Introduction To Machine Learning for HEP

A. Sznajder

UERJ
Instituto de Fisica

January - 2019

Outline

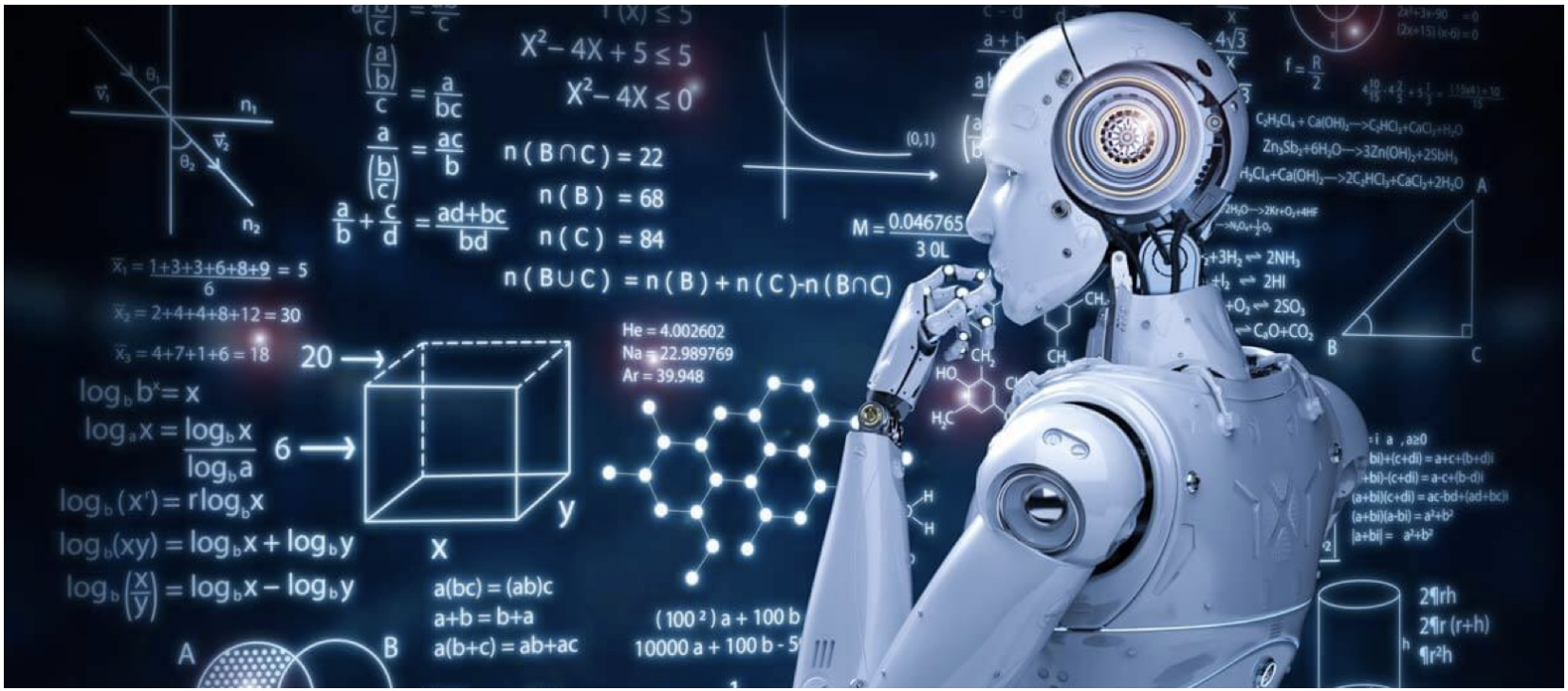
Machine Learning Software

- General ML library (Python):
  1. `https://scikit-learn.org/stable`
- Deep learning libraries:
  1. `https://www.tensorflow.org` (Google)
  2. `https://pytorch.org` (Facebook)
  3. `https://www.microsoft.com/en-us/cognitive-toolkit` (Microsoft)
  4. `https://mxnet.apache.org` (Apache)
  5. `https://github.com/Theano/Theano` (Univ.Montreal)
- High level deep learning API:
  1. `https://keras.io` (Tensorflow ,CNTK,Theano)
  2. `https://docs.fast.ai` (Pytorch)
- Converting ROOT trees to Python numpy arrays or panda data frames
  1. `https://github.com/scikit-hep/root_numpy`
  2. `https://github.com/scikit-hep/root_pandas`

Bibliography

- Neural Networks and Deep Learning, M.Nielsen : `http://neuralnetworksanddeeplearning.com`
- EPFL EE559 class, F.Fleuret : `https://fleuret.org/ee559/`
- Stanford University CS231 class, A.Karpathy : `http://cs231n.stanford.edu`
- Stanford University CS229 class, A.Ng : `http://cs229.stanford.edu`
- Introduction to machine learning, Murray: `http://videolectures.net/bootcamp2010_murray_iml`
- Machine Learning HEP School, A.Rogozhnikov: `https://indico.cern.ch/event/497368`
- Machine Learning , M.Kagan: `https://indico.cern.ch/event/726959`
- Pisa School on Future Colliders, S.Valecorsa: `https://indico.cern.ch/event/669093`
- VBSCOST Ljubljana ML Training, M.Pierini: `https://indico.cern.ch/event/775229/contributions`
- ML4HEP at Univ.Zurich, G.Kasieczka: `https://indico.cern.ch/event/757837/contributions`

What is Machine Learning ?



Introduction to 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 algorithms are capable of making predictions without being explicitly programmed to do so. In HEP one is familiar mainly support vector machines (SVN), boosted decision trees (BDT) and neural networks (NN)

1) Neural Network Topologies

- Feed Forward NN
- Recurrent NN

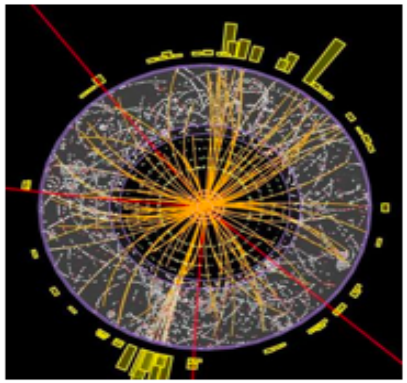
2) Learning Paradigms

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

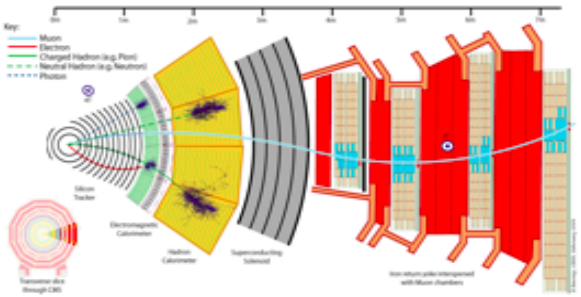
3) Neural Networks Architectures

- Multilayer Perceptron(MLP)
- Convolutional Network (CNN)
- Recurrent Network (RNN)
- Long Short Time Memory(LSTM)
- Residual Network (ResNet)
- Variational Autoencoder(VAE)
- Generative Adversarial Network(GAN)

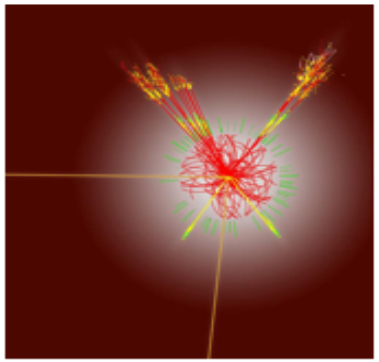
Machine Learning in HEP



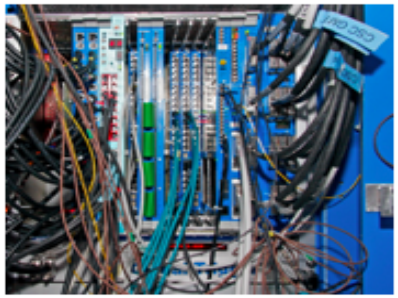
Deep Kalman, LSTMs, GNN



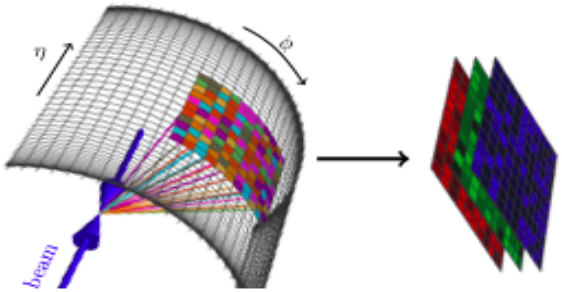
Generative Models, Adversarial Networks



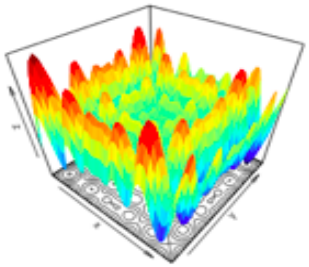
FCN, Recurrent, LSTMs



Deep ML +FPGA



Convolutional NN



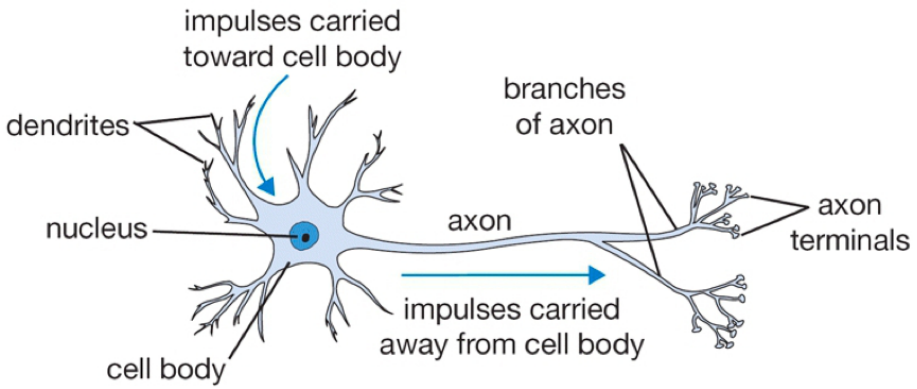
Multiobjective Regression

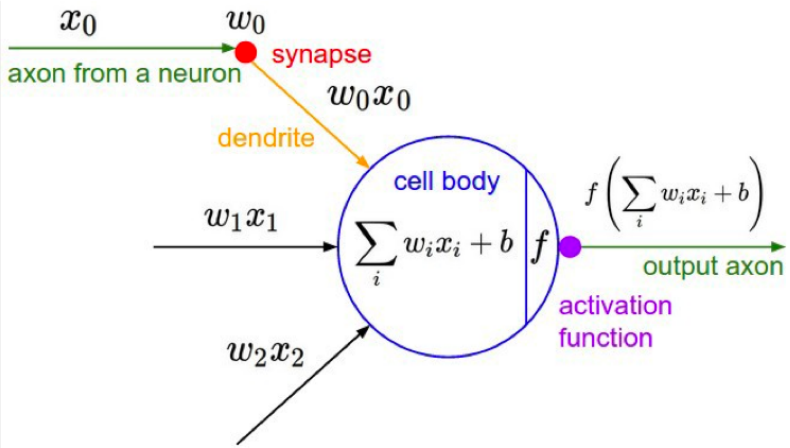
18/18

28

Neural Networks

Artificial Neural Networks (NN) are computational models vaguely inspired [1] 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, ...

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

Artificial Neuron

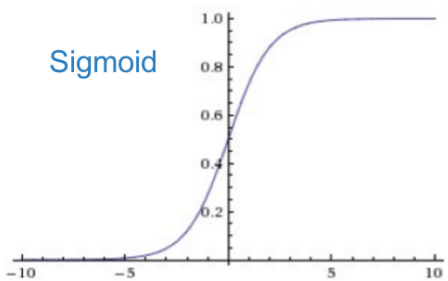
Artificial Neuron (Node)

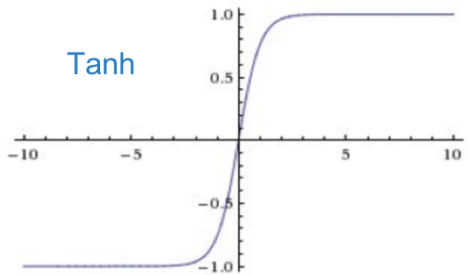
Each node of a NN receives inputs $\vec{x} = \{x_1, ..., x_n\}$ from other nodes or an external source and computes an output y according to the expression

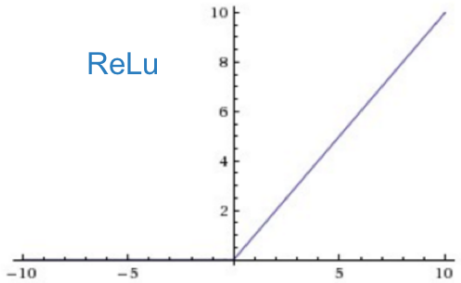
$$y = F\left(\sum_{i=1}^{n} W_i x_i + B\right) = F\left(\vec{W}.\vec{x} + B\right) \quad (1)$$

, where W_i are connection weights, B is the treshold and F the activation function [2]

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







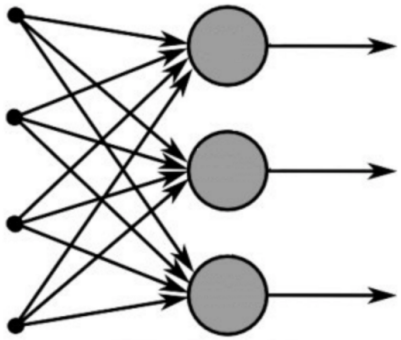
[2] The neuron output is sometimes called the activation

Neural Network Topologies

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

Feed Forward Networks

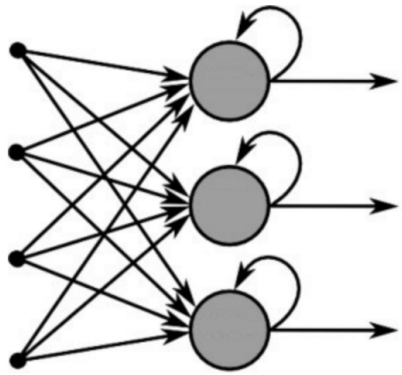
A feedforward NN is an neural network wherein connections between the nodes do not form a cycle. In a feed forward network information always moves one direction, from input to output, and it never goes backwards. Feedforward NN can be viewed as mathematical models of a function $F : \mathbb{R}^N \rightarrow \mathbb{R}^M$



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. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequential input data.

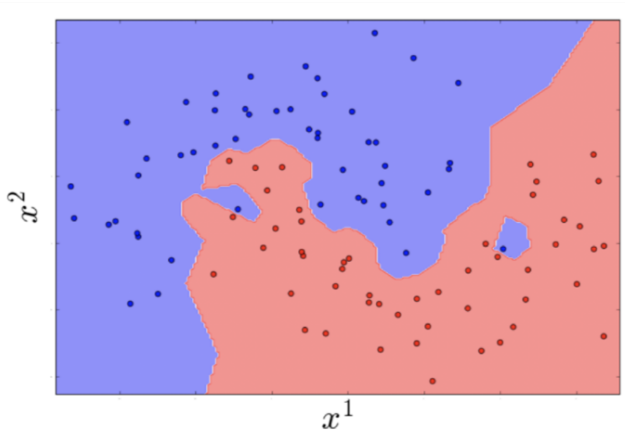


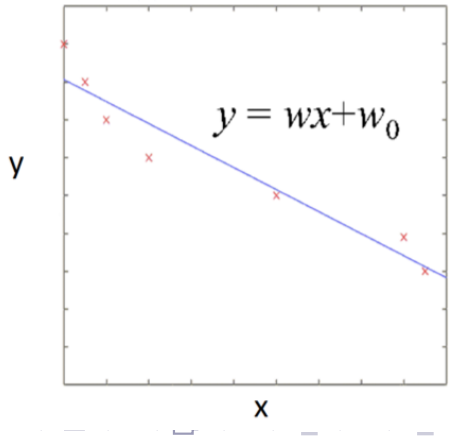
Recurrent Neural Network

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 data $D = \{(\vec{x}_1, \vec{t}_1), ..., (\vec{x}_n, \vec{t}_n)\}$ with features $\{\vec{x}_i\}$ and targets $\{\vec{t}_i\}$, the algorithm finds a mapping $\vec{y}_i = F(\vec{x}_i)$
- **Classification:** $\{\vec{t}_1, ..., \vec{t}_n\}$ (finite set of labels)
- **Regression:** $\vec{t}_i \in \mathbb{R}^n$

