Improved Selective Background Monte Carlo Simulation at Belle II with Graph Attention Networks and Weighted Events

Boyang Yu

Master′s Thesis
at the Faculty of Physics
Ludwig–Maximilians–Universität München
Chair of Elementary Particle Physics

Supervisors:
Prof. Dr. Thomas Kuhr
Dr. Nikolai Hartmann

Munich, September 14, 2021

Verbesserte selektive Untergrund-Monte-Carlo-Simulation bei Belle II mit Graph-Attention-Networks und gewichteten Ereignissen

Boyang Yu

Masterarbeit
an der Fakulität für Physik
Ludwig–Maximilians–Universität München
Lehrstuhl für Experimentelle Flavorphysik

Betreuer:
Prof. Dr. Thomas Kuhr
Dr. Nikolai Hartmann

München, September 14, 2021

Abstract

When measuring rare processes such as $B \to K^{(*)}\nu\bar{\nu}$ or $B \to l\nu\gamma$, a huge luminosity is required, which means a large number of simulations are necessary to determine signal efficiencies and background contributions. However, this process demands high computation costs while most of the simulated data, in particular in case of background, are discarded by the event selection. Thus filters using neural networks are introduced after the Monte Carlo event generation to speed up the following processes of detector simulation and reconstruction. Merely filtering out events will however inevitably introduce bias. Therefore statistical methods are invested to deal with this side effect.

This thesis extends previous work by James Kahn and Yannick Bross by the usage of advanced graph neural network architectures and the introduction of weighted events.

In this work, I first study optimizations of the performance of graph neural network structures by implementing advanced architectures with attention mechanisms and validate them on large datasets. Then I present different ways to avoid the bias with the help of event weights in post-processing and finally compare the biases after each strategy by checking weighted distributions of several observables relevant for physics analysis.

Contents

Chapter 1

The BELLE II Experiment

The standard model (SM) is so far the best-tested fundamental theory to describe nature. However, there still remain many primary questions to be answered. To complement the SM, many new physics (NP) scenarios have been proposed and experiments in high energy physics (HEP) are designed for confirmations of new particles and new processes. The experiments in HEP can be divided mainly by their focus into two categories: a) At the energy frontier, such as the Large Hadron Collider (LHC), and b) At the intensity frontier, such as the Belle II, which provides the context for this work.

1.1 SuperKEKB and BELLE II Detector

KEK[10] is the abbreviation for "the Japanese High-Energy Accelerator Research Organisation". It operated KEKB from 1998 to 2010, which is a 3 km circumference asymmetric electron-positron collider with instantaneous luminosity of $2.11 \times 10^{34}\,\mathrm{cm}^{-2}\mathrm{s}^{-1}$. KEKB is known as a B factory because it operates at a center of mass energy equivalent to the $\Upsilon(4S)$ resonance that predominantly decays into B-mesons. The corresponding Belle experiment concentrated on the characteristics of $B\bar{B}$ pairs and arrived at its highlight in 2008 when Makoto Kobayashi and Toshihide Maskawa were awarded the Nobel prize in physics for their theory to explain CP-violation which was confirmed by the Belle experiment. The measured level of CP-violation is however not sufficient. Therefore scientists were looking for a much deeper understanding of the related phenomena.

As an upgraded version of KEKB, SuperKEKB (Figure 1.1 left) has an increased instantaneous luminosity by about a factor of 40 to $8 \times 10^{35}\,\mathrm{cm}^{-2}\mathrm{s}^{-1}$. Traditionally a higher luminosity was reached by increasing the beam currents. However, SuperKEKB was designed based on ideas of Pantaleo Raimondi from the Italian SuperB project, to use a large crossing-angle at the interaction point and squeezing to nanometer-scale to increase luminosity (Figure 1.2). The integrated luminosity is expected to reach $50\,\mathrm{ab}^{-1}$ which is 50 times more data than Belle.

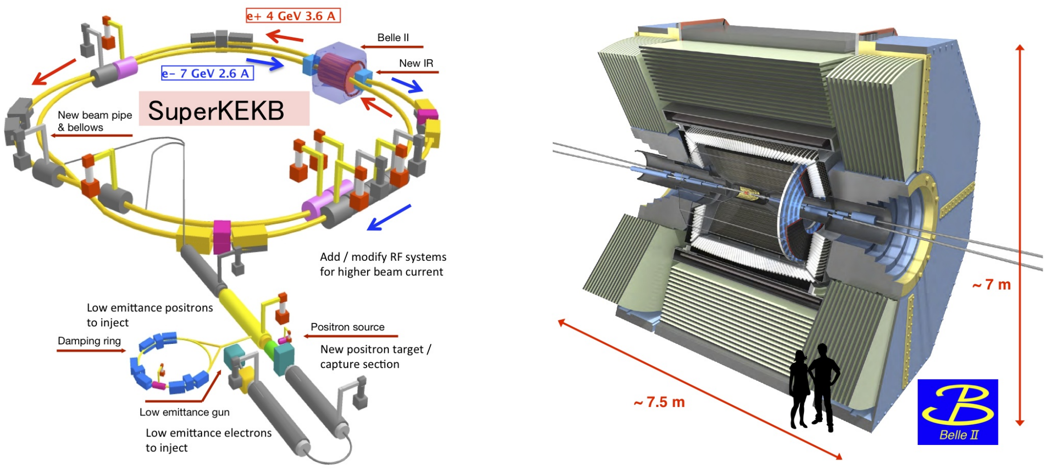

e+ 4 GeV 3.6 A
e- 7 GeV 2.6 A
Belle II
New IR
SuperKEKB
New beam pipe & bellows
Add / modify RF systems for higher beam current
Low emittance positrons to inject
Positron source
Damping ring
New positron target / capture section
Low emittance gun
Low emittance electrons to inject
~ 7 m
~ 7.5 m
Belle II

Figure 1.1: SuperKEKB and Belle II

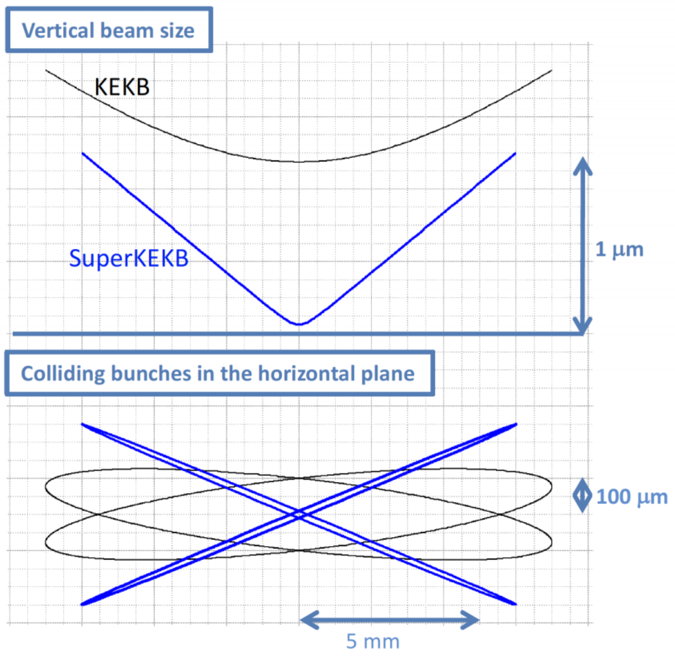


Figure 1.2: Schematic view of the nanobeam collision scheme [12]

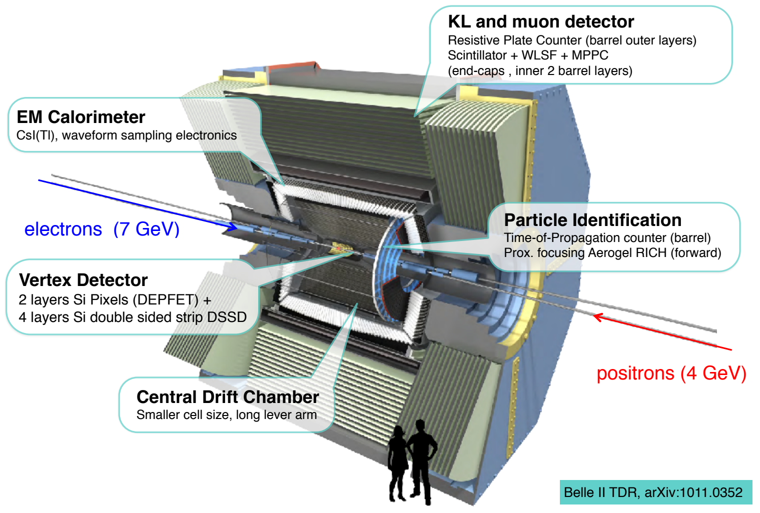

KL and muon detector
Resistive Plate Counter (barrel outer layers)
Scintillator + WLSF + MPPC
(end-caps , inner 2 barrel layers)
EM Calorimeter
CsI(Tl), waveform sampling electronics
electrons (7 GeV)
Particle Identification
Time-of-Propagation counter (barrel)
Prox. focusing Aerogel RICH (forward)
Vertex Detector
2 layers Si Pixels (DEPFET) +
4 layers Si double sided strip DSSD
positrons (4 GeV)
Central Drift Chamber
Smaller cell size, long lever arm
Belle II TDR, arXiv:1011.0352

Figure 1.3: 3D-Schematic view of Belle II dector [16]

Because of the higher rate of collisions by the SuperKEKB accelerator, the Belle detector was upgraded to Belle II (Figure 1.1 right). The Belle II detector is comprised of the following sub-detectors: Vertex detector (VXD), Central drift chamber (CDC), Particle identification (PID), Electromagnetic calorimeter (ECL), K-Long and muon detector (KLM) (Figure 1.3).

As mentioned before, the most prominent feature of this project is its high luminosity, especially when compared with other experiments (Figure 1.4). However, this also produces challenges for data processing to tremendous dataset it generates, which is also the motivation of this work.

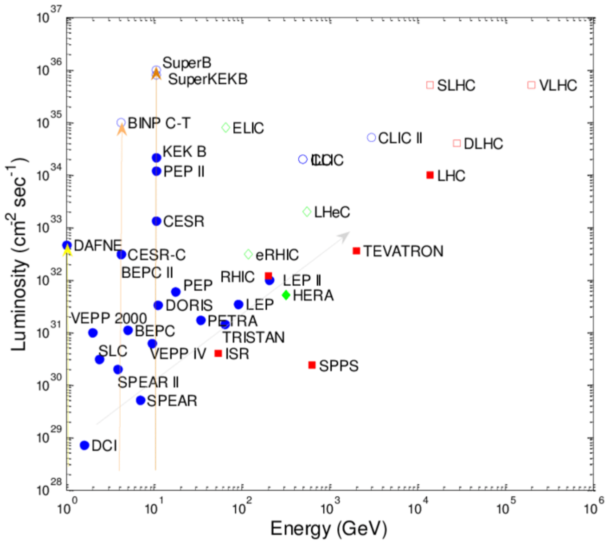


Figure 1.4: Luminosity versus energy of colliders [13]

1.2 Monte Carlo Simulation

To analyse the experiment data, simulations are constructed in parallel, using Monte Carlo (MC) experiments. Virtual signals are generated through **MC** and detector simulation (**Det. Sim.**), then experience the same processing as real data to get physical characteristics extracted. Finally these two results are compared in order to match physical quantities to observations from the experiment (Figure 1.5). Simulated data passes through the following steps in a Belle II analysis:

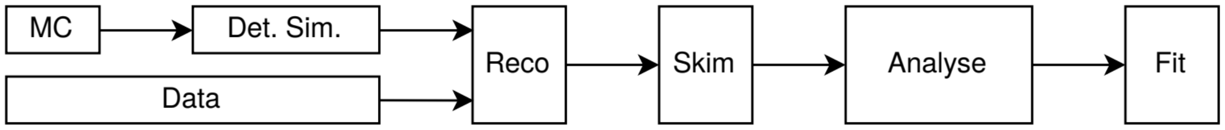


Figure 1.5: Data flow within a Monte Carlo based experiment. [16]

These steps are:

- **MC**: In this step, hadron decays and the following quark hadronisations are simulated using EvtGen package [30] and Pythia [35] separately. This simulates the collision event of electron and positron as well as the subsequent decays to stable final state particles (FSP) in vacuum.
- **Det. Sim.**: During detector simulation, the simulated decays from the MC stage virtually interact with the Belle II detector with the help of the GEANT4 toolkit [11]. The output format of this step is identical to that of the real experiment output (detector hits, track candidates, etc.), except for the information of MC simulations.
- **Data**: Real experiment outputs from Belle II detectors. Independent of the above simulations.
- **Reco**: Reconstruct the detector information into particle candidates for easier further analyses. Details in Chapter 1.3.
- **Skim**: Select suitable events to be used for further analyses. Requirments can be much different for different uses. In this study this is specified as Full Event Interpretation (FEI) for Hadronic B^0 channel. Details will come in Chapter 1.4.
- **Analyse/Fit**: Further studies and comparisons between MC simulations and real experiment data. Not relevant in this work.

Due to the expensive computational cost for **Det. Sim.** and **Reco** steps as well as the low retention rate ($\mathcal{O}(10^{-7}$ *to* $10^{-1}))$ for background during the **Skim** process, I follow the work of James [25] to add a neural network (Figure 1.6, NN in red) as a filter directly after Monte Carlo generation to pre-decide which particles will be kept to save computation in the following data flow (see Chapter 4).

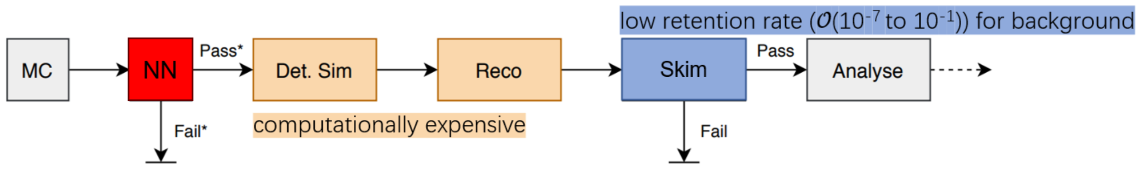

MC
NN
Pass*
Fail*
Det. Sim
Reco
computationally expensive
low retention rate (𝒪(10⁻⁷ to 10⁻¹)) for background
Skim
Pass
Fail
Analyse

Figure 1.6: Simulation with smart background selection.

