Machine Learning Techniques applied to High Energy Physics

Sebastian Pina Otey

Grupo AIA/IFAE

30 November 2017

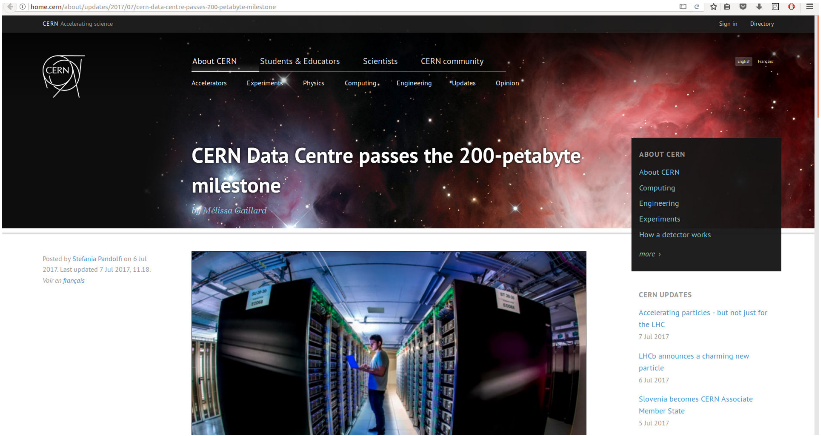
GRUPO AIA

Institut de Física d'Altes Energies

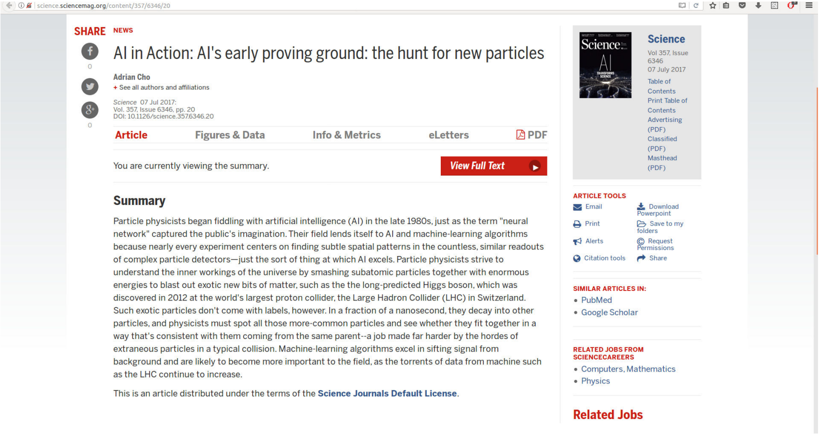
Outline

Contents

Motivation


home.cern/about/updates/2017/07/cern-data-centre-passes-200-petabyte-milestone
CERN Accelerating science
Sign in
Directory
CERN
About CERN
Students & Educators
Scientists
CERN community
English
Français
Accelerators
Experiments
Physics
Computing
Engineering
Updates
Opinion
CERN Data Centre passes the 200-petabyte milestone
by Mélissa Gaillard
ABOUT CERN
About CERN
Computing
Engineering
Experiments
How a detector works
more ›
Posted by Stefania Pandolfi on 6 Jul 2017. Last updated 7 Jul 2017, 11.18.
Voir en français
CERN UPDATES
Accelerating particles - but not just for the LHC
7 Jul 2017
LHCb announces a charming new particle
6 Jul 2017
Slovenia becomes CERN Associate Member State
5 Jul 2017

Motivation



science.sciencemag.org/content/357/6346/20

SHARE

0

0

NEWS

AI in Action: AI's early proving ground: the hunt for new particles

Adrian Cho

+ See all authors and affiliations

Science 07 Jul 2017:
Vol. 357, Issue 6346, pp. 20
DOI: 10.1126/science.357.6346.20

Article | Figures & Data | Info & Metrics | eLetters | PDF

You are currently viewing the summary.

