University of Oxford

Lincoln College

Doctorate of Philosophy

Particle Physics

Advanced Machine Learning Applications for the Higgs and Heavy Flavour Quarks at ATLAS

Candidate

Maxence DRAGUET

Supervisor

Daniela BORTOLETTO

2020-2024

CONTENTS

List of Abbreviations

AI Artificial Intelligence
ATLAS A Toroidal LHC ApparatuS
BDT Boosted Decision Trees
CNN Convolutional Neural Network
DL Deep Learning
DT Decision Trees
FPGA Field-Programmable Gate Array
GAN Generative Adversarial Network
HEP High Energy Physics
LHC Large Hadron Collider
MC Monte Carlo
ML Machine Learning
MVA Multivariate Analysis
PU Pile-up
RL Reinforcement Learning
VAE Variational Auto-Encoder

Abstract

This confirmation of status report summarises some of the work I have carried out as part of the $VH(H \to c\bar{c})$ combined analysis of Run 2, with data from 2016 to 2018, as well as the development of the new jet flavour tagger called *DL1d*, designed to identify *b*- and *c*-jets with high efficiency.

CHAPTER 1

MACHINE LEARNING & DEEP LEARNING

1.1 Introduction

This chapter is entirely dedicated to a review of relevant Machine Learning (ML) and Deep Learning (DL) methods in the context of High Energy Physics (HEP). As for every other fields of science and technology, the recent advancements in the domain of Artificial Intelligence (AI) have introduced a shift of paradigm in particle physics. Before continuing with the review, some definitions of these different terms is suggested to decrypt their meaning.

1.1.1 Definitions

Artificial Intelligence

Artificial Intelligence encapsulates any piece of software, any *programs*, that mimics an aspect of human intelligence. A non-exhaustive list of these aspects include:

- *Reasoning*, the ability to conduct logical thoughts down to conclusions,
- *Inferring*, the ability to connect logical statements to induce new statements,
- *Creativity*, the ability to generate new information.
- *Acting*, the ability to change the environment, particularly studied in the field of robotics.

AI is a large field of study that contains among many the area of robotics, natural language processing, computer vision, generative models, reinforcement learning. Artificial intelligence is broadly separated into three levels of performance, of which only the first one is currently accessible:

1. Narrow Intelligence: representing artificial intelligence capabilities on a specific problem, for which the underlying software is uniquely trained. This field includes *reactive AI*, where a model would be trained to output an optimal move based on current conditions only (e.g., the IBM chess player Deep Blue), and *limited memory AI*, where a model is able to draw knowledge from past data to make decision (e.g., any ML model, such as the OpenAI chatGPT chat-box).

2. Artificial General Intelligence: representing an artificial intelligence capable of matching human problem-solving skills. In particular, this hypothetical setting would let a machine learn new tasks on its own and extrapolate from its existing knowledge.

3. Artificial Super Intelligence: describes a hypothetical type of intelligence able to exceed human abilities and exhibit independent control of thoughts.

The inception of reactive AI can be found in the research into games in the 50s and 60s. This approach saw the rise of algorithms capable of searching for the optimal move in large search spaces of possible actions using heuristics, human-passed knowledge of useful features of the specific environment of the game (e.g., the point system for chess pieces, with a queen being worth more than a simple pion). In this reactive approach, neither the rules of the game nor the decision process are learnt. The former is forced into the search logic and the latter is the outcome of the search process.

Machine Learning

Machine Learning underpins the field of narrow AI with limited memory capabilities. It represents a shift of paradigm in AI: moving away from human-declared logic-based rules written in a specific syntax

$$\textit{If } x \textit{ happens, do } y,$$

it automates the representation and update with mathematical models of both the dataspace ($\mathscr{D}$) and the learning process:

$$\forall x \in \mathscr{D}, \text{ do } f(x) = \hat{y}; \text{ update } f(x) \text{ given } (x,y).$$

In this respect, both the internal representation of the rules and the decision-making is underpinned by the trained mathematical model f.
Two general steps are combined into a model defined by learnable parameters:

1. Learning: the parameters of the model are updated based on a specified training or fit procedure, depending on whether the training will be progressively exposed to the data points of a training dataset or directly exposed to entirety of the set. The objective is to align the output of the model with the expected behaviour: given the couple (x,y), let $f(x) = \hat{y} \to y$ under training convergence - this means the model f has to become an unbias estimate of the label y.