1.3 Tagging Method

To measure the branching fraction of rare processes such as $B \to K^{(*)}\nu\bar{\nu}$ or $B \to l\nu\gamma$, the corresponding signal must be identified amongst all e^+e- collision events. In the study, Monte Carlo simulations are used to tune the analysis for identifying and isolating the signal and rejecting the non-signal (background) events, requiring a large simulation sample.

During the reconstruction, signal events have to be selected with the help of the so called tagging method (Figure 1.7):

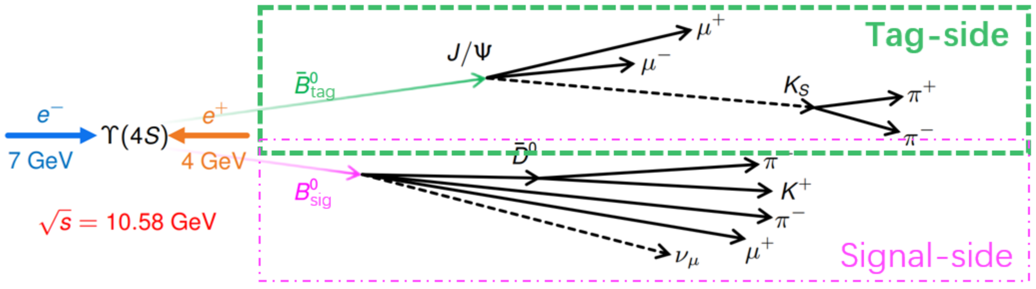


Figure 1.7: Tagging Method.

1. Reconstruct generically decaying B mesons, B_{tag}, using the Full Event Intergretation (FEI) software (see Chapter 1.4).

2. Reconstruct signal-side B mesons, B_{sig}, from experiment data.

3. Pair two mesons each from one of the above to produce an $\Upsilon(4S)$ resonance.

4. Send the reconstructed $\Upsilon(4S)$ decays into Skimming process (see Chapter 1.5) to filter out the background events while retaining as many signal events as possible.

1.4 Full Event Interpretation

FEI is an exclusive tagging algorithm for the Belle II experiment which enables precise measurement of otherwise inaccessible B meson decay-modes. With the help of machine learning it can automatically identify plausible B meson decay chains based on the experiment data from detector. FEI provides a greater efficiency to enable a larger effective sample size in the measurement [26].

The hierarchical procedure of the FEI reconstruction is shown in Figure 1.8. The inputs are reconstructed tracks and clusters. These informations are then translated as FSP and combined to form intermediate particles and finally used to reconstruct B mesons. Using a series of fast Boosted Decision Trees (fBDT, see Chapter 2.3), the FEI determines the likelihood of each reconstructed candidate based on its properties as well as its parent status. Therefore the output of the FEI is a list of reconstructed B mesons for each input event, associated with their signal probabilities (sigProb) indicating the likelihood of their correct reconstructions. For B^0 decays, the object used in this study, the reconstruction efficiency is 0.24%.

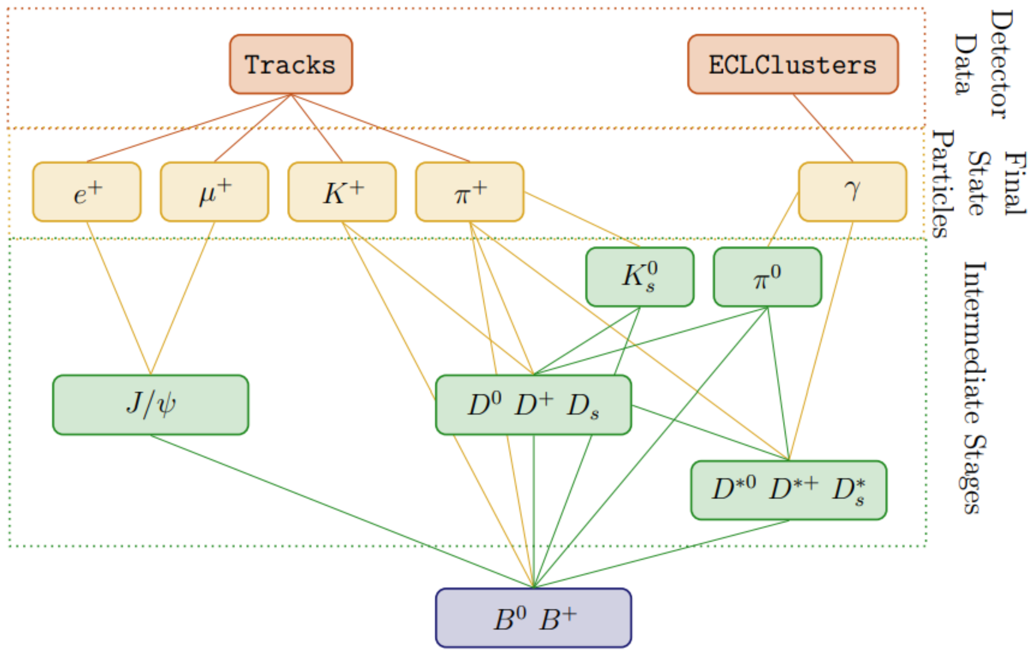


Figure 1.8: Schematic overview of the FEI. [27]

1.5 Skimming

Due to the high luminosity of Belle II and the corresponding large data volume for analyses, Skims are used to reduce the number of irrelevant samples by applying a simple selection at event level. Skimming is applied for both MC simulated and real data. Information on the reconstructed B candidates is calculated and added to the kept events.

This study uses the FEI skims for reconstruction of hadronic B^0 (see Chapter 1.4) where two sets of selection criteria are carried out before and after the FEI reconstruction:

Skim pre-cuts ([1]: fei_precuts) at event-level are applied before running the FEI to reduce computation time:

- $n_{\text{cleaned tracks}} \geq 3$
- $n_{\text{cleaned ECL clusters}} \geq 3$
- Visible energy of event (CMS frame) $> 4\,\text{GeV}$
- $2\,\text{GeV} < E_{\text{cleaned tracks \& clusters in ECL}} < 7\,\text{GeV}$

where cleaned tracks and clusters are defined as:

- Cleaned tracks : $d_0 < 0.5\,\text{cm}$, $|z_0| < 2\,\text{cm}$, and $p_T > 0.1\,\text{GeV}$
- Cleaned ECL clusters : $0.296706 < \theta < 2.61799$, and $E > 0.1\,\text{GeV}$

After FEI there is a Tag side selection ([1]: feiHadronicB0):

- $M_{bc} > 5.24\,\text{GeV}$
- $|\Delta E| < 0.2\,\text{GeV}$
- signal probability > 0.001

where M_{bc} is the beam-constrained mass of the reconstructed B_{tag} defined as $M_{bc} = \sqrt{E_{\text{beam}}^2 - p_{B_{\text{tag}}}^2}$, ΔE is the reconstructed energy difference $\Delta E = E_{B_{\text{tag}}} - E_{\text{beam}}$, and signal probability is the correct reconstruction-probability output by the FEI. The retention rate of the Hadronic B^0 channel used in this study is about 5.86%.

Chapter 2

Machine Learning

The method of selective Monte Carlo simulation used in this work relies on Machine Learning. Therefore some basic ideas of Machine Learning (ML) are introduced in this chapter. I will first explain the definition and show some general characteristics of ML. Then I go into details to introduce two important branches of ML: Neural Networks and Decision Trees, which are applied in this work.

2.1 Machine Learning Basics

As a sub-field of Artificial Intelligence (AI), ML provides machines the ability to learn without explicitly being programmed (Figure 2.1). It is a form of applied statistics with increased emphasis on the use of computers to statistically estimate complicated functions and a decreased emphasis on proving confidence intervals around these functions [22].

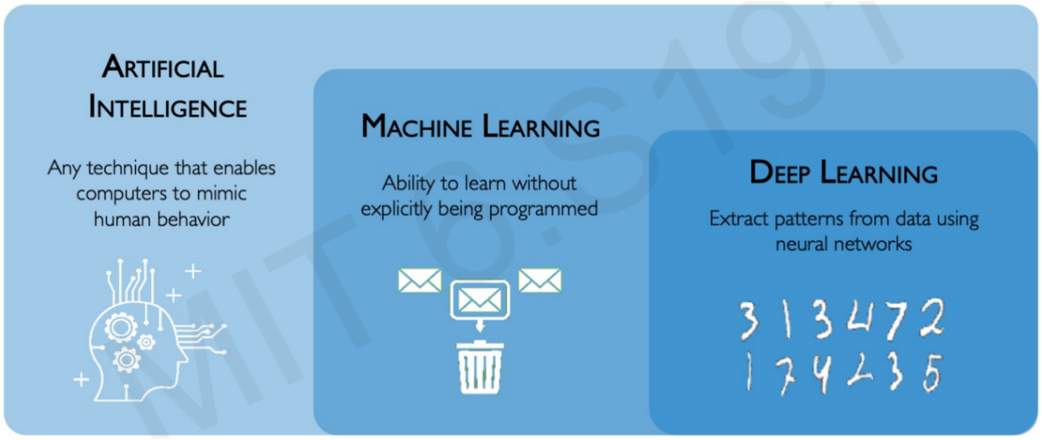

ARTIFICIAL INTELLIGENCE
Any technique that enables computers to mimic human behavior
MACHINE LEARNING
Ability to learn without explicitly being programmed
DEEP LEARNING
Extract patterns from data using neural networks
MIT 6.S191

Figure 2.1: Relations of Artificial Intelligence, Machine Learning and Deep Learning [6]

A succinct definition of "Learning" is given by Mitchell [33]: "A computer program is said to **learn** from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**." as visualized in the Figure 2.2.

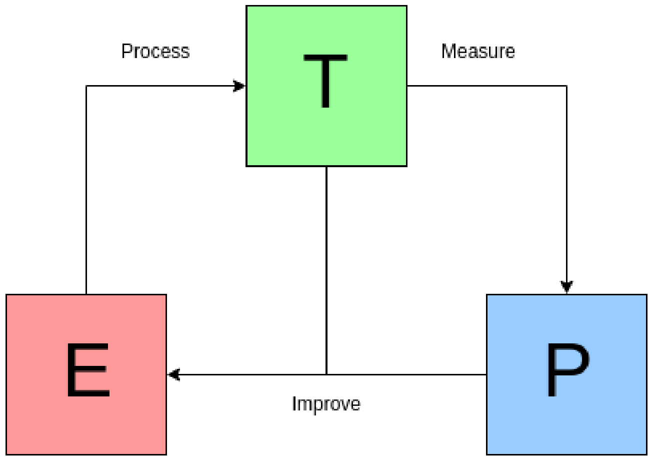


Figure 2.2: The Mitchell Paradigm [2]

ML shows an easier way to deal with problems that are too difficult to solve with fixed programs designed by human beings, while **Learning** is the process to acquire the ability to perform the task.

The way an ML system processes an example is usually understood as a **Task**, where an example is a set of n features $x \in \mathbb{R}^n$ measured from an object to be processed by our ML system. Most common tasks include **Classification**: $f : \mathbb{R}^n \to \{1, \ldots, k\}$, with k classes; **Regression**: $f : \mathbb{R}^n \to \mathbb{R}$ to find a function, etc..

Experience is a set of knowledge that an ML system has access to. According to different experiences, scientists categorize ML algorithms into several areas (Table 2.1):

Algorithm	Experiences
Supervised learning	**Data** and **Label**
Unsupervised learning	**Data** alone
Semi-supervised learning	**Data**, partly with **Label**
Reinforcement learning	**Data** and **Rule**

Table 2.1: Categories of Machine Learning algorithms according to their Experience

Performance defines a metric to evaluate the abilities of an ML algorithm. Very often used are **Accuracy**, **Error**, **Cross Entropy**, and so on, comparing the predictions of ML with the expected answers.

In this work two different systems of ML are used, one is Filter Prediction (See Chapter 4.2) with Neural Networks, the other is True-Positive Recognition with Decision Trees (See Chapter 5.2.2): (Table 2.2)

	Background Recognition	True-Positive Recognition for (Cutting-)Reweighting Method
Task	**Classify** Target (Pass) events and Background (Fail) events	**Classify** True-Positive (TP) events and False-Negative (FN) events
Experience (both **Supervised**)	**Data**: Graphs of Events **Label**: Pass/Fail	**Data**: Features of Pass-Events **Label**: TP/FN
Performance	Sampling Method: **Speedup Loss** Reweighting Method: **Cross-Entropy Loss**	**Binomial Deviance Loss**
Tools	(Graph) Neural Networks	(Gradient Boosting) Decision Trees

Table 2.2: Machine Learning Systems used in this work

2.1.1 Generalization

To train an ML model, the model outputs changing with small updates of model parameters is compared with the target result in terms of a certain metric. This metric is called **loss function** and should be suited to the task at hand. This is what is minimized or maximized during the training. After one comparison, the program should decide how to update the matrices in order to get closer to the task and then give another try with some new small changes. The update rule is called **Optimizer**. Within a wide range of optimizers, most are gradient based to be used in the training of NNs where efficient calculation of the gradient is allowed by backpropagation. Each update is called a **step** and all the steps to go over the dataset one time is called an **epoch**. The maximum number of epochs is usually restricted to control the training time, while another way to monitor is the amplitude of the last update. During the training process, it is very important to know when to stop and how large one step, the **learning rate**, should be taken. With smaller learning rate the training is more guaranteed to converge, but at a cost of training time (Figure 2.3).

The training process I introduced above is carried out just on a single training set, which simply defines an optimization problem. However, to measure how well an ML model generalizes, the performance of the trained model is evaluated on new inputs which are previously unseen by the system and are independent of the training data. This defines

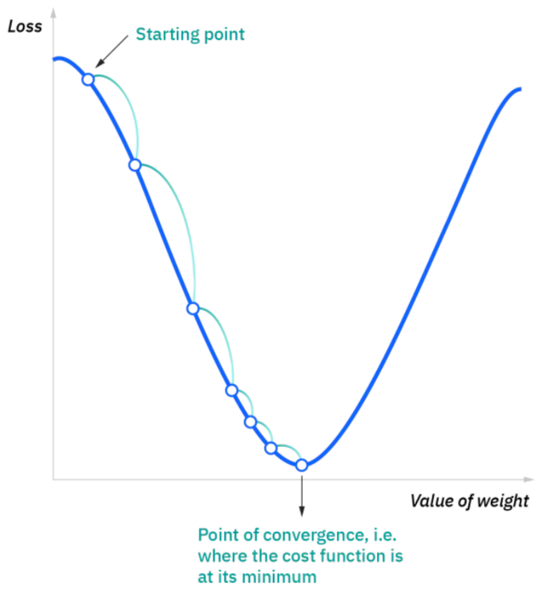

Loss
Starting point
Value of weight
Point of convergence, i.e. where the cost function is at its minimum

Figure 2.3: Ideal Training Process [4]

the generalization error or test error.