View Full Text

Summary

Particle physicists began fiddling with artificial intelligence (AI) in the late 1980s, just as the term "neural network" captured the public's imagination. Their field lends itself to AI and machine-learning algorithms because nearly every experiment centers on finding subtle spatial patterns in the countless, similar readouts of complex particle detectors—just the sort of thing at which AI excels. Particle physicists strive to understand the inner workings of the universe by smashing subatomic particles together with enormous energies to blast out exotic new bits of matter, such as the the long-predicted Higgs boson, which was discovered in 2012 at the world's largest proton collider, the Large Hadron Collider (LHC) in Switzerland. Such exotic particles don't come with labels, however. In a fraction of a nanosecond, they decay into other particles, and physicists must spot all those more-common particles and see whether they fit together in a way that's consistent with them coming from the same parent--a job made far harder by the hordes of extraneous particles in a typical collision. Machine-learning algorithms excel in sifting signal from background and are likely to become more important to the field, as the torrents of data from machine such as the LHC continue to increase.

This is an article distributed under the terms of the **Science Journals Default License**.

Science
Vol 357, Issue 6346
07 July 2017
Table of Contents
Print Table of Contents
Advertising (PDF)
Classified (PDF)
Masthead (PDF)

ARTICLE TOOLS
Email
Download Powerpoint
Print
Save to my folders
Alerts
Request Permissions
Citation tools
Share

SIMILAR ARTICLES IN:
- PubMed
- Google Scholar

RELATED JOBS FROM SCIENCECAREERS
- Computers, Mathematics
- Physics

Related Jobs

Motivation

- Many data points, with a large number of variables available to make complex models are available, which escalate too fast for classical statistics.
- Since its beginning, machine learning has been around high energy physics (HEP), proving to be a very useful tool for classification of daata.

Motivation

- Many data points, with a large number of variables available to make complex models are available, which escalate too fast for classical statistics.
- Since its beginning, machine learning has been around high energy physics (HEP), proving to be a very useful tool for classification of daata.
- The computational power is now here to obtain our own algorithms through a personal computer, opening new opportunities for experimenting and expanding these algorithms for a specific interest.

What is machine learning?

Machine learning is the science of getting computers to act without being explicitly programmed.

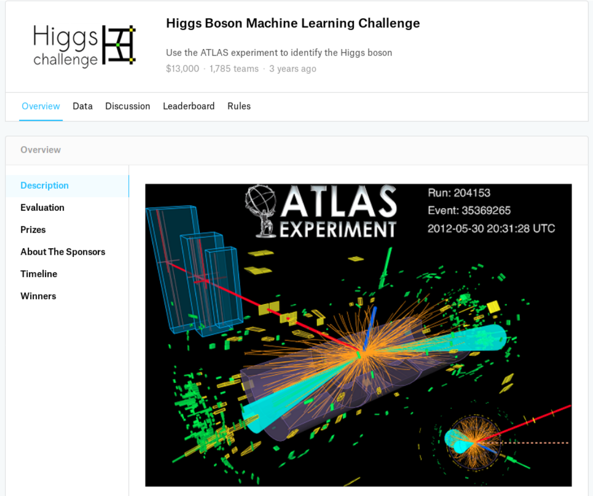
What is machine learning?

Machine learning is the science of getting computers to act without being explicitly programmed.

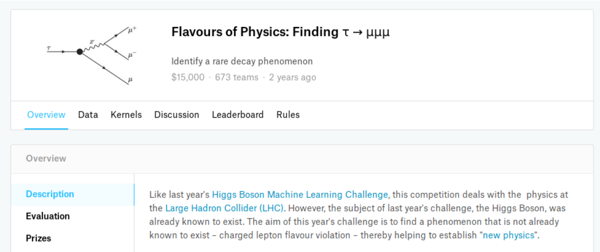
Machine learning algorithms can be classified into three groups:

- Supervised learning: The algorithm is presented with example inputs and their desired outputs, with the goal set to learn a general rule that maps inputs to outputs.
- Unsupervised learning: No labels are given to the learning algorithm, leaving it on its own to find structure in its input.
- Reinforcement learning: Learns how to take actions in an environment so as to maximize some notion of cumulative reward.

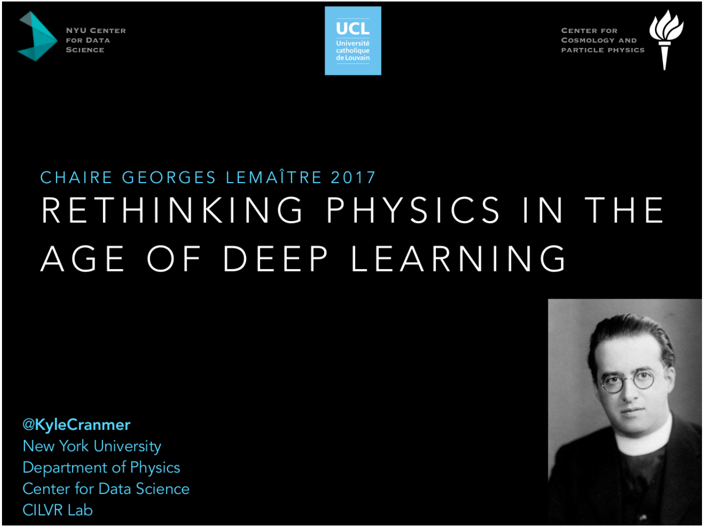
Machine Learning in HEP



Machine Learning in HEP

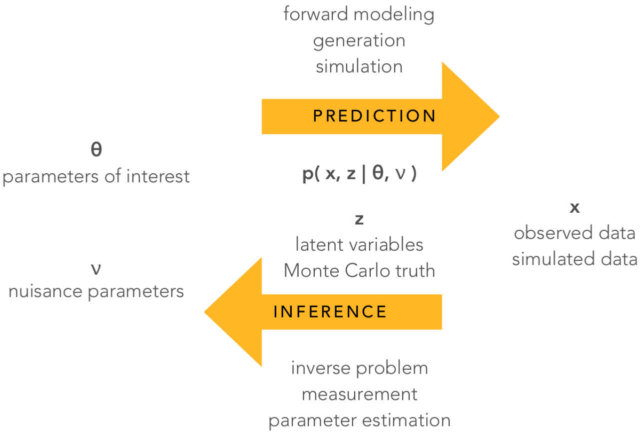


Machine Learning in HEP



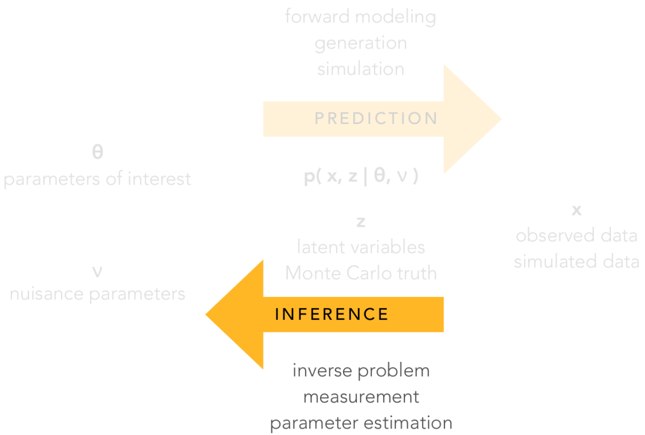
Machine Learning in HEP

THE PLAYERS



Machine Learning in HEP

THE PLAYERS



Machine Learning in HEP

Objetive: Perform Likelihood-Free Inference

Contents

Supervised learning

- Is a branch of machine learning, training a **model** Θ for making a **prediction** y given an observation $x \in \mathbb{R}^d$, $\Theta(x) = \hat{y}$.

Supervised learning

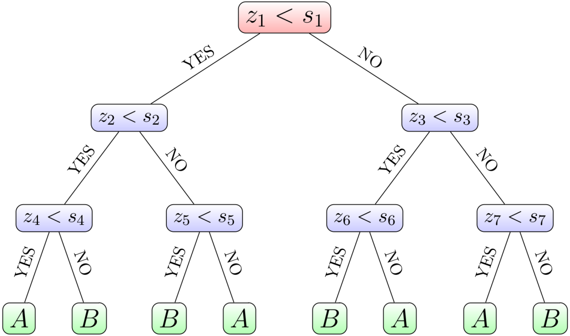
- Is a branch of machine learning, training a **model** Θ for making a **prediction** y given an observation $x \in \mathbb{R}^d$, $\Theta(x) = \hat{y}$.
- For this purpose, a training set of data points (x_i, y_i), $i = 1, \dots, N$ is given, hence the model learns from examples.
- The observations x are called **features**, while the prediction variable y is called the **target**. The target is usually one dimensional.
- If y is discrete, the problem is a **classification**, with a binary classification being the most common case. If y is continuous, the problem is a **regression**.

Supervised learning

- Is a branch of machine learning, training a **model** Θ for making a **prediction** y given an observation $x \in \mathbb{R}^d$, $\Theta(x) = \hat{y}$.
- The model has a set of **parameters** θ, which are adjusted through the algorithm to make the prediction $\hat{y}$ as close as possible to y using the training set, $\Theta(x; \theta) = \hat{y}$.
- The simplest example is the linear regression.

Trees

Consists in a sequence of conditions which, typically, cut our feature space of observed events into disjoint partitions of it.



Example of binary classification tree. z_i is a particular feature, $z_i \in \{x_1, \ldots, x_d\}$.

Constructing Trees

The **probability** of an observation landing on each of the nodes is given by $P(t) = N_t/N,\ t \in \{0, L, R\}$, where N_0 is the number of samples in the parent node, $N_{L/R}$ is the one of left/right subsets after splitting.

In a binary classification tree, we say that a node is pure when it only contains data from a single class. When growing a tree, we try to obtain pure nodes. However, if the probability of that node $P(t)$ is too low, we are likely overfitting the data.

Constructing Trees

The **probability** of an observation landing on each of the nodes is given by $P(t) = N_t/N,\ t \in \{0, L, R\}$, where N_0 is the number of samples in the parent node, $N_{L/R}$ is the one of left/right subsets after splitting.

In a binary classification tree, we say that a node is pure when it only contains data from a single class. When growing a tree, we try to obtain pure nodes. However, if the probability of that node $P(t)$ is too low, we are likely overfitting the data.

Probability is not a good measure for deciding splits directly! But it still will be helpful.

Constructing Trees

Node impurity:

$$i(t) = \phi(P(A|t), P(B|t)),$$

where $\phi(p, q)$ is bounded to $0 \leq \phi(p, q) \leq 1/2$. It has to be symmetric, and satisfy that $\phi(1/2, 1/2) = 1/2$ (maximum impurity when both classes are equally likely) while $\phi(1, 0) = \phi(0, 1) = 0$ (minimum when the node is pure).

Constructing Trees

Node impurity:

$$i(t) = \phi(P(A|t), P(B|t)),$$

where $\phi(p, q)$ is bounded to $0 \leq \phi(p, q) \leq 1/2$. It has to be symmetric, and satisfy that $\phi(1/2, 1/2) = 1/2$ (maximum impurity when both classes are equally likely) while $\phi(1, 0) = \phi(0, 1) = 0$ (minimum when the node is pure).
An example of function ϕ is the **binary cross-entropy**

$$\phi(p, q) = -\frac{p \log_2 p + q \log_2 q}{2}.$$

Constructing Trees

Scaled node impurity:

$$I(t) = P(t)i(t).$$

Maximizing the **impurity gain**,

$$\Delta I = I(t_0) - I(t_R) - I(t_L),$$

will be the criteria of splitting.

Constructing Trees

Stopping criteria:

- The node is pure.
- Maximum depth, which, when reached, stops the tree from splitting.
- Maximum number of leaves acquired.
- Maximum possible impurity gain is below some threshold.
- Size of node is less than allowed.

Constructing Trees

Stopping criteria:

- The node is pure.
- Maximum depth, which, when reached, stops the tree from splitting.
- Maximum number of leaves acquired.
- Maximum possible impurity gain is below some threshold.
- Size of node is less than allowed.

Additionally, one can prune the tree, which consists in getting rid of the branches of a node and substitute it for its parents node as a leaf. It is a form of regularization in order to penalize the tree for being too complex.

General principle

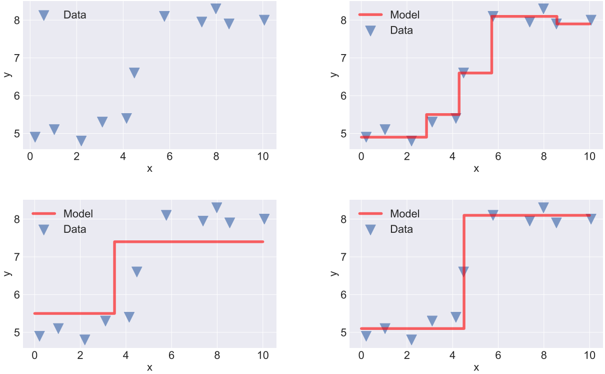
$$\mathrm{Obj}(\Theta, \theta) = L(\theta) + \Omega(\Theta),$$

where L is the **training loss function** and Ω is the **regularization term**.

General principle

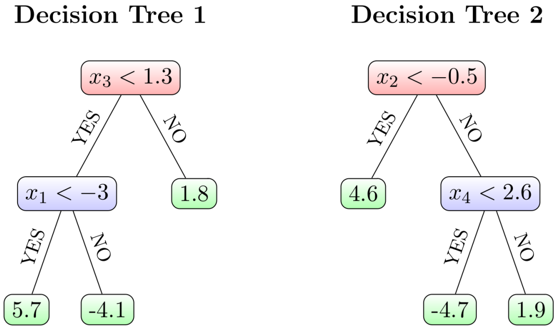
$$\mathrm{Obj}(\Theta, \theta) = L(\theta) + \Omega(\Theta),$$

where L is the **training loss function** and Ω is the **regularization term**.

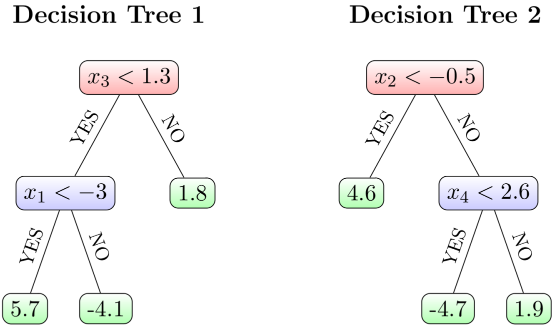


Ensemble of trees

Ensemble of trees



Ensemble of trees



Example: Consider the above decision trees and a new data point $x = \{1.5, -0.7, 2.5, 4.1\}$. The prediction for the first tree is 1.8, while for the second, it is 4.6. The total prediction of the ensemble is $1.8 + 4.6 = 6.4$.

Ensemble of trees

An ensemble model is written as

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i),\ f_k \in \mathcal{F},$$

where t is the number of trees, x_i the $i-$th observation and $\mathcal{F}$ is the functional space of all possible decision trees for regression.
The objective function for t trees becomes

$$\mathrm{Obj}(\theta)^{(t)} = \sum_{i}^{N} l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_k),$$

where l is the loss function for an individual sample ($L = \sum_i l$).

Gradient Boosting Trees

Gradient Boosting Trees

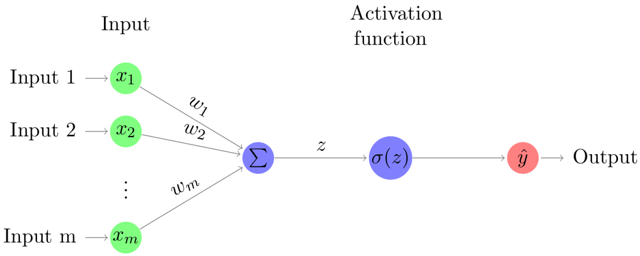
- **Boosting**: To optimize the objective function we cannot apply directly a gradient method as in standard optimization. Instead we will consider $t-1$ trees learned, and see how we can build the $t-$th tree to optimize the objective function from there, i.e., we see how to add one tree at a time optimally.

Gradient Boosting Trees

- **Boosting**: To optimize the objective function we cannot apply directly a gradient method as in standard optimization. Instead we will consider $t-1$ trees learned, and see how we can build the $t-$th tree to optimize the objective function from there, i.e., we see how to add one tree at a time optimally.
- **Gradient**: To find the optimal weights of the tree's leaves, the gradient of the objective function is used.

Contents

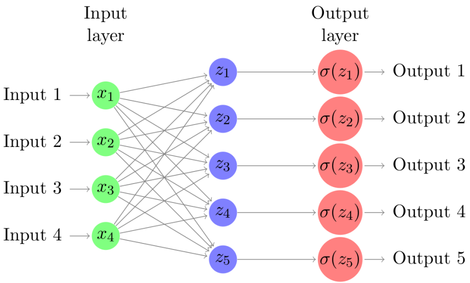
McCulloch and Pitts neuronal model



$$\hat{y} = \sigma(z), \quad z = \sum_{i=1}^{m} x_i w_i,$$

where σ is the **activation function** and w_i are the weights of the network.

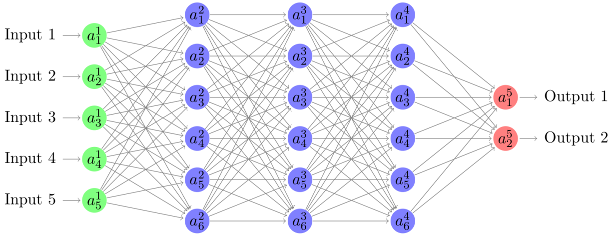
Perceptron



$$z_j = \sum_{i=0}^{m} w_{ji} x_i = w_{j0} + \sum_{i=1}^{m} w_{ji} x_i = b_j + \sum_{i=1}^{m} w_{ji} x_i,$$

where b_j is the **bias term**.

Deep Neural Network



$$a_j^l = \sigma(z_j^l) = \sigma\left(\sum_i w_{ji}^l a_i^{l-1} + b_j^l\right),$$

or in matrix notation,

$$a^l = \sigma(z^l) = \sigma(w^l a^{l-1} + b^l).$$

Backpropagation

Consider the cost (loss) function C. Instead of

$$\frac{\partial C}{\partial \theta_i} \simeq \frac{C(\theta_i + \varepsilon) - C(\theta_i)}{\varepsilon},$$

we use the **backpropagation** method,

$$\delta^L = \nabla_{a^L} C \odot \sigma'(z^L), \tag{1}$$

$$\delta^l = \left(\left(w^{l+1} \right)^T \delta^{l+1} \right) \odot \sigma'(z^l), \tag{2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j, \tag{3}$$

$$\frac{\partial C}{\partial w^l_{jk}} = \delta^l_j a^{l-1}_k. \tag{4}$$

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an example of how to update the weights using backpropagation. M random samples of the training data are selected and used:

$$w^l \to w^l - \frac{\eta}{M} \sum_{m=1}^{M} \delta^{m,l} \left(a^{m,l-1}\right)^T,$$

$$b^l \to b^l - \frac{\eta}{M} \sum_{m=1}^{M} \delta^{m,l},$$

where η is the learning parameter. There are other optimizers, being Adam the most popular one.

Generative Adversarial Networks

GANs are a kind of **generative models**.

Generative Adversarial Networks

GANs are a kind of **generative models**.
We will consider them to be any model that, from a training set sampled from a distribution p_{data}, learns to produce an estimation of such distribution in any form of it. The resulting estimation will be denoted as p_{model}. This estimation can be an explicit form of the distribution. It might also be a mechanism only able to sample new data from it. It might be both.

Generative Adversarial Networks

GANs are a kind of **generative models**.

- Training and sampling from generative models shows how we can represent and manipulate high-dimensional probability distributions, such as pictures and videos.
- They can be integrated in reinforcement learning, creating new realistic final goals for the algorithm.
- Train with missing data and make prediction on missing ones.
- Multi-modal outputs are enabled for machine learning, e.g., for predicting the next frame in a video.
- Generate realistic samples for some distribution. This is required for many tasks, including single image super-resolution, interactive art creation, image-to-image translation, etc.

Generative Adversarial Networks

GANs have recently been also applied to physical problem, such as the reconstruction of three-dimensional porous media , simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters and creating Virtual Universes. In all three cases, the need of a large amount of complex simulations drives to resort to GANs for a faster emulation algorithm, to obtain new data.

GAN framework

The idea behind GANs is a game (in a mathematical sense) between two players: a *generator* (a function G, representing the first NN) and a *discriminator* (a function D, representing the second NN). The first one learns to produce samples imitating the distribution of the training data set. The later one learns to distinguish real data, from the training set, from fake one, produced by the generator. Hence the game consists in the discriminator classifying data into real or fake one, while the generator has to try to trick the discriminator by generating more realistic data. It is an adversarial situation.

$$Z \xrightarrow{G} X \xrightarrow{D} \{0, 1\}.$$

GAN framework

The networks have each a cost function C that depend on the parameters of the networks. The discriminator wishes to minimize $C^{(D)}\left(\theta^{(D)},\theta^{(G)}\right)$ while only controlling $\theta^{(D)}$. On the other hand, the generator wishes to minimize its own cost function $C^{(G)}\left(\theta^{(D)},\theta^{(G)}\right)$ while only controlling $\theta^{(G)}$.

GAN framework

The networks have each a cost function C that depend on the parameters of the networks. The discriminator wishes to minimize $C^{(D)}\left(\theta^{(D)}, \theta^{(G)}\right)$ while only controlling $\theta^{(D)}$. On the other hand, the generator wishes to minimize its own cost function $C^{(G)}\left(\theta^{(D)}, \theta^{(G)}\right)$ while only controlling $\theta^{(G)}$.
The solution to a game is a Nash equilibrium, which in our case is a tuple $\left(\theta^{(D)}, \theta^{(G)}\right)$ which is a local minimum of $C^{(D)}$ with respect to $\theta^{(D)}$ and a local minimum of $C^{(G)}$ with respect to $\theta^{(G)}$.

GAN framework

The networks have each a cost function C that depend on the parameters of the networks. The discriminator wishes to minimize $C^{(D)}\left(\theta^{(D)}, \theta^{(G)}\right)$ while only controlling $\theta^{(D)}$. On the other hand, the generator wishes to minimize its own cost function $C^{(G)}\left(\theta^{(D)}, \theta^{(G)}\right)$ while only controlling $\theta^{(G)}$.
The solution to a game is a Nash equilibrium, which in our case is a tuple $\left(\theta^{(D)}, \theta^{(G)}\right)$ which is a local minimum of $C^{(D)}$ with respect to $\theta^{(D)}$ and a local minimum of $C^{(G)}$ with respect to $\theta^{(G)}$.
If we rewrite it as a zero-sum game, the optimal parameters for the generator are

$$\theta^{(G)*} = \arg\min_{\theta^{(G)}} \max_{\theta^{(D)}} V\left(\theta^{(D)}, \theta^{(G)}\right).$$

GAN applied to HEP

- The data was provided by F. Sanchez, and contains the energy E and momentum p_x, p_y and p_z produced in a neutrino event in the T2K experiment.

GAN applied to HEP

- The data was provided by F. Sanchez, and contains the energy E and momentum p_x, p_y and p_z produced in a neutrino event in the T2K experiment.
- From 10 million data points, we select randomly 10 thousand, a 0.1% of the data, and try to learn its distribution via a GAN.
- This will be later contrasted with a different sample of 500 thousand points.
- The data was scaled into the interval [-1,1].

GAN applied to HEP

Generator:

- The initial input dimension of z is 20 to ensure capturing the whole p_{data} space. Each dimension of z follows a standard normal distribution.
- The last activation function of the generator network was the hyperbolic tangent function, mapping the output into the interval [-1,1]. The last layer has dimension 4, as we want the output to be in the same space as the training data.
- We have 4 hidden layers in the network of dimension the same as the input, 20. The layers are connected in a dense way, meaning that all neurons of one layer are connected to all neurons of the next layer.
- The activation of the hidden layers is the leaky ReLU function.

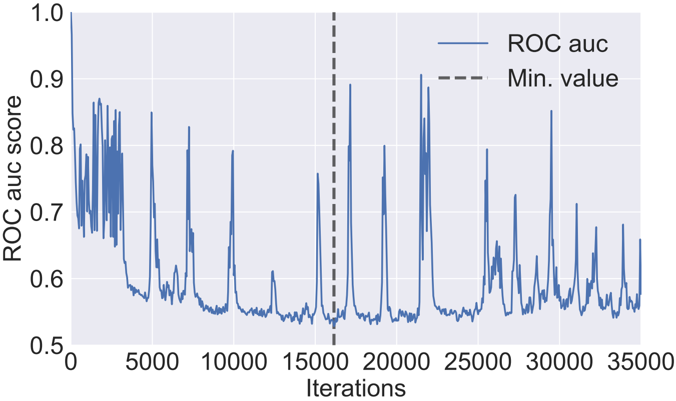
GAN applied to HEP

Discriminator:

- The input dimension is 4, since we have 4 variables.
- The output has dimension 1, and indicates the probability of being a sample drawn from the p_{data} distribution. The last activation function is hence a sigmoid to map it to the interval [0,1].
- The network has 3 hidden layers, of dimension $4 \cdot 6$. They are all also connected in a dense way.
- As for the generator, the activations for the hidden layers is the leaky ReLU function.

GAN applied to HEP

ROC auc score for a classifier trying to distinguish real from simulated data. It was computed every 50 iterations.



GAN applied to HEP

- The ROC auc score for p_{model} vs p_{data} is 0.5284.
- The ROC auc score for p_{model} vs p_{total} is 0.6014.
- The ROC auc score for p_{data} vs p_{total} is 0.5034.

Why? Let us take a look at the individual densities of the variables.

GAN applied to HEP

Individual densities: