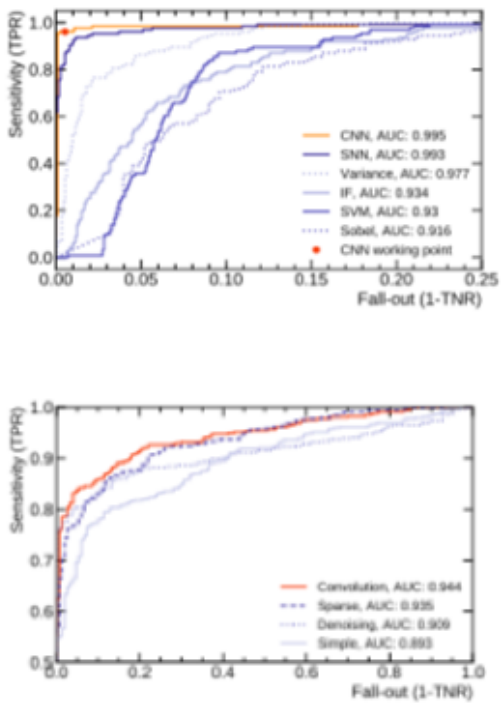
AE can be used for anomaly detection by training on a single class , so that every anomaly gives a large reconstruction error

Detector Quality Monitoring (DQM)

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





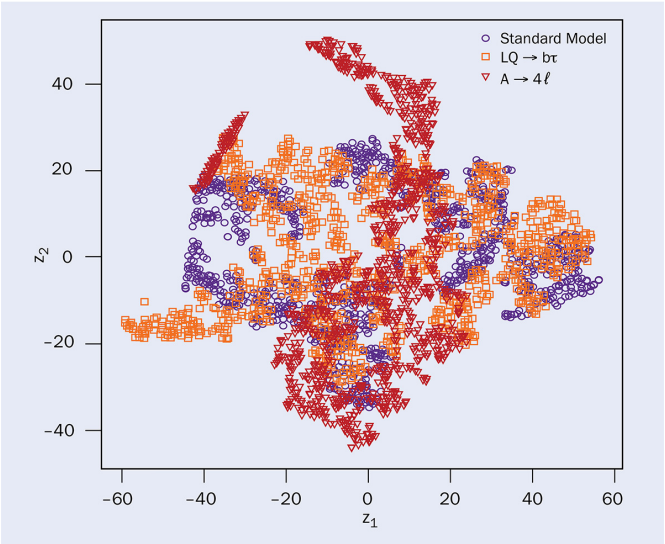


Autoencoder Application in HEP: Anomaly Detection at LHC (BSM)

AE can watch for signs of new physics at the LHC that have not yet been dreamt up by physicists [43]

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.

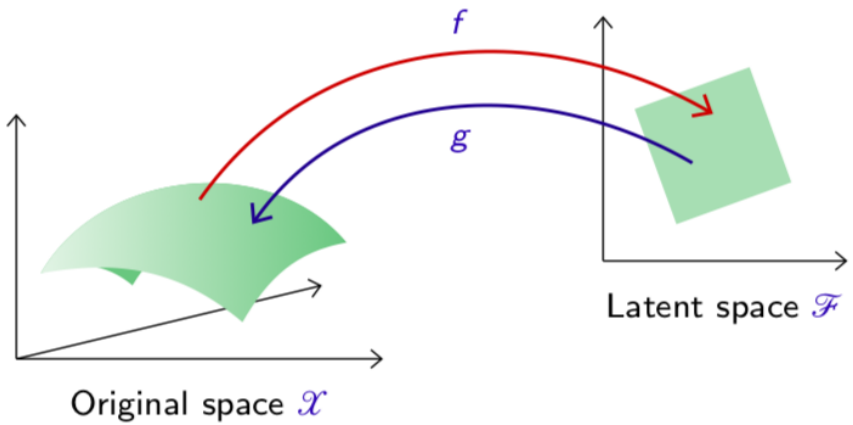


Outliers could go into a data stream of interesting events to be further scrutinised

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

Generative Autoencoder

An autoencoder combines an encoder f from the original space $\mathscr{X}$ to a latent space $\mathscr{F}$, and a decoder g to map back to $\mathscr{X}$, 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)

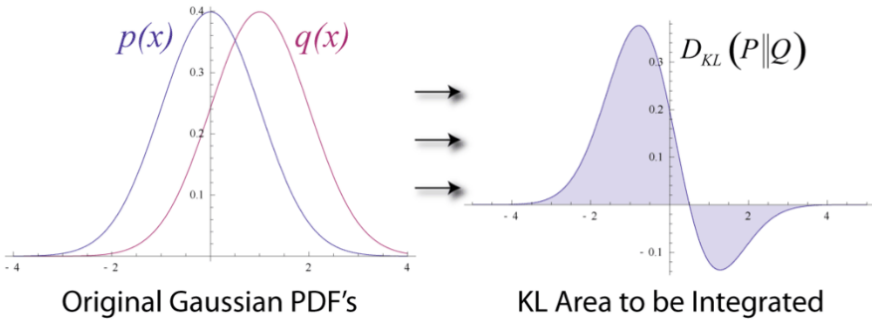
Variational Autoencoder(VAE)

A VAE [45] is an autoencoder with a loss function penalty Kullback–Leibler (KL)divergence [46] [47] 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 measure of how one probability distribution differs from a second, reference distribution

Kullback–Leibler Divergence (Relative Entropy)

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



Original Gaussian PDF's KL Area to be Integrated

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

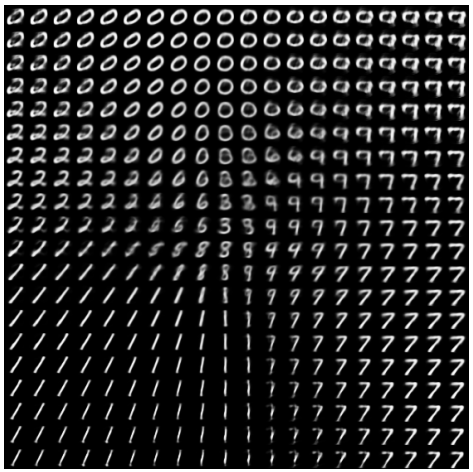
Variational Autoencoder(VAE)

VAE Loss

The VAE loss [48] 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.

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

Bellow we have an example of a set of a VAE generated numbers obtained by gaussian sampling a 2D latent space



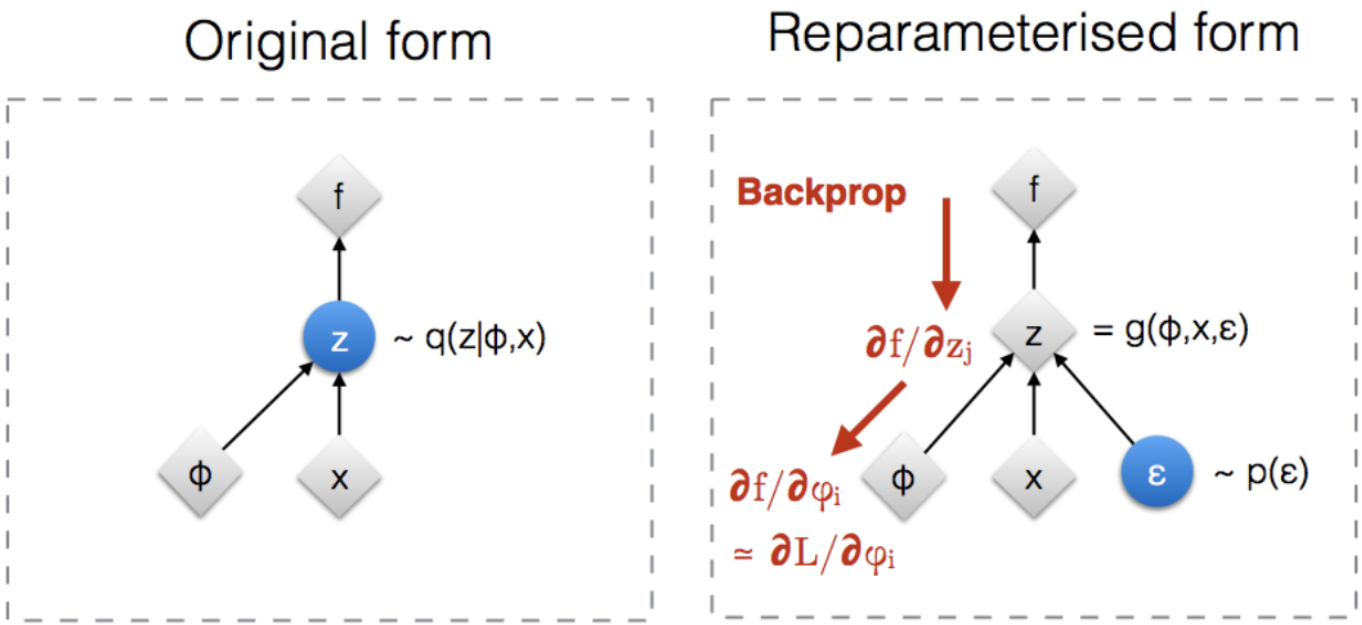
KL loss is a regularization term that helps learning ”well formed”latent space