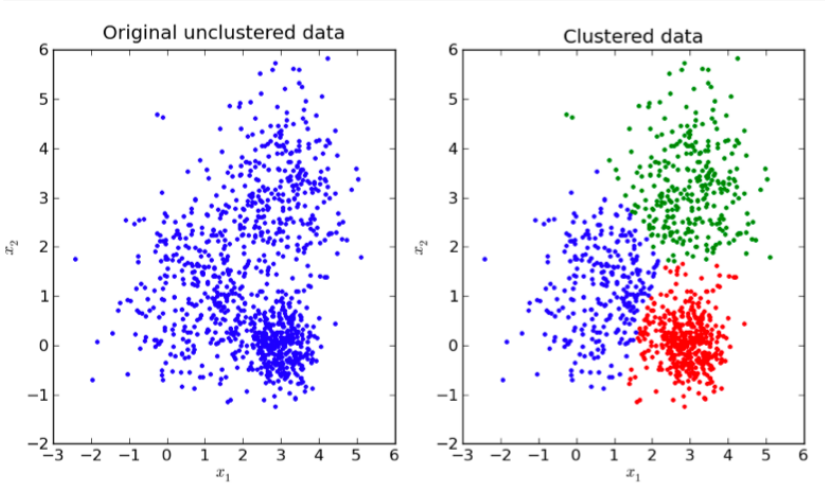


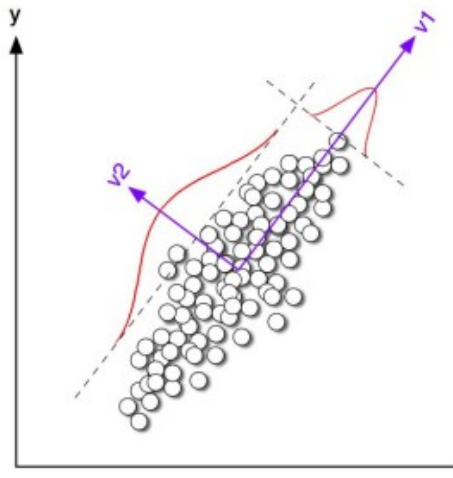


Unsupervised Learning

Unsupervised Learning

- No target output is used at training and the NN finds patterns within input data
- Given some data $D = \{\vec{x}_1, ..., \vec{x}_n\}$, but no labels, find structures in the data
  - **Clustering:** partition the data into sub-groups $D = \{D_1 \cup D_2 \cup ....D_k\}$
  - **Dimensional Reduction:** find a low dimensional representation of the data with a mapping $\vec{y} = F(\vec{x})$, where $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ with $n >> m$



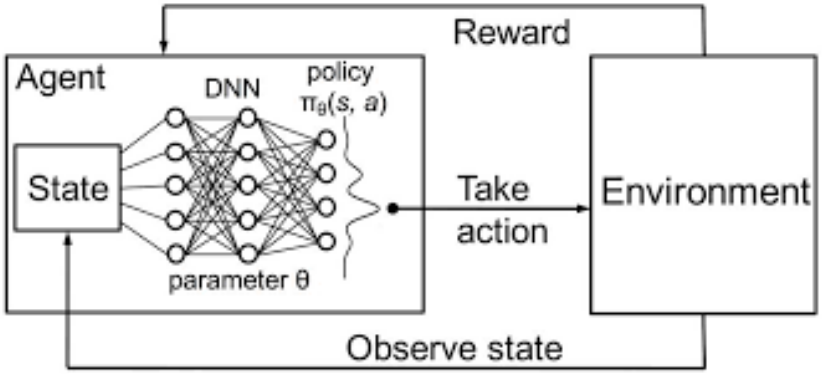


Reinforcement Learning

Reinforcement learning (RL)

Similar to supervised learning but instead of a target output, a reward is given based on how well the system performed. The algorithm take actions in an environment so as to maximize some notion of cumulative reward.

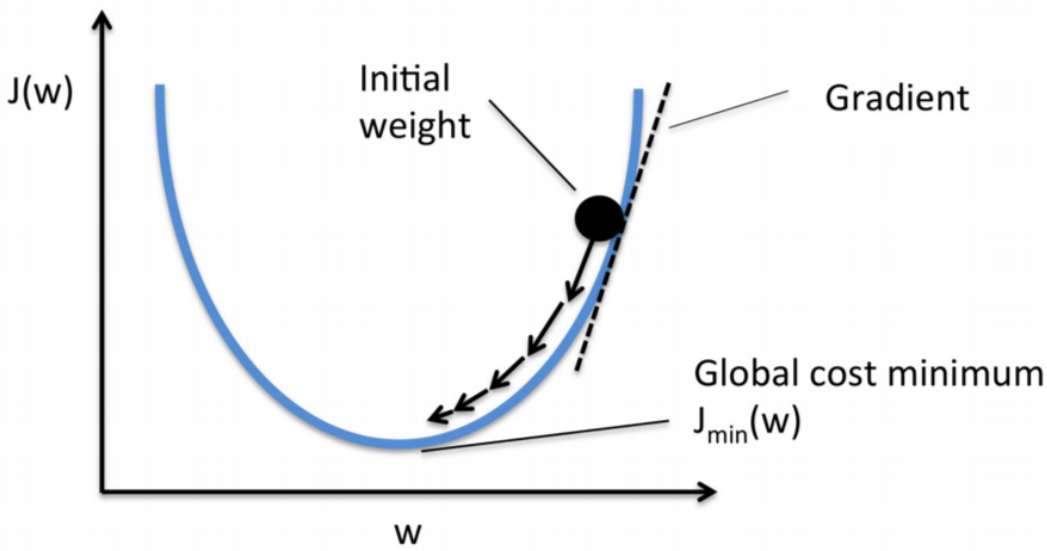
- Inspired by behaviorist psychology and strongly related with how learning works in nature
- Maximize the reward the system receives through trial-and-error
- Algorithm learns to make the best sequence of decisions to achieve goal.
- Requires a lot of data, so applicable in domains where simulated data is readily available: gameplay, robotics, self-driving vehicles
- Reinforcement learning algorithms: Q-learning, policy gradients ...



Supervised Learning - Training Process

Learning as an Error Minimization Problem

1. Random NN parameters (weights and bias) initialisation
2. Choose a 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 - Loss Function

Loss Function

The Loss function quantifies the error between the NN output $F(\vec{x})$ and the desired target output $\vec{t}$.
Loss over the datasample (Cost) is the mean loss over examples of the sample $\{(\vec{x}_i, \vec{t}_i)\}$

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

Squared Error Loss (Regression)

$$L = \frac{1}{2} \left[F(\vec{x}) - \vec{t} \right]^2$$

Cross Entropy Loss (Binary Classification)

If we have binary targets $t \in \{0, 1\}$:

$$L = -t \log \left[F(\vec{x}) \right] - (1 - t) \log \left[1 - F(\vec{x}) \right]$$

Log-Likelihood Loss (Multi-Classification)

If we have m classes with probabilities $p_i \in \{0, 1\}$:

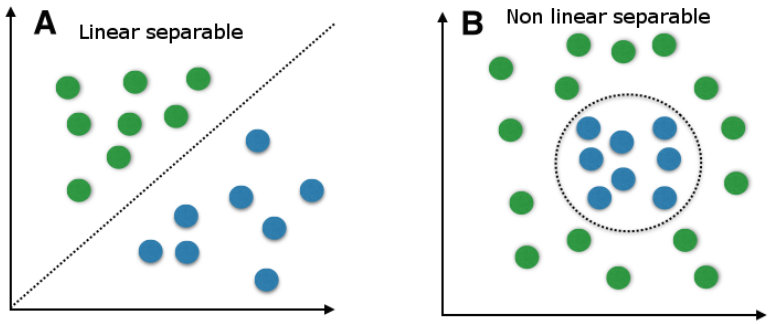
$$L = -\frac{1}{m} \log \left(\prod_{i=1}^{m} p_i \right) = -\frac{1}{m} \sum_{i=1}^{m} log(p_i)$$

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 [3]

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



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

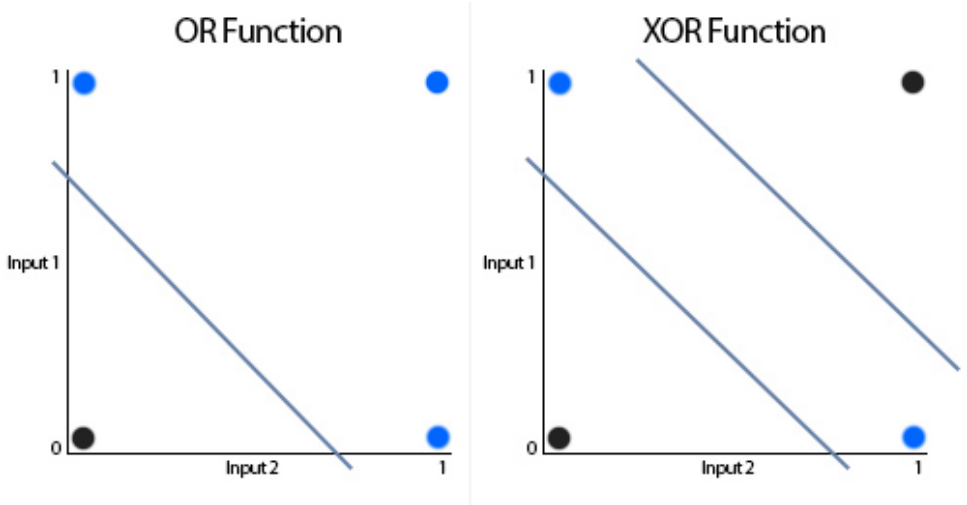
Initialize the weights and for each example j in training set $D = \{(\vec{x}_1, t_1), \ldots, (\vec{x}_m, t_m)\}$, where t_j are output targets, and perform the following steps for each $(\vec{x}_j, t_j) \in D$

1. Calculate the actual output and 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

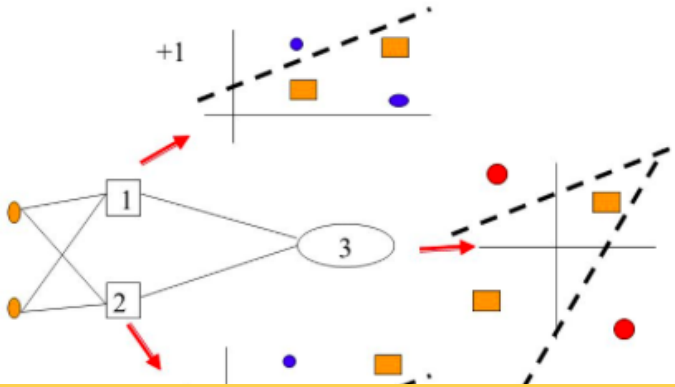
[3] Equation of a plane in $\mathbb{R}^n$ is $\vec{W}.\vec{x} + B = 0$

Perceptron and XOR Problem

Perceptrons limitations with non linearly separable problems makes its unable to learn the Boolean *XOR* function



We need a multi layer architecture to solve the *XOR* problem in two-stages



Multilayer Perceptron

The Multilayer Perceptron(MLP) is a fully connected NN with at least 2 layers (hidden and output). The layers uses nonlinear activation functions F which can differ between layers [4] . The MLP is the simplest feed forward neural network and has a model [5]

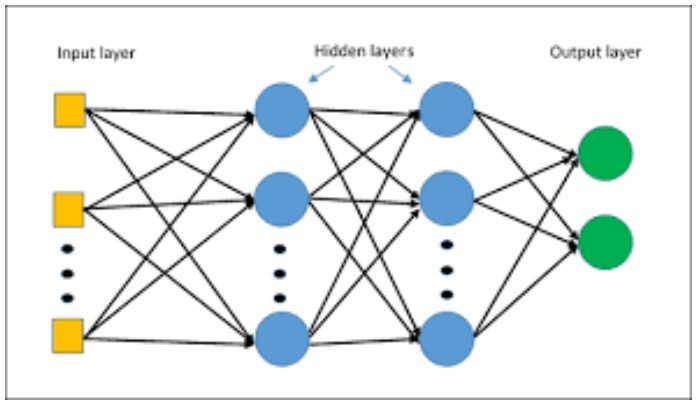
Multilayer Perceptron Model

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 parametrization of a mapping (function) $F_{w,b} : \mathbb{R}^n \rightarrow \mathbb{R}^m$

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

[5] 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

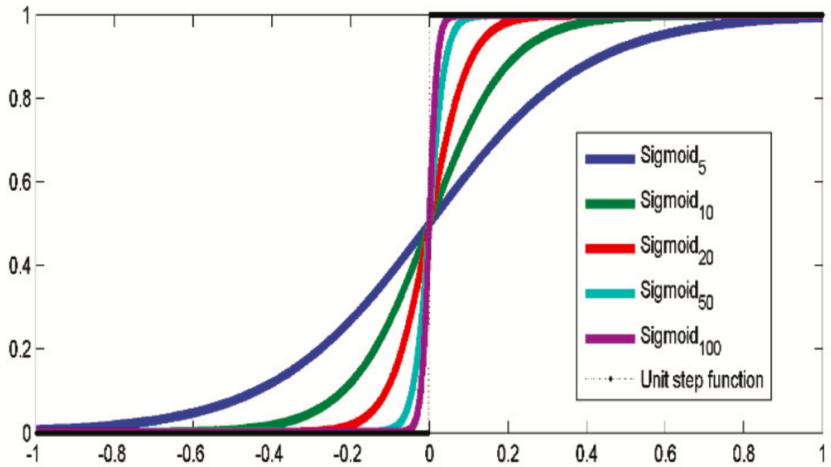
A single hidden layer feed forward neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden neurons [6]

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

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



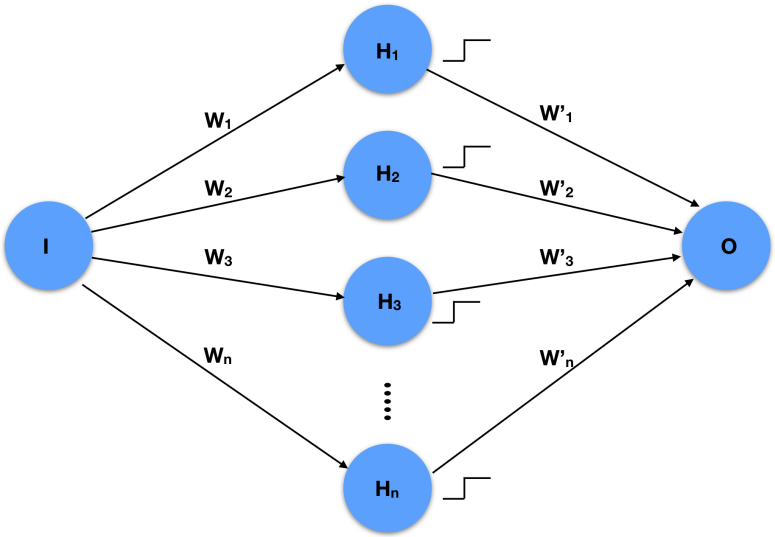
[6] Cybenko,G.(1989) Approximations by superpositions of sigmoidal functions, Math.ofCtrl.,Sig.,andSyst.,2(4),303
Hornik,K.(1991) Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks, 4(2), 251

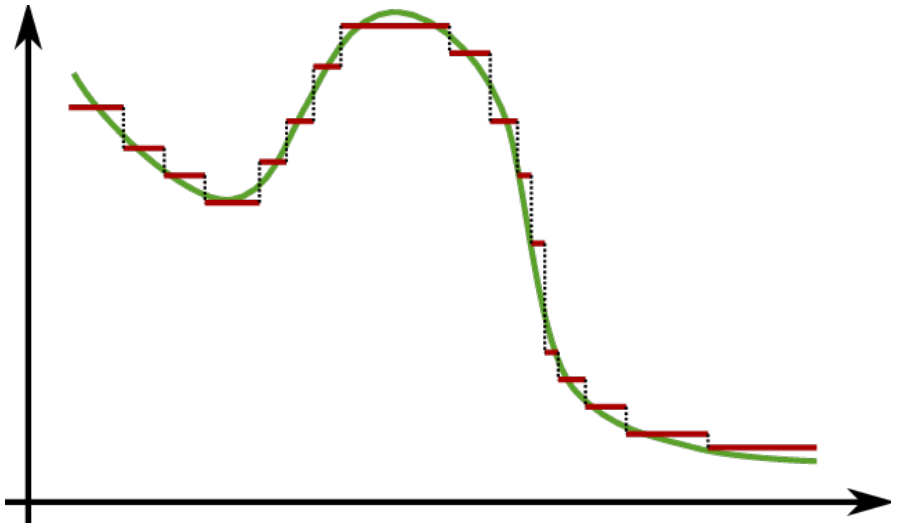
Multilayer Perceptron as a Universal Approximator

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 !

Gradient Descent Method (Loss Minimization)

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

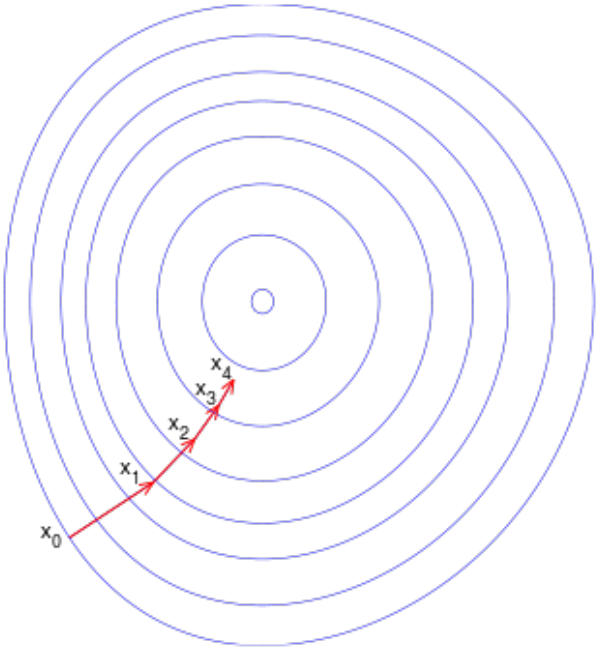
Gradient Descent

A multi-variable function $F(\vec{x})$ decreases fastest in the direction of its negative gradient $-\nabla F(\vec{x})$ [7].