A well-trained ML model should have both low training error and test error to be neither **underfitted** nor **overfitted**. One takes this into account when tuning the parameters (such as the number of units, the number of layers, etc.) of an ML algorithm. When the model is not even able to perform well on the training set, it is underfitted. Overfitting occurs when the training error is satisfying, but test error is much larger. The tendency of a model to overfit or underfit is reflected by the **capacity** of the model, which measures the ability of a model to memorize the detailed properties of the training set (Figure 2.4). A more complicated model, e.g. having larger weight matrices, tends to have higher capacity, indicating a better performance on training set but meanwhile a higher danger of overfitting.

Another system to describe this consists of **Bias** and **Variance**. Bias is defined as the difference between the expectation value of estimated results by the model and the true value:

$$\mathrm{Bias}(\hat{\theta}_{\mathrm{model}}) = \mathbb{E}(\hat{\theta}_{\mathrm{model}}) - \theta_{\mathrm{true}}$$

Variance is simply the one we are familiar with:

$$\mathrm{Var}(\hat{\theta}_{\mathrm{model}})$$

For an estimator, bias and variance reflect two different sources of error. Bias shows the expected deviation from the real function, while variance measures the deviation from

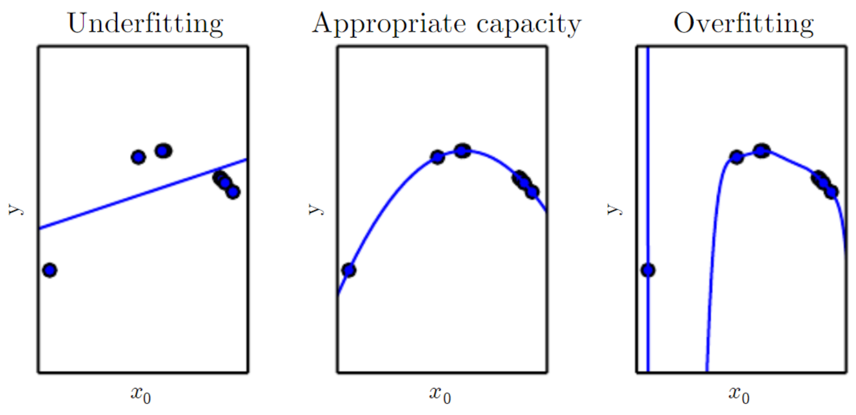


Figure 2.4: Underfitting and Overfitting: Dots are data and curves are trained models. [22]

the expected simulated value that every single sampling data is likely to cause. One measurement to consider both of these two criteria is the **mean squared error** (MSE):

$$\begin{aligned}\text{MSE} &= \mathbb{E}[(\hat{\theta}_{\text{model}} - \theta)^2] \\ &= \text{Bias}(\hat{\theta}_{\text{model}})^2 + \text{Var}(\hat{\theta}_{\text{model}})\end{aligned}$$

Best estimators with both bias and variance satisfying should also have low MSE. Now come back to the previous concepts of capacity, underfitting and overfitting. When generalization error, or test error, is measured by the MSE, increasing capacity helps to decrease bias at the cost of increasing variance. The optimal model with minimum test error can be found in the middle of the U-shaped curve (Figure 2.5).

2.1.2 Gradient Descent

Gradient Descent is a common method to be used in the updating of ML models. The idea is based on approximating the loss function with its first derivative: $f(\vec{x} + \vec{\epsilon}) \approx f(\vec{x}) + \vec{\epsilon} \cdot \nabla f(\vec{x})$ with f the loss function and $\vec{\epsilon}$ the step length, or the *learning rate*. The loss function reaches its **local minimum** or **local maximum** at **critical point** $\vec{x}_c$ satisfying $\nabla f(\vec{x}_c) = 0$ or approximately $f(\vec{x}_c + \vec{\epsilon}) - f(\vec{x}_c) \approx 0$ as shown in Figure 2.6.

With a **descending gradient** $\nabla f(\vec{x}) \leq 0$, the **local minimum** can be ensured with $\nabla f(\vec{x}_{min}) = 0$ and $f(\vec{x}_{min}) \leq f(\vec{x}_{min} + \vec{\epsilon})$.

In a learning process, the **global minimum** instead of a **local minimum** of the loss function is what people expect to reach (Figure 2.7). So the learning rate should not be too small to disable the update to "jump out of" a local minimum or to slow down the convergence of the loss function too much (Figure 2.8, left).

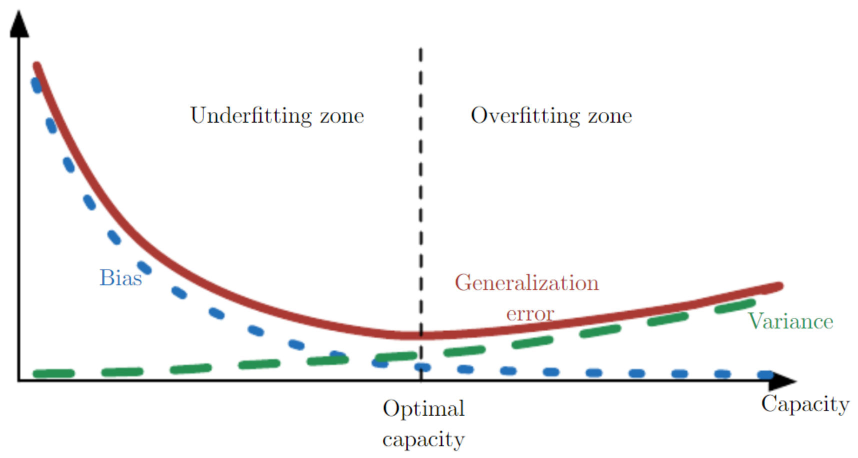


Figure 2.5: As capacity increases, bias (dotted) tends to decrease and variance (dashed) tends to increase, yielding another U-shaped curve for generalization error (bold curve). [22]

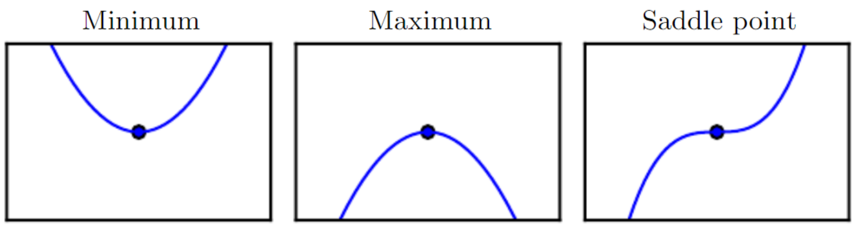


Figure 2.6: Types of critical points in one dimension. [22]

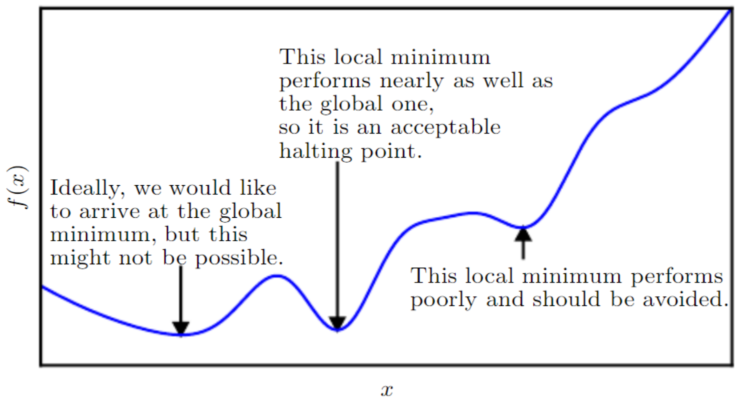

This local minimum performs nearly as well as the global one, so it is an acceptable halting point.
Ideally, we would like to arrive at the global minimum, but this might not be possible.
This local minimum performs poorly and should be avoided.
$f(x)$
x

Figure 2.7: Local minimums and global minimum in one dimension. [22]

However, the learning rate should also not be too large to overshoot the minimum (Figure 2.8, right). Therefore it is important to set a suitable learning rate to fit for the task.

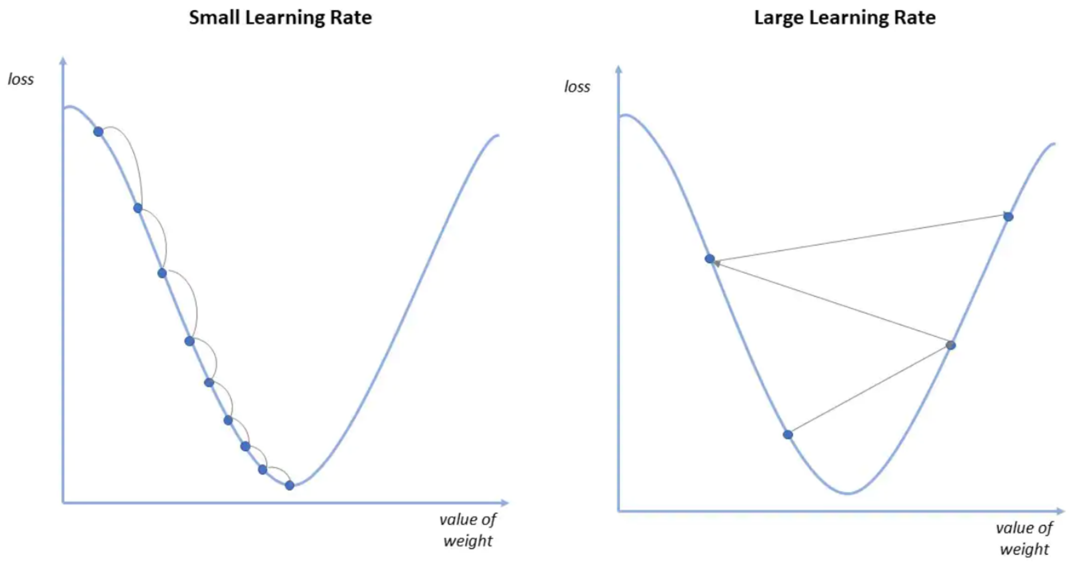


Figure 2.8: Small and large learning rates in one dimension. [4]

2.2 Neural Networks

Neural Networks (NN), also called Artificial neural networks (ANN), are computing systems inspired by information processing and distributed communication nodes in biological neural networks that constitute human and animal brains. Information or signals are proceeded by nodes, or artificial neurons, and transmitted through edges, or connections, to other nodes which usually belong to the next layer.

As a part of deep learning, one neural network often consists of multiple layers, therefore “deep”, with different functions or purposes. With the help of non-linear activation function in each node, the whole NN system is able to learn complicated tasks. Thus it is widely used in many areas of physics such as high energy physics, medical physics, condensed matter physics and so on [15].

2.2.1 General Concept

The most fundamental structure of an NN is the artificial neuron, or called Perceptron (Figure 2.9).

The data from m inputs $\{x^1, x^2, ..., x^m\}$ are summed up with weights $\{w_1, w_2, ..., w_m\}$ together with a bias b to form an intermediate value $z = \sum_{i=1}^{m} x^i \cdot w_i + b \equiv \vec{w} \cdot \vec{x} + b$ which

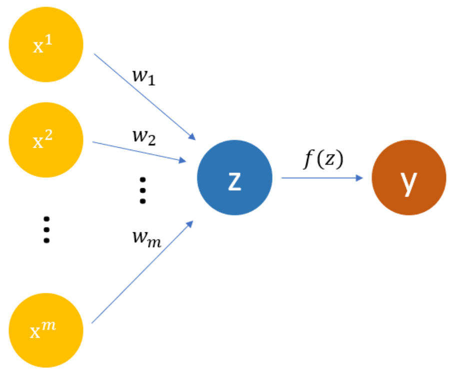


Figure 2.9: An artificial neuron – Perceptron

will then get activated by f, the so called activation function, usually non-linear, to form the final result of this neuron $y = f(z)$.

A more complicated case is called Feed-Forward Networks (Figure 2.10) where there are not only one set of weights $\vec{w}$ and bias b, but n (not necessarily equals to the number of the input neurons m) sets stored in the matrix $W \in \mathbb{R}^n_m$ and vector $\vec{b} \in \mathbb{R}^n$.

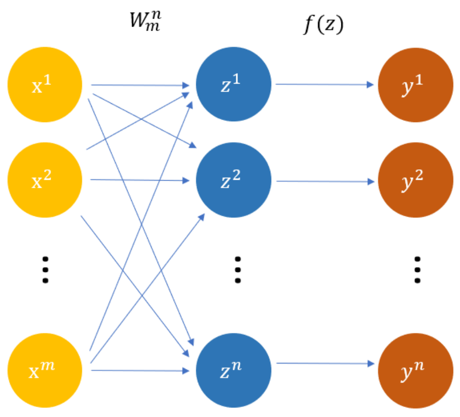


Figure 2.10: Feed-Forward Networks

Now the functions turn out to be:

$$
\begin{aligned}
\mathbb{R}^m &\rightarrow \mathbb{R}^n \\
\vec{x} &\rightarrow \vec{z} = W\vec{x} + \vec{b} \\
\mathbb{R}^n &\rightarrow \mathbb{R}^n \\
\vec{z} &\rightarrow \vec{y} = f(\vec{z})
\end{aligned}
$$

In order to learn complicated tasks, more intermediate layers are often added between input and output. This is finally the most common used structure: Deep Neural Networks (Figure 2.11). The layers in between build up hidden layers.