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

Variational Autoencoder(VAE)

VAE Reparametrization Trick

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



Variational Autoencoder(VAE)

Feature Disentanglement(Factorization) in VAE

49

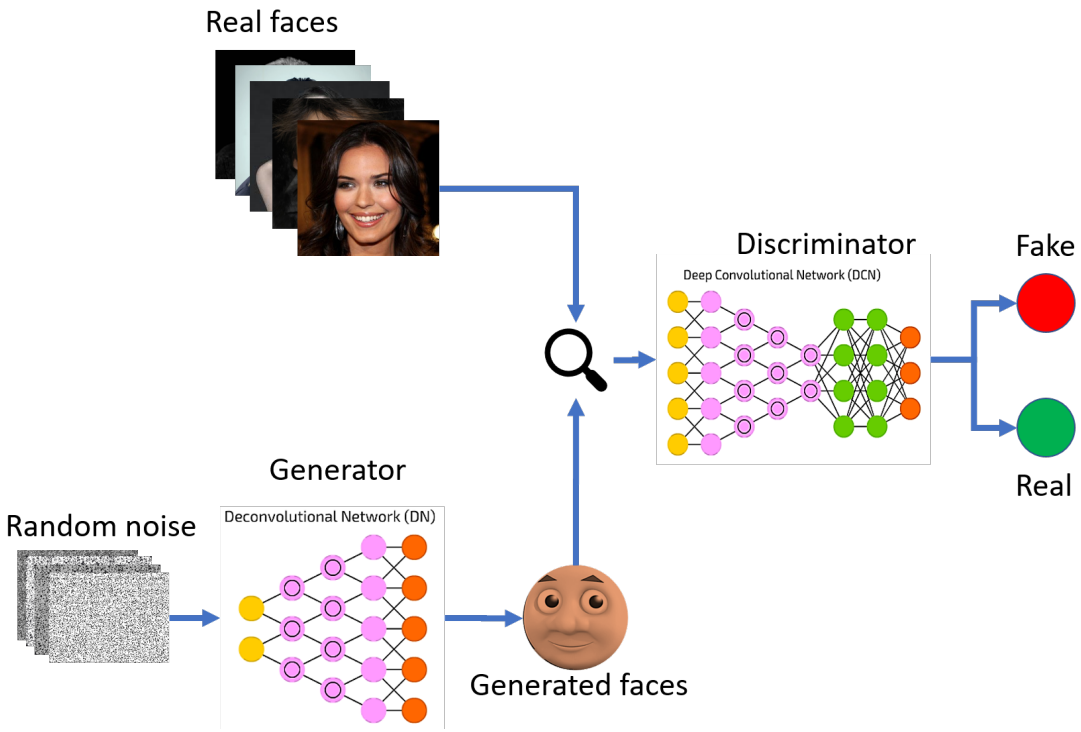
(a) Emotion (smile) (b) Gender (c) Azimuth

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

Generative Adversarial Network(GAN)

Generative Adversarial Networks (GAN)

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

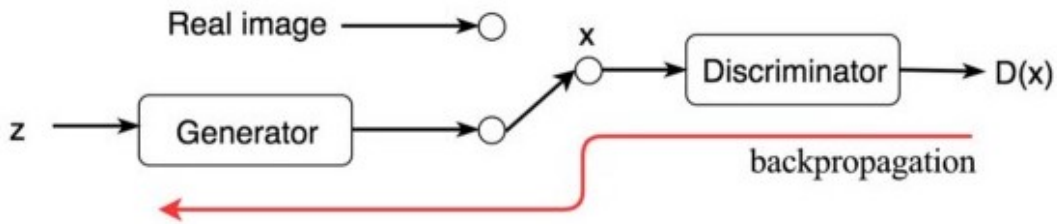


Generative Adversarial Network(GAN)

GAN adversarial training [50] 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

