Statistics for Particle Physicists
LPC 2021, Fermilab

Lecture 14: Introduction to Machine Learning – Part 1

Harrison B. Prosper

Department of Physics, Florida State University

17 November, 2021

Outline

Outline

Machine learning is now a huge field, which is impossible to cover in three lectures. Instead of attempting the impossible, I'll focus on the key ideas and a few key models.

Here is the tentative plan:

- Today: Decision trees (DT), deep feed forward neural networks (DFFNN), empirical risk minimization.
- After the US Thanksgiving Holiday:
  - Monday: Convolutional neural networks (CNN), auto-encoders (AE) and reinforcement learning (RL).
  - Wednesday: Graph neural networks (GNN), sequence-to-sequence models (LSTMs), and quantile regression.

- 4th century BC Laws of logic (Aristotle)
- 15th century 1456 – Gutenberg Bible

- 18th century 1763 – Bayes' theorem (Thomas Bayes)
- 19th century 1832 – first programmable calculator (Charles Babbage)
- 20th century
  - 1943 – Invention of neural networks (McCulloch and Pitts)
  - 1950 – Turing Test (Alan Turing)
  - 1956 – Artificial Intelligence (AI) coined (John McCathy)
  - 1974 – Back-propagation algorithm (Paul Werbos)
  - 1997 – Defeat of chess world champion Gary Kasparov by Deep Blue, an IBM supercomputer

- 21st century
  - 2006 – Successful training of deep neural networks (Hinton, Osindero, Teh)
  - 2010 – Demonstration that lots of data and lots of computing power are sufficient for the successful training of deep neural networks, and that very deep models are the key (Ciresan, Meier, Gambardella, Schmidhuber)
  - 2017 – `AlphaGoZero` achieves superhuman `Go` player status entirely through self-play (`DeepMind`)
  - 2020 – `GPT-3` API released (but not the code) – a 175 billion parameter language model (`OpenAI`)

Outline

A decision tree (DT) is a set of if then else statements that form a tree-like structure that can be used to partition data, for example, signal (S) and background (B) events, into two classes.

Algorithm: recursively partition the space of observations x into regions of diminishing impurity.

A common measure of impurity is the Gini Index[1]:

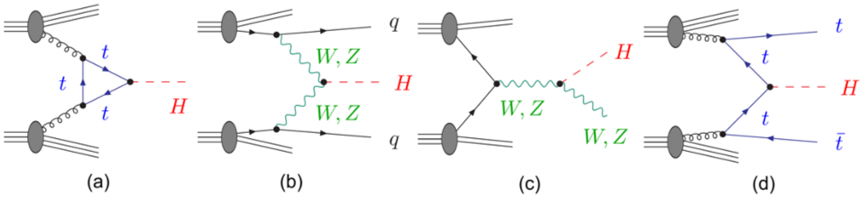
$$\begin{aligned} \text{Gini Index } &= p\,(1-p), \text{ where } p \text{ is the purity} \\ p &= S/(S+B), \end{aligned}$$

where S and B, in this case, would be the signal and background counts, respectively, in a given data subset. The Gini index is defined so that

$$\begin{aligned} p &= 0 \text{ or } 1 \quad \text{implies maximum purity and} \\ p &= 0.5 \quad \text{implies maximum impurity.} \end{aligned}$$

[1] Corrado Gini, 1884 –1965

Example ($pp \to H \to ZZ \to 4\ell$ VBF vs. ggF)



http://www.scholarpedia.org/article/The_Higgs_Boson_discovery

(a) Gluon fusion (ggF)

(b) Vector boson fusion (VBF)

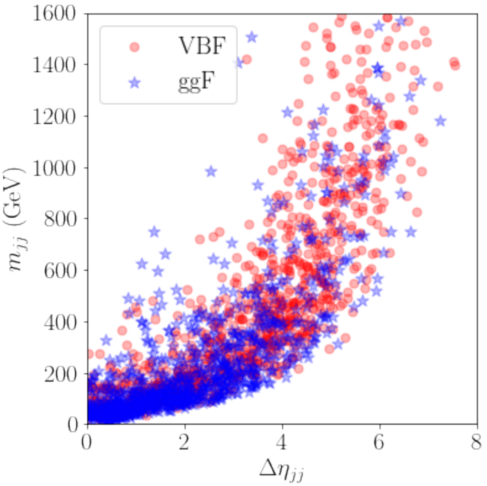
(c) Associated production (VH)

(d) Top anti-top fusion (ttH)

Let's apply decision trees to separate VBF events from ggF events.