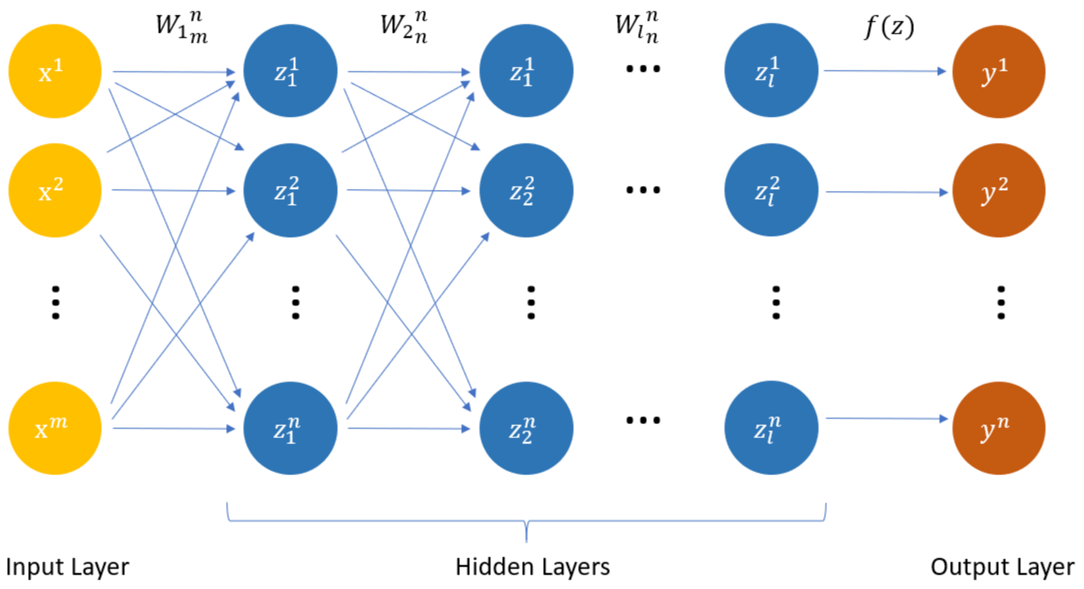


Figure 2.11: Deep Neural Networks with n **units** and l (hidden) **layers**

In this example, all the hidden layers have the same dimension. It is however not necessary in practice. Therefore the weight matrices between each layers can have different shapes decided by the size of its local input and output. Non-linear activation functions can also be added in between.

Training an NN is in fact updating the weight matrices. By slightly changing the values in the matrices, the new output is compared with the target result in terms of loss function in order to decide the update of next step.

2.2.2 Graph Neural Networks

In order to meet the demand of representing data with graph structures, information scientists invented the Graph Neural Networks (GNN) model [34]. In general the model can be represented with a mapping $\tau(\mathbf{G}, n) \in \mathbb{R}^m$ with $\mathbf{G}$ the graph, n one of the nodes building the graph and m the dimension of output vectors. According to the dependency of function τ on node n, GNN can be divided into two categories:

$$\begin{aligned} \textit{Node-focused} &\rightarrow \textit{n-dependent} \\ \textit{Graph-focused} &\rightarrow \textit{n-independent} \end{aligned}$$

In this work, only *graph-focused* GNN is relevant, whose function can be simplified as $\tau(\mathbf{G}) \in \mathbb{R}^m$.

A graph $\mathbf{G}$ has the structure $\mathbf{G} = (\mathbf{N}, \mathbf{E})$, where $\mathbf{N}$ stands for the set of *nodes* and $\mathbf{E}$ stands for the set of *edges*. An *edge* $(u, v) \in \mathbf{E}$ connects two *nodes* $u, v \in \mathbf{N}$. Here the *nodes* u and v can be the same to build a *self-loop*; all the *edges* (u, v) and (v, u) in a graph can be the same to form an *undirected graph* or somewhere distinguishable to form a *directed graph*. In this work, *directed graphs* with *self-loops* are used as basic structure for the NN filter. Each *node* or *edge* is attached with a label $\mathbf{l}_n \in \mathbb{R}^{l_N}$ for *node* n or $\mathbf{l}_{(n_1,n_2)} \in \mathbb{R}^{l_E}$ for *edge* (n_1, n_2)

Take the following graph as an example (Figure 2.12):

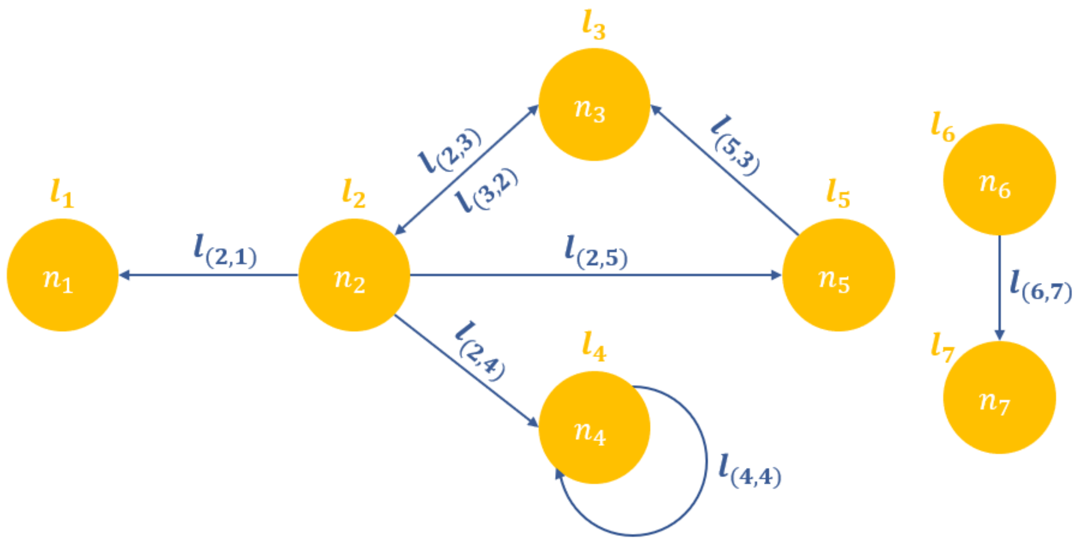


Figure 2.12: Example of a graph

There are overall:

- 7 *nodes*: n_1, n_2, ... , n_7
- 8 *edges*: (n_2, n_1), (n_2, n_3), ... , (n_6, n_7) simplified as $(2, 1)$, $(2, 3)$, ... , $(6, 7)$
- 7+8 *labels*: l_1, l_2, ... , l_7, $l_{(2,1)}$, $l_{(2,3)}$, ... , $l_{(6,7)}$

Note that in this context **labels** are also referred to as **features** in some other literature and in the Python library [3] used in this work, so not to be confused with *labels* as targets in supervised learning.

With graphs well prepared, it is time to consider training processes. A supervised learning framework specified for this work can be written as:

$$\mathcal{L} = \{(\mathbf{G}_i, \mathbf{F}_i, \mathbf{t}_i) \mid \mathbf{G}_i = (\mathbf{N}_i, \mathbf{E}_i);\ \mathbf{F}_i = \mathbf{F}_{\mathbf{N}_i} \in \mathbb{R}^m;\\ \mathbf{t}_i \in \{0,1\},\ 1 < i < p\}$$

where $\mathbf{G}_i = (\mathbf{N}_i, \mathbf{E}_i)$ is one of the p graphs with node $\mathbf{N}_i$ and edge $\mathbf{E}_i$; $\mathbf{F}_i = \mathbf{F}_{\mathbf{N}_i}$ is the corresponding features with m dimensions but only attributing to the nodes; $\mathbf{t}_i$ is the *target*, or *label* in terms of training, for the i-th event. As shown in the example above (Figure 2.12), graphs with separable parts $\{n_1 - n_5, n_6 - n_7\}$ are allowed to form single graph. In other words, all the graphs inside the learning set can be combined into a unique disconnected graph $\mathbf{G}$. Therefore the framework can be packed as:

$$\mathcal{L} = \{(\mathbf{G}, \mathbf{F}, \mathbf{T}) \mid \mathbf{G} = (\mathbf{N}, \mathbf{E});\ \mathbf{F} = \{\mathbf{F}_{\mathbf{N}_i}\} \in \mathbb{R}^{m\times p};\ \mathbf{t} \in \{0,1\}^p\}$$

This encapsulation will be referred to as **batching** during pre-processing (See Chapter 4.1).

In order to make use of GNN, it is necessary to define a method to update the network, or to find the relation between $\mathbf{G}$ and $\mathbf{T}$. One of the simplest graph neutral network models is given by Graph Convolutional Networks (GCN) [28].

The mathematical expression of graph convolution ([3]: GraphConv) with self loop can be written as:

$$h_i^{(l+1)} = \sigma(b^{(l)} + \sum_{j\in\mathcal{N}_i} \frac{1}{c_{ji}} h_j^{(l)} W^{(l)})$$

where $\mathcal{N}_i$ stands for all the neighbouring nodes of the node i, $h_i^{(l)}$ is the value of node i in the l-th step, $W^{(l)}$ is the corresponding weight matrix, c_{ji} is a normalisation factor and σ is the activation function.

The whole network gets upgraded through convolutions at every single node with its surroundings. The similarities of graph structure and convolution mechanism with tree structure of particle decays encourage the investment of GNN in this work.

2.2.3 Attention Mechanisms

The updates in GCN are always symmetric over adjacent nodes but in practice, a more flexible summation can be realised by *Attention Mechanisms*. Already before the suggestion of Graph Attention Networks (GAT) [36], many NN algorithms benefited from this idea. The attention mechanisms have three major properties ensuring the reasonableness of choosing it for tasks like particle decays:

- The algorithm is fast through parallel computation of node neighbor pairs;
- It is robust to the degrees of different nodes by specifying weights to the neighbors;

- The model is tolerant to the shapes of graphs and therefore is able to deal with completely unseen structures

The Attention Mechanism is defined as a mapping $\vec{a} \in \mathbb{R}^{2F'}$ of two neighboring nodes $\vec{h}_i, \vec{h}_j \in \mathbb{R}^F$ on a real score in $\mathbb{R}$ (Figure 2.13 left). The coefficients α_{ij} are defined as:

$$\alpha_{ij} = \text{softmax}_j(\sigma(\vec{a}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]))$$

with a global weight matrix $\mathbf{W} : \mathbb{R}^F \to \mathbb{R}^{F'}$, F' the feature's dimension of outputs, and an activation function σ. The symbol $\|$ is the concatenation operation.

This coefficient α_{ij} can be seen as the attention that has to be payed on the connection between node i and j (or self-loop when $i = j$). Using all of these coefficients, the update rule is defined as:

$$\vec{h}_i^{(l+1)} = \sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W}^{(l)} \vec{h}_j^{(l)}$$

The design of GATs also allows multihead. Naively speaking it is a training of several GCNs at the same time on the same graph. These GCNs are parallelized in different *Attention Heads* (Figure 2.13 right).

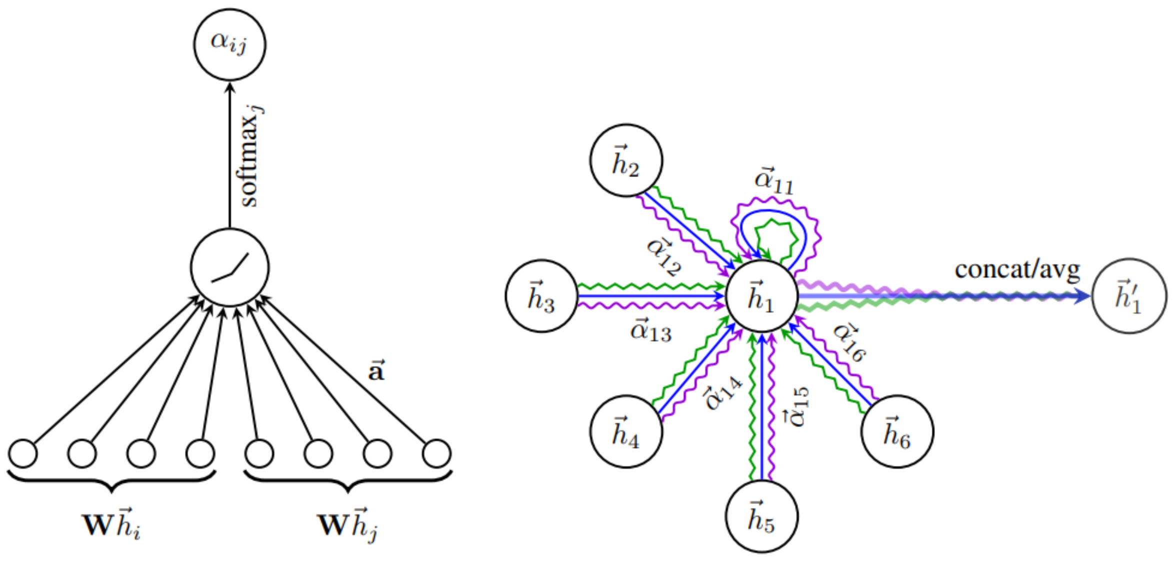


Figure 2.13: **Left**: The Attention Mechanism for GAT; **Right**: Multihead Attention

Studying the evolution of Attention Mechanism will help to understand the propagation of information during the training process. A visualization will be shown in Chapter 4.2.2.

2.3 Decision Trees

As mentioned in Chapter 1.4 and Table 2.1, Decision Trees (DT) are the basic structure of FEI skims and the Cutting-Reweighting Method which is a graphical supervised learning method used for classification and regression. It can learn simple decision rules inferred from the data features to create a tree-structured model that predicts the value of a target.

As shown in Figure 2.14, each node of the decision tree represents a specific area in the input space, and each child of the node breaks that area into one subregion. Repeating divisions, the whole space is subdivided into non overlapping regions represented by leaves. The training of a DT can be considered non-parametric if it is able to learn a tree of arbitrary size.

The diagrams on the right describe how a decision tree works.

Top: Each node of the tree makes the choice between 0 and 1 according to certain criteria attached to this node and passes the information it gets from its parent node to the chosen child node on the left or on the right. Internal nodes representing Input space are drawn as circles, while leaf nodes constructing Output space as squares.

Bottom: The entire 2-D space is divided by the tree into 7 regions. The internal nodes of the tree (circles in both diagrams) are drawn along the dividing lines used to categorize the examples. And leaf nodes are drawn in the center of each region of examples they receive.

The result of the training is then a piecewise-constant function, with one piece per leaf. At least one training sample is required to define each leaf. Therefore the learning ability of a decision tree is strongly restricted by the distribution of training examples.