Choosing an initial point $\vec{x}_0$, one can use a recursion formula to obtain a sequence of points $\{\vec{x}_1, \ldots, \vec{x}_n\}$ leading to the 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 $F(\vec{x}_0) \geq F(\vec{x}_1) \geq \ldots \geq F(\vec{x}_n)$ indicates it converges to local minimum !



Gradient Descent uses only first order derivatives, which is efficiently and simply calculated by backpropagation. Minimization methods like Newton and BFGS needs second order derivative(Hessian), which is computationally costly and memory inefficient.

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

Stochastic Gradient Descent

Stochastic Gradient Descent(SGD)

SGD [8] is called stochastic because data samples are selected randomly (shuffled), instead of the order they appear in the training set. This allows the algorithm to try different ”minimization paths”at each training epoch

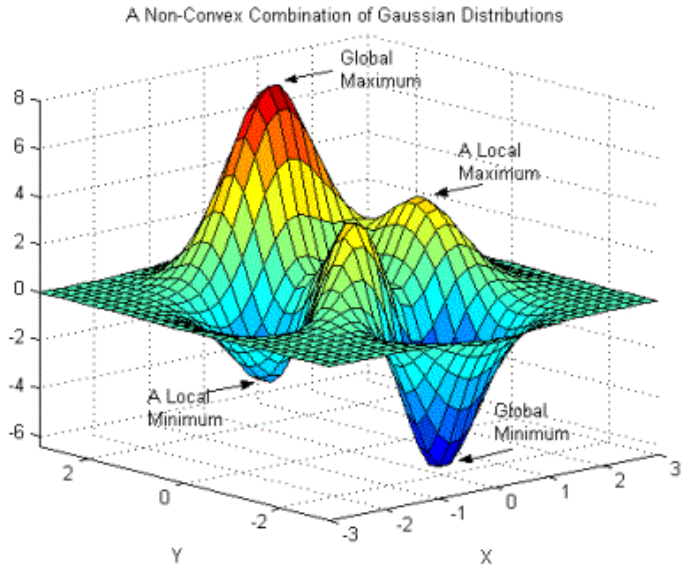
- It can also average gradient with respect to a set of events (minibatch)
- Noisy estimates average out and allows “jumping” out of bad critical points
- Scales well with dataset and model size

Convex Loss

- Single global minimum
- Iterations toward minimum

Non-Convex Loss

- Get stuck in local minima
- Convergence issues
- Adaptive variants

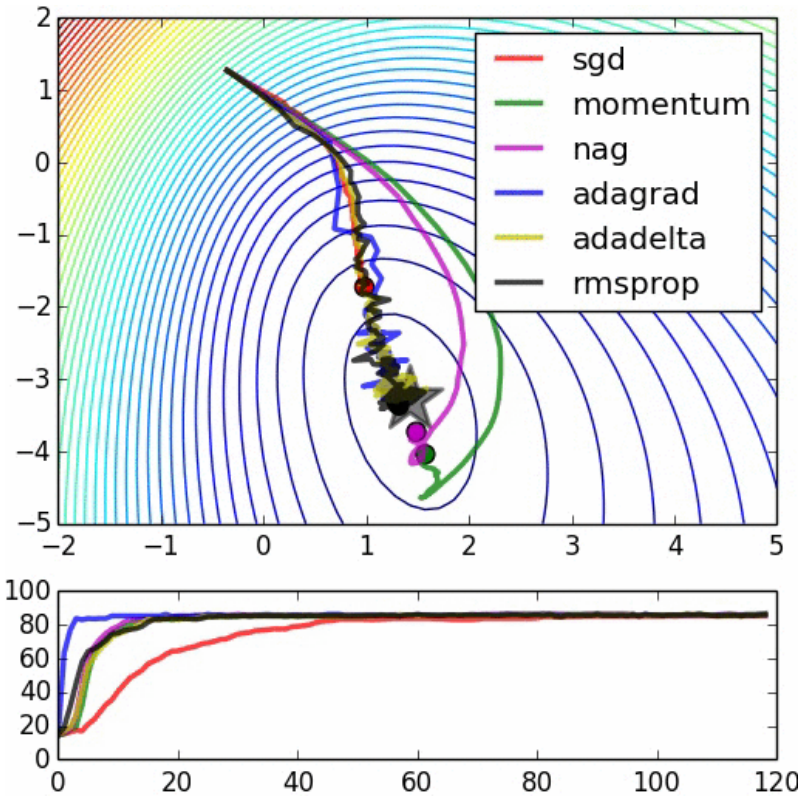


[8] https://deepnotes.io/sgd-momentum-adaptive

SGD Algorithms Improvements [10]

SGD Algorithms [9] :

- **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



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

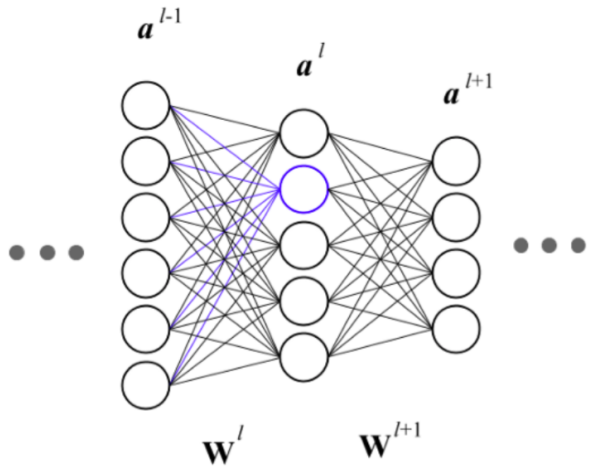
Backpropagation

Backpropagation is a gradient descent calculation technique for multilayer networks under supervised learning. An error is computed at the network output and distributed backwards to each layer, as a consequence of recursive use of the chain rule [11]

MLP Loss Function

Consider a MLP with n layers (not counting input layer) with a quadratic loss. One can view the MLP loss as a n layer composite function of the weights, where $W_{ij}^{(l)}$ connects the neuron j in layer $(l-1)$, to neuron i in layer l. Defining the activation $a_i^{(l)} = F(W_{ij}^{(l)} a_j^{(l-1)})$ as the output of neuron i in layer (l), we have

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



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

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

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

Backpropagation Formulas

The backpropagation master formulas [12] for the gradients of the loss function in layer-l is given by

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

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

[12] 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. Choose an element from the dataset $D = \{(x_i, t_i)\}$
3. Perform a feedforward pass, computing the arguments $z_k^{(l)}$ and activations $a_k^{(l)}$ for all layers
4. Determine the network output error: $\delta_i^{(n)} = \left[a_i^{(n)} - t_i\right] F'(z_i^{(n)})$
5. Backpropagate the output error: $\delta_k^{(l)} = \left(\sum_m \delta_m^{(l+1)} W_{mk}^{(l+1)}\right) F'(z_k^{(l)})$
6. Compute the loss gradients using the neurons error and activation: $\frac{\partial L}{\partial W_{kj}^{(l)}} = \delta_k^{(l)} a_k^{(l-1)}$
7. Update the weights according to the gradient descent: $\Delta W_{kj}^{(l)} = -\lambda \frac{\partial L}{\partial W_{kj}^{(l)}}$
8. Return to step (2)

So far we have focused on fully connected neural network, but this reasoning can be applied to other NN architectures (different neuron 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 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 and tune hyper-parameters

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

Test

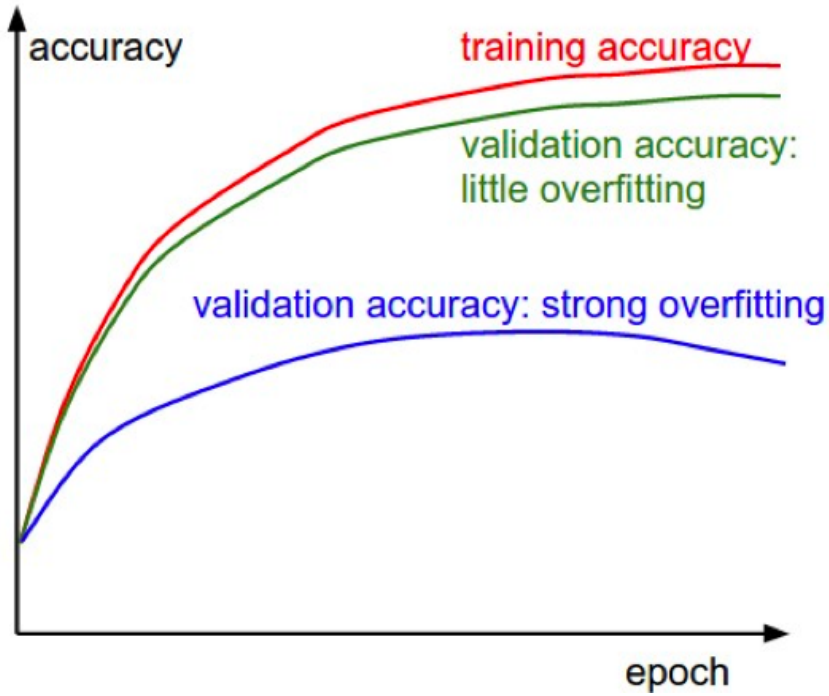
Final performance evaluation after finished training and hyper-parameters are fixed. Use the test dataset for an independent evaluation of performance obtaining a ROC curve

Overtrainging(Overfitting)

Overtraining(Overfitting)

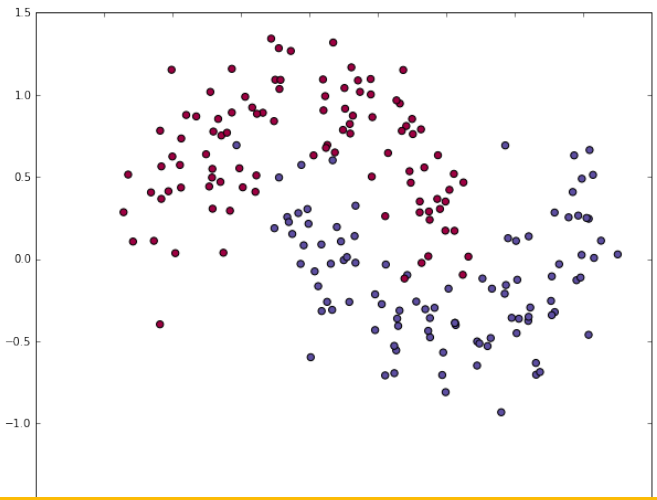
Gap between training and validation accuracy indicates the amount of overfitting

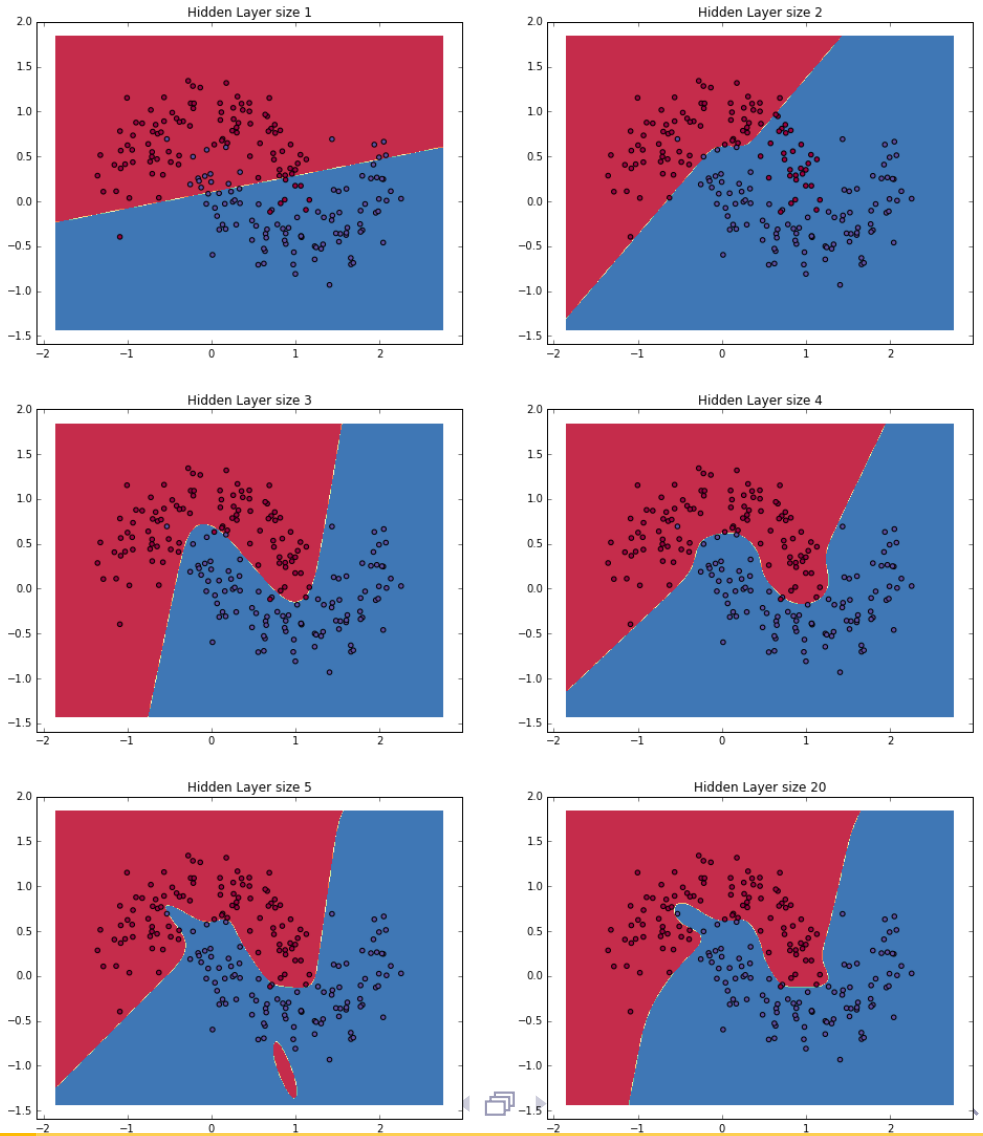
- If validation error curve shows small accuracy compared to training it indicates overfitting $\Rightarrow$ add regularization or use more data.
- If validation accuracy tracks the training accuracy well, the model capacity is not high enough $\Rightarrow$ use larger model



Overfitting - Hidden Layer Size X Decision Boundary [13]

- 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

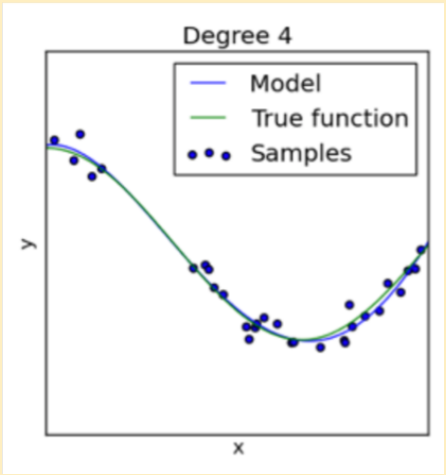




Overfitting and Underfitting - Bias Variance Tradeoff

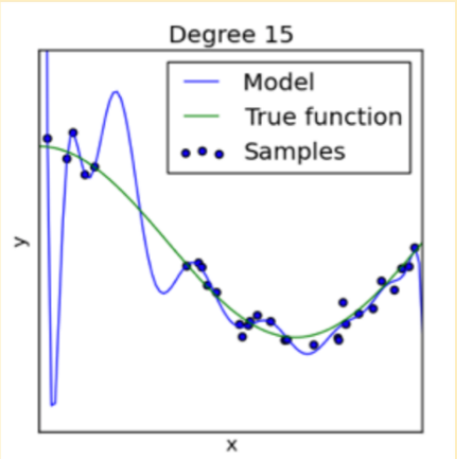
Proper fitting

Correct identification of outliers (noise) $\Rightarrow$ better generalization



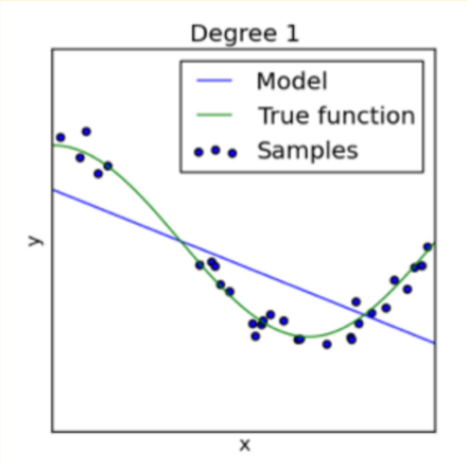
Overfitting

Large NN with too many DOF



Underfitting

Small NN with too few DOF



Bias Variance Tradeoff

- **Generalization error = systematic error (bias or accuracy) + sensitivity of prediction (variance or uncertainty)**
- **Complex models overfit** : captures more data points having a lower bias, while it ”moves”more to capture the points having a larger variance. So, won’t deviate systematically from data (low bias) but will be very sensitive to data (high variance)
- **Simple models underfit** : will deviate from data(high bias) but will not be influenced by peculiarities of data (low variance)

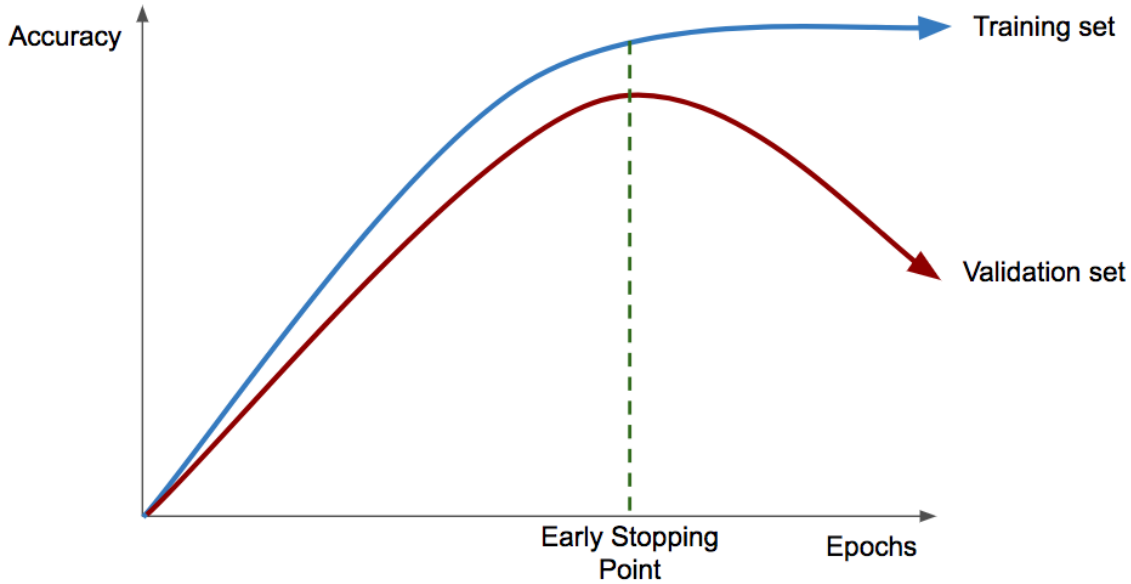
Regularization

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.