2. Inferring: a trained model has to give its output on a new data point: $f(a) = \hat{b}$.

The training process will depend on the type of model being deployed. These can be broadly separated into two fields:

- Classical machine learning: includes decision trees (XGBoost, Boosted Decision Trees (BDT) or Multivariate Analysis (MVA), random forest, ...), Support Vector Machine (SVM), logistic regression, kernel methods, ...

- Deep Learning (DL): these methods are based on a core module call the Artificial Neural Network. This module is stacked into layers of given width, meaning a given number of neurons, and several layers of such modules are then connected along depth.

Deep Learning

Deep Learning corresponds to a specific set of methods who have quickly grown in popularity around 2010, with widely advertised results on competitive benchmark tasks in pattern recognition, as exemplified by the super-human performance of the *DanNet* [1] model based on Convolutional Neural Network (CNN)s [2]. The basis of any DL method is the Artificial Neural Network, a logical unit

built to mimic the functioning of a human neuron. These neurons are then combined into layers of any numbers of neurons (the width of the layer) and the layers themselves are stacked into depth, with deeper layer receiving as input the output of earlier neurons. Different DL models are constructed by modifying the structure of the layers - in particular, the input, output, and activation function used - and the transfer the information between neurons, be that between layers, depth-wise, or between neurons, width-wise.
The task of a ML model can be mutlifold the amount of human intervention [3]:

- Classification: the task of assigning a label to a data sample, e.g., this jet is labelled a b-jet. The general case is multiclass, with n labels possible, while a particular and common case is the binary classification case ($n = 2$).
- Regression: the task of predicting a continuous variable based on a data sample, e.g., the momentum of the particle is 15 GeV/c.
- Feature extraction: given a dataset with specific features, reconstruct new features, e.g., given a set of tracks, reconstruct the secondary vertex. This case is a multidimensional case of regression combined with classificiation (do the tracks share a vertex?).
- Generation: given a sample of 1 million $t\bar{t}$ events, sample 10 new data points from the underlying statistical model.
- Anomaly detection: identify and flag rare events in a dataset

There exist different paradigms of ML models, divided mostly along the lines of the amount of human intervention [3]:

- *Supervised learning*: the data used for training is endowed with the information the model must extrapolate. It is therefore restrain to generate information target directly by the learning process. Classificiation and regression are the most common examples.
- *Unsupervised learning*: the data is not endowed with the information the model must learn to infer. The model is therefore trained with an objective to optimise without human intervention, and must discover patterns and insights without any guidance. Generative models and clsutering are prime examples.
- *Semi-supervised learning*: also called *weak supervision*, is a paradigm combining the two above. The model is mostly unsupervised but can benefit from some labelled cases or human input (a technique also named *active learning*). A prime example is that a clustering tasks followed by a classification of the formed clusters. This is particularly fruitful when the cost of labelling the data is expensive, as is the case with real human data but thankfully not so in the case of particle physics data.
- *Reinforcement Learning*: this paradigm of ML is dedicated to the setting of a game theoretic environment. An agent must explore its environment and can act by choosing a specific policy. In Reinforcement Learning (RL), the objective is for the agent to learn how to construct the most optimal policy to satisfy a reward function.

Given how important DL methods have become in all technological fields, this chapter is primarily dedicated to introducing some of its approaches relevant to HEP. This form of AI is indeed specifically well-suited to the setting of the A Toroidal LHC ApparatuS (ATLAS) experiment, as it enjoys:

- large datasets of both real and simulated data are readily available,

- thanks to advanced Monte Carlo (MC)-based simulation programs, the simulated data points are faithful representations of the real data,
- the data and data-model from which the data originates is well understood in physics, the former coming from measurements from well-calibrated detectors and the second from crafted theories of the field.

1.2 Machine Learning Methods for Physics

High-energy physicists enjoy a special relationship with ML methods. Experimental particle physics largely relies on statistical analyses of complex and large datasets, be that simulated using MC methods or collected from sophisticated detector apparata. A typical physics analysis can be described as the succession of four main steps:

1. Data collection: real data is collected from a detector exposed to the underlying physics desired, e.g., at CERN placing and callibrating the ATLAS detector at an interaction point of the Large Hadron Collider (LHC) to collect proton-proton collision data.
2. Simulated data is generated to match the condition of collection of the real data - in terms detector effects and operational conditions such as energy, Pile-up (PU) and luminosity. This simulated data englobes the best of our current theorical knowledge of the law of physics.
3. The detector of a modern particle physics experiment is a complex set of sub-detector system sensitive to different physical phenomena, as described in chapter This low-level information collected by different device must be processed and recombine to generate *objects*, aggregated information that often hold physical interpretation. For examples, from hits in the tracking detector a track can be fitted and some of its physical properties, such as $p_T$reconstructed. This task corresponding to a mapping from *low*-level $\rightarrow$ *high*-level measurement information can benefit from ML in many ways. Broadly, ML-based method can offer scalable, efficient, and precise solutions to this object reconstruction step.
4. An analysis strategy is established, with objective to similarly restrict the full dataset of both simulated and real data to a portion of the data-space that is most sensitive to the search signal or process. The sensitivity aspect underlies the need to take into account limited knowledge of the theorical physics, limited precision of the apparata, limited statistics of both simulations and data collected. To optimise the analysis, selection rules are derived based on physically accessible information, e.g., the centre-of-mass energy, presence of leptons, the transverse momentum p_T, and other high-level object reconstructed in the previous step.
5. With the optimally selected set of real and simulated data, a statistical model is built to quantify the agreement of the measured data with the expectations from the theory under the conditions of the experiment. This is most often achieved through a likelihood computation.

Modern advanced machine learning has the potential to improve all steps of this process:

1. The operationial side of running the detector and the accelerator pipeline can benefit from RL methods for improved control of the different electronic device. Triggers, an essential component of the ATLAS experiment can be upgraded to use sophisticated DL model running online thanks to a hardware back-bone built on Field-Programmable Gate Array (FPGA)s.
2. Simulating a dataset through MC approach is a computationally intensive task. Each event must pass through a selection of probabilistic step, with only a sastifying all requirements ending in the usable sample. This process can be speed-up and optimised significantly, but the costs

remain significant to generate sample of sufficient statistics. Generative AI has the potential to accelerate this step by giving statistical model that are possible to sample efficiently. Generative Adversarial Network (GAN) and Variational Auto-Encoder (VAE) have been shown to perform the sampling step in a competitive amount of time. However, a key limitation of these stasticial approach is their limited ability to encorporate the sophisticated theorical model required to simulate the data, with any discrepancy or unclosure introducing levels of disagreement that are counter-productive for the final objective of the physics analyses.

3. DL is particularly well-suited for the object reconstruction task. Important examples in ATLAS are identifying particles in the detector (e.g., τ identification), reconstructing missing transverse energe (E_T) in the triggers, and classifying heavy-flavour jets - as exemplified in the next chapter of this thesis dedicated to flavour tagging.

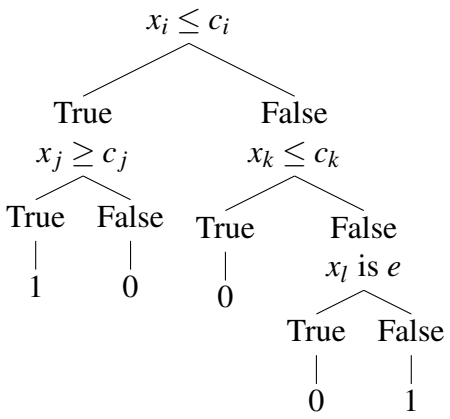
4. Historically, physicists have relied on a cut-base approach to select their data: they painfully analyse relevant variables to the physics problem to try and identify the best features to use to restrict the dataspace through manually-set restrictions. For example, in a measurement of Z-bosons decaying to two charged leptons l^+l^- search, restricting the invariant mass of the lepton pair $m_{l^+l^-}$ to lie close the Z-boson rest mass $m_Z \approx 91.19$ GeV/c^2 is beneficial, as most of the search process will be found there. Machine learning is able to entirely bypass this need, learning directly from an appropriate set of signal and background datasets a transformation of the input data features to a discriminant optimising the separation of signal from background.

5. The likelihood function of the constructed statistical test, verifying the level of agreement between the real data and the theory through the simulated sample, can be directly learnt by a model given access to both sets. Furthermore, anamoly detection settings, such as those in the search for unknown resonances, can be derived using model ML in an unsupervised setting, thereby automating the discovery process and requiring only real data.

One of the focus of this thesis can be broadly summarised as contributing to step 2 in the aformentioned list: developping DL-tools for improved object reconstruction. The analysis presented in the latter part of this document also introduces some classical ML technique of data selection - step 3. The rest of this chapter consists in a review of the relevant ML methods.

1.2.1 Decision Trees

Decision Trees (DT), also called textitClassification and Regression Trees (CART) are the bread and butter of any data analysis. They are simple to train, give a good ground performance for both classification and regression tasks, and are white box model - meaning they are easy to interpret. Underlying the model is a recursive partitioning approach of the input space [3]. Labelling a partition step as *node*, the tree structures emmerges from a *root* state that is subsquently partitioned along different branches with one *leaf* per final region. The splits are done along a feature of the input space, and the method accept both discrete categorical values (e.g., the label of a lepton as e, μ, τ) and continuous values (e.g., m_l). A simple classification tree structure can be represented as the following tree:



At each node there is a learnt condition with x_i, x_j, x_k being continuous features of the dataset that are cut on c_i, c_j, c_k and c_l is a categorical feature (is the lepton an electron). The leaf values are the output of the tree in different regions defined by the combination of successive selections - here a binary variable. An example of a thus defined region is shown in Figure 1.1, where a tree with two nodes is able to isolate most of the blue class from the red class with the region limited by green lines, corresponding to both conditions $x_1 \geq c_2$ and $x_2 \geq c_2$ being satisfied.

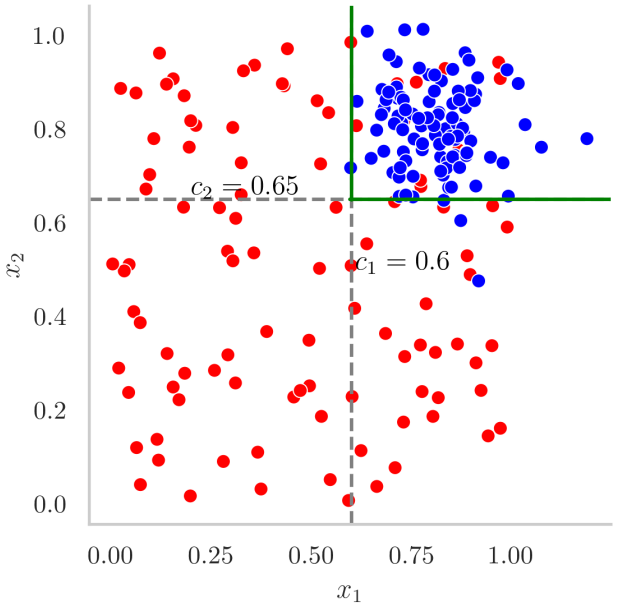


Figure 1.1: Binary classification with two features. A decision tree applies two successive cuts c_1 and c_2 to isolate the blue class from the red.

Finding the optimal set of partitions of a dataset is an NP-complete problem and therefore intractable for large datasets. To build a tree, a greedy approach must be adopted - meaning a heuristically successively choosing the most optimal step at each stage with no guarantee to find a local optimum with no guarantee of reaching a global optimum. The chosen split is selected based on a defined *cost* by Equation 1.1.

$$(j^*, t^*) = \arg \min_{j \in \{1,\ldots,D\}, t \in T_j} \min \left(\text{cost}(\{x_i, y_i : x_{ij} \leq t\}) + \text{cost}(\{x_i, y_i : x_j > t\}) \right) \tag{1.1}$$

where T_j is the set of possible thresholds, x_j, y_j are the features and label (or regressive objective). For categorical variable, the inequality $x_j >< t$ would be typically converted in a value equality $x_j == t$. The *cost* function will depend on the objective of the tree, with the regression case using the error function

$$cost(D) : \sum_{i \in D} (y_i - \bar{y})^2,$$

and for a classification the typical loss would be one of the following:

- **Missclassification rate**: $\frac{1}{|D|}\sum_{i\in D}\mathbb{I}(y_i \neq \hat{y})$, where D is the data in the leaf of the tree.
- **Statistical entropy**: by defining the class-condidition probability $\pi_c = \frac{1}{|D|}\sum_{i\in D}\mathbb{I}((y_i) \neq c)$, where $\mathbb{I}$ is the identity function ($\mathbb{I}(x) = 1$ if x is True, else 0) the entropy over the (C) classes is defined as in Equation 1.2.

$$H(\vec{\pi}) = -\sum_{c=1}^{C} \pi_c \log \pi_c. \tag{1.2}$$

- **Information Gain**: an equivalent formulation to the entropy, where the gain in information that should be maximised is the relative change in entropy by adding a selection on feature X_j:

$$\text{Gain}(X_j < t, Y) = H(Y) - H(Y|X_j < t)$$

- **Gini**: computes the expected error rate:

$$\sum_{c=1}^{C} \pi_c(1-\pi_c).$$

The pseudocode algorithm to train a DT with this update rule is summarised in Algorithm 1.

Algorithm 1 Recursive Procedure to Train a Decision Tree [3]

```
function FITTREE(node, D, depth)
    node.prediction ← mean({y_i : i ∈D})
    (j*, t*, D_L, D_R) ← split(D)
    if not worthSplitting(depth, cost, D_L, D_R) then
        return node
    else
        node.left ← FITTREE(node, D_L, depth + 1)
        node.right ← FITTREE(node, D_R, depth + 1)
        return node
    end if
end function
```

DT can overfit a dataset and regularisation serves as an important step to avoid this unwanted behaviour. For trees, a natural procedure to avoid overtraining is to interrupt the growth of the tree when it is no longer worth doing so - a criterion that is hard to decide *a priori* - or to *prune* the tree, which implies the removes nodes or branches that contribute little to the overall performance. A simpler way to regularise the performance by reducing the variance of the estmate of the model is to train several trees with different subsets of the data chosen randomly with replacement and aggregate the results into a single prediction, for example by taking the average over each base learners:

$$y(x) = \frac{1}{N_l}\sum_{i=1}^{N_l} y_i(x),$$

for N_l base learner making prediction $y_i(x)$ given the input features x. This statistical technique of using ensemble of predictors is refered to as *bagging*. To further decorrelate the performance of the different predictors, these can be reduced to a subset of the input features, thereby forming a *random foret*.
For the following discussion, the model is built on a training dataset $\{(x_1, y_1), ..., (x_N, y_N)\}$ with input vectors $x_i \in \{\mathbb{R} \otimes \mathbb{D}\}^d$ of d features that are real or discrete ($\mathbb{D}$) and $y \in \mathbb{R}^d$ is a d-dimension real vector that serves as output to be predicted by the model.

1.2.2 Boosted Decision Trees

Another extension to the simple decision trees is to introduce the concept of *boosting*, a method called Boosted Decision Trees (BDT) and, importantly in particle physics, MVA. Boosting is a greedy algorithm leveraging a weak learner or predictor (e.g., a DT) and applying it sequentially to weighted versions of the data, with a larger weight given to missclassified / miss-regressed datapoints. This method is hugely popular in data science, having earned the title *"best off-the-shelf classifier in the world"* [4]. Two particularly useful approaches are adaptive boosting (AdaBoost) [5] and gradient boosting [6], both combining an ensemble of L weak learners f_i ($i = 1, ..., L$) into a strong learner

$$F(x) = \sum_{i=1}^{L} f_i(x).$$

AdaBoost: combines the L weak learners h_i with adaptive weights α_i to improve the ensemble performance as

$$\hat{f}(x) = \sum_{i=1}^{L} \alpha_i h_i(x),$$

where $\hat{f}$ is the boosted model, and the successive boosting stages $f_T = \sum_{i=1}^{T \leq L} \alpha_i h_i(x)$ define stronger and stronger boosted variants of the model. At each iteration m of the training process ($m = 1, ..., L$), a weight w_i^m is assigned to each training sample (indexed by i) proportional to the current error or loss: $L(y_i, f_{t-1}(x_i))$, where the typical case for AdaBoost is binary classification with $y_i \in -1, 1$. The error in AdaBoost is the exponential loss on the datapoint of Equation 1.3.

$$E = L(y, f_m(x)) = \sum_{i=1}^{N} \exp\left(-y_i f_m(x_i)\right) = \sum_{i=1}^{N} \exp\left(-y_i (f_{m-1}(x_i) + \alpha_m h_m(x_i))\right) \tag{1.3}$$

Equation 1.3 can be re-expressed to highlight the weight $w_{i,m}$ applied to each datapoint (x_i, y_i) at step m as :

$$\sum_{i=1}^{N} w_{i,m} \exp\left(-\alpha_m y_i h_m(x_i)\right),$$

where $w_{i,m} = \exp\left(-y_i f_{m-1}(x_i)\right)$.
The weak learner at step m is applied to the weighted version of the dataset with weights $w_{i,m}$ with optimal weights α_m found by minimising the error to be

$$\alpha_m = \frac{1}{2} \log \frac{1 - \varepsilon_m}{\varepsilon_m},$$

where ε_m is the ratio of miss-classified weights: $\varepsilon_m = \sum_i w_{i,m} \mathbb{I}(y_i \neq h_m(x_i)) / \sum_i w_{i,m}$. The AdaBoost algorithm is summarised in Algorithm 2.

Algorithm 2 Adaboost for Binary Classification with Exponential Loss [3]

Initialize weights: $w_i = \frac{1}{N}$, where N is the number of samples.
for $m = 1$ to M **do**
  Fit a classifier $h_m(x)$ to the training set using weights w.
  Compute $\varepsilon_m = \sum_i w_{i,m} \mathbb{I}(y_i \neq h_m(x_i)) / \sum_i w_{i,m}$.
  Compute $\alpha_m = \log\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right)$.
  Update weights: $w_i \leftarrow w_i \exp(\alpha_m \cdot \mathbb{I}(y_i \neq h_m(x_i)))$.
end for
return $f(x) = \text{sgn}\left(\sum_{m=1}^{M} \alpha_m h_m(x)\right)$

Gradient boosting: is a generic approach which does not require a specific derivation for each loss function. The objective is to minimise the empirical risk, the expected value of the loss function L on the training set as shown in Equation 1.4:

$$\hat{f} = \arg\min_f \mathbb{E}_{x,y} L(y, f(x)) \tag{1.4}$$

where $f(x) = (f(x_1), ..., f(x_N))$ is the output of the learner on the training set. As the name suggests, the approach leverages gradient descent to find the optimal $\hat{f}$. At step m, the gradient of the loss $L(f)$ is evaluated at $f = f_{m-1}$ as

$$g_{i,m} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f=f_{m-1}},$$

which is then used to update the learner with a step

$$f_m = f_{m-1} - \alpha g_m$$

of step-length α_m chosen to minimise the residual loss $L(y, f_{m-1} - \alpha_m g_m)$.