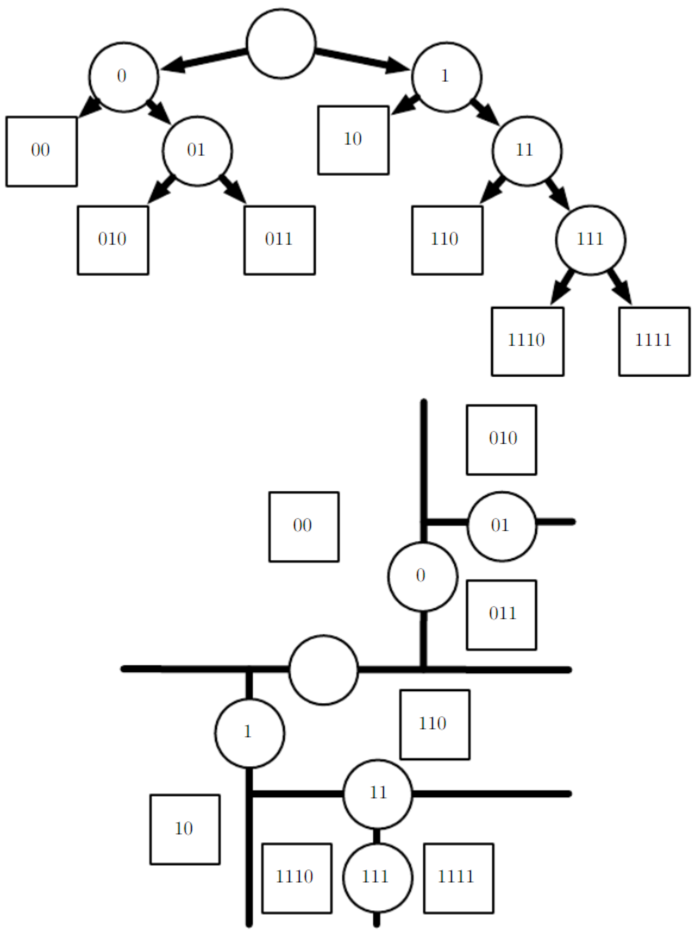


Figure 2.14: An example of Decision Trees. **Top**: Decision Tree; **Bottom**: Space Division [22]

2.3.1 Gradient Boosting Decision Trees

There are usually several ways to optimize an ML model. Decision trees are most powerful in ensembles, referred to as **boosting**. One of the most common methods to train boosted decision trees is Gradient Boosting (GB) which depends on Gradient Descent.

Coming back to DTs, the basic building block of a Gradient Boosting Decision Tree (GBDT) is a *weak learner* h_m: a decision tree regressor with fixed size. The output can be predicted by the sum of such weak learners:

$$\hat{y_i} = F_M(x_i) = \sum_{m=1}^{M} h_m(x_i)$$

with M the number of estimators and given input x_i with label y_i.
One extra estimator can be seen as a small update of the function:

$$F_m(x) = F_{m-1}(x) + h_m(x)$$

and the update h_m should minimize the summed loss L_m:

$$h_m = \arg\min_h L_m = \arg\min_h \sum_{i=1}^{n} l(y_i, F_{m-1}(x_i) + h(x_i))$$

the loss function l can be approximated as:

$$l(y_i, F_{m-1}(x_i) + h_m(x_i)) \approx l(y_i, F_{m-1}(x_i)) + h_m(x_i) \left[\frac{\partial l(y_i, F(x_i))}{\partial F(x_i)}\right]_{F=F_{m-1}}$$

the criteria of h_m can be then simplified as:

$$h_m \approx \arg\min_h \sum_{i=1}^{n} h(x_i) g_i$$

with g_i the gradient $\left[\frac{\partial l(y_i, F(x_i))}{\partial F(x_i)}\right]_{F=F_{m-1}}$. Therefore the gradient descent amounts to fitting the weak learner h_m to match the negative gradient g_i .

2.3.2 Hyperparameters

Apart from the *learning rate* I mentioned above, many other parameters are also playing decisive roles and require careful tuning for an ideal performance of the model. These parameters that are to be decided before the training process are called **Hyperparameters**. In this work, the function *GradientBoostingClassifier* from scikit-learn library [8] is tuned and invested for GBDT. It has the following major hyperparameters:

- **loss**: The loss function to minimize in the training.

- **learning_rate**: Relative contribution of each weak learner
- **n_estimators**: The number of weak learners.
- **criterion**: Chosen among **friedman_mse**, **mse** and **mae**, this parameter instruct the measurement of the quality of a split in each weak learner. It influences the training of each regressor.
- **min_samples_split**: The minimum number of necessary samples to split an internal node.
- **min_samples_leaf**: The minimum number of necessary samples to build a leaf node.
- **max_depth**: The maximum depth of each weak learner.

Some of the parameters can have correlated effects: **learning_rate** vs **n_estimators** as well as **min_samples_split** vs **min_samples_leaf** vs **max_depth**. These relations should be taken care of during the tuning.

Chapter 3

Statistical Tools

Once Neural Network Filters are ready, one has to consider how to make use of them and how to evaluate their performances. In this Chapter the two main directions of **post-processing**: *Sampling Methods* and *Reweighting Methods* will be first introduced. Then some statistical metrics used in tuning, training and evaluation are explained together in one section.

The output of the NN Filter is designed as a real number $p \in [0, 1] \subset \mathbb{R}$, indicating the predicted probability of the corresponding event to pass the later applied **FEI skims**. To make use of these probabilities, two parallel directions are tried out: **Sampling**, where all the events have the chance to be selected by a stochastic process according to their scores and **Reweighting**, where only the events with scores larger than certain threshold will be kept. Chosen events of both methods will be attached with some weights for further analysis. Both methods are aimed to avoid bias caused by false negatives in the filtering process. In contrast to previous approaches that try to mitigate the bias during training [14], with these methods the bias is corrected by the use of event weights.

3.1 Sampling Method

Sampling methods select members from the population to take part in a statistical study or survey. The distribution may be biased if the samples aren't randomly selected and therefore will lower the reliability of the conclusions [5].

In the context of HEP simulations, people are often more curious about the distributions of observables instead of detailed information of individual events. With the help of sampling methods, samples are selected randomly so that their characteristics will form new distributions without the need that every single event is correctly chosen or abandoned. Further more, background events are expected to be rejected as many as possible to save the calculations in the following steps.

There are many different sampling methods suitable for different tasks: **Simple Random Sampling**, **Rejection Sampling**, **Stratified Sampling**, **Importance Sampling**, **Metropolis-Hastings** and so on [31]. I only invested **Importance Sampling** in this work in order to make use of the NN outputs as probability distribution function (PDF). In the context of this paper, **Sampling Methods** merely refers to **Importance Sampling**.

Suppose the following quantity is wanted:

$$\theta = \int_{\mathcal{X}} h(x)\pi(x)dx = E_\pi[h(X)]$$

where $\mathcal{X}$ is the support of the random variable X, $\pi(x)$ the PDF and $h(x)$ the target character to be checked. In a **Simple Random Sampling**, rare events from $\mathcal{X}$-space are equal-possibly selected regardless of their PDF and contribute only a little on the final result θ. This will cause a waste of time and computation in practice. Marshall [32] suggested that the focus should be set on the region(s) of "importance" in order to save computational resources. This becomes particular essential for Monte Carlo computation with high-dimensional models. A sample is generated with the following **Importance Sampling** algorithm:

- Draw $x^{(1)}, ..., x^{(m)}$ from a *trial distribution* $g(\cdot)$.
- Calculate the *importance weight*

$$\omega^{(j)} = \pi(x^{(j)})/g(x^{(j)}), \text{ for } j = 1, ..., m.$$

- Approximate θ by

$$\hat{\theta} = \frac{\omega^{(1)}h(x^{(1)}) + \cdots + \omega^{(m)}h(x^{(m)})}{\omega^{(1)} + \cdots + \omega^{(m)}}$$

In order to make the estimation error $\hat{\theta} - \theta$ small, $g(x)$ has to be as "close" in shape to $\pi(x)h(x)$ as possible. In this work the probability to select a certain event is given by the NN output $p_{NN}(x)$. Consider the total number of events passing the skim
$N = \sum_x p_{skim}(x)p_i(x)$ where $p_i(x)$ is the distribution of all the random observables that the event generation process tries to approximate. Due to the requirement of detector simulation and reconstruction for every given x, the evaluation of $p_{skim}(x)$ is very expensive. Therefore the probability $p_{NN}(x)$ is added in the sampling from $p_i(x)p_{NN}(x)$ instead of real processes through *importance weight*: $\omega(x) = \frac{p_i(x)}{p_i(x)p_{NN}(x)} = \frac{1}{p_{NN}(x)}$.

The following (Figure 3.1) shows the histograms and the corresponding deviation curve of the observable M_{bc}, one of the features attached to each event with *true* label, using the sampling method with which the bias on test data in this work can be avoided by construction.

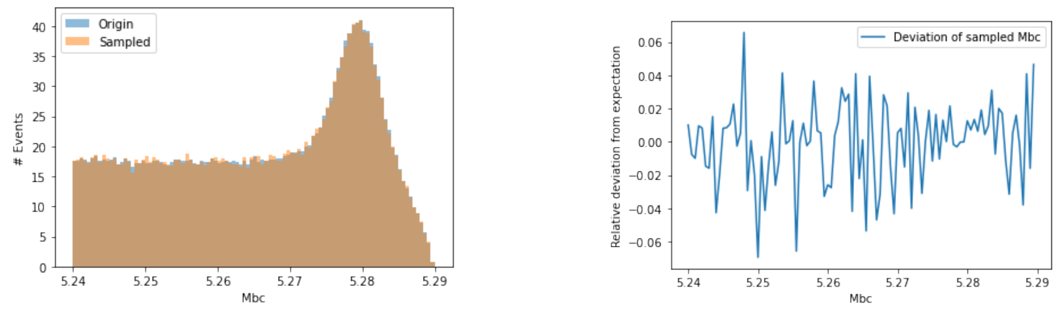


Figure 3.1: Histograms (left) and the deviation (right) of M_{bc} for **Sampling Method**

3.2 Reweighting Method

Another method of post-processing that corrects for the bias from false negative filtering decisions is the **Reweighting Method**. In contrast to the **Sampling Method**, **Reweighting Methods** can be used to correct a potential bias after a hard-cut filtering decision without introducing randomness.

Reweighting is a general procedure, but in particle physics it is usually utilized to modify output of MC simulation to reduce disagreement with real experiment data. This process is also known as **Calibration**.

Take the following histograms as an example.

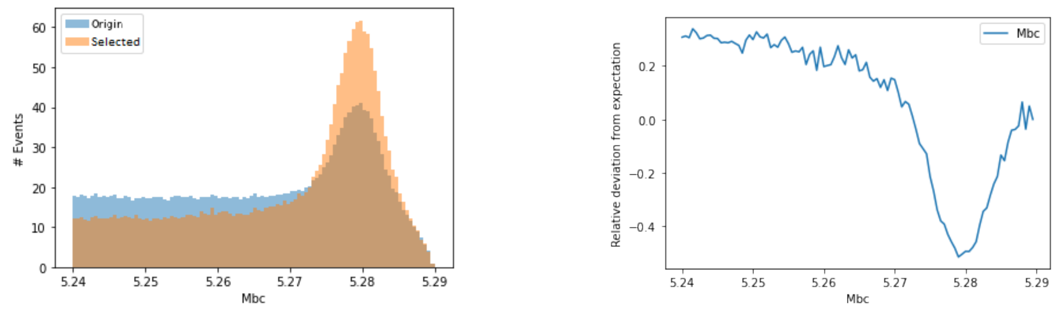


Figure 3.2: Histograms (left) and the deviation (right) of M_{bc} after selection

The left plot shows the histograms of the feature M_{bc}. The histogram for *true* events is colored in blue - *Origin*, while the one for *true-positive* events after a hard-cut selection is colored in orange - *Selected*. Both histograms are normalized and it is clear to recognize a much higher peak as well as a lower plateau in the *Selected* one. The relative deviation of the two histogram can also be seen from the figure on the right. This difference is known as **bias**.

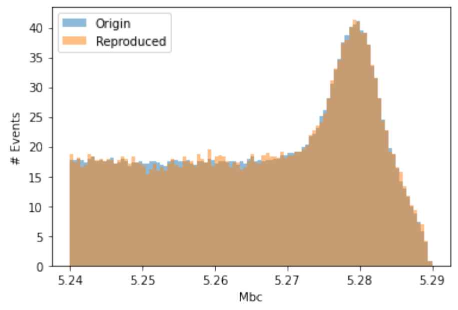
In order to fix this bias, the **Reweighting Method** is introduced. After the selection, each kept event will be attached with a weight w to enhance or decrease its importance. I tried out two methods to calculate the weight w: **GBDT Reweighting** and **Histogram Reweighting**.

- In **GBDT Reweighting**, a GBDT classifier (See Chapter 2.3.1) is trained with selected features of events from the training set to distinguish between **True-Positive** (TP) events and **False-Negative** (FN) events:

True-Positive (TP)	Label=*true*	NN score $\geq$ Threshold
False-Negative (FN)	Label=*true*	NN score $<$ Threshold

Table 3.1: Training of GBDT classifier

Then the events from the test set will be scored by the well trained classifier. The inverse scores will be used as weights, with the classifier output probability $w = \frac{1}{p_{\text{clf}}}$. The performance of this method is shown below. Compared to the *Selected* one (Figure 3.2), the bias is almost eliminated and the fluctuations around 0 are symmetric along the whole range of M_{bc}.