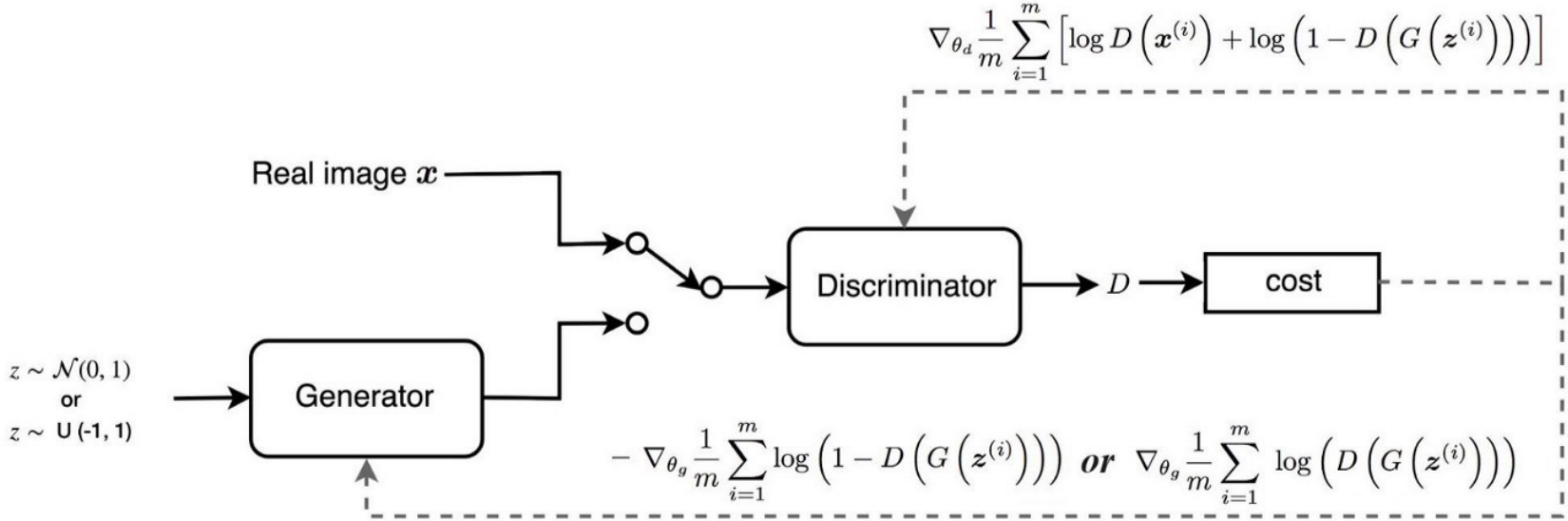
1. The discriminator receives as input samples synthesized by the generator and real data . It is trained just like a classifier, so if the input is real, we want output=1 and if it's generated, output=0
2. The generator is seeded with a randomized input that is sampled from a predefined latent space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



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

Generative Adversarial Network(GAN)

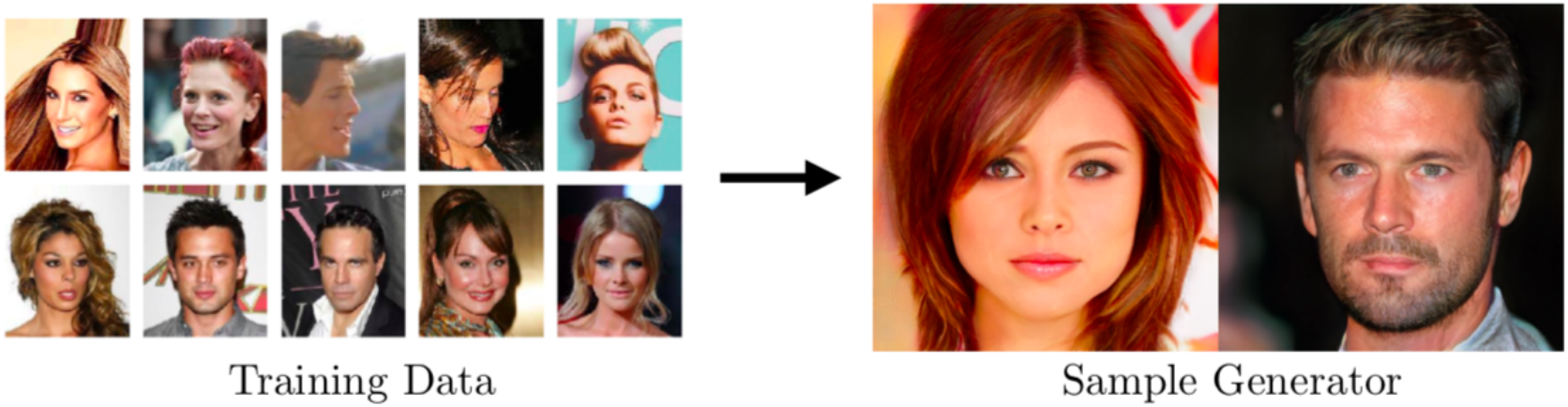
GAN training algorithm is ilustrated in more detail by the diagram bellow [51]

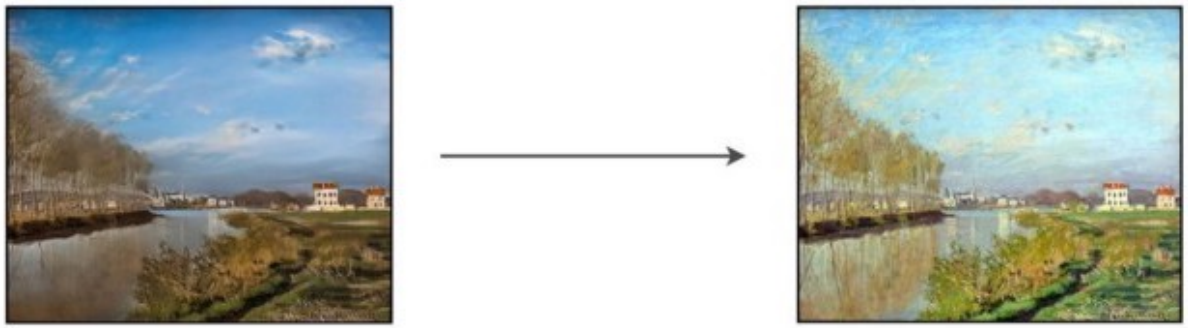


[51] https://medium.com/@jonathan_hui

Generative Adversarial Network(GAN)

GANs are quite good on faking celebrities images [52] or Monet style painitings [53] !



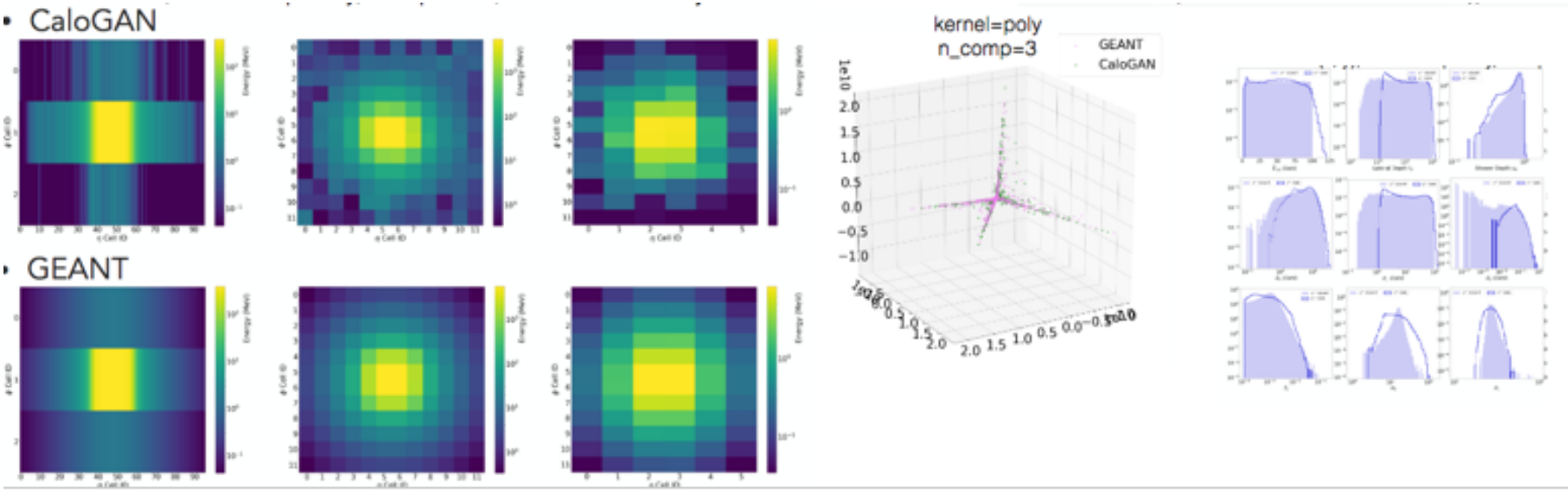


[52] https://research.nvidia.com/publication/2017-10_Progressive-Growing-of
[53] https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

GAN Application in HEP: MC Simulation

CaloGAN

Simulating 3D high energy particle showers in multi-layer electromagnetic calorimeters with a GAN [54]



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

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)

The End !!!