For the specific case of gradient boosted decision trees, at step m a DT $h_m(x)$ is fitted to the pseudo-residuals. This DT h_m at step m defines J_m disjoint regions through its leaves with predictions b_{jm} in each $j = 1, ...J_m$ regions:

$$h_m(x) = \sum_{j=1}^{J_m} b_{jm} \mathbf{1}_{R_{jm}}(x),$$

where $\mathbf{1}_{R_{jm}}(x)$ is the indicator function - equals to 1 when $x \in R_{jm}$ and 0 otherwise. The update to the model is chosen so that:

$$f_m(x) = f_{m-1} + \gamma_m h_m(x),$$

with γ_m selected by minimising the empirical risk of the updated model:

$$\gamma_m = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \gamma h_m(x_i)).$$

Algorithm 3 Gradient Boosting [3]

Initialize $f_0(x) = \arg\min_\alpha \sum_{i=1}^N L(y_i, h(x_i; \alpha))$
for $m = 1$ to M **do**
  Compute the gradient residual $\forall i$: $g_{i,m} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x_i)=f_{m-1}(x_i)}$
  Train weak learner h_m on the dataset $\{(x_i, r_{i,m})\}_{i=1}^N$
  Compute α_m by minimising $\sum_{i=1}^N (g_{i,m} - \alpha_m h_m(x_i))$
  Update $f_m(x) = f_{m-1}(x) + \nu \alpha_m h_m(x)$
end for
return $f(x) = f_M(x)$

The full algorithm for Gradient Boosting is mentioned in Algorithm 3, where the update rule is added a *learning rate* hyperparameter ν to introduce regularisation and reduce the risk of overfitting. By keeping $0 < \nu \leq 1$, reduces the ability of the model to fully adapt to the training error, thereby improving generalisation. The price is a slower updating of the model and therefore a more demanding computational time. Further regularising techniques are bootstrap aggregation - training each weak learn on a random subset of the data -, limiting the number of leaves, or more generally penalising model of larger complexity - removing branches that do not reduce the loss by a minimal amount.

BDT resist better to overtraining thanks to the regularisation effect of boosting and the different techniques described in thi section. An undiserable feature of boosting is the loss of direct interpretability of the decision.

1.2.3 Artificial Neural Network

1.2.4 Multilayer Perceptron

1.2.5 Recurrent Neural Network

1.2.6 Convolutional Neural Network

1.2.7 Graph Neural Network

1.2.8 The rise of the Transformer

1.2.9 Generative Models: GAN & VAE

1.2.10 Reinforcment Learning

1.3 Training and Optimising Deep Learning Models

BIBLIOGRAPHY

[1] Dan Cireşan, Ueli Meier, and Juergen Schmidhuber. "Multi-column Deep Neural Networks for Image Classification". In: *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Feb. 2012). DOI: 10.1109/CVPR.2012.6248110.

[2] Yann LeCun et al. "Handwritten Digit Recognition with a Back-Propagation Network". In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989. URL: %5Curl%7Bhttps://proceedings.neurips.cc/paper_files/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf%7D.

[3] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020.

[4] Leo Breiman. "Bagging predictors". In: *Machine Learning* 24.2 (1996), pp. 123–140. DOI: 10.1007/BF00058655. URL: %5Curl%7Bhttps://doi.org/10.1007/BF00058655%7D.

[5] Yoav Freund and Robert E Schapire. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1997.1504. URL: https://www.sciencedirect.com/science/article/pii/S002200009791504X.

[6] Jerome H. Friedman. "Greedy function approximation: A gradient boosting machine." In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. DOI: 10.1214/aos/1013203451. URL: https://doi.org/10.1214/aos/1013203451.