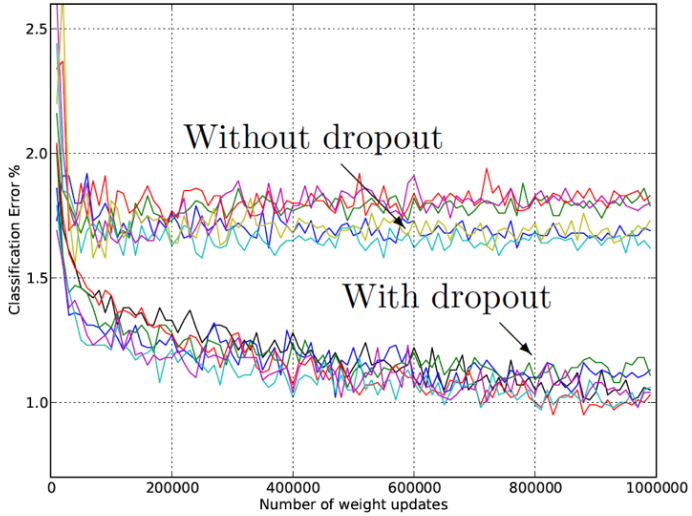


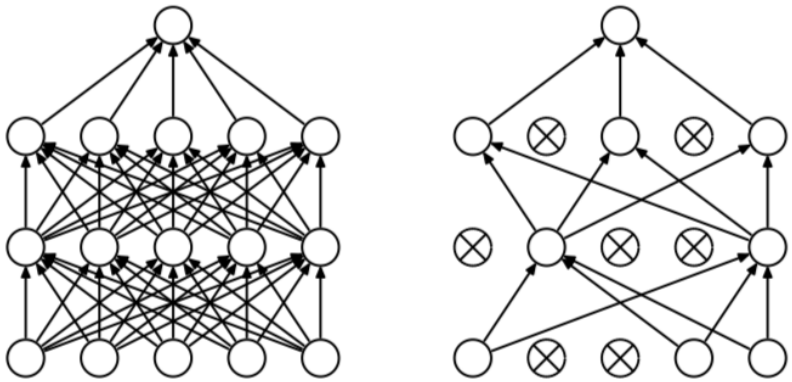
Dropout Regularization

Dropout Regularization

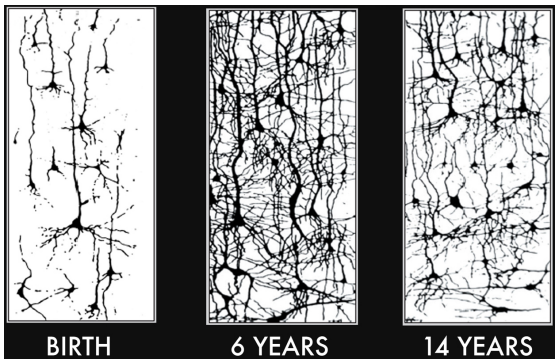
Regularization inside network that remove nodes randomly during training

- Avoid co-adaptation on training data
- Essentially a large model averaging procedure





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



L1 & L2 Regularization

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

$$L(\vec{w}) = \frac{1}{2}\left|\vec{o} - \vec{t}\right|^2 + \alpha\Omega(\vec{w}) \tag{4}$$

L2 Regularization

- Adds a term $\Omega(\vec{w}) = |\vec{w}|^2$ to loss function
- Keep weights from getting too large and saturating activation function
- Works by penalising large weights
- Use all inputs with small contributions instead of just a few larger weights

L1 Regularization

- Adds a term $\Omega(\vec{w}) = |\vec{w}|$ to loss function
- Works by keeping weights sparse

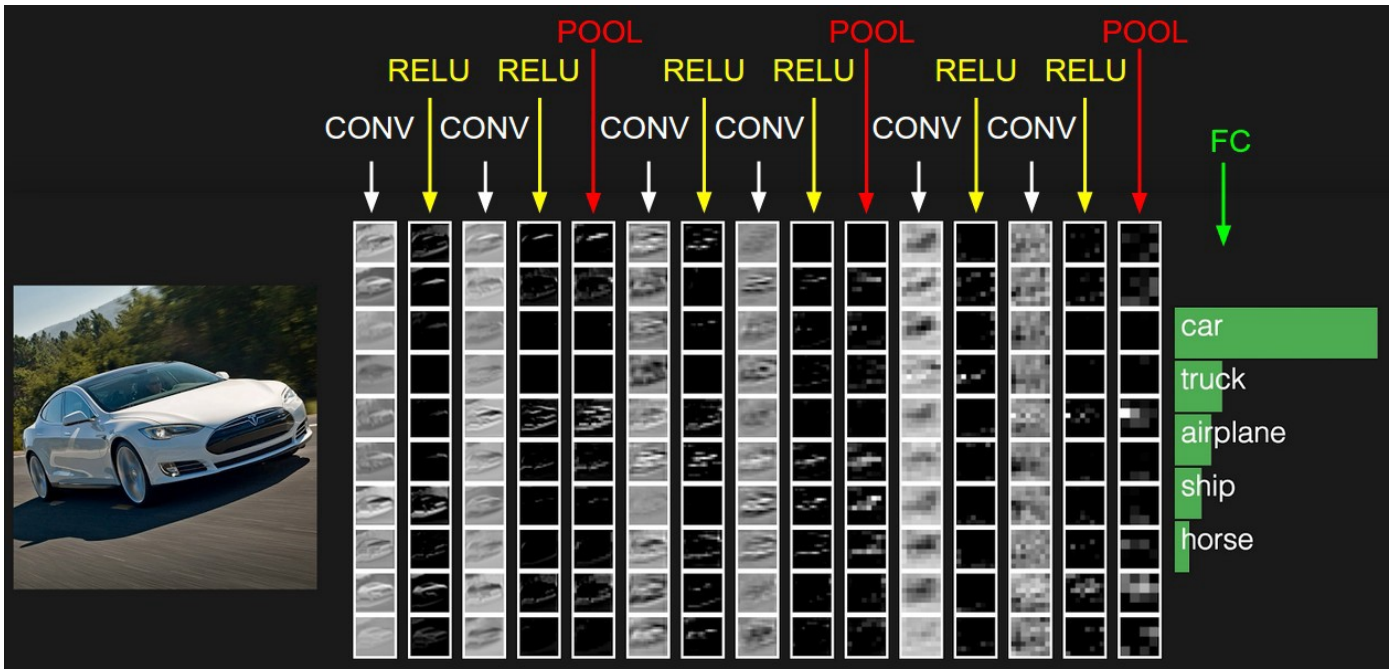
The combined $L1 + L2$ regularization is called Elastic

Deep Neural Networks - Need for Depth

Universal approximation theorem says a single hidden layer is enough to learn any function. But as data complexity grows, one needs exponentially large hidden layer to capture all the structure in data

Deep Neural Networks(DNN)

DNN , as oposed to shallow NN, have more than one hidden layer. They factorize the data features, distributing its representation across the layers.



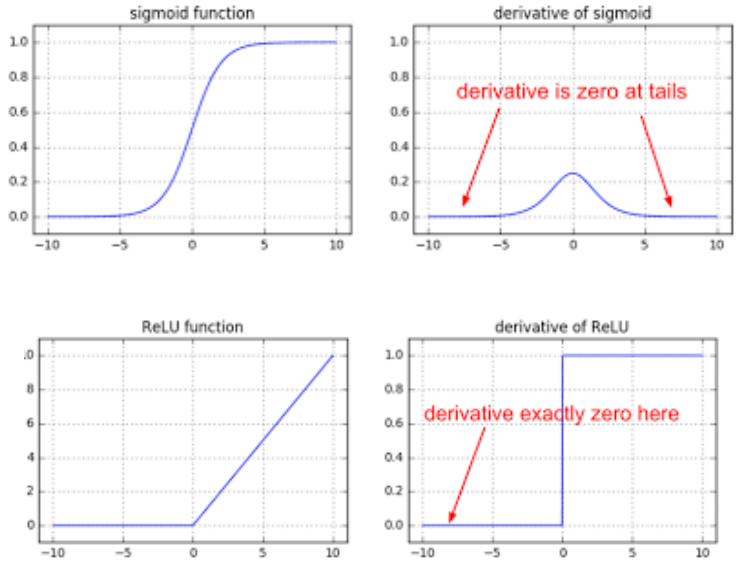
Deep Learning

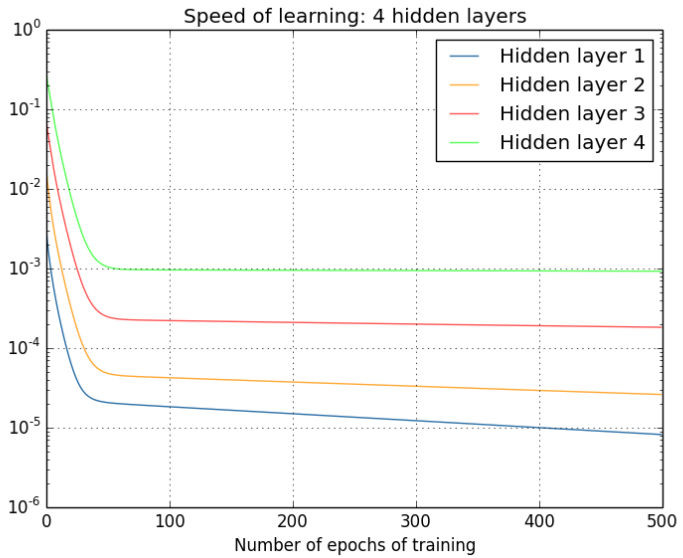
Vanishing Gradient Problem [14]

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

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

For sigmoid and tanh F' is asymptoticaly zero, so gradients gets vanishing small as we move backwards through the layers. Then, earlier layers learns much slower than later layers !!!





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

Deep Learning

Depth makes it difficult for DNN to learn and only with recent training procedures it became possible to train them. Deep learning is a collection of tools and procedures to train DNNs

Deep Learning (DNN Training)

- Use of ReLU activation in hidden layers to avoid vanishing gradients
- For regression use linear activation in ouput layer and quadratic loss
- For binary classification use sigmoid activation in ouput layer and binary cross entropy loss [15]
- For multiclass classification use softmax activation in ouput layer and log likelihood loss
- SGD with batch normalization and regularization
- Use large training data samples and data augmentation
- Powerfull computing: GPU(Nvidia), TPU(Google), FPGA(Amazon EC2) ...

[15] http://neuralnetworksanddeeplearning.com/chap3.html

Activation Functions [16]

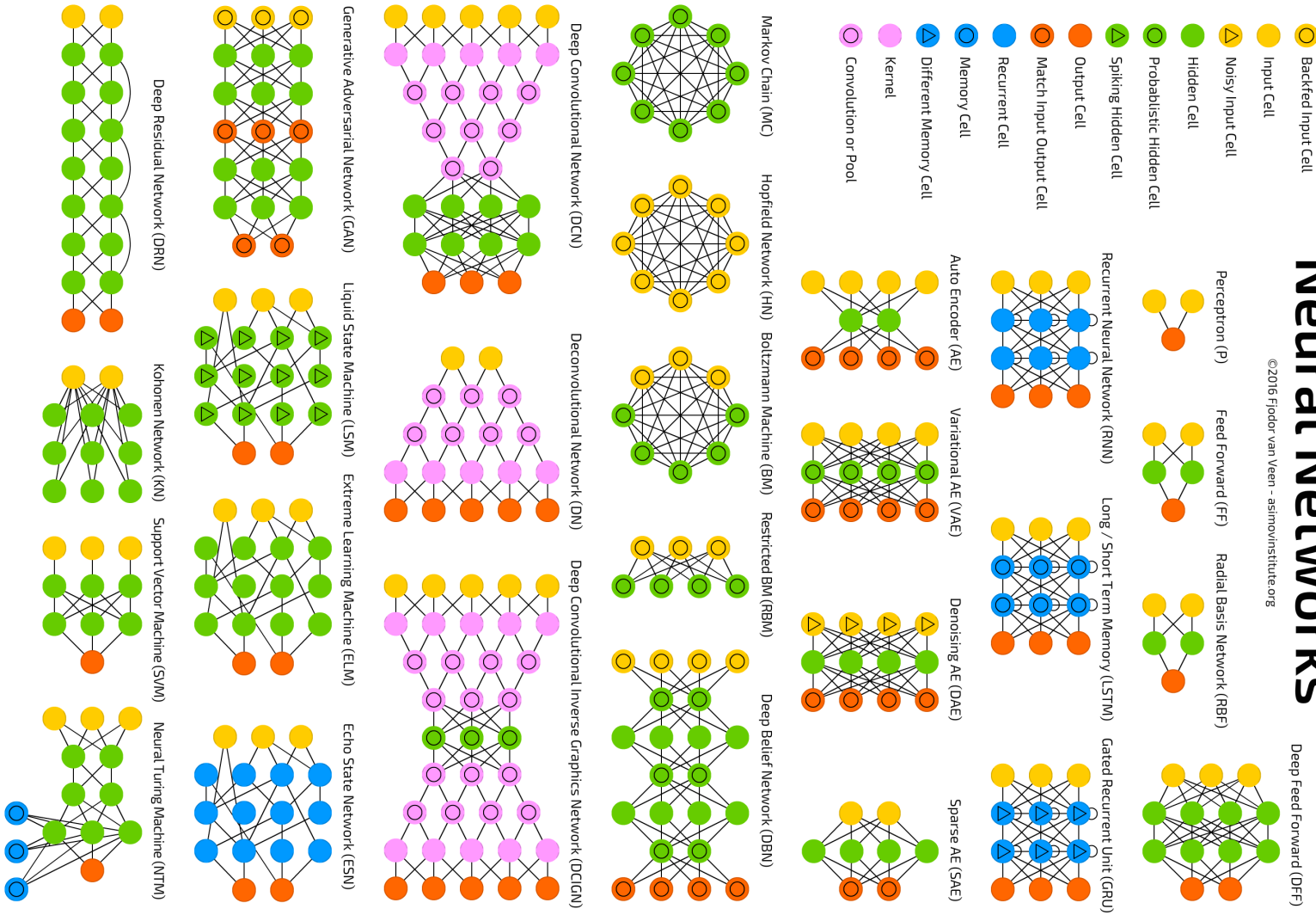
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Oftentimes, we want output vector as probability distribution over mutually exclusive labels (multiclass). In this case we use a squashing output layer called softmax $f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$

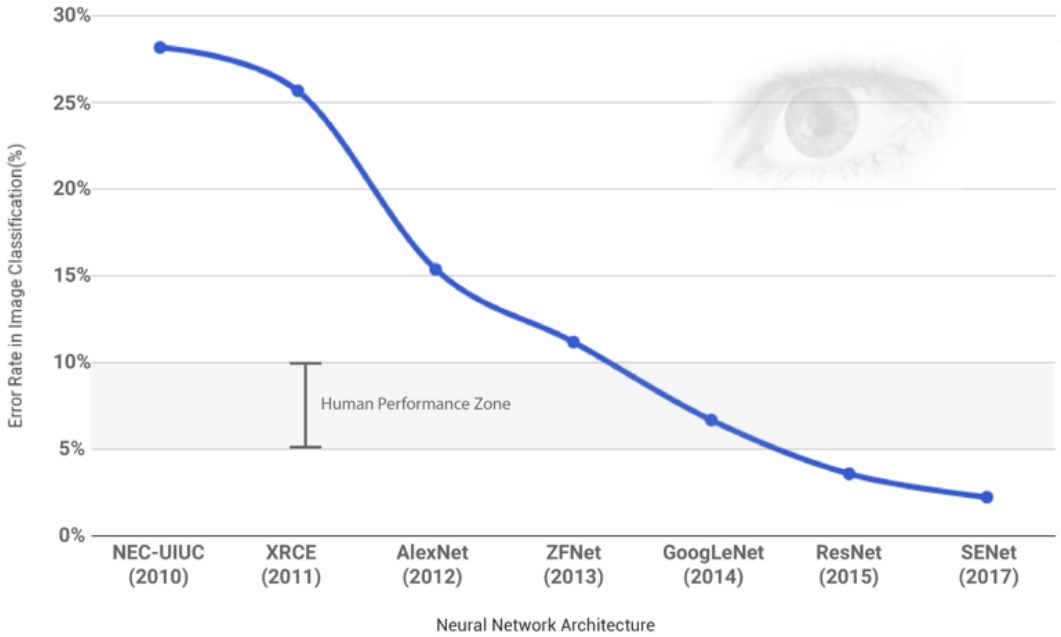
[16] https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

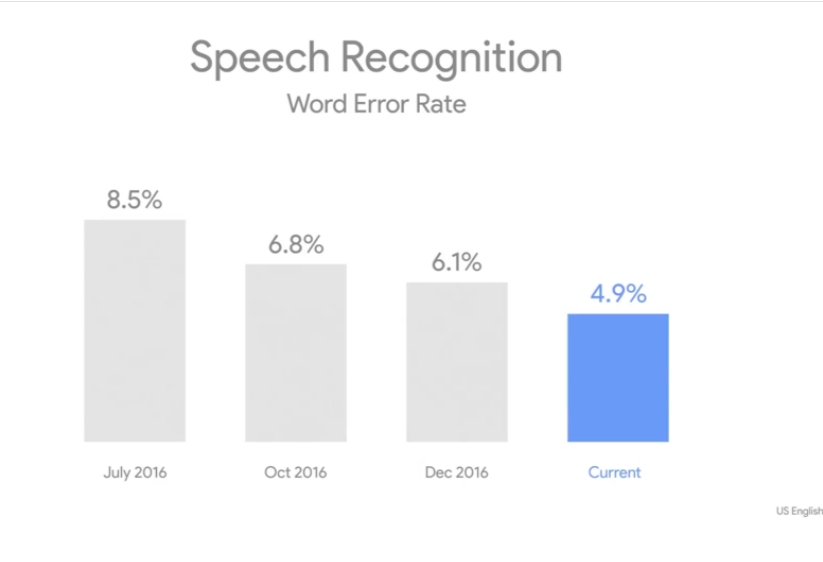
Neural Network Zoo [17]

Structure of NN and nodes connectivity can be adapted for the problem at hand



DNN versus Humans



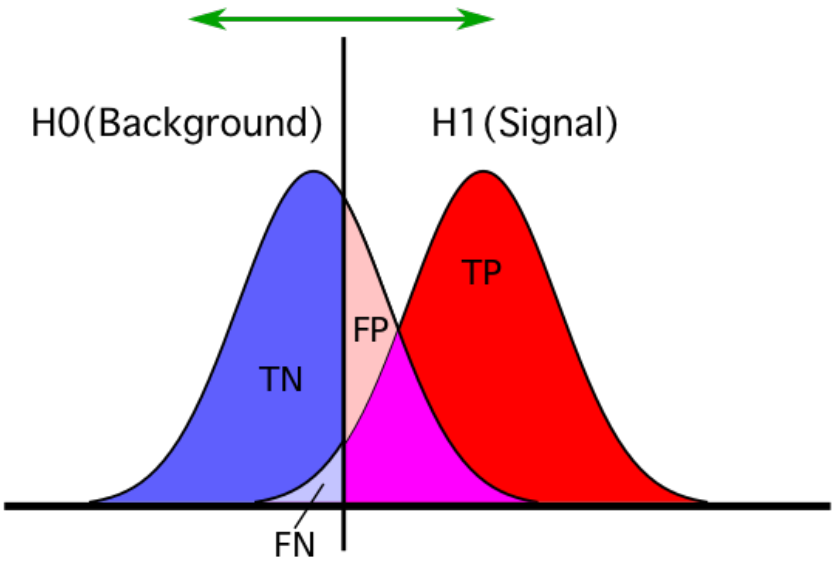


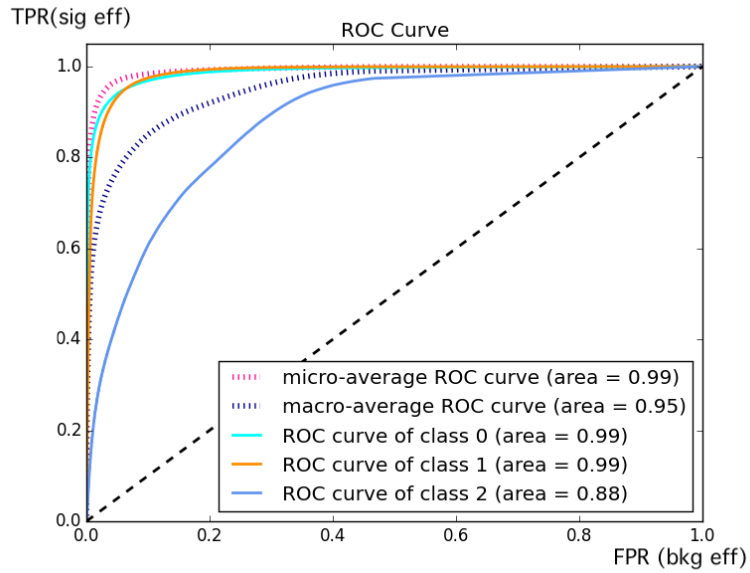
https://arxiv.org/pdf/1409.0575.pdf

Classifier Perfomance: ROC

ROC

A Receiver Operating Characteristic (ROC) curve, is a plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied

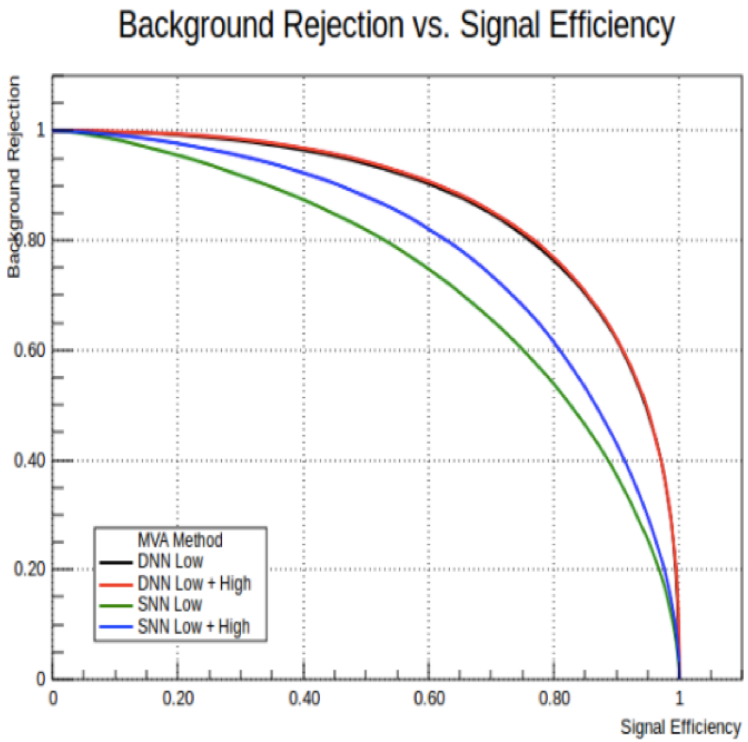




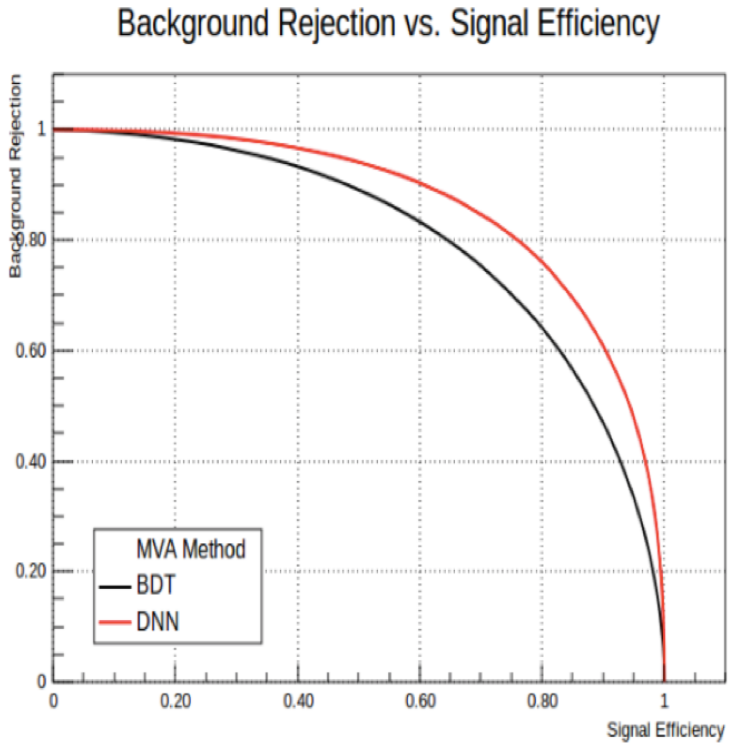
The Area Under the Curve (AUC) is sometimes used as a classifier figure of merit

DNN Perfomance [18]

Low versus high level (feature engineered) variables



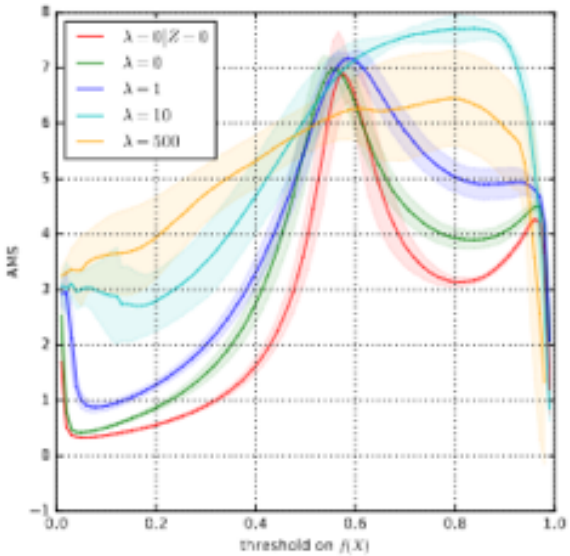
BDT versus DNN using same set of variables



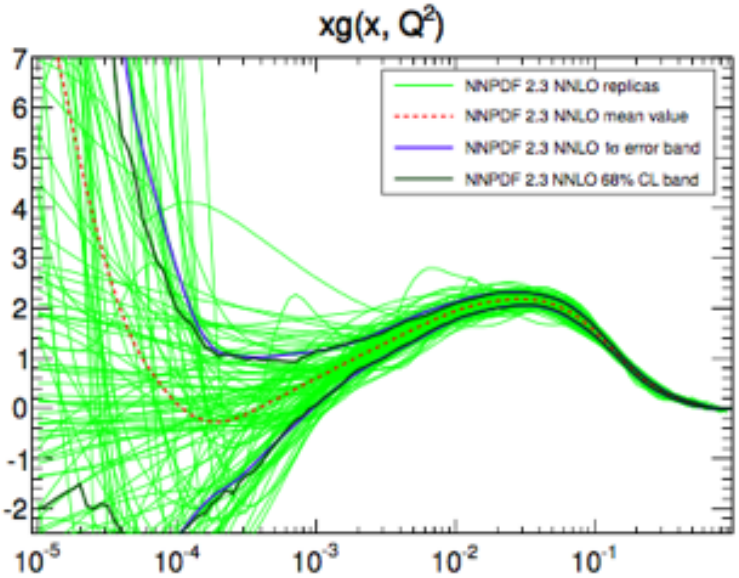
[18] https://arxiv.org/pdf/1402.4735.pdf

HEP Application: NNPDF Regression

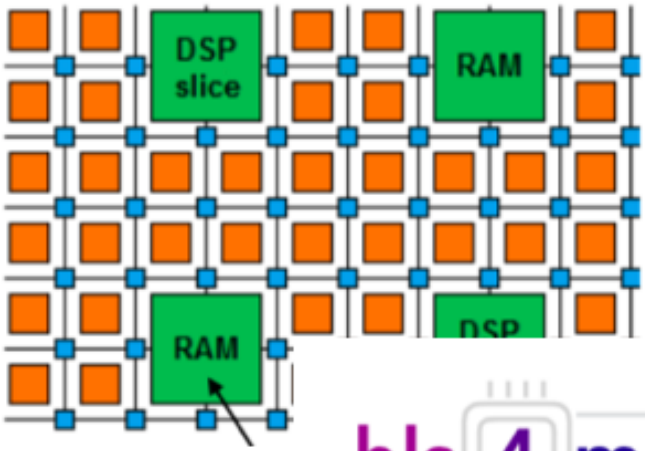
G. Louppe et al., 2016

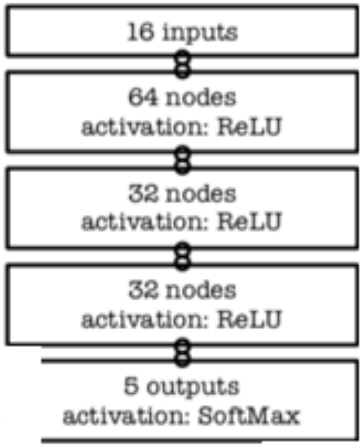


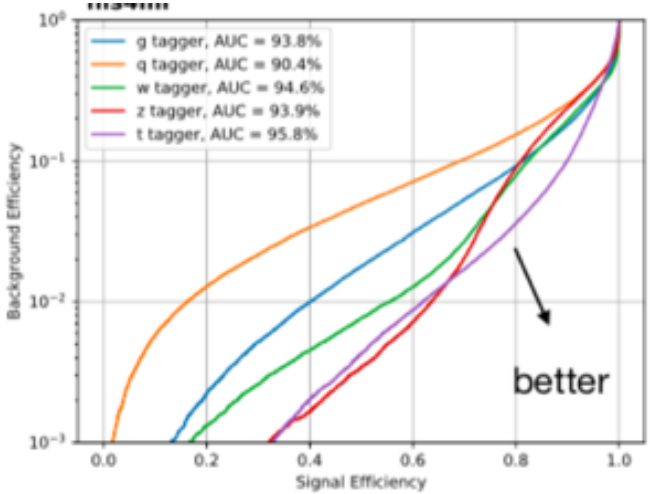
NNPDF Collaboration



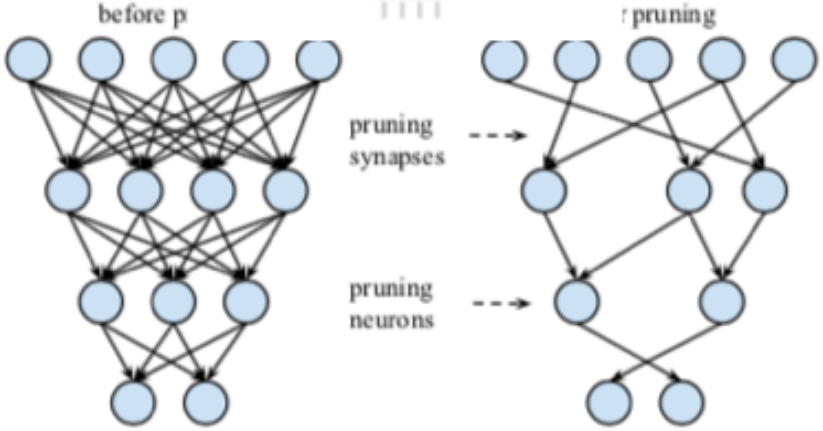
DNN Application in HEP: L1 Trigger

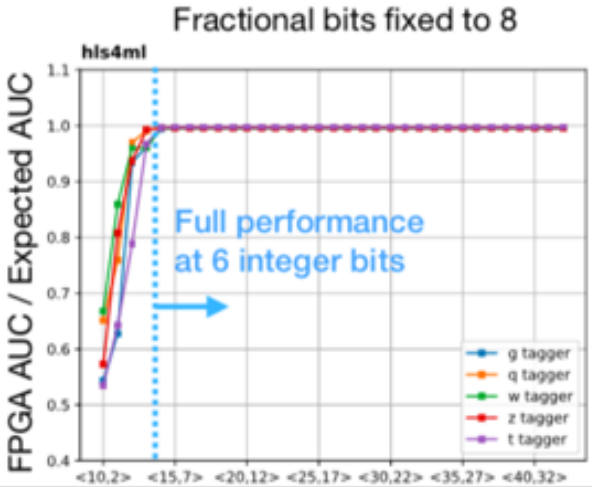






hls 4 ml





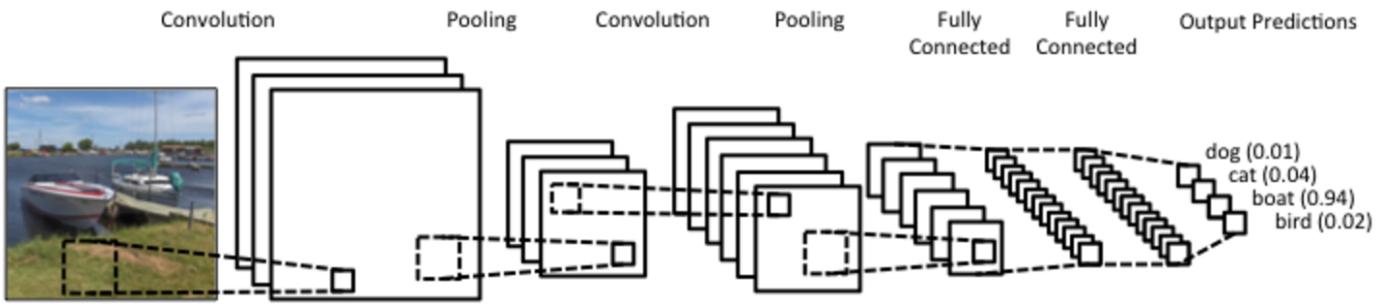
Convolutional Neural Network

When dealing with high-dimensional inputs such as images, learning features with fully connected networks is computationally too expensive (ex: 1000x1000 image has $O(10^6)$ pixels).
Convolutional neural network [19] emulate the behavior of a visual cortex, where individual neurons respond to stimuli only in a restricted region of the visual field.

Convolutional Neural Network (CNN or ConvNet)

A CNN mitigate the challenges of high dimensional inputs by restricting the connections between the input and hidden neurons. It conects only a small contiguous region of input nodes, exploiting local correlation. Its architecture is a stack of distinct and specialized layers:

1. Convolutional
2. Pooling (Downsampling)
3. Fully connected (MLP)



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

CNN - Convolutional Layer

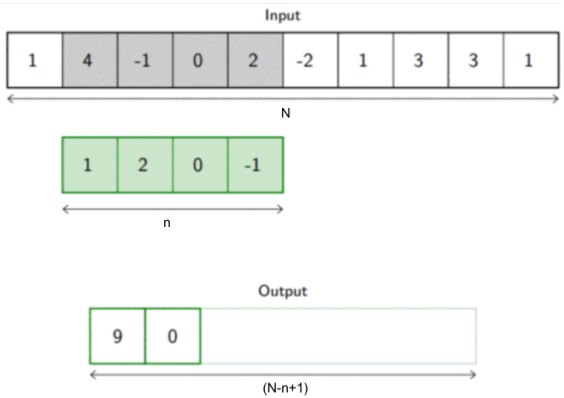
Convolutional Layer

CNN convolutional layer is its core building block and its parameters are learnable filters (kernels), corresponding to a small receptive field. A convolution is like a sliding window transformation, where the window is the filter

1D Convolution

Convolution of N neurons $x_i^{(l)}$ from layer-l and a filter w_a of size n gives an output of size $(N - n + 1)$

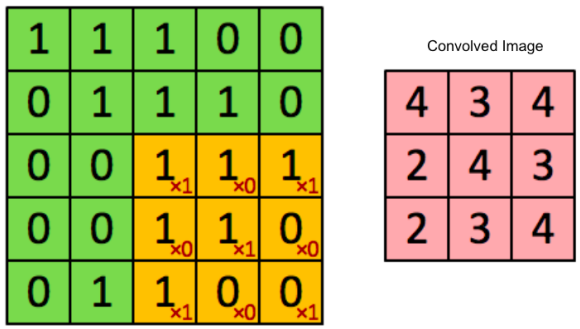
$$y_i^{(l+1)} = \sum_{a=0}^{n-1} w_a x_{i+a}^{(l)}$$



2D Convolution

Convolution of $N \times M$ neurons $x_{ij}^{(l)}$ from layer-l and a filter w_{ab} of size $n \times m$ gives an output of size $(N - n + 1)(M - m + 1)$

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



Convolution Filters

An image convolution [20] with filters designed to produce some effect (sharpen, blurr) or detect some image feature (edges, texture)

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	

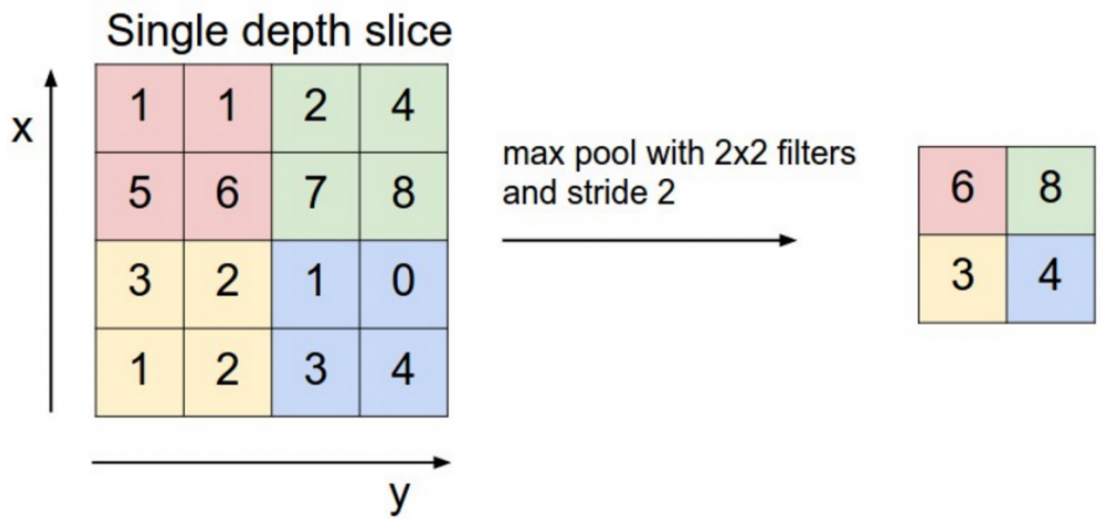
	-2	-1	0	
	-1	1	1	
	0	1	2	

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

CNN - Pooling Layer

Pooling(Downsampling) Layer

The pooling (downsampling) layer has no learning capabilities and serves only to decrease the representation size, reducing the number of parameters and amount of computation in the network. It partitions the input in a set of **non-overlapping** regions and, for each sub-region, it outputs a single value (ex: max 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

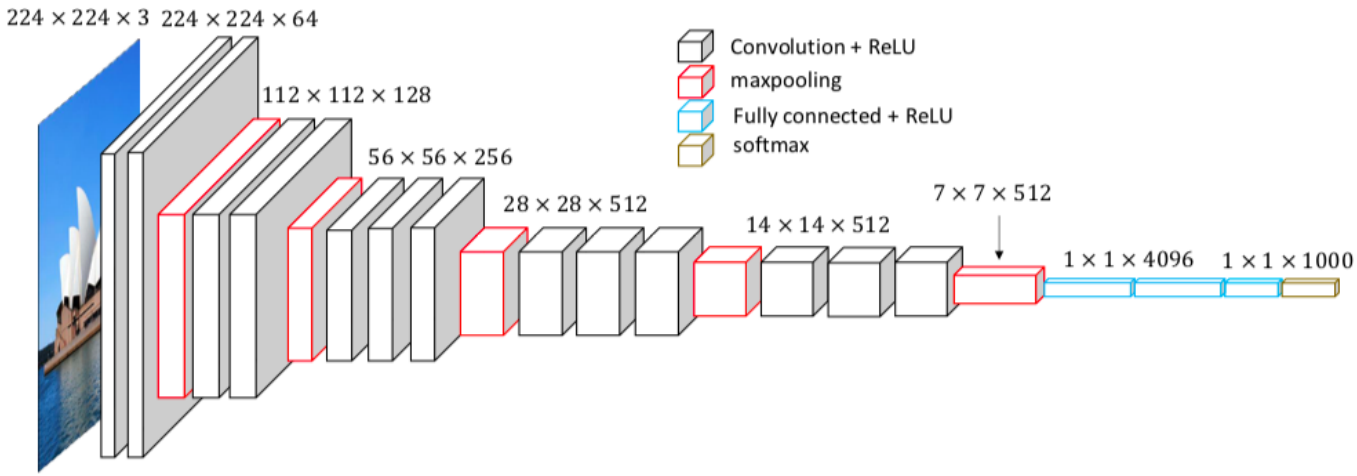
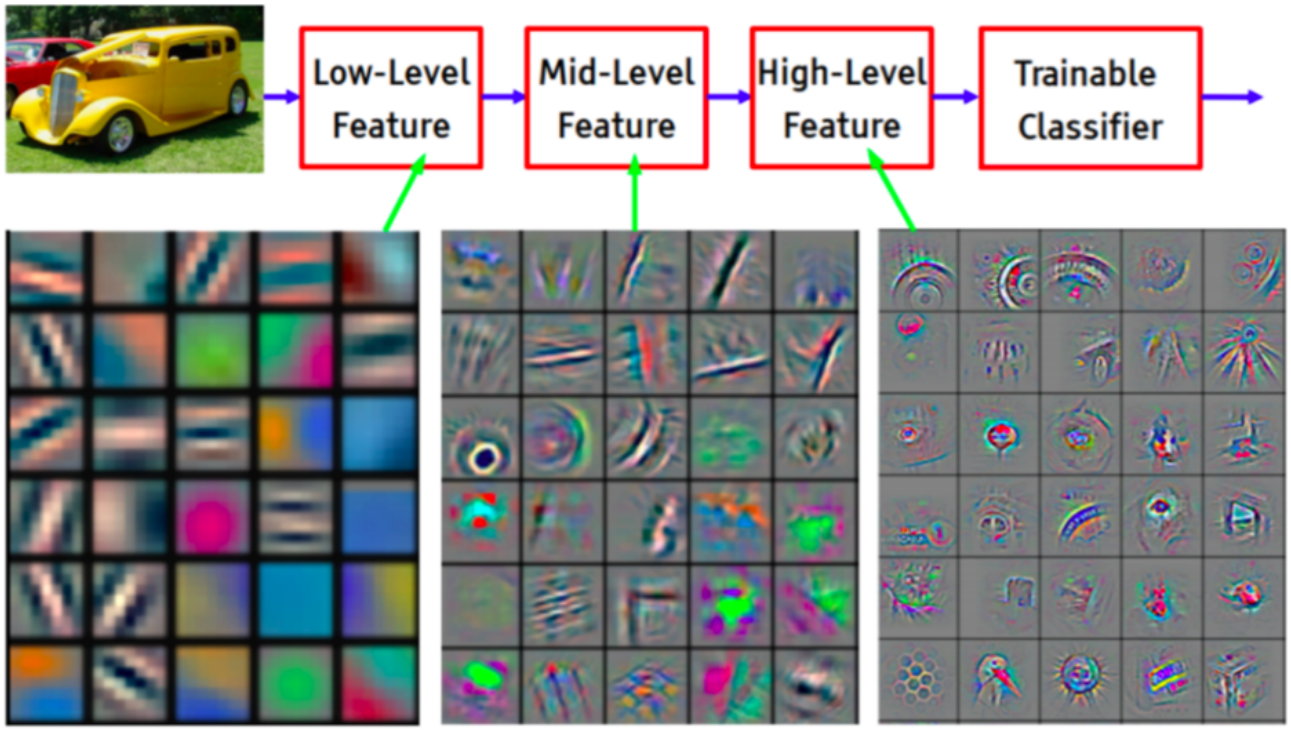


Figure 2: The architecture of VGG16 model .

CNN Feature Visualization

Each CNN layer is responsible for capturing a different level of features as can be see from from ImagiNet [21]

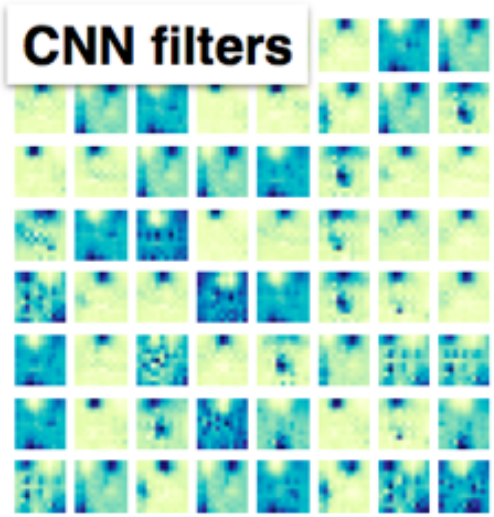


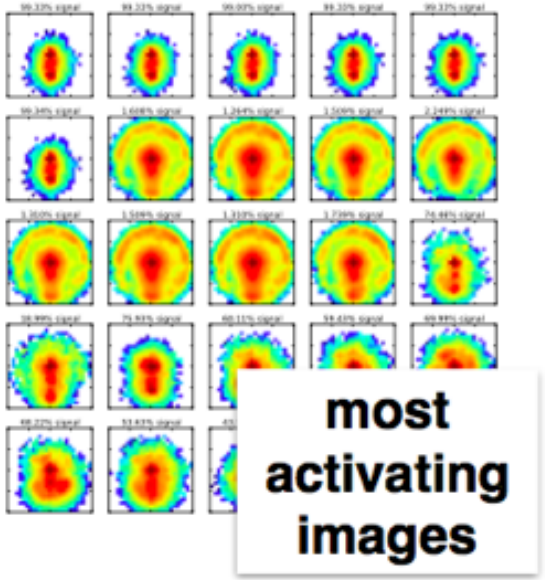
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

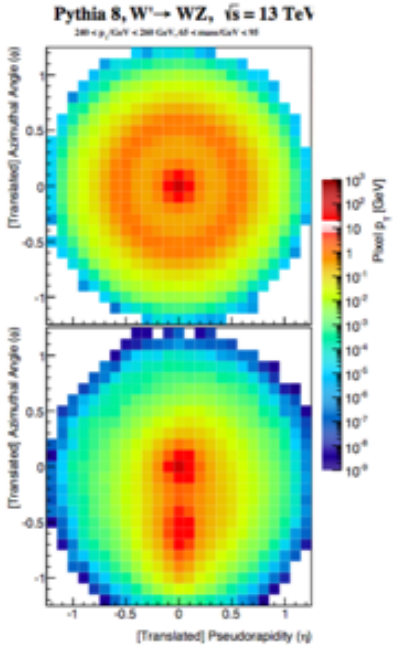
[21] https://arxiv.org/pdf/1311.2901

CNN Application in HEP: Jet ID

Jet images with convolutional nets



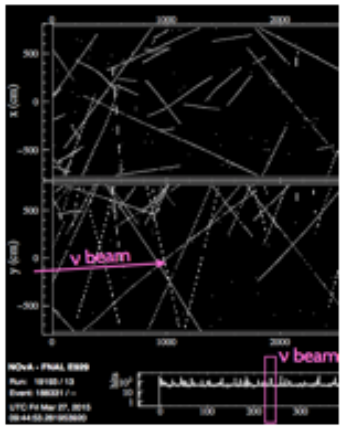


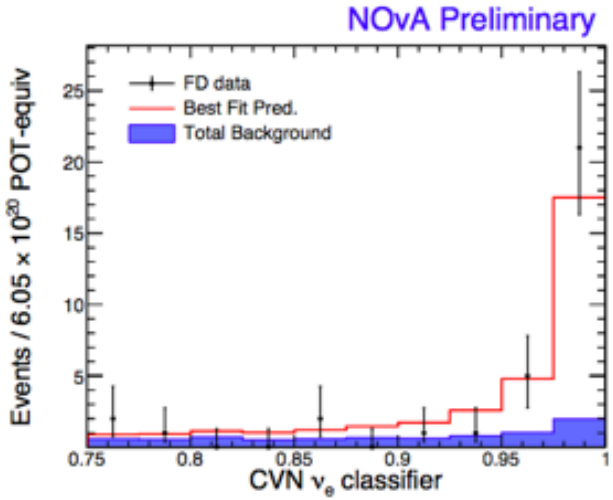


L. de Oliveira et al., 2015

CNN Application HEP: Neutrino ID

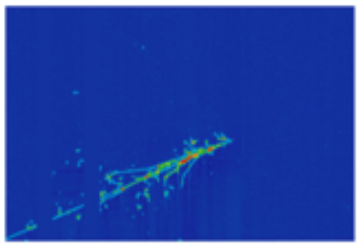
Neutrinos with convolutional nets

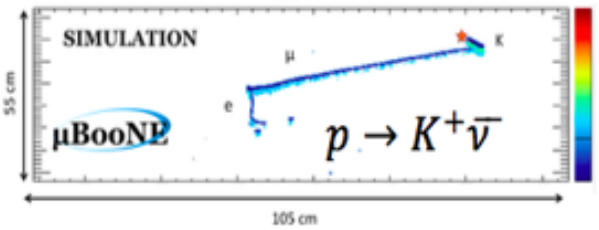


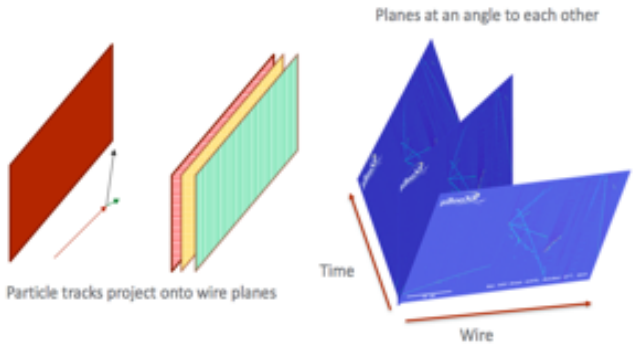


An equivalent increased exposure of 30%

Aurisiano et al. 2016







μBooNE

Recurrent Neural Network(RNN)

Sequential data is a data stream (finite or infinite) which is interdependent (ex: text , speech and video). Feed forward networks can't learn any sort of correlation between previous and current input !

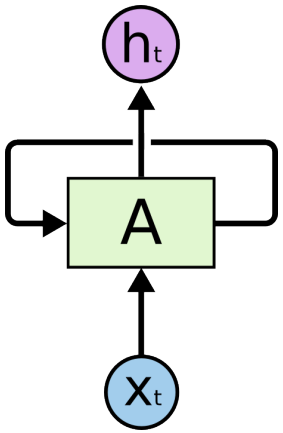
Recurrent Neural Networks (RNN)

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

RNN Layers and Loss [22]

The hidden layer output depends not only on the current input, but also on the entire history of past inputs. The output layer is squashed by a softmax and error evaluated by a log loss

1. **Hidden Layer:** $\vec{h}^{[t]} = Tanh(\vec{W}_{xh}\vec{x}^{[t]} + \vec{W}_{hh}\vec{h}^{[t-1]} + b_h)$
2. **Output Layer:** $p^{[t]} = Softmax(\vec{W}_{hy}\vec{h}^{[t]} + b_y)$
3. **Loss Function:** $L = -log(p^{[t]})$



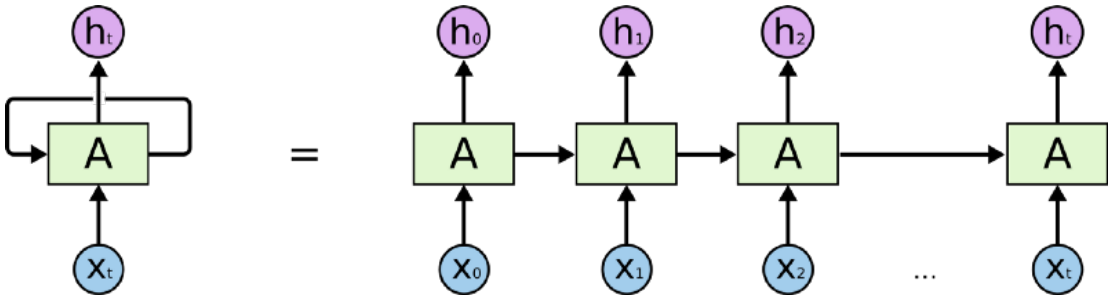
[22] https://eli.thegreenplace.net/2018/understanding-how-to-implement-a-character-based-rnn-language-model

Recurrent Neural Network(RNN)

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.

RNN Unrolling

Unrolling is a visualization and conceptual tool, which views a RNN hidden layer as a sequence of layers that you train one after another with backpropagation.



Backpropagation Through Time (BPTT)

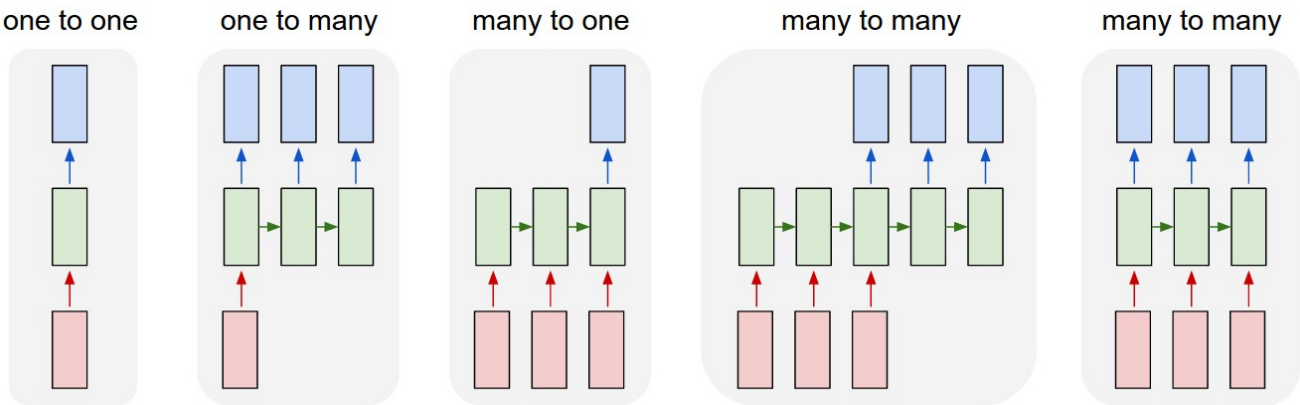
Backpropagation through time is basically a fancy buzz word for doing backpropagation on an unrolled RNN. In BPTT the error is back-propagated from the last to the first timestep, allowing the error calculation for each timestep and weights updating

Recurrent Neural Network(RNN)

RNN Properties

- RNNs combine the input vector with their state vector to produce a new state vector
- It can take as input variable size data sequences, as the recurrent transformation can be applied as many times as we like
- RNN allows one to operate over sequences of vectors: sequences in the input, the output, or in the most general case both

Bellow, input vectors are in red, output vectors are in blue and green vectors hold the RNN's state [23]



There are no constraints on the sequences lengths because the recurrent transformation (green) can be applied as many times as we like

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

Long Short Term Memory(LSTM)

Long Short-Term Memory (LSTM)

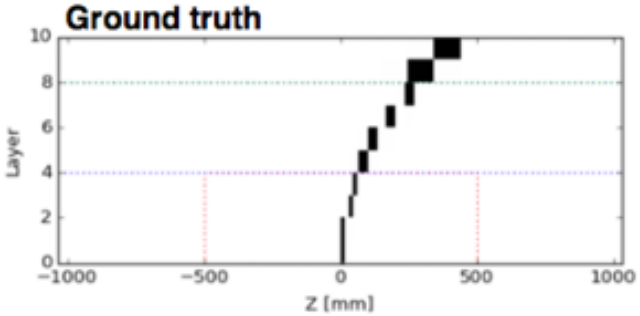
Long Short Term Memory [24] networks are a special kind of RNN, capable of learning long-term dependencies. LSTM neurons categorize data into short term and long term memory cells, enabling them figure out what data is important and should be remembered and what can be forgotten.

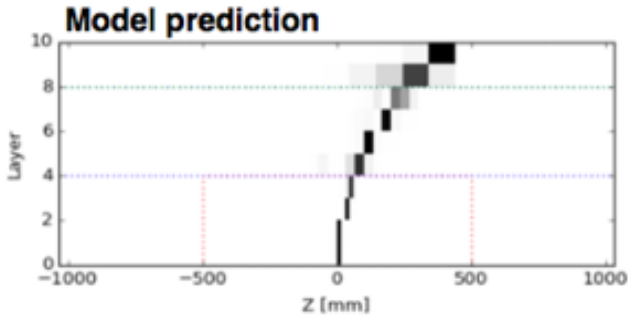
- LSTM are suited to learn from important experiences that have very long time lags in between
- In an LSTM you have three gates: input, forget and output gate.
- The gates determine whether or not to let new input in (input gate), delete unimportant information (forget gate) or to let it impact the current output (output gate).

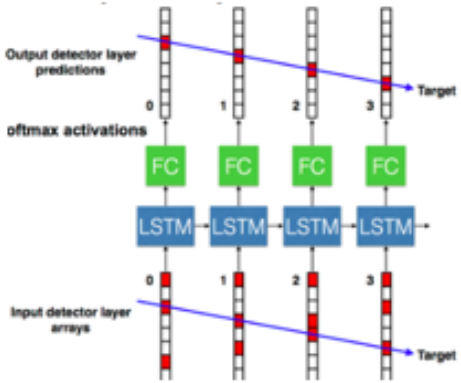
[24] http://colah.github.io/posts/2015-08-Understanding-LSTMs

LSTM Application in HEP: Tracking

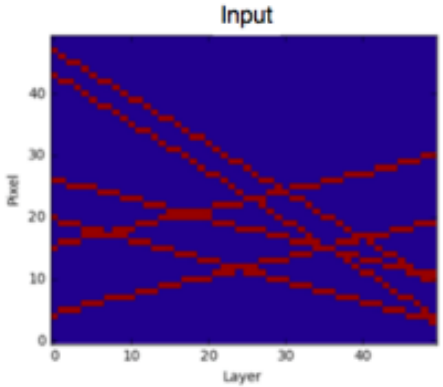
Tracking with recurrent nets (LSTM)

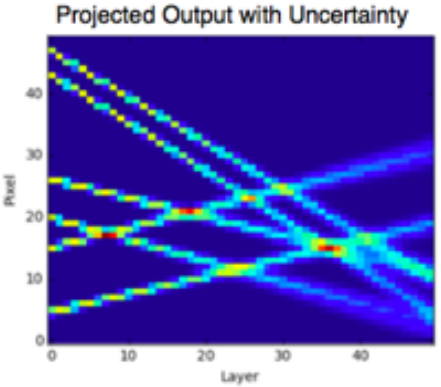






Time dimension (state memory)





HEP.TrkX, CHEP 2018

Residual Neural Network (ResNEt)

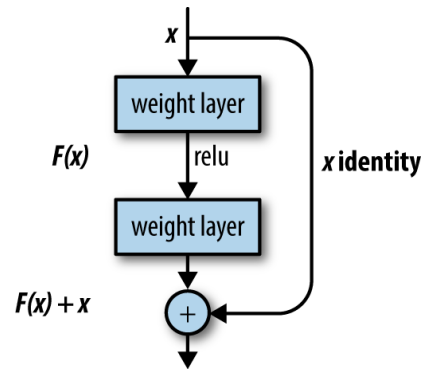
Training very deep networks is difficult because of the vanishing gradients problem. ResNets [25] try avoid this by reusing activations from a previous layer until the layer next to the current one have learned its weights.

Residual Neural Network (ResNet)

A Residual Neural Network is a network inspired on the pyramidal cells of cerebral cortex. The network skip connections or short-cuts to jump over some , using as inputs not only the previous neuron output, but also it's input.

ResNet Neurons

$$a^{(l)} = F(z_i^{(l)}) + a^{(l-2)} \text{ , where } z^{(l)} = W^{(l-1,l)} a^{(l-1)}$$

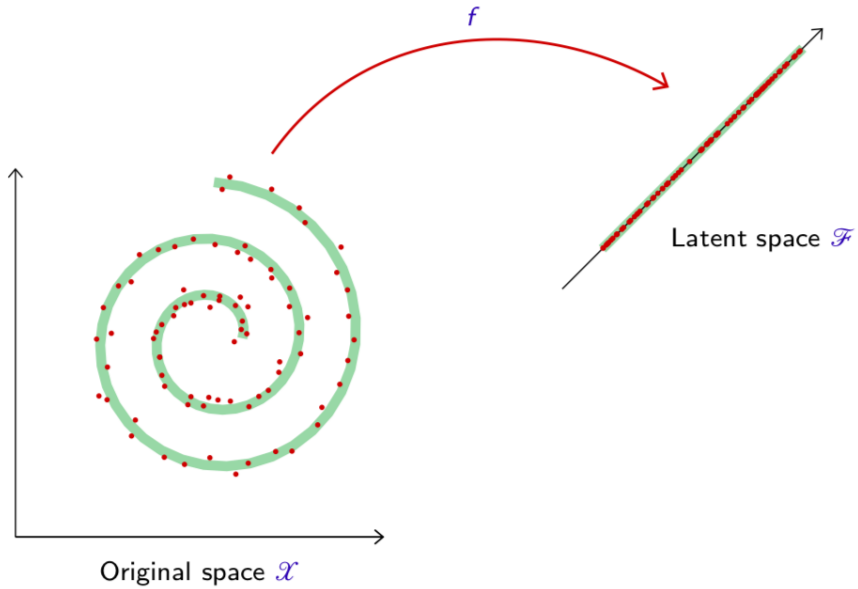


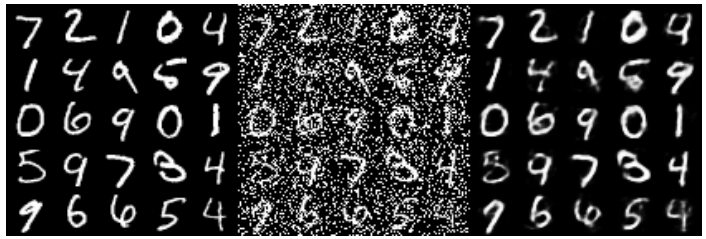
In the initial training stage the weights will adapt to mute layers, collapsing the network into fewer layers and making it easier to learn. Then as it learns it gradually expands the layers.

[25] https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

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



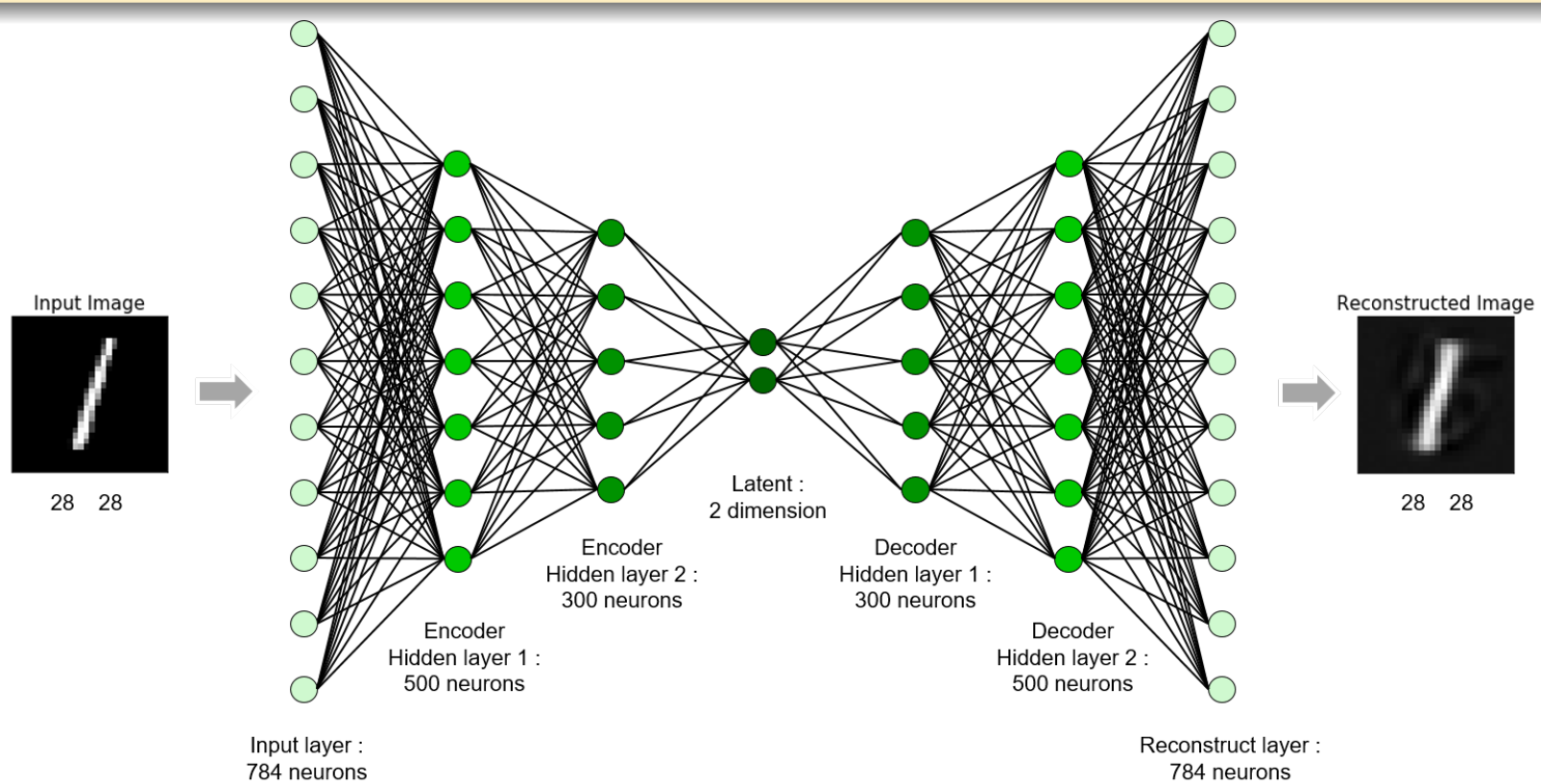


Traditionally, autoencoders were used for dimensionality reduction and denoising.
Recently, autoencoders are being used also in generative modeling

Autoencoder

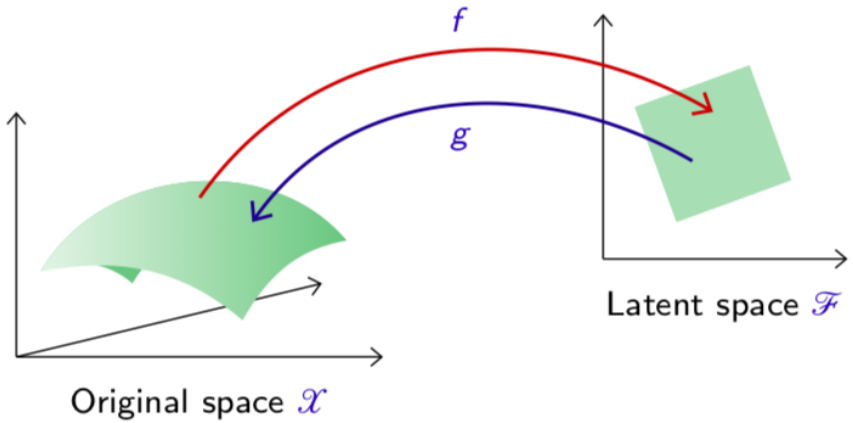
Autoencoder(AE)

An Autoencoder(AE) is a neural network that is trained to attempt to copy its input to its output in an *unsupervised way*. It may be viewed as consisting of two parts: an encoder function $l = f(x)$ and a decoder that produces a reconstruction $y = g(l)$. In doing so, it learns a representation(encoding) of the data set features l in a latent space.



Autoencoder (AE)

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.

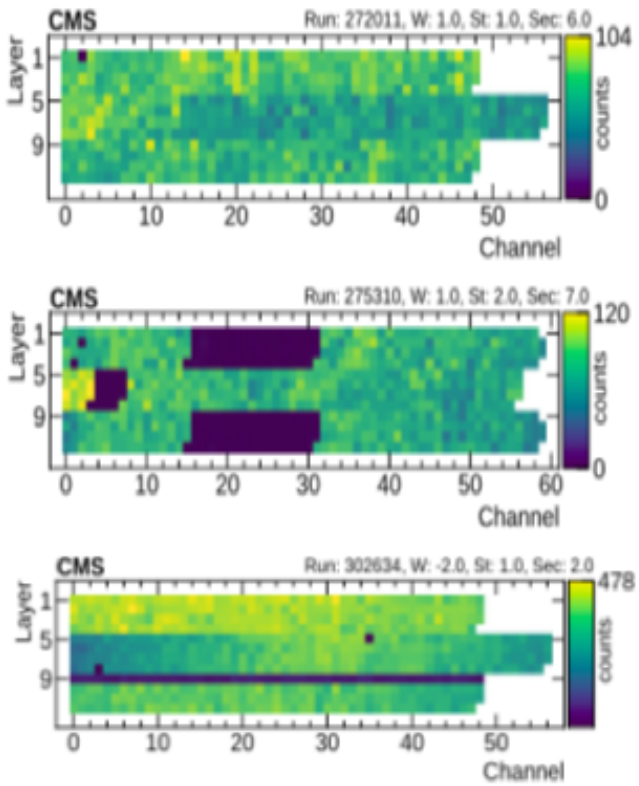


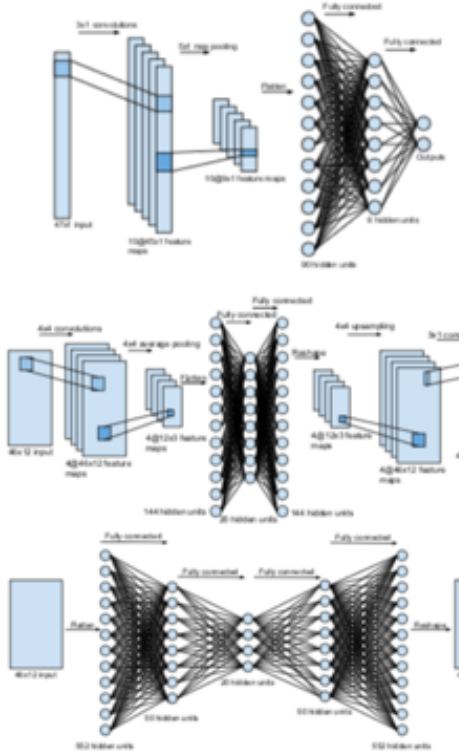
Autoencoder Loss Function

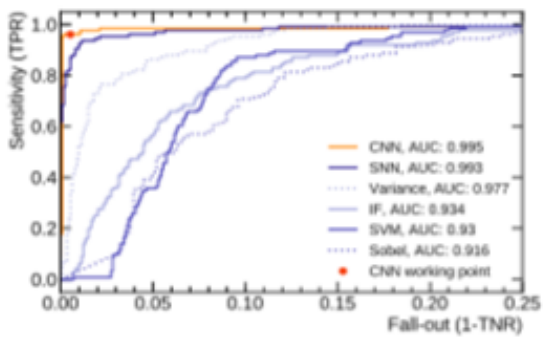
$$L = \| X - f \circ g(X) \|^2$$

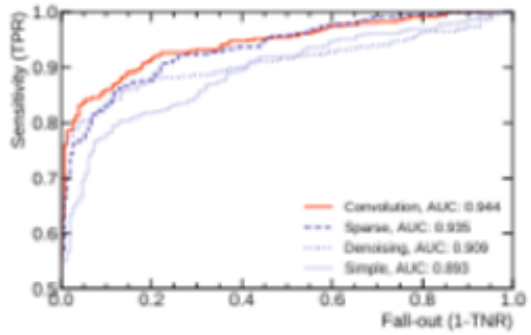
- An AE becomes useful by having a latent hidden layer smaller than the input layer, forcing it to create a compressed representation of the data by learning correlations.
- AE layers can be feed forward, convolutional or even recurrent, depending on the application
- One can train an AE on images and save the encoded vector to reconstruct (generate) it later by passing it thorugh 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 !

Autoencoder Application in HEP: DQM









Variational Autoencoder(VAE)

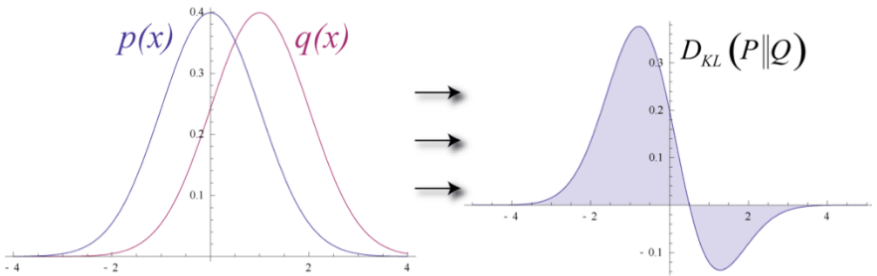
Variational Autoencoder(VAE)

A Variational Autoencoder(VAE) is an autoencoder with a loss function penalty (KL divergence) 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 Kullback–Leibler(KL) divergence, or relative entropy, is a measure of how one probability distribution differs from a second, reference distribution

Kullback–Leibler Divergence

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



Original Gaussian PDF's KL Area to be Integrated

VAE Loss

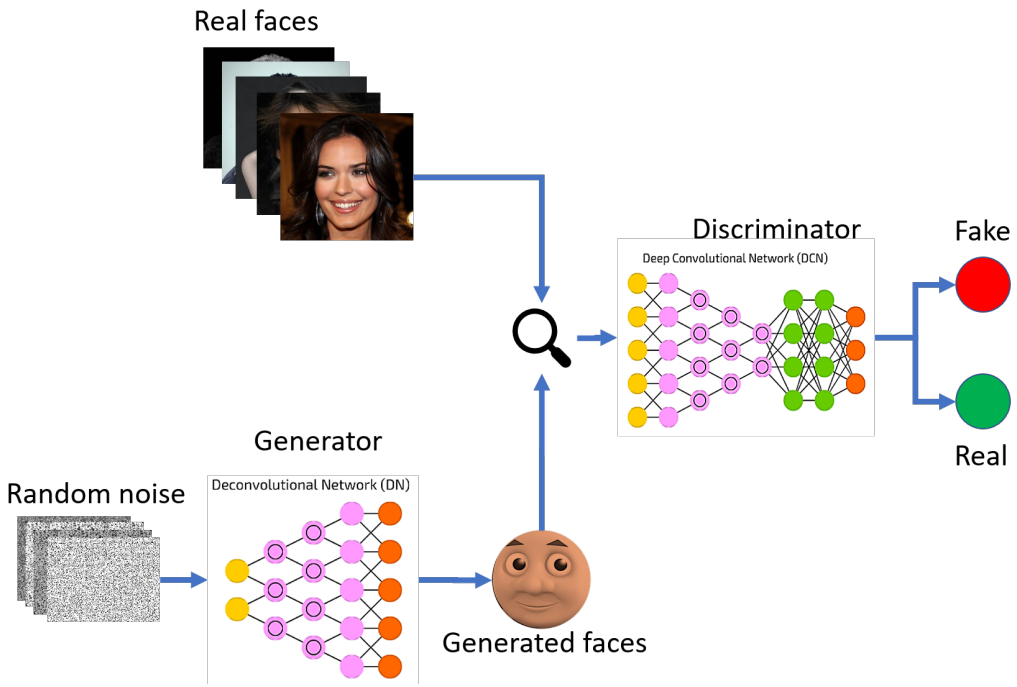
The VAE loss function is composed of a mean squared error (generative) loss that measures the reconstruction accuracy, and a KL divergence (latent) loss that measures how close the latent variables match a gaussian.

$$L = \| x - f \circ g(x) \|^2 + D_{KL}(I|\hat{I})$$

Generative Adversarial Network(GAN)

Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GANs) are composed by two neural networks, where one generates candidates (generative), while the other classifies them (discriminative). Typically, the generative network learns a map from a latent space to data, while the discriminative network discriminates between generated and real data.

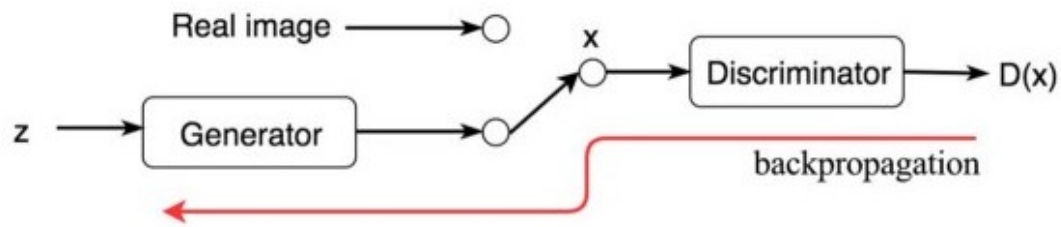


Generative Adversarial Network(GAN)

The GAN training [26] aims to increase the discrimination error rate, ”fooling”the discriminator network, by generating images that looks like real. Menwhile, the discriminator is constantly trying to find differences between the generated and real images. In practice, the two neural networks compete by “training” each other

GAN Training

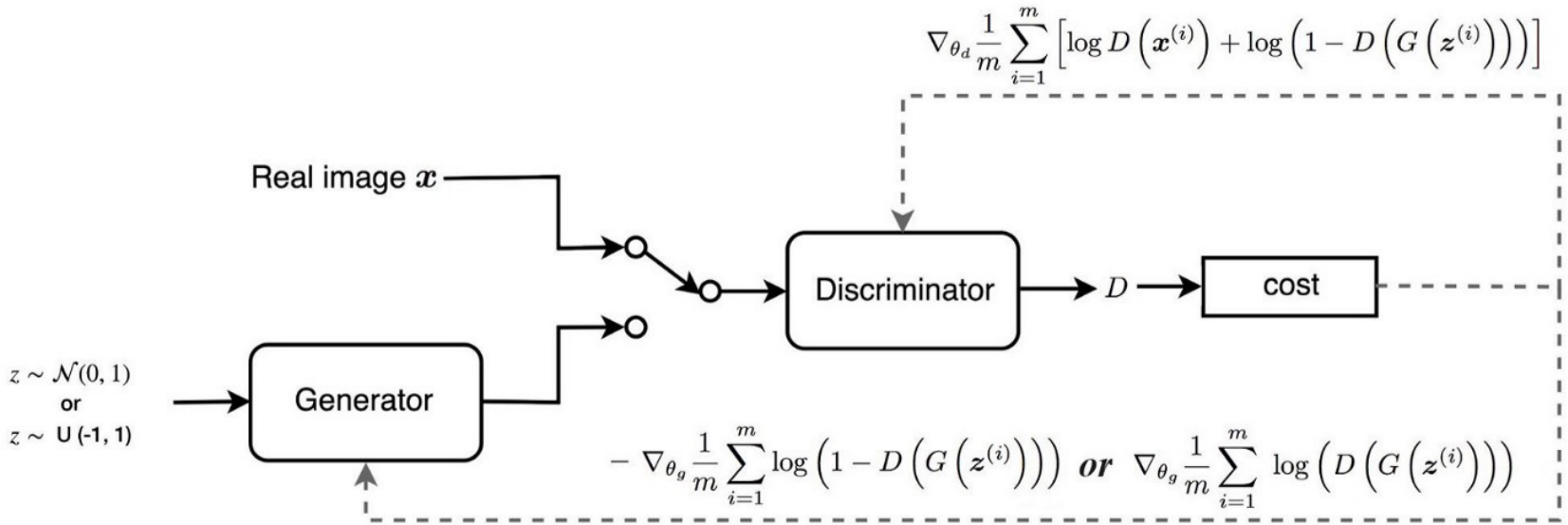
1. The generator is seeded with a randomized input that is sampled from a predefined latent space (ex: multivariate normal distribution).
2. Samples synthesized by the generator and real data are evaluated by the discriminator. It’s trained just like a classifier, so if the input is real, we want output=1 and if it’s generated, output=0
3. We want the generator to create images with discriminator output=1. So we can train the generator by backpropagating this target value all the way back to the generator, i.e. we train the generator to create images that the discriminator thinks are real !
4. Both networks are trained in alternating steps and in competition to improve themselves.



[26] 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 [27]



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

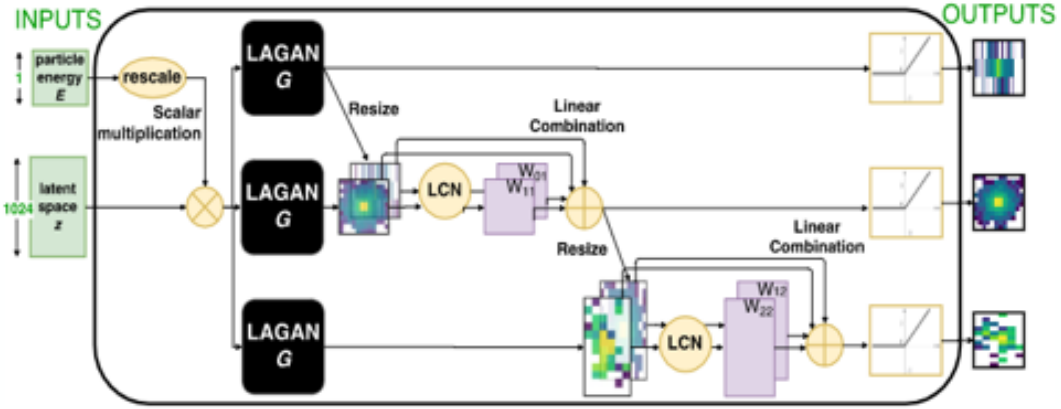
Generative Adversarial Network(GAN)

GANs are quite good on faking celebrities images and Monet style painitings !

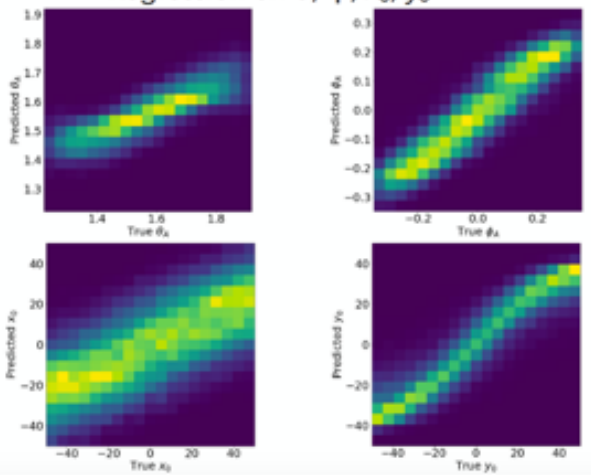
Training Data
(CelebA)

Sample Generator
(Karras et al, 2017)

GAN Application in HEP: MC Simulation

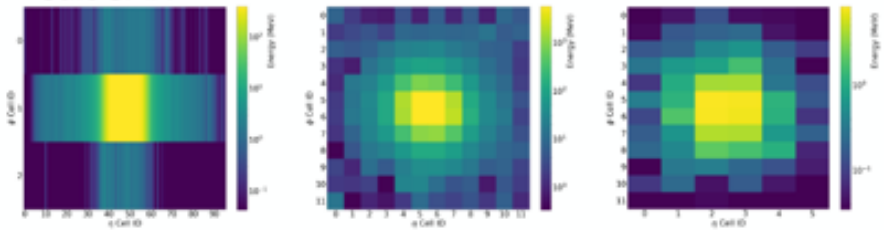


- Regression on θ, φ, x_0, y_0

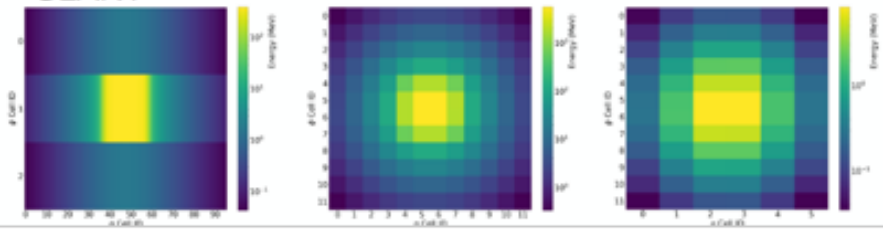


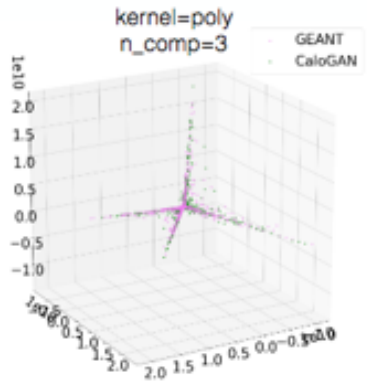
Dataset: 5°; Net: soft sparsity, multiplied E, Conv. attn. and layers

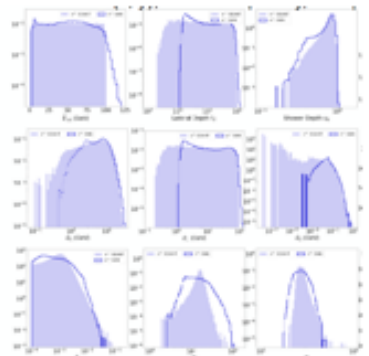
- CaloGAN



- GEANT







L. de Oliveira et al., 2017