Machine Learning in HEP

A. Sznajder

UERJ
Instituto de Fisica

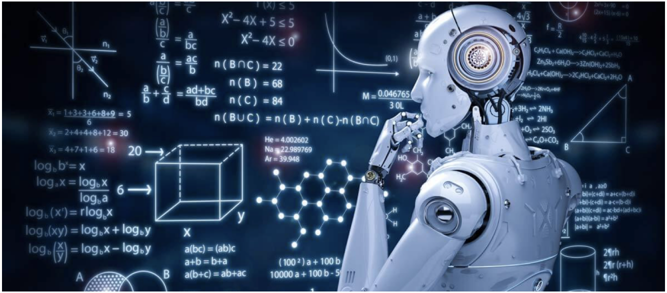
12 de março de 2020

Outline

What is Machine Learning ?

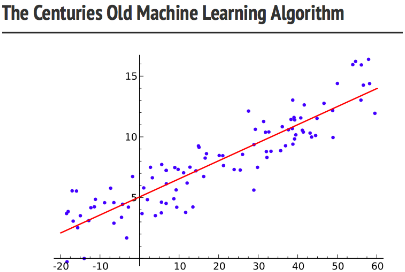
Machine Learning (ML)

Machine learning (ML) is the study of computer algorithms capable of building a model out of a data sample. It learns from examples , 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



Linear regression (Source)

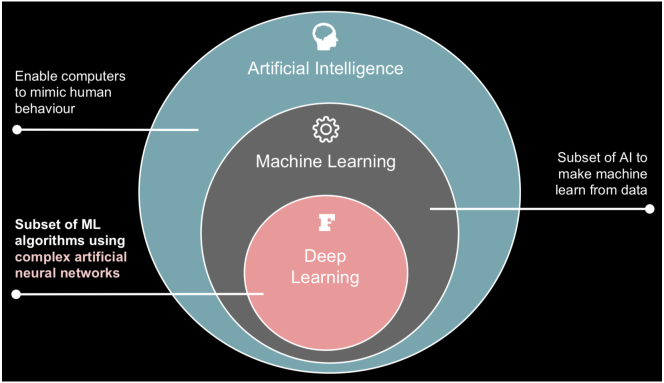
Linear regression is a bit too wimpy a technique to solve the problems of image , speech or text recognition, but what it does is essentially what supervised ML is all about: learning a function from a data sample

---

[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



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

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

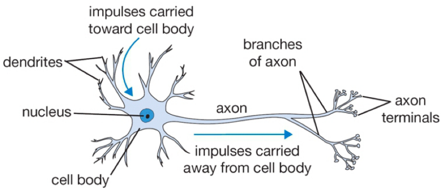
- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

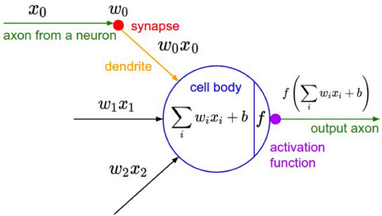
3) Neural Networks Architectures

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

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

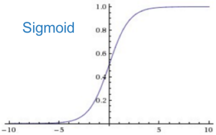
Artificial Neuron (Node)

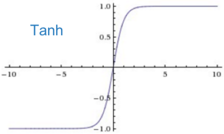
Each node of a NN receives inputs $\vec{x} = \{x_1, \ldots, x_n\}$ from other nodes or an external source and computes an output y according to the expression

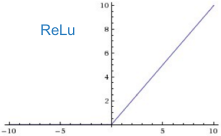
$$y = F\left(\sum_{i=1}^{n} W_i x_i + B\right) = F\left(\vec{W}.\vec{x} + B\right) \tag{1}$$

, 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

---

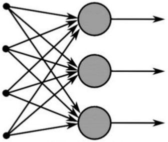
[3] Nonlinear activation is fundamental for nonlinear decision boundaries

Neural Network Topologies

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

Feed Forward Networks

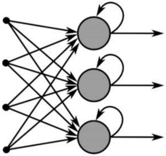
In a Feedforward Neural Network (FFN) connections between the nodes do not form a cycle. The information always moves one direction, from input to output.



Feed-Forward Neural Network

Recurrent Neural Network

A Recurrent Neural Network (RNN) can have connections between nodes in the same layer or with themselves. RNNs can use their internal state (memory) to process sequential data.



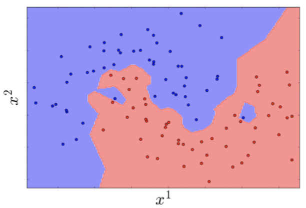
Recurrent Neural Network

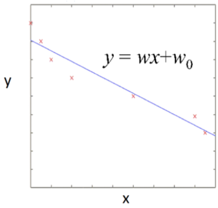
A NN layer is called a dense layer to indicate that it's fully connected.