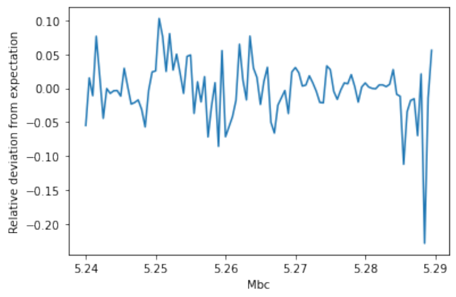


Figure 3.3: Histograms (left) and the deviation (right) of M_{bc} after **GBDT Reweighting**

- In **Histogram Reweighting**, the first step is again the training of GBDT classifier with the above mentioned data. However, the scores of test samples will be counted into histograms. By comparing the score histogram of **Positive** (Labels=*true*) events with the score histogram of **True-Positive** events, a mapping can be found for each bin of the histograms:

$$w_{\text{bin},i} = \frac{h_{\text{P},i}}{h_{\text{TP},i}}$$

Finally each event will be attached with a weight according to the bin where the score of this event lies in:

$$w = w_{\text{bin,arg}_\text{i}\,(p_{\text{clf}} \in \text{bin}_\text{i})}$$

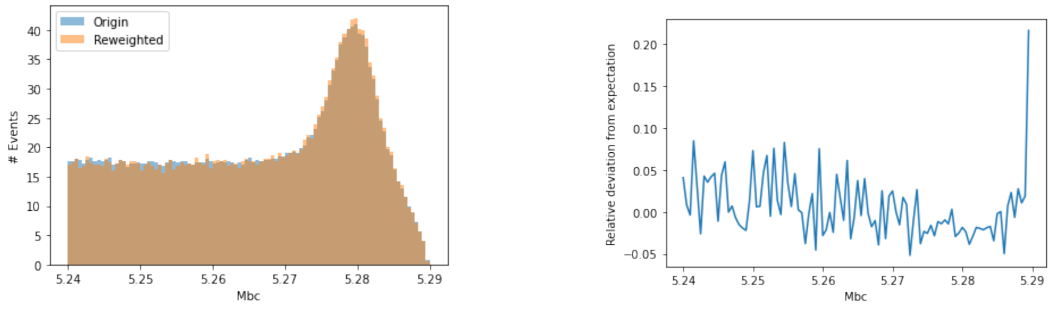


Figure 3.4: Histograms (left) and the deviation (right) of M_{bc} after **Histogram Reweighting**

The performance of **Histogram Reweighting** is shown in the Figure 3.4. The result is satisfying with a relative low deviation, compared to the unreweighted one (Figure 3.2).

To summarize, both the **Histogram Reweighting** (Figure 3.4) and the **GBDT Reweighting** (Figure 3.3) show a good performance, but **GBDT Reweighting** requires less calculation so I choose this for the final analysis. Both of the methods can be further studied and upgraded, e.g. using *quantile* to decide for the **bining** in the **Histogram Reweighting** for a more precise reweighting, and so on.

3.3 Metrics

Several metrics contribute in this project. The **Calibration Curve** is an important measurement for the reweighting and is invested as a criterion in the tuning of GBDT classifier. The **ROC curve and AUC value** illustrate how well a binary classifier performs and are used in both tuning and comparison of neural network models. The **Chi-squared Test** [20] compares two distributions quantitatively to show their similarity and appears in both tuning of the classifier and evaluation of the final weighted distribution of features. The **Cross Entropy** [17] serves as the loss function during the training and evaluation processes. The **Kullback-Leibler Divergence** [29] and the **Kolmogorov-Smirnov Test** [24] are also metrics to evaluate the similarity of two distributions and are used in the final evaluation of bias.

3.3.1 Calibration Curve

A Calibration Curve shows how well the probabilistic prediction of a classifier is calibrated. It is often used in the comparison of different classifiers or different configurations of the same model, thus useful in the tuning of classifiers. The method is generally invested to compare binary classifiers that leave probabilities of incoming events belonging to one of the two categories (skl[8]: calibration_curve).

As a 2-D plot, a calibration curve has an x axis representing the average predicted probability in each bin from the probability space $[0, 1] \subset \mathbb{R}$ and an y axis being the fraction of positives, i.e. the proportion of samples belonging to the class "1" in each bin.

The calibration curve for an ideal classifier is exactly a straight line connecting $(0, 0)$ and $(1, 1)$, or follows the function $y = x$, meaning that for the samples whose predictions are given as x, the ratio of these samples being positive is y with $y = x$. In practice, the closer to the function $y = x$ a curve lies , the better a classifier is calibrated.

Following is an example (Figure 3.5) of two tunings of the **GBDT Classifier** for reweighting task (See Chapter 3.2). The orange line lies nearer to the central curve therefore the corresponding method should be chosen for a better calibration.

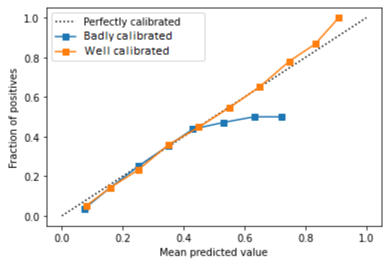


Figure 3.5: Calibration Curves for comparing two methods

3.3.2 ROC Curve and AUC Value

ROC Curve [18] is the abbreviation of receiver operating characteristic curve. It helps to evaluate the quality of a binary classification. The ROC curve is generated by plotting the rate of true positive (TP) $\frac{\mathrm{TP}}{\mathrm{TP+FN}} = \frac{\mathrm{TP}}{\mathrm{P}}$ against the rate of false positive $\frac{\mathrm{FP}}{\mathrm{FP+TN}} = \frac{\mathrm{FP}}{\mathrm{F}}$. With the definition of TP, FP, FN, TN explained in the confusion matrix (Table 3.2).

Prediction / Label	Positive	Negative
Pass	True-Positive (TP)	False-Negative (FN)
Fail	False-Positive (FP)	True-Negative (TN)

Table 3.2: Confusion Matrix

An illustration of ROC curve is shown in figure 3.6. A random classifier which is unable to classify between categories shows a straight line connecting (0,0) and (1,1). A ROC curve of any classifier should lie above this line, the further the better. In the ideal case a perfect classifier will cover the whole triangular area decided by the three points

(0,0), (0,1) and (1,1). To quantify the performance of a classifier with the help of its ROC curve, the area under this curve, also known as AUC value, is measured and used as a criterion. The AUC value ranges from 0.5 to 1. A better classifier also has a higher AUC value.

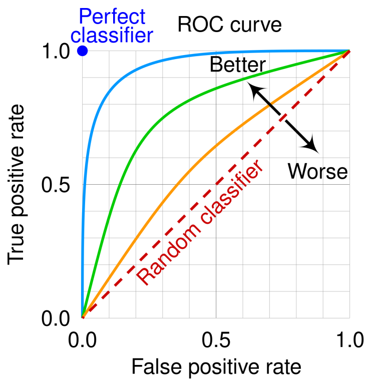


Figure 3.6: The ROC space for a "better" and "worse" classifier.

3.3.3 Chi-squared Test

As a statistical hypothesis test, the chi-squared test [20], or χ^2 test, is valid to check if the statistic follows the **chi-squared distribution** under the null hypothesis. The **chi-squared distribution** with k degrees of freedom is defined as the sum of k independent, standard normal random variables $\{Z_i\}$:

$$\begin{aligned} Q &= \sum_{i=1}^{k} Z_i^2, \\ Z_i &\sim \mathcal{G}(\mu, \sigma^2) \\ \Rightarrow Q &\sim \chi_k^2 \end{aligned}$$

The distribution depends strongly on k and approximates to Gaussian distribution for large k.

A widely used variant of chi-squared test is named after Pearson [21] which can be applied to detect statistically significant differences between the distributions of two categorical datasets. The statistic is calculated as follows:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} = N \sum_{i=1}^{n} \frac{(O_i/N - p_i)^2}{p_i}$$

with O_i and E_i the number of observed and expected samples in each category or bin and p_i the ratio between E_i and N, n is the number of categories or bins and N is the total number of the expected samples. This calculation is used in this work to compare the

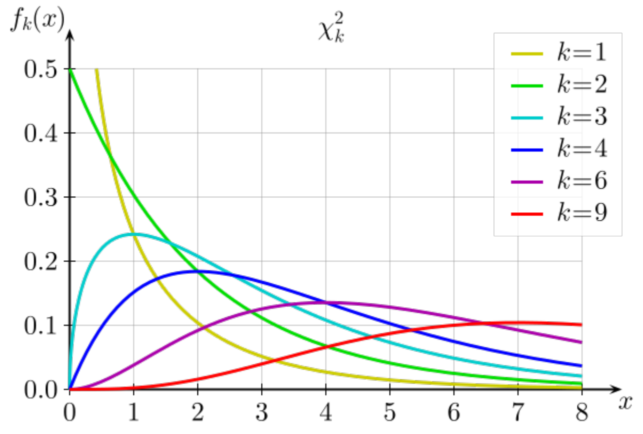


Figure 3.7: Probability density function of Chi-square distribution

histogram of weighted features of chosen samples with the histogram of the features from the events with *True* labels.

Another variant, the reduced chi-squared statistic is also invested which is suited for the sampling method through consideration of the statistical uncertainty. It requires the variance σ_i^2 instead of the target E_i in the denominator:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{\sigma_i^2}$$

The variance σ_i^2 is acquired by the histogram of features with squared weights.

3.3.4 Cross Entropy

The cross entropy [17] between two probability distributions based on the same set of events shows the average number of bits needed in the identification of an event from the set through a function generated from the observed dataset, instead of the expected distribution.

The cross entropy for discrete probability distributions p and q with the same support $\mathcal{X}$ is:

$$H(p,q) = -\sum_{x\in\mathcal{X}} p(x)\log q(x)$$

In this work, the binary cross entropy function provided by Pytorch [7] is used. For a binary classification problem, the cross entropy is given by:

$$l(x,y) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log x_i + (1-y_i)\log(1-x_i))$$

with N the number of samples, y the labels or expectations and x the predictions.

3.3.5 Kullback-Leibler Divergence

The Kullback-Leibler divergence, D_{KL}, also called relative entropy, measures the deviation of one probability distribution from another one. The fomular is very similar to the entropy:

$$D_{\mathrm{KL}}(P\|Q) = -\sum_{x\in\mathcal{X}} P(x)\log\left(\frac{P(x)}{Q(x)}\right)$$

The entropy function from SciPy [9] is used in this work, which has only a different prefactor than the original definition:

$$S(p,q) = \sum_{i=1}^{N} p_i \log\left(\frac{p_i}{q_i}\right)$$

With N the number of events, q the target or expected probability distribution and q the calculated or predicted probability distribution.

3.3.6 Kolmogorov-Smirnov Test

Different from the metrics above, the Kolmogorov-Smirnov Test (KS test) [24] considers the cumulative distribution instead of the probability distribution. It can quantify a distance between the empirical distribution of the sample and the cumulative distribution of the reference distribution, or between the cumulated distributions of two sets of samples.

In order to compare two discrete one-dimensional probability distributions, the KS statistic is:

$$D_{n,m} = \sup_x |F_{1,n}(x) - F_{2,m}(x)|$$

where $F_{1,n}$ and $F_{2,m}$ are the two empirical cumulative distribution functions of the first and second sample, sup is the supremum function and $n,\ m$ are the sizes of the two samples.

To meet the demand of comparing weighted distributions, a varied version with the help of statistical uncertainty is used in the work:

$$\begin{aligned} D_{n,m} &= \sup_x |F_{1,n}(x)W_{1,n}(x) - F_{2,m}(x)W_{2,m}(x)|; \\ N_{\mathrm{reduced}} &= \frac{N_{1,\mathrm{eff}} N_{2,\mathrm{eff}}}{N_{1,\mathrm{eff}} + N_{2,\mathrm{eff}}} \end{aligned}$$

with the two cumulated weight distributions $W_{1,n}$ and $W_{2,m}$ of ordered weights w_1 and w_2, $N_{1,\mathrm{eff}} = \frac{(\sum_{i=1}^{n} w_{1,i})^2}{\sum_{i=1}^{n} w_{1,i}^2}$ and $N_{2,\mathrm{eff}} = \frac{(\sum_{i=1}^{m} w_{2,i})^2}{\sum_{i=1}^{m} w_{2,i}^2}$ the effective sizes of the two sets of samples. With the help of the general reduced sample size N_{reduced}, **p-value**s can be found, which is the probability of obtaining predicted results as least extreme as the results actually observed, under the assumption of the null hypothesis (Figure 3.8). The **p-value** can be calculated by using an asymptotic form of the distribution of the KS statistic [24]. The null hypothesis in this case means that the two samples originate from the same distribution.

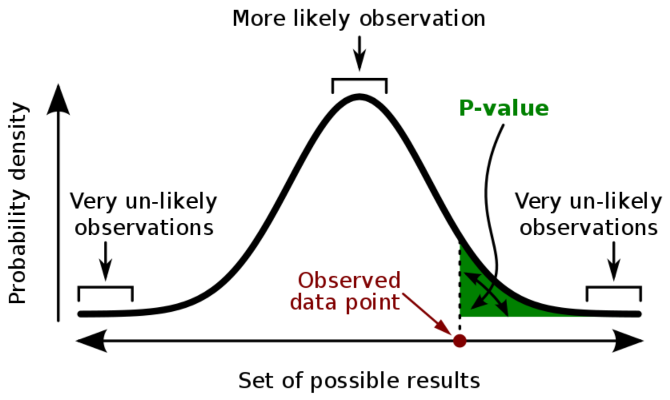


Figure 3.8: **p-value** is the area colored in green

Chapter 4

Neural network architecture optimization

From this chapter I will start to introduce the experimental procedure in this study. Starting with **Pre-processing**, operations including the manufacture of graphs from samples, batching and one unsuccessful trying of clustering will come in the first part. Next follows the main part: **Neural Network Filter**, where I explain the design, visualisation, tuning and comparing periods of the neural network model used in this work.

4.1 Pre-processing

The purpose of this study is to improve selective background Monte Carlo simulation with Graph Attention Networks and weighted events. Therefore datasets to be suitable for both NN simulations and weighting methods, or post-processing, are required. The starting point of this work is a dataset based on the FEI skim following the previous studies [25][14]. Several features - **Generated Variables** attached to each event are chosen to build graphs contributing to the training and evaluation processes, while some other features - **Physics Observables** are selected for post-processing. Considering the different graph structures of different decay events, I also tried to cluster the graphs and treat them with separate models. This attempt did not improve results but it showed on the other hand that the capacity of the GAT structure is high enough to deal with different decays and the method is therefore expected to generalize on other data sets from other skims.

4.1.1 Dataset

The FEI skims (See Chapter 1.4) was chosen in the previous works as the study object mainly because of its high enough retention rate (about 5.86%) ensuring enough data for classification tasks. The data is shared on the Belle II grid and the selected skim is study independent, meaning that the achievement of these works can be directly used by others. Further more, the availability of reconstructed B mesons in the FEI skim helps to detect bias by allowing a statistical comparison of true positives and false negatives.

The dataset is constructed from simulated $\Upsilon(4S) \rightarrow B^0\bar{B}^0$ events where the skim selection corresponds to the hadronic B^0 FEI, containing roughly 7.5×10^5 events in total. The samples that are able or unable to pass this skim are signed with *True* or *False* respectively. The numbers of kept and discarded events are controlled to be equal and sum up to about 1.5×10^6 events. During this study, 9×10^5 samples are randomly chosen to build **training set**, 1×10^5 and 5×10^5 for **validation set** and **test set**, respectively. **Training set** and **validation set** are used in the learning process of neural network models as well as GBDT classifiers, while the **test set** only appears in the final evaluation of different performances including the ability of classification, speedup and features weighting.

The original information generated by Monte Carlo has a rooted tree structure with each node representing a single particle and the mother particle index on each node the connection between particles. The information of each particle is carried on each node as shown in the example (Figure 4.1).

A number of generator-level features of each particle are selected for the building of data sets. According to their usages they can be divided into **Generated Variables** and **Physics Observables**.

The **Generated Variables** take part in the construction of the input for the graph neural network structures, extracting the information to distinguish between background and survived decays. These features will be collected over nodes from graphs to calculate global features which will be decisive in the final classification. Following are the **Generated Variables**:

- **PDG ID**: Officially called PDG particle numbering scheme published by the Particle Data Group [23]. Each type of particle, including all known elementary particles (electrons, W bosons, . . .), composite particles (mesons, baryons, . . .), atomic nuclei and also hypothetical particles beyond the Standard Model, has a unique code (See the second column of Figure 4.1). Particles and antiparticles are assigned as positive and negative separately. The range of the original PDG ID is 1 to $\pm$ 1000020040, which is too large for efficient one-hot coding. Therefore they are tokenized to match integers from 1 to the number of different PDG IDs. The tokenized values will be fed into each **Node** to represent the corresponding particle type.

- **Mother array index**: The array index of the mother particles used to build adjacency matrix representing the graph.

- **Production Time**: The time interval from the initial $\Upsilon(4S)$ generation to the production of each particle in ns.

- **Energy**: The energy of each particle in GeV.

```
 1   300553 (Upsilon(4S))    E: 1.100e+01 m: 1.058e+01 p:(...) v:(...)
 2       521 (B+)            E: 5.511e+00 m: 5.279e+00 p:(...) v:(...)
 4        -421 (anti-D0)     E: 1.939e+00 m: 1.865e+00 p:(...) v:(...)
10           310 (K_S0)      E: 5.913e-01 m: 4.976e-01 p:(...) v:(...)
20             211 (pi+)     E: 2.379e-01 m: 1.396e-01 p:(...) v:(...)
21            -211 (pi-)     E: 3.534e-01 m: 1.396e-01 p:(...) v:(...)
11           211 (pi+)       E: 3.484e-01 m: 1.396e-01 p:(...) v:(...)
12          -211 (pi-)       E: 7.756e-01 m: 1.396e-01 p:(...) v:(...)
13           111 (pi0)       E: 2.234e-01 m: 1.350e-01 p:(...) v:(...)
26              22 (gamma)   E: 1.660e-01 m: 0.000e+00 p:(...) v:(...)
27              22 (gamma)   E: 5.742e-02 m: 0.000e+00 p:(...) v:(...)
 5         413 (D*+)         E: 2.453e+00 m: 2.010e+00 p:(...) v:(...)
14           411 (D+)        E: 2.239e+00 m: 1.870e+00 p:(...) v:(...)
28            -321 (K-)      E: 8.801e-01 m: 4.937e-01 p:(...) v:(...)
29             211 (pi+)     E: 5.853e-01 m: 1.396e-01 p:(...) v:(...)
30             211 (pi+)     E: 7.737e-01 m: 1.396e-01 p:(...) v:(...)
15            22 (gamma)     E: 2.138e-01 m: 0.000e+00 p:(...) v:(...)
 6         313 (K*0)         E: 1.111e+00 m: 8.617e-01 p:(...) v:(...)
16           321 (K+)        E: 8.508e-01 m: 4.937e-01 p:(...) v:(...)
17          -211 (pi-)       E: 2.683e-01 m: 1.396e-01 p:(...) v:(...)
 3      -521 (B-)            E: 5.493e+00 m: 5.279e+00 p:(...) v:(...)
 7          43 (Xu0)         E: 2.052e+00 m: 8.646e-01 p:(...) v:(...)
18           211 (pi+)       E: 4.797e-01 m: 1.396e-01 p:(...) v:(...)
19          -211 (pi-)       E: 1.572e+00 m: 1.396e-01 p:(...) v:(...)
 8          11 (e-)          E: 1.621e+00 m: 5.110e-04 p:(...) v:(...)
 9         -12 (anti-nu_e)   E: 1.820e+00 m: 0.000e+00 p:(...) v:(...)
```

Figure 4.1: Example of a Υ(4S) decay structure within an event, as represented by the Belle II software. The first two columns represent the array index and particle information (PDG code and human readable name) respectively. The indentation level of the second column corresponds to the depth of the particle within the decay, with all particles originating from the initial Υ(4S). The remaining columns represent the (non-exhaustive) array of data for each particle, just as a visual demonstration of the data structure [25]

- **Momentum**: The three-dimensional momentum p_x, p_y, p_z of each particle in GeV/c.
- **Position**: The three-dimensional coordinate x, y, z of each particle in m. Particles with one of the coordinates larger than 10 m are discarded because they are generated beyond the detector and provide no information.

The **Physics Observables** on the other hand are not used in the training of the neural networks even though they are also attached to each event. These features are used to evaluate the kept events after NNs in terms of several dimensions that are vital in the further analysis. The selection of the following 14 **physics observables** depends on the previous study [14] where those features showed the strongest bias among 29 features.

- **M**$_{\text{bc}}$: The beam-constrained mass of the reconstructed B meson (See Chapter 1.5).

- $\mathbf{E}^{\text{CMS}}_{\text{visible}}$: The missing energy of the event in the center-of-mass system.
- $\mathbf{R}_2$: Ratio of the 2^{nd} to the 0^{th} order Fox Wolfram moments. [19]
- $\mathbf{R}_4$: Ratio of the 4^{th} to the 0^{th} order Fox Wolfram moments. [19]
- **No. tracks**: Number of tracks in the event.
- **No. MC Particles**: Number of generator-level particles in the event. This quantity is not a physics observable, but still included in this list since it is not used to train the NN filter but only used in the evaluation of bias and reweighting.
- $\mathbf{\Delta E}$: The reconstructed energy difference (See Chapter 1.5)
- $\mathbf{E}_{\text{photon}}$: The energy of all the photons in the event combined in the laboratory system.
- $\mathbf{M}^2_{\text{miss}}$: The missing mass of the event squared.
- $\mathbf{E}^{\text{CMS}}_{\text{miss}}$: The missing energy of the event in the center-of-mass system.
- **sphericity**: The event sphericity is defined as the linear combination of the two smallest eigenvalues of the sphericity tensor: $S = \frac{3}{2}(\lambda_2 + \lambda_3)$.
- **thrust**: The thrust reflects the shape of the events. In general it ranges from 0.5 to 1, with the lower limit corresponding to the spherically symmetric events and upper limit indicating the two back-to-back jets.
- **aplanarity**: The event aplanarity is defined as $\frac{3}{2}$ of the third sphericity eigenvalue.
- $\mathbf{P}_{\text{backward Hemisphere}}$: Total momentum of the particles flying in the direction opposite to the thrust axis.

Again according to the performance shown in this project, the first 8 features above are chosen to derive weights in the Reweighting method to minimize the bias, including $\mathbf{M}_{\text{bc}}$, $\mathbf{E}^{\text{CMS}}_{\text{visible}}$, $\mathbf{R}_2$, $\mathbf{R}_4$, **No. tracks**, **No. MC Particles**, $\mathbf{\Delta E}$ and $\mathbf{E}_{\text{photon}}$.

4.1.2 Graphs and Batches

With the help of the **mother array index** carried by each particle, graphs are built to represent decay events. In this work all the decays are expected to be generated by $\Upsilon(4S) \rightarrow B^0\bar{B}^0$ (Figure 4.2). Each node in the graph contains all the information of the corresponding particle including its preprocessed **generated variables** and **physics observables**. One decay event can be completely represented by one graph and is matched with one boolean label telling whether this decay can pass the FEI skim (See Chapter 1.4).

After the construction of the graphs, all the events and their corresponding labels are shuffeled and subpacked into batches to ensure a better randomness and efficiency in the upcoming training, validation and test processes.

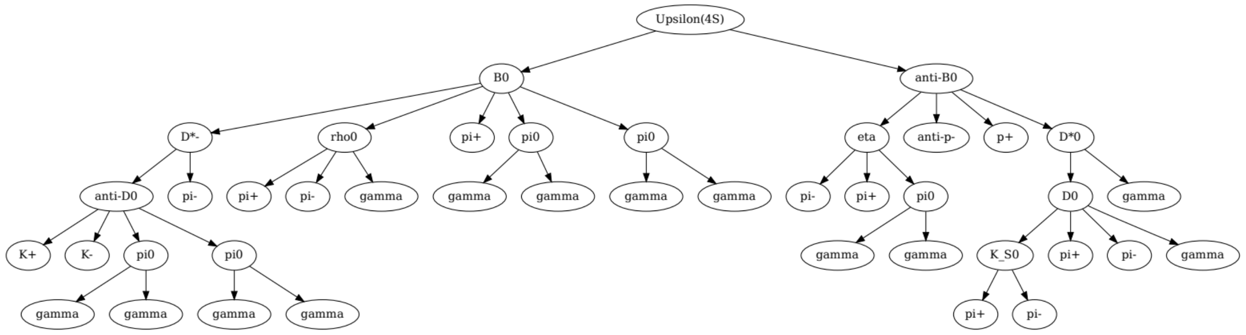


Figure 4.2: An example of $\Upsilon(4S) \to B^0\bar{B}^0$ decay

4.1.3 Clustering

Even though all the decay events come from $\Upsilon(4S) \to B^0\bar{B}^0$, the shapes of their decay trees can be different from each other. The idea of this method is to cluster the events with similar decay trees in order to train a different neural network corresponding to each cluster parallelly. The scores of the test events are then determined by the neural network that was trained for the corresponding category.

depth	particle(s)	n_d	$d \cdot n_d$
0	$\Upsilon(4S)$	1	0
1	B^0, $\bar{B}^0$	2	2
2	D^{*-}, ρ^0, …, D^{*0}	9	18
3	$\bar{D}^0$, π^-, …, γ	14	42
4	K^+, K^-, …, γ	10	40
5	γ, γ, …, π^-	6	30

Table 4.1: Calculation of the shape vector for the graph from figure 4.2

The first step is to abstract the "shape" out of a graph. Here I define a vector $\vec{r} \in \mathbb{N}^{D+1}$ with $\vec{r} = \bigoplus_{d=0}^{D} d \cdot n_d$ where d is the depth, or the distance of a particle to the root of the tree, D is the max depth and n_d is the total number of particles at depth d. Take the graph in figure 4.2 as an example, the calculation is shown in the table 4.1. The shape of this graph is then represented by the vector $\vec{r} = (0, 2, 18, 42, 40, 30)$. The first two components of each vector are discarded as they are always $(0, 2)$. All the vectors are padded with zeros into a space that is large enough for all the graphs, e.g. $\vec{r}' = (18, 42, 40, 30, 0, \ldots, 0)$.

Next, the clusters are decided using the **KMeans** ([8]:cluster.KMeans) method on the shape vectors. It aims to find k centroids in the shape vector space so that the sum of all the distances from each shape vector to its nearest centroid is minimized. To assign clusters it will tell, to which centroid a vector lies closest, in other words, to which cluster a vector belongs. After some tunings the number of clusters k is set to be 5. It is hard to visualize the result of high dimensional clustering, so I print the average of the one dimensional projection, or the mean $r =< \vec{r'} > \in \mathbb{R}$, in each cluster for visualization (Figure 4.3 left). And also show the number of events in each cluster from each dataset to ensure that there is a sufficient number of training events for each category (Figure 4.3 right).

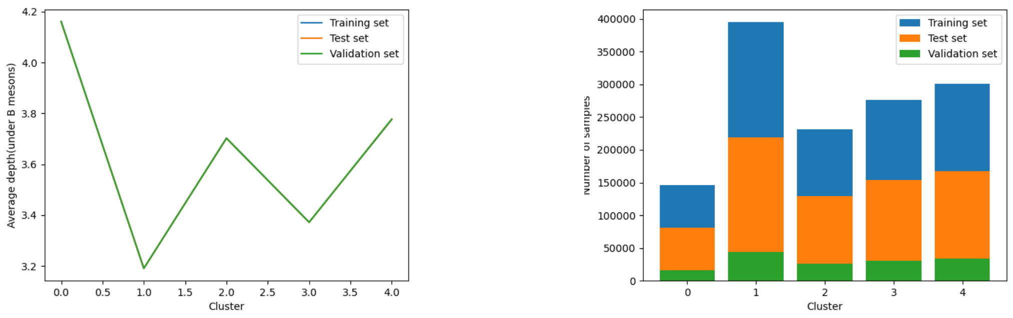


Figure 4.3: (Left) Average of 1-D projection of shape vectors in each cluster. (Right) Number of samples in each cluster from each dataset.

Finally, the events in each cluster are treated separately to tune a unique neural network filter. In this work only the number of layers used for GAT module (See Chapter 4.2) is tuned in this step. According to their loss, the best number of layers are selected as:

Cluster	Best number of layers
0	10
1	10
2	10
3	8
4	12

Table 4.2: Separate tuning for each cluster

This result shows that there is indeed a difference of the best neural network structure regarding of different graph shapes. The difference is however very small when I check the concrete loss of different configurations. The largest down side of this method is that the performances of all the neural networks on their corresponding clusters are worse than the performance of only one general neural network trained on the whole data set. Therefore the clustering method is finally abandoned but it shows that the current neural network structure is strong enough to deal with the decays that have different shapes.

4.2 Neural Network Filter

In this chapter I will first introduce the structures and hyperparameters of the neural network filters used in this work. The filters are divided into several parts and then compared piece after piece. Starting from Graph Convolutional Network which is similar to what was used in the previous study [14], the Graph Attention mechanism is then added as improvement. Finally the training of node features and global features are generalized in a customized module to further enlarge the capacity. Next, I will show the visualisation of attention weights which can help to understand the learning process of the graph neural network structure and the graph attention mechanism. Finally after some tuning processes, the performances of different structures are compared by loss, accuracy and AUC values. The best model with the best configuration is then decided and ready for further analysis.

4.2.1 Architecture and Hyperparameters

There are overall four networks being compared in this work. All of them are based on graph neural networks (See Chapter 2.2.2). A general view of all the structures is shown in the table 4.3.

Name	Kernel	Training of Node/Global Features	Pooling
GCN(sep)	GraphConv	Separated	Global Average Pooling
GAT(sep)	GATConv	Separated	Global Average Pooling
GAT(gen)	GATModule	Generalized	Global Average Pooling
GATGAP(gen)	GATModule	Generalized	Global Attention Pooling

Table 4.3: Overview of all the GNN structures in this work.

Every neural network starts from an input structure. In this work, two slightly different settings are designed for separated and generalized cases (Figure 4.4). The part before concatenation are the same, which corresponds to the preprocessing explained in the section 4.1. After concatenation the **Generated Variables** of each node are used as node features. In the separated training, these node features will be transformed by three Dense layers, also called a Deep Neural Network structure with 3 hidden layers (See Chapter 2.2.1), which are expected to extract the information of each particle. In the generalized training, the initial node features take part in only the first run of the kernel. The kernel will output a set of upgraded node features after each round, who will then take part in the next run instead of the initial one. On the other hand, the graph represented by the adjacency matrix provides the information on how to connect to each other for each particle. Together with the node features attached to each particle, all the data that is expected to be able to characterize the whole decay event is passed to the kernel.

The kernel should learn most of the information and be the most important part in each neural network. This part is repeated several times, controlled by the hyperparameter: *the number of layers*, in one network. For separated training, the kernels are simply compact structures provided by the Deep Graph Library [3]. But for generalized training, it is customized to be able to update node features and global features parallely in each run.

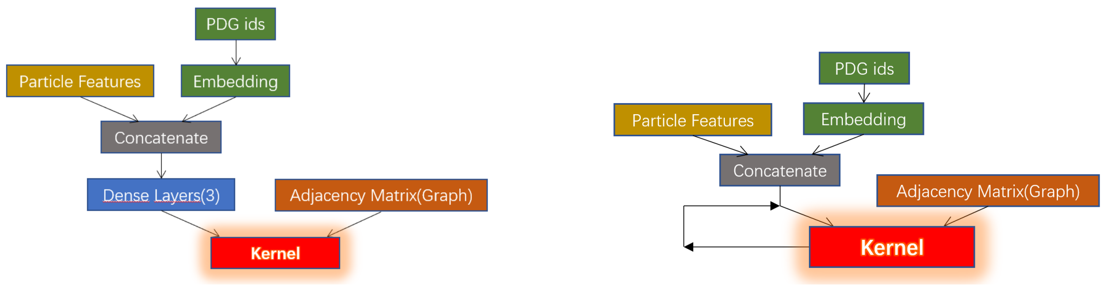


Figure 4.4: Input structure for separated training (left) and generalized training (right).

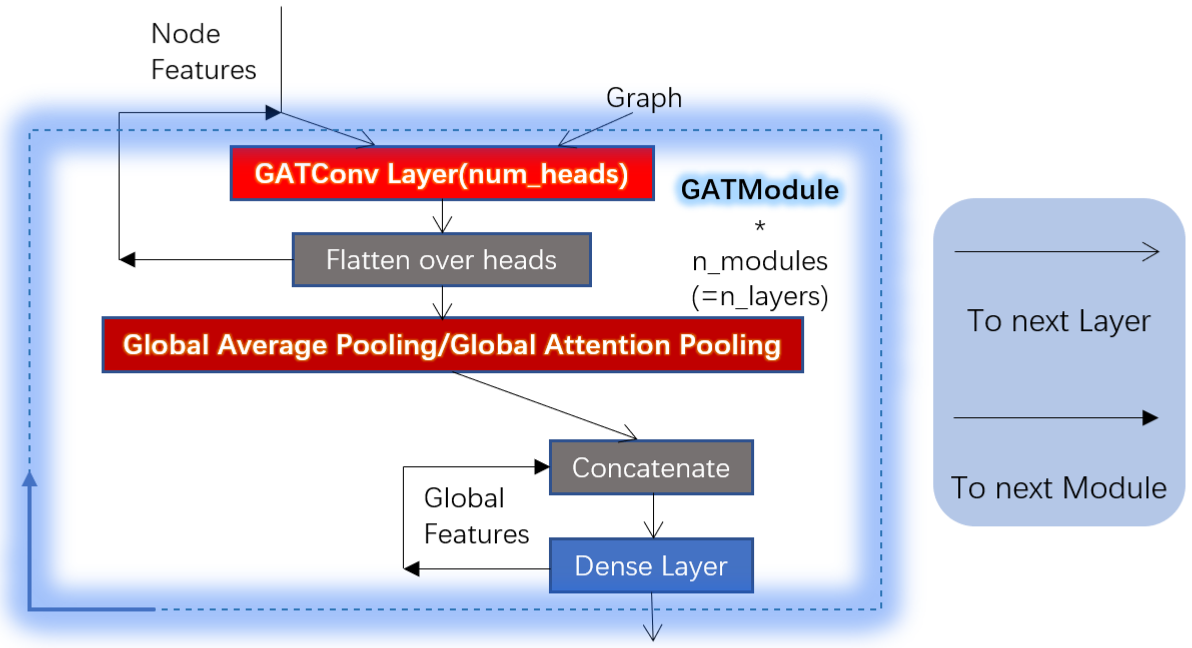


Figure 4.5: GATModule

The kernels for GCN(sep) and GAT(sep) are introduced in Chapter 2.2.2 (Graph Convolutional Networks) and Chapter 2.2.3 (Graph Attention Networks). The customized one for generalized methods bases on Graph Attention Networks as well. The main difference of GATModule (Figure 4.5) to the first two kernels is the integration of the training of node/global features, allowing a higher flexibility and capacity. After Graph Attention (GATConv Layer in the picture), the node features on each node are updated. On the one hand they are used as the new input node features in the next run of GATModule, on the other hand they are sent into the pooling layer, who will collect the features from all the nodes in the graph and form a set of global features that are graph structure independent, meaning that for all kinds of graphs the shape of global features stays the same. However, the rule to sum over all the nodes can be graph structure dependent, allowed by using Global Attention Pooling instead of Global Average Pooling. Similar to Graph Attention, Global Attention Pooling is also able to learn the weight distribution itself and treats the nodes inside a graph unequally. The fresh global features will be concatenated with the old one, initialized as 0, from last run and mapped together by a Dense layer to form a set of new global features as an update for the next run or as the final output after the last run.

Finally all the information is summarized to give a score $p \in \mathbb{R}$ (Figure 4.6). For separated methods the data is still stored in each node after the kernel, therefore a Global Average Pooling serves for the naive collection all over the graph and together with a Dense layer to form a set of global features. One more averaging over heads is extra added for GAT(sep) model to fit for the multi-attention-head structure given by GATConv. The output of GATModule is already the well trained global features, so it needs no further manufactures. In the last layer of each model, the global features is mapped to only one value, known as the neural network output or the score.

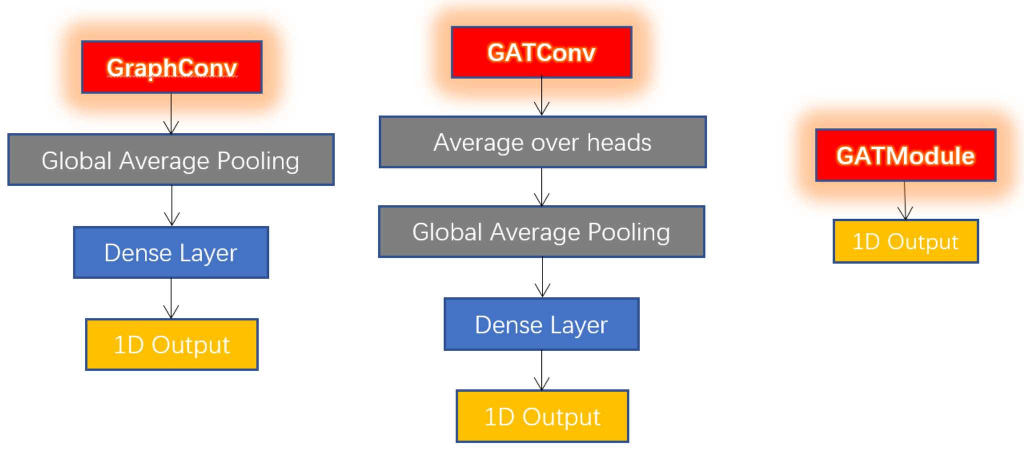


Figure 4.6: Output structure for GCN(sep) (left), GAT(sep) (middle) and GAT(GAP)(gen) (right).

The complete structures of models with separated training are shown in figure 4.7 and of models with generalized method are in figure 4.8. Before comparing the performances of different models, the best configuration of hyperparameters has to be decided, which is known as the tuning process.

The hyperparameters for each model are in general the same, except for GCN(sep) where *the number of heads* is invalid.

- *The number of heads*: Parameter for Graph Attention Network to decide the number attention heads. Visualisations in section 4.2.2 will show that a large number of heads is unnecessary.

- *The number of layers/modules*: The repeated time of the kernel in each model. For separated trainings it is the number of graph convolutional/attention layers while for generalized case it refers to the number of the whole GAT Module.

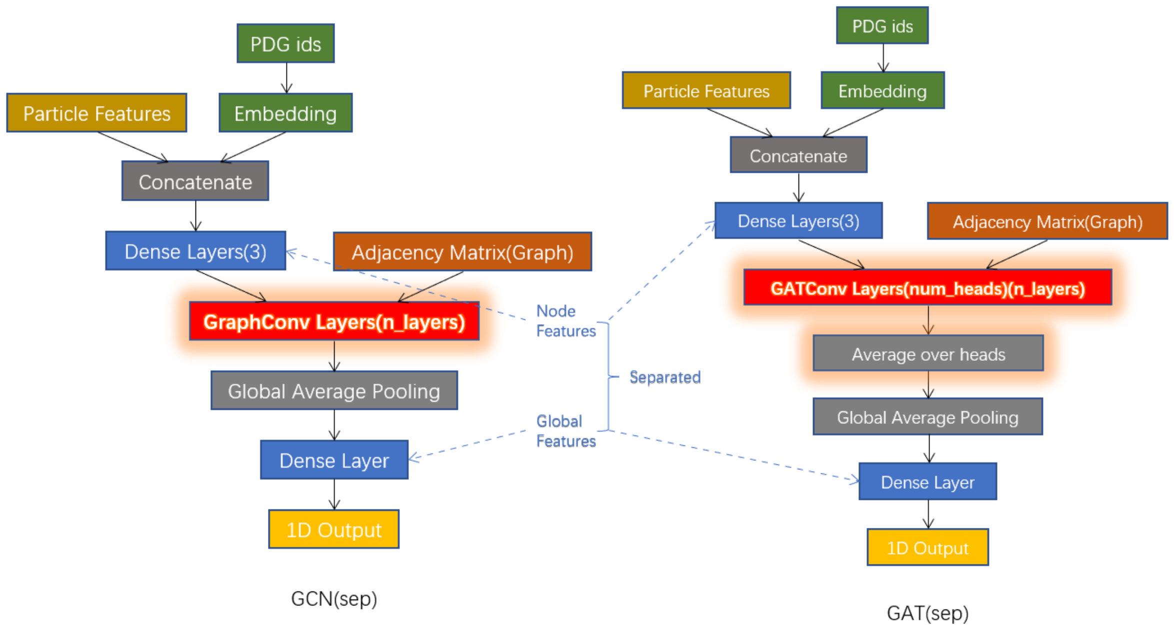


Figure 4.7: Models with separated method: GCN(sep) (left) and GAT(sep) (right).

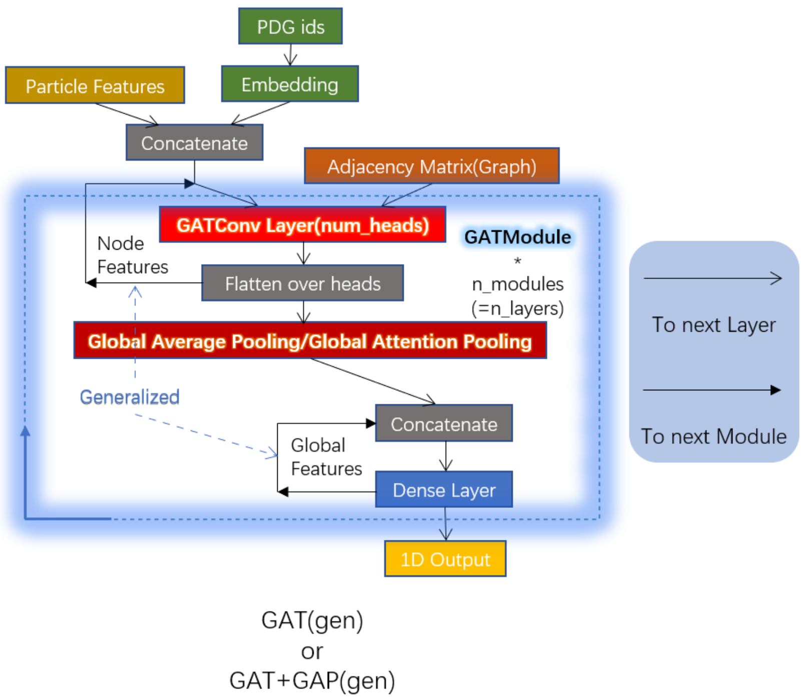


Figure 4.8: Models with generalized method: GAT(gen) with Global Average Pooling and GATGAP(gen) with Global Attention Pooling.

- *The number of units*: Parameter for graph convolutional/attention layers as the output size. In each model it defines the size of global features.
- *Batchsize*: The number of events used in each step of update during the training. A large batchsize will reduce the training time for one epoch but takes the risk of slow convergence due to a low number of updates in each epoch.

In the tuning in section 4.2.3, I will study the influence of *the number of heads* and *the number of layers/modules* on loss, accuracy and AUC values, as well as the trade off between *the number of units* and *batchsize* considering AUC values, training times and memory requirements.

4.2.2 Visualisation of Attention Weights

Before tuning, the training of graph attention layers is monitored in order to visualise the distribution and evolution of attention weights. All the nodes and edges in the graph are colored with different shades mapped to its weight. A darker color means a higher weight, indicating that the corresponding edge or node is given high attention in that step. Figure 4.9 is an example of visualisation.