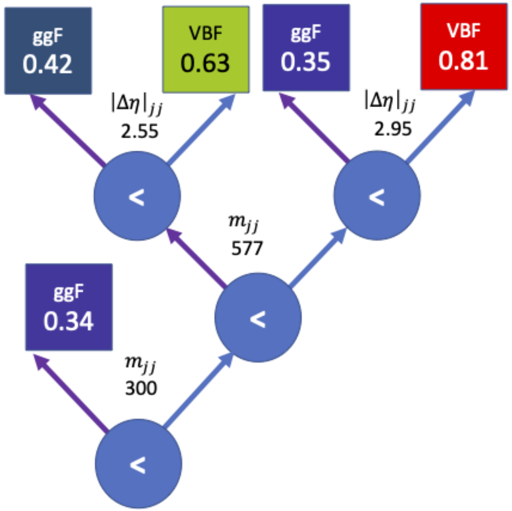
Example ($pp \to H \to ZZ \to 4\ell$ VBF vs. ggF)

If we require 2 forward jets in an event, we can compute the 2-jet variables $|\Delta\eta|_{jj}$ and m_{jj}, the di-jet pseudo-rapidity difference and di-jet mass, respectively, whose distribution is shown in the plot on the right.



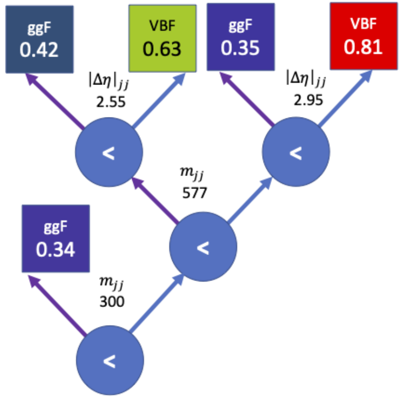
Example ($pp \to H \to ZZ \to 4\ell$ VBF vs. ggF)

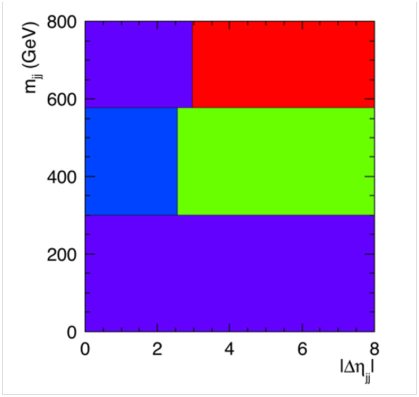
1. For each variable, find the partition that gives the greatest decrease in impurity.
2. Choose the best partition and split the data along that partition into two subsets.
3. Repeat 1 and 2 for each subset.



Example ($pp \to H \to ZZ \to 4\ell$ VBF vs. ggF)

A decision tree is also just a multi-dimensional histogram.





Unfortunately, decision trees are unstable.

A small change in the data can change a tree completely.

Also, since decision trees only provide a piecewise constant approximation to functions, they are not as effective as other machine learning models.

In 1997, AT&T researchers Freund and Schapire[2] published an algorithm called `AdaBoost` that produced highly effective classifiers by combining many decision trees.

`AdaBoost` was the first successful method to boost (i.e., enhance) the performance of poorly performing classifiers (called weak learners) by averaging their outputs:

$$f(x, \theta) = \sum_n a_n \, T(x, \theta_n),$$

where $T(x, \theta_n)$ are decision trees.

For our example, the key idea is to build trees sequentially using weighted events. The classifier $f(x, \theta)$ based on the first k trees determines which of the weighted events are correctly or incorrectly classified. The weights of misclassified events are increased relative to the weights of correctly classified events and a new tree is created.

[2] Y. Freund and R. E. Schapire, Journal of Computer and Sys. Sci. 55 (1), 119 (1997)

Here are details of the `AdaBoost` algorithm. The indicator function $\mathbb{I}[\,X\,] = 1$ if X is true and zero if false. Note also that $y_n\, f(x_n, \theta) > 0$ if x_n is correctly classified.