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:** $\vec{t}_i \in \mathbb{R}^n$

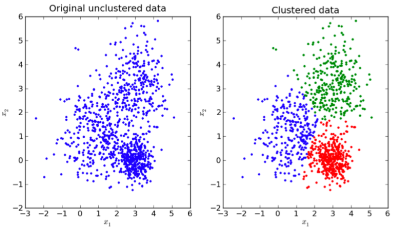


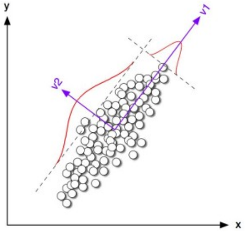


Unsupervised Learning

Unsupervised Learning

- No labeled data is used at training and the NN finds patterns within input data
- Given some data $D = \{\vec{x}_1, \ldots, \vec{x}_n\}$, but no labels, find structures in the data
  - **Clustering:** partition the data into sub-groups $D = \{D_1 \cup D_2 \cup \ldots D_k\}$
  - **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 $n >> m$



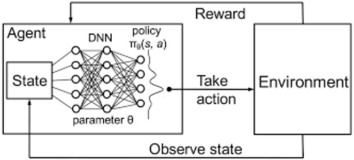


Reinforcement Learning

Reinforcement Learning

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 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: robotics, self-driving vehicles, gameplay ...

Reinforcement Learning

Google Deepmind neural network playing Atari game [4]

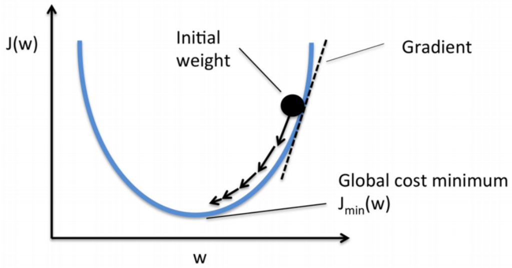
---

[4] https://www.youtube.com/watch?v=TmPfTpjtdgg

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

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 [5] (Classification)

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

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

Cost (Objective) Function

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

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

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

---

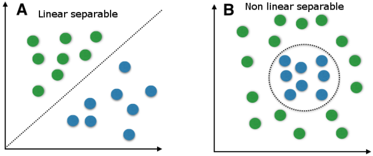
[5] For $m = 2$ we have the Binary Cross Entropy $L = -t \log y - (1 - t) \log(1 - y)$

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 [6]

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



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

Perceptron Learning Algorithm

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

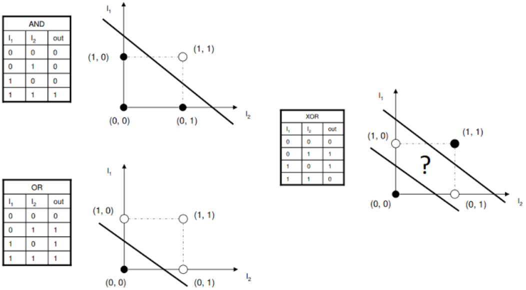
---

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

Perceptron and XOR Problem

Problem:

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



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 [7] . It is the simplest feed forward NN. [8]

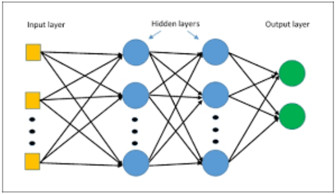
Multilayer Perceptron Model

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

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



A MLP can be seen as a parametrization of a mapping $F_{w,b} : \mathbb{R}^n \rightarrow \mathbb{R}^m$

---

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

[8] 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

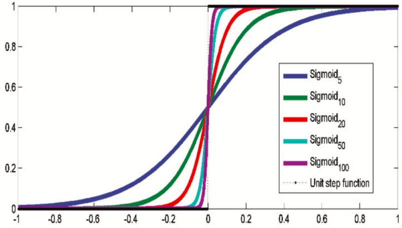
A single hidden layer feed forward neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden neurons [9]

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

Sigmoid → Step Function

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

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



---

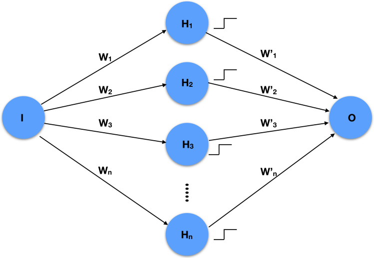
[9] Cybenko,G.(1989) Approximations by superpositions of sigmoidal functions, Math.ofCtrl.,Sig.,andSyst.,2(4),303
Hornik,K.(1991) Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks, 4(2), 251

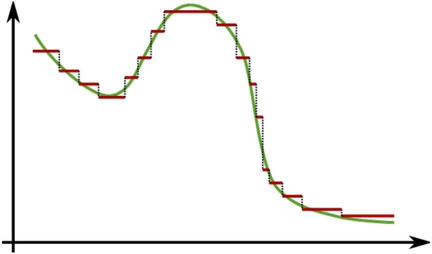
Multilayer Perceptron as a Universal Approximator

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

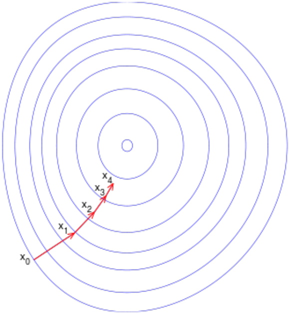
Gradient Descent

A multi-variable function $F(\vec{x})$ decreases fastest in the direction of its negative gradient $-\nabla F(\vec{x})$ [10].