Algorithm 1 `AdaBoost`

```
    initial training sample: $\mathbb{T} \leftarrow (x_1, y_1), \cdots (x_N, y_N)$
    normalize weights: $\sum_{n=1}^{N} w_{1,n} = 1$
    create weighted sample $\mathbb{T}_1 \leftarrow \mathbb{T}$ using $w_{1,n}$
    while $k \in [1, \cdots K]$ do
        create tree: $T_k(x, \theta)$ using $\mathbb{T}_k$
        $f(x, \theta) \leftarrow T_1(x, \theta)$ if $k = 1$ else $\sum_{j=1}^{k-1} a_j\, T_j(x, \theta)$
        compute error rate: $\epsilon_k \leftarrow \sum_{n=1}^{N} w_{k,n}\, \mathbb{I}[\, y_n\, f(x_n, \theta) \leq 0\,]$
        compute coefficient: $a_k \leftarrow \log[\,(1 - \epsilon_k)/\epsilon_k\,]$
        re-weight: $w_{k+1,n} \leftarrow w_{k,n}\, \exp(-a_k\, y_n\, f(x_n, \theta)\,/\,2)$
        normalize weights: $\sum_{n=1}^{N} w_{k+1,n} = 1$
    end while
    $f(x, \theta) \leftarrow \sum_{k=1}^{K} a_k\, T_k(x, \theta)$
```

Several methods have since been developed for averaging weak learners, the most popular of which (used mostly with decision trees) are:

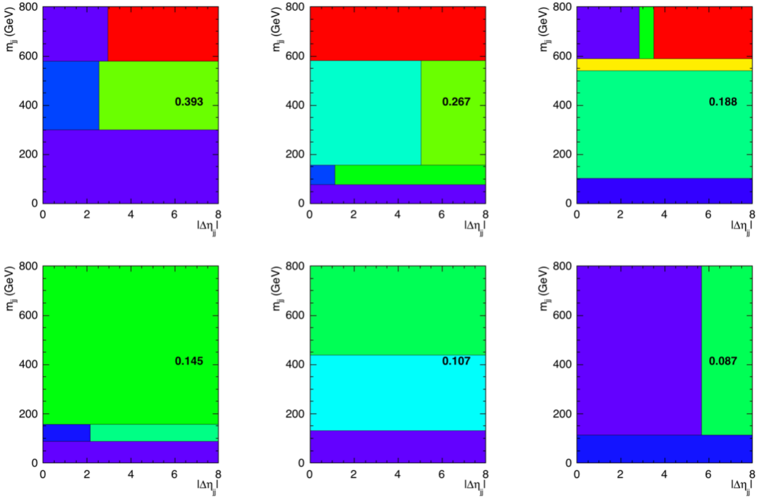
- Bagging: each tree is trained on a bootstrap[3] sample drawn from the training data.
- Random Forest: bagging with randomized trees.
- Boosting: each tree trained on a different re-weighting of the training data.

Let's apply `AdaBoost` to our VBF vs. ggF classification problem.

[3] A bootstrap sample is a sample of size N drawn, with replacement, from another of the same size. Duplicates can occur and are allowed.

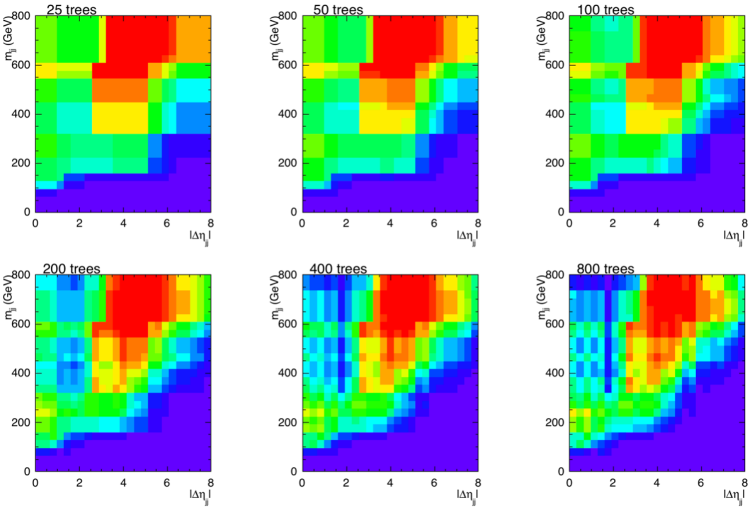
Example ($pp \to H \to ZZ \to 4\ell$ VBF vs. ggF)

First 6 decision trees.

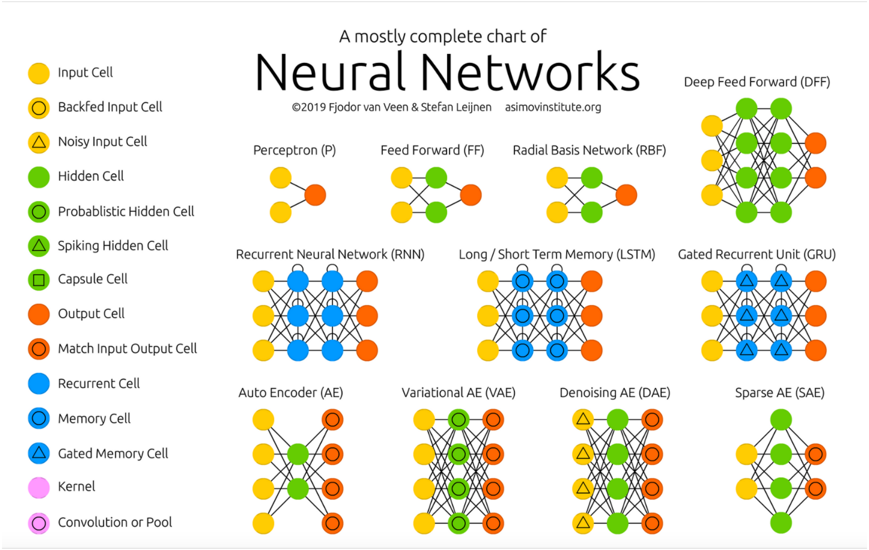


Example ($pp \to H \to ZZ \to 4\ell$ VBF vs. ggF)

Average over multiple trees.



Outline



https://www.asimovinstitute.org/neural-network-zoo

Neural Networks

The basic elements of a feed-forward neural network are node functions

$$h(b_j + \sum_{i=1}^{I} w_{ji} x_i),$$

where b_j and w_{ji} are parameters with $j = 1, \cdots, H$ and $x_i, i = 1, \cdots, I$, are inputs. Define the matrices

$$\mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_H \end{bmatrix}, \mathbf{w} = \begin{bmatrix} w_{11} & \cdots & w_{1I} \\ \vdots & \vdots & \vdots \\ w_{H1} & \cdots & w_{HI} \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_I \end{bmatrix},$$

then we can write the vector of node functions as

$$\mathbf{h}(\mathbf{b} + \mathbf{w}\mathbf{x}),$$

where it is understood that the column matrix $\mathbf{h}$ is defined by vectorizing the function $h(*)$, i.e., applying it element-wise to the column matrix $\mathbf{b} + \mathbf{w}\mathbf{x}$.

A shallow feed-forward neural network (NN), (I, H, K), with I inputs, H hidden nodes, and K outputs is the non-linear function

$$\mathbf{n}(\mathbf{x}) = \mathbf{g}(\mathbf{b}_1 + \mathbf{w}_1\, \mathbf{h}_1(\mathbf{b}_0 + \mathbf{w}_0 \mathbf{x})),$$

where $\mathbf{g}$ is the vectorized function $g(*)$, $\mathbf{d}_1$ is a $K \times H$ matrix and $\mathbf{w}_0$ is a $H \times I$ matrix. The non-linearity is introduced via functions $h(*)$ such as

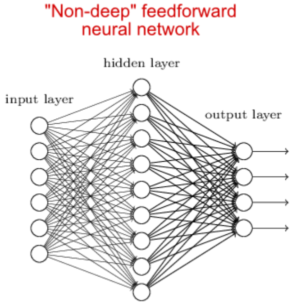
$$\begin{aligned} h_1(z) &= \tanh(z), \\ h_1(z) &= \max(0, z) \quad \text{the rectified linear unit (reLU) function,} \end{aligned}$$

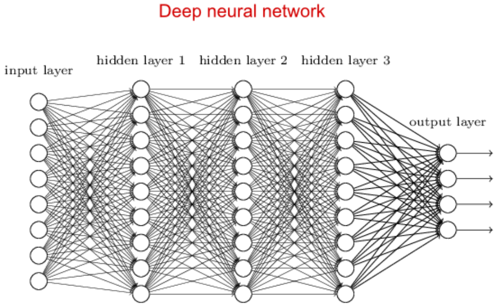
and, depending upon the application, $g(*)$ is either the softmax (if there are multiple outputs that must sum to one), the identity, or the logistic function,

$$g(z) = 1 \,/\, (1 + e^{-z}),$$

when there is only one output. The parameters $\theta = (\mathbf{b}_1, \mathbf{w}_1, \mathbf{b}_0, \mathbf{w}_0)$ are often referred to as weights.

A deep (feed-forward) neural network (DNN) contains two or more hidden layers that are compounded recursively: the matrix **h** of node function values becomes the inputs for the next layer. In general, each layer has its own pair of matrices **b** and **w** and the number of nodes per layer can vary.





http://neuralnetworksanddeeplearning.com/chap5.html

Here is the mathematical form of the above deep neural network:

$$\mathbf{n}(\mathbf{x}) = \mathbf{g}(\mathbf{b}_3 + \mathbf{w}_3\,\mathbf{h}_3(\mathbf{b}_2 + \mathbf{w}_2\,\mathbf{h}_2(\mathbf{b}_1 + \mathbf{w}_1\,\mathbf{h}_1(\mathbf{b}_0 + \mathbf{w}_0\mathbf{x}))))$$

Outline

As noted in earlier lectures, machine learning models $f(x, \theta)$ are typically trained, that is, fitted to data, by minimizing the empirical risk function, that is, the average loss,

$$R_N = \frac{1}{N}\sum_{i=1}^{N} L(y_i, f_i) + C(\theta), \text{ where } f_i \equiv f(x_i, \theta),$$

and $L(y, f)$ is a loss function and $C(\theta)$ is a constraint on the parameters θ.

The empirical risk approximates the risk functional

$$R[f] = \int \int L(y, f)\, p(y, x)\, dy\, dx, \tag{1}$$

where $p(y, x)\, dy\, dx$ is the joint probability of the training data[4].

[4]We have assumed that the effect of the constraint vanishes in the limit $N \to \infty$.

$$R[f] = \int \int L(y, f)\, p(y, x)\, dy\, dx,$$

The functional $R[f]$ is of the form

$$\begin{aligned} R[f] &= \int F(x, f)\, dx, \\ \text{where } F(x, f) &\equiv \int L(y, f)\, p(y, x)\, dy, \\ &= p(x) \int L(y, f)\, p(y \,|\, x)\, dy. \end{aligned} \tag{2}$$

$R[f]$ is analogous to an action and $F(x, f)$ to a Lagrangian. So we can apply the theory developed for the principle of least action.

The calculus of variations shows that if $f(x, \theta)$ is sufficiently flexible, then there exists an extremal function $f(x, \theta^*)$ that minimizes $R[f]$, that is, which solves

$$\frac{\delta R}{\delta f} = \int \frac{\partial L}{\partial f} \, p(y \,|\, x) \, dy = 0, \tag{3}$$

assuming $p(x) > 0$.

We therefore conclude that

> *given sufficient training data T, a sufficiently flexible function f, and an average loss function R_N that is amenable to minimization, $f(x, \theta^*)$ approximates a mathematical quantity determined solely by the form of the loss function $L(y, f)$ and the conditional probability $p(y \,|\, x) \, dy$ of the training data.*

In particular, this result does *not* depend on the details of the machine learning model. Indeed, the model does not have to be a neural network.

Example (Binary cross-entropy loss: $L(y, f) = -y \log f - (1 - y) \log(1 - f)$)

$$\int \frac{\partial L}{\partial f} \, p(y \,|\, x) \, dy = 0,$$

$$\int \left[-\frac{y}{f} + \frac{(1 - y)}{(1 - f)} \right] p(y \,|\, x) \, dy = 0,$$

$$\text{therefore, } f(x, \theta^*) = \int y \, p(y \,|\, x) \, dy.$$

For classification, y has two discrete values, namely, 0 and 1. We, therefore, conclude that $f(x, \theta^*) = p(y = 1 \,|\, x)$.

Whether the model is a lowly shallow neural network or a state-of-the-art deep neural network, provided both are sufficiently flexible, sufficient training data are used, and the minimizer works well, both models will approximate the class probability $p(y = 1 \,|\, x)$.

Outline

- Decision trees are unstable. However, it is possible to average over the outputs of many decision trees and thereby form highly effective models $f(x, \theta)$ for classification. It is also possible to use such models for regression.

- Deep neural networks, of which there are many different classes, are highly non-linear functions, typically constructed recursively. Today, we considered the simplest class of model: deep feed forward neural networks.

- Given a sufficiently flexible model $f(x, \theta)$ and a good minimizer, the two quantities that determine what the model approximates is the loss function and the probability distribution of the data $p(y, x)\,dx\,dy$.