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 !



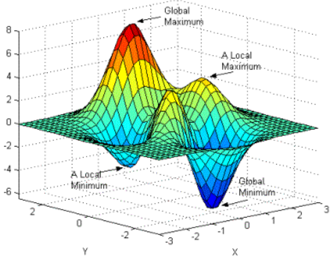
Gradient Descent uses 1^o derivatives, which is efficiently and simply calculated by backpropagation. Methods like Newton uses 2^o derivative(Hessian), which is computationally costly and memory inefficient.

---

[10] 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 [11] 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

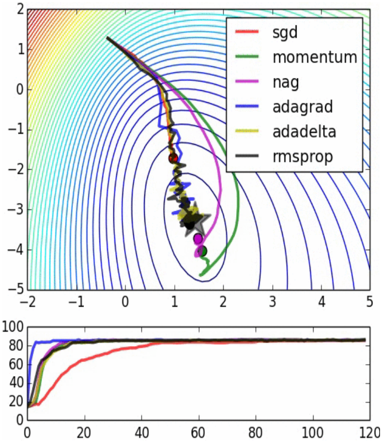
---

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

SGD Algorithm Improvements [13]

SGD Variants [12] :

- **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



---

[12] http://danielnouri.org/notes/category/deep-learning
[13] 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 [14]

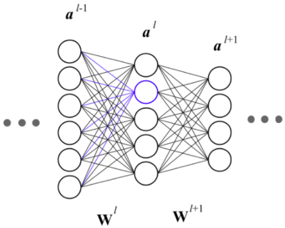
MLP Loss Function

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.



---

[14] 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

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

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

Backpropagation Formulas

The backpropagation master formulas 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 $\delta_k^{(l)} = \dfrac{\partial L}{z_k^{(l)}}$

→ 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: $\dfrac{\partial L}{\partial W_{kj}^{(l)}} = \delta_k^{(l)} a_k^{(l-1)}$
6. Update the weights according to the gradient descent: $\Delta W_{kj}^{(l)} = -\lambda \dfrac{\partial L}{\partial W_{kj}^{(l)}}$

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 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

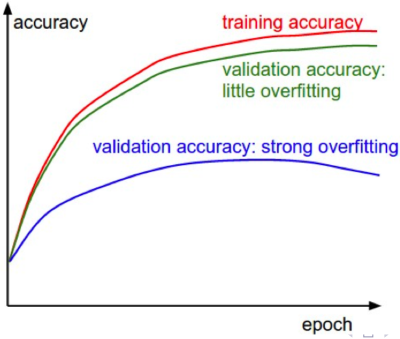
Final performance evaluation after finished training and hyper-parameters are fixed. Use the test dataset for an independent evaluation of performance obtaining a ROC curve

Overtrainging(Overfitting)

Overtraining(Overfitting)

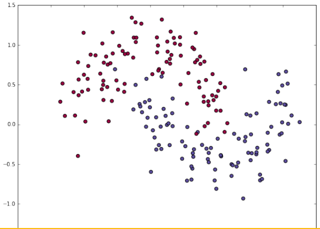
Gap between training and validation accuracy indicates the amount of overfitting

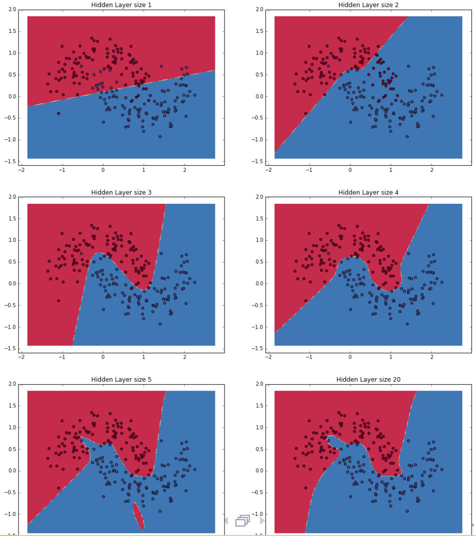
- If validation error curve shows small accuracy compared to training it indicates overfitting ⇒ add regularization or use more data.
- If validation accuracy tracks the training accuracy well, the model capacity is not high enough ⇒ use larger model



Overfitting - Hidden Layer Size X Decision Boundary [15]

- Hidden layer of low dimension captures well the general trend of data.
- Higher dimensional layers 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





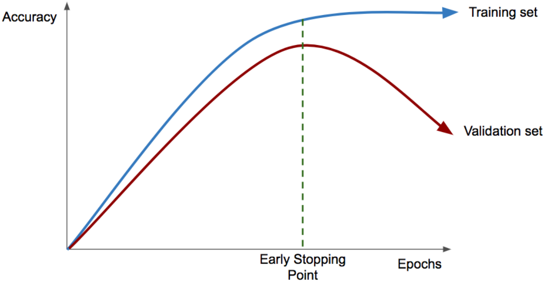
Regularization

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.

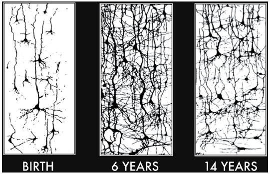


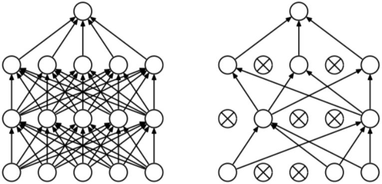
Dropout Regularization

Dropout Regularization

Regularization inside network that remove nodes randomly during training.

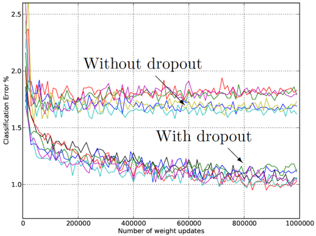
- It's ON in training and OFF in validation and testing
- Avoid co-adaptation on training data
- Essentially a large model averaging procedure





(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

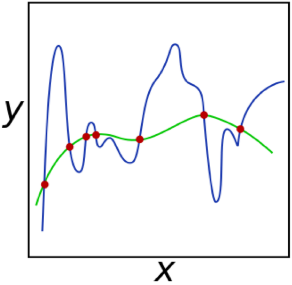
Works by keeping weights sparse

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

L2 Regularization

Works by penalising large weights

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

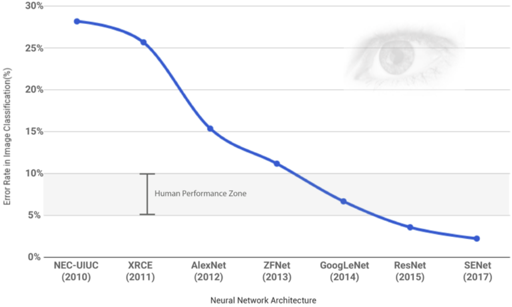


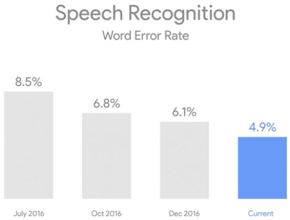
The combined $L1 + L2$ regularization is called Elastic

Why Deep Learning ?
and
Why Now ?

Why Deep Learning ?

Image and Speech Recognition perfomance (DNN versus Humans)

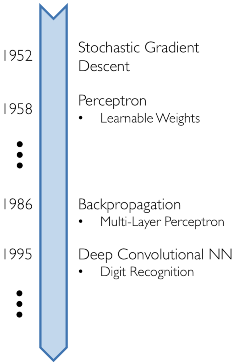




https://arxiv.org/pdf/1409.0575.pdf

Why Now ?

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



1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

⋮

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

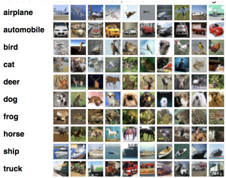
⋮

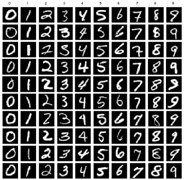
The main catalysts for the current Deep Learning revolution have been:

- **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 [16]





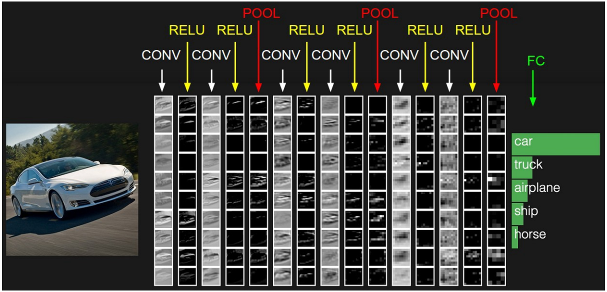
---

[16] 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 [17]



⇒ DNN allows a hierarchical representation of data features !

---

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

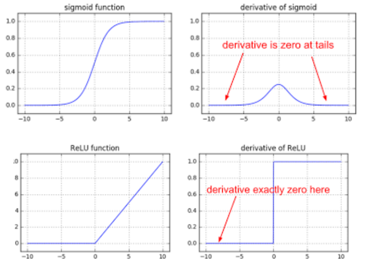
Deep Learning - Vanishing Gradient Problem

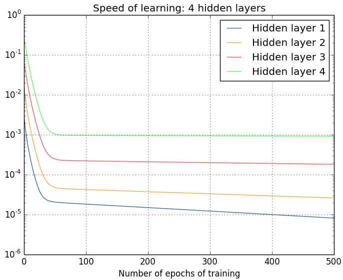
Vanishing Gradient Problem [18]

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 !!!





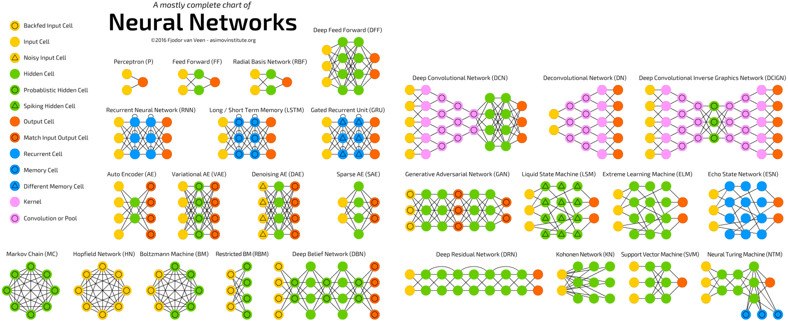
---

[18] 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 [19]



---

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

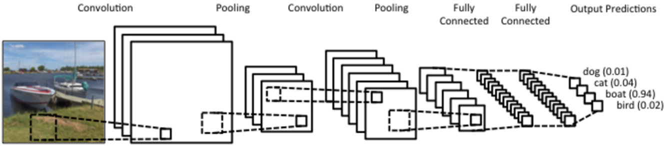
Convolutional Neural Network

For high-dimensional inputs as images, fully connected NN is computationally too expensive (ex: 1000x1000 image has $O(10^6)$ pixels). Convolutional neural network [20] emulate the visual cortex, where neurons respond to stimuli only in a restricted region of the visual field.

Convolutional Neural Network (CNN or ConvNet)

CNN mitigates 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)



---

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

CNN - Convolutional Layer

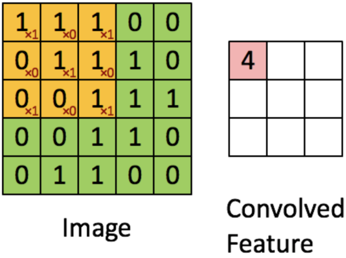
Convolutional Layer

CNN convolutional layer [21] is its core building block and its parameters are learnable filters (kernels), that extracts image features corresponding to a small receptive field.

A convolution is like a sliding window transformation that applies a filter to extract local image features

Discrete Convolution

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

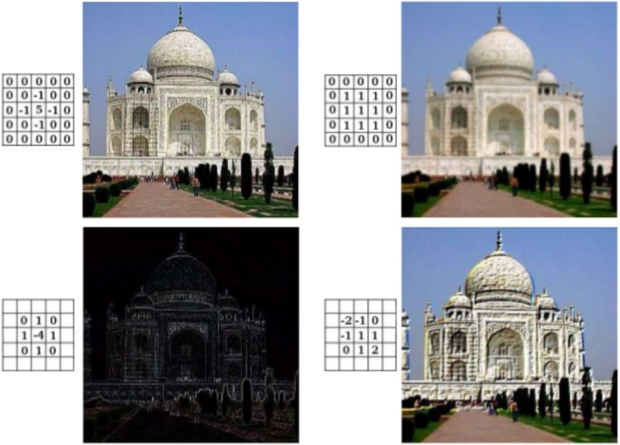


---

[21] 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) [22]



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	

---

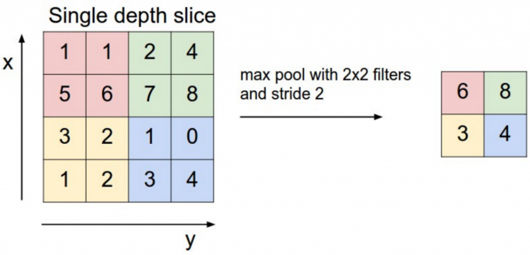
[22] https://docs.gimp.org/2.6/en/plug-in-convmatrix.html

CNN - Pooling Layer

Pooling(Downsampling) Layer

The pooling (downsampling) layer [23] has no learning capabilities and serves the purpose of decreasing the representation size (reduce computation)

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



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

---

[23] 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

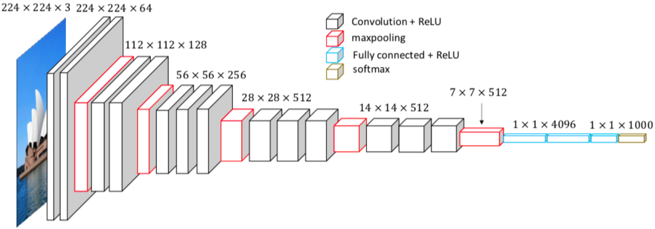
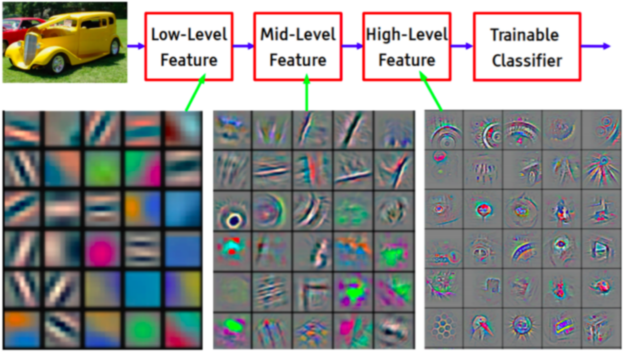


Figure 2: The architecture of VGG16 model .

CNN Feature Visualization

Each CNN layer is responsible for capturing a different level of features as can be seen from ImagiNet [24]



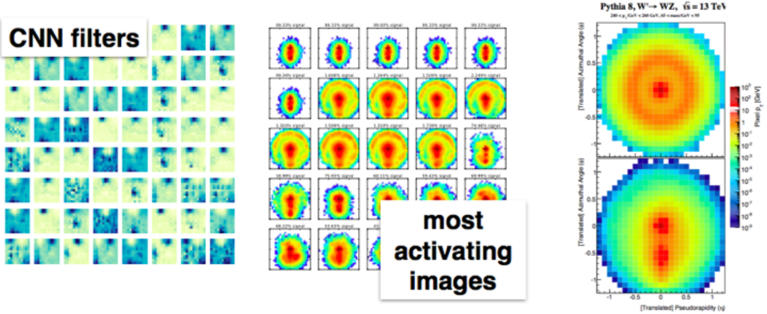
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

---

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

CNN Application in HEP: Jet ID

Jet images with convolutional nets

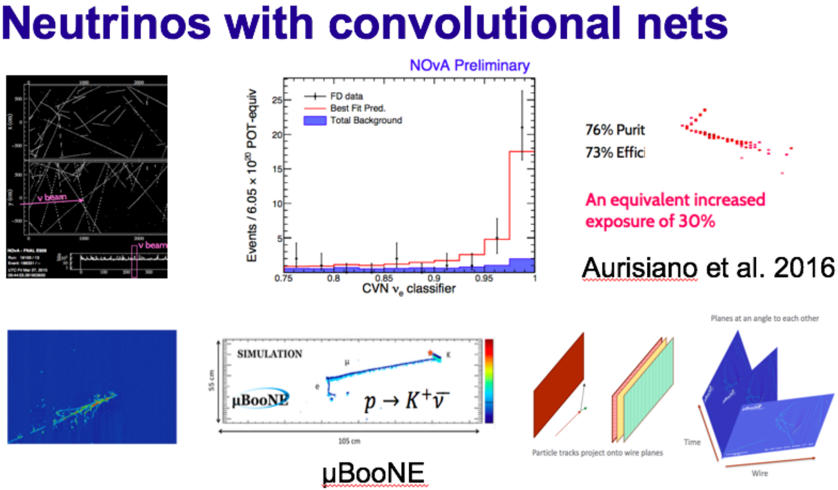


CNN filters

most activating images

L. de Oliveira et al., 2015

CNN Application HEP: Neutrino ID



Neutrinos with convolutional nets

NOvA Preliminary

76% Purit
73% Effici

An equivalent increased exposure of 30%

Aurisiano et al. 2016

μBooNE

Sequential Data

Sequential Data

Sequential data is an interdependent data stream (ex: text , audio and video)

The food was good, not bad at all.

vs.

The food was bad, not good at all.

Sequences and Inteligence

- Sequences are very important for inteligence
- Brain is all the time predicting what comes ahead ...
- If prediction is wrong the brain learns and corrects
- We memorize sequences for alphabet, words, phone numbers and not just symbols !

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

Recurrent Neural Network(RNN)

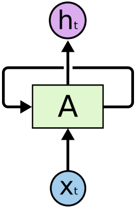
Recurrent Neural Networks (RNN)

RNNs are networks that use feedback loops to process sequential data [25]. These feedbacks allows information to persist, which is an effect often described as memory.

RNN Cell (Neuron [26])

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



---

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

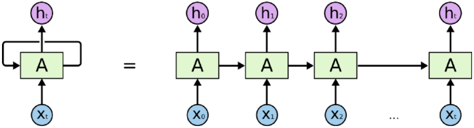
[26] https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks

Recurrent Neural Network(RNN)

RNNs [27] 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 hidden layer as a sequence of layers that you train one after another using backpropagation.



Backpropagation Through Time (BPTT)

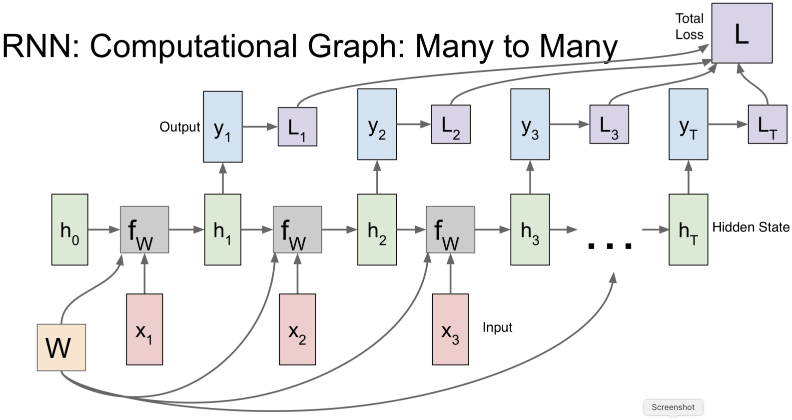
Backpropagation through time is just a fancy buzz word for backpropagation on an unrolled RNN. The error is back-propagated from the last to the first timestep

RNNs unfolded in time can be seen as very deep networks ⇒ difficult to train !

---

[27] `http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-`

Recurrent Neural Network(RNN)



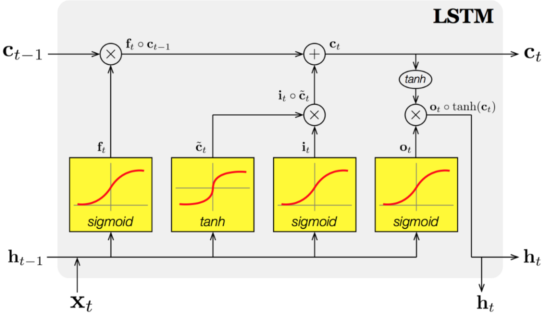
The same set of weights are used to compute the hiden state and output for all time steps

Long Short Term Memory(LSTM)

Long Short-Term Memory (LSTM) Network

LSTM [28] is a special 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

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

---

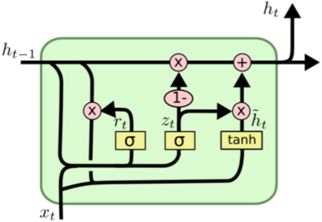
[28] `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

Gated Recurrent Network (GRU)

Gated Recurrent Network (GRU)

GRU [29] 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



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

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

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

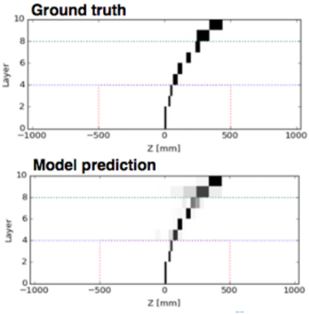
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

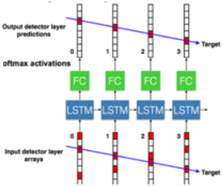
---

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

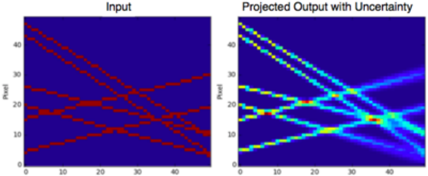
LSTM Application in HEP: Tracking

Tracking with recurrent neural networks (LSTM) [30]





Time dimension (state memory)

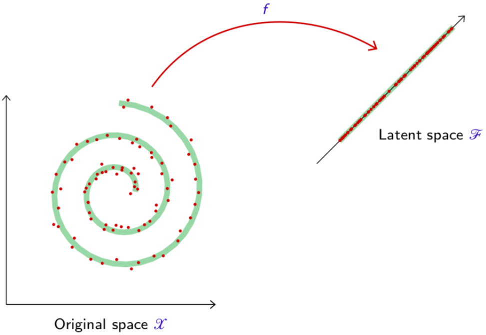


---

[30] `https://heptrkx.github.io`

Autoencoder

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



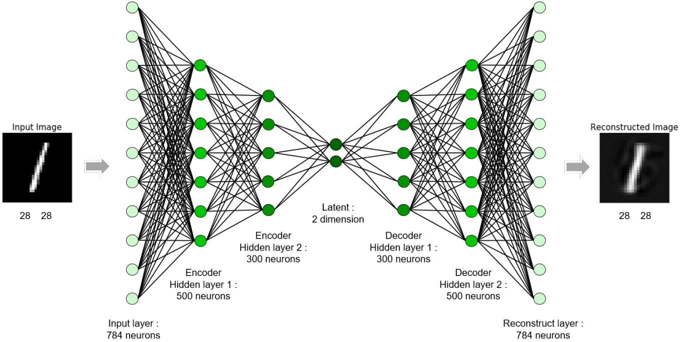
Traditionally, autoencoders were used for dimensionality reduction and denoising. Recently autoencoders are being used also in generative modeling

Autoencoder(AE)

Autoencoder(AE)

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

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



$\Rightarrow$ Understanding is data reduction

Denoising Autoencoder (DAE)

Denoising Autoencoder (DAE)

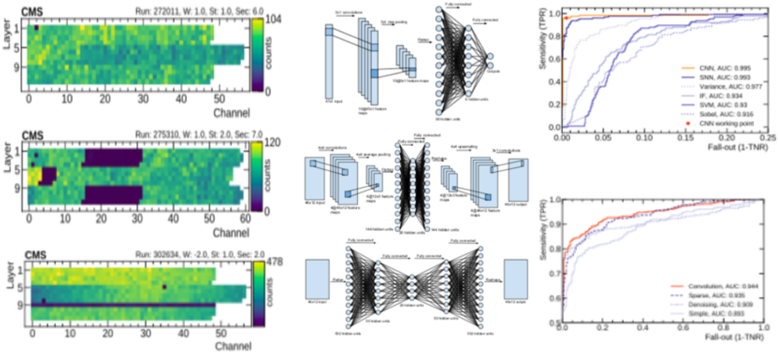
The DAE is an extension of a classical autoencoder where one corrupts the original image on purpose by adding random noise to it's input. The autoencoder is trained to reconstruct the input from a corrupted version of it and then used as a tool for noise extraction

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

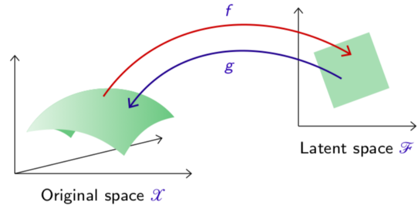
Detector Quality Monitoring (DQM)

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



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

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

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)

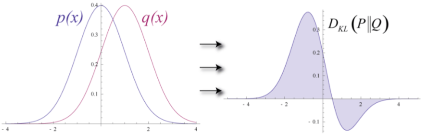
Variational Autoencoder(VAE)

A VAE [31] is an autoencoder with a loss function penalty Kullback–Leibler (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 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

---

[31] `http://kvfrans.com/variational-autoencoders-explained`

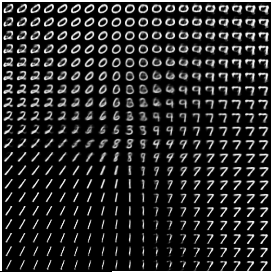
Variational Autoencoder(VAE)

VAE Loss

The VAE loss [32] 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.

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

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



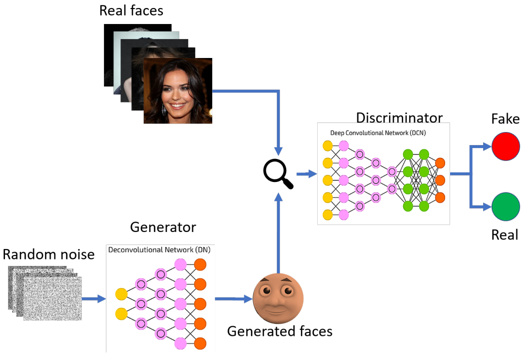
---

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

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.

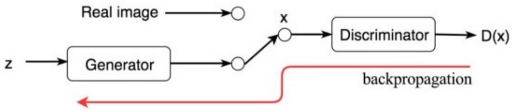


Generative Adversarial Network(GAN)

GAN adversarial training [33] 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

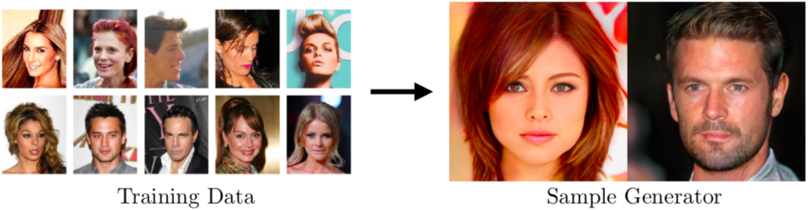


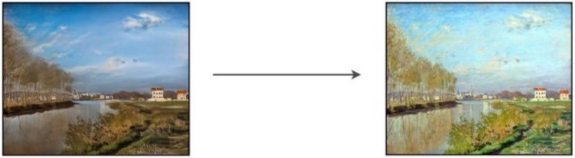
---

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

Generative Adversarial Network(GAN)

GANs are quite good on faking celebrities images [34] or Monet style painitings [35] !





---

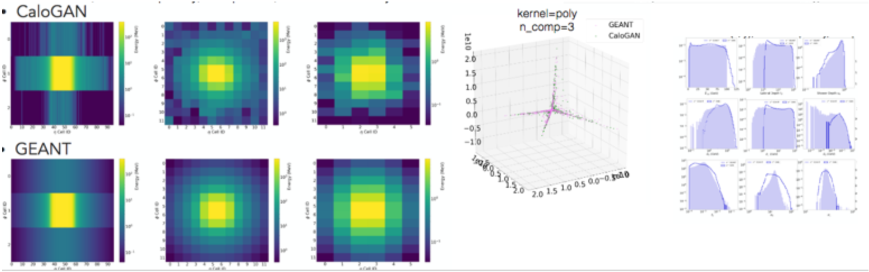
[34] https://research.nvidia.com/publication/2017-10_Progressive-Growing-of

[35] 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 [36]



---

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

Additional Topics:

- Hyperparameter Optimization: Grid search , Random search and Bayesian Optimization ...
- New architectures: Capsule Networks , Graph Neural Networks , Brain reverse engineering ...
- New computing frameworks: GPU , TPU , FPGA (Amazon EC2)
- What's the NN learning ? Debuging a CNN ?
    - Which pixels of a picture most contributed to a prediction?
    - Which pixels were in contradiction to prediction?
    - "Sensitivity analysis and heatmaps (based on "input gradient" df/dxi).

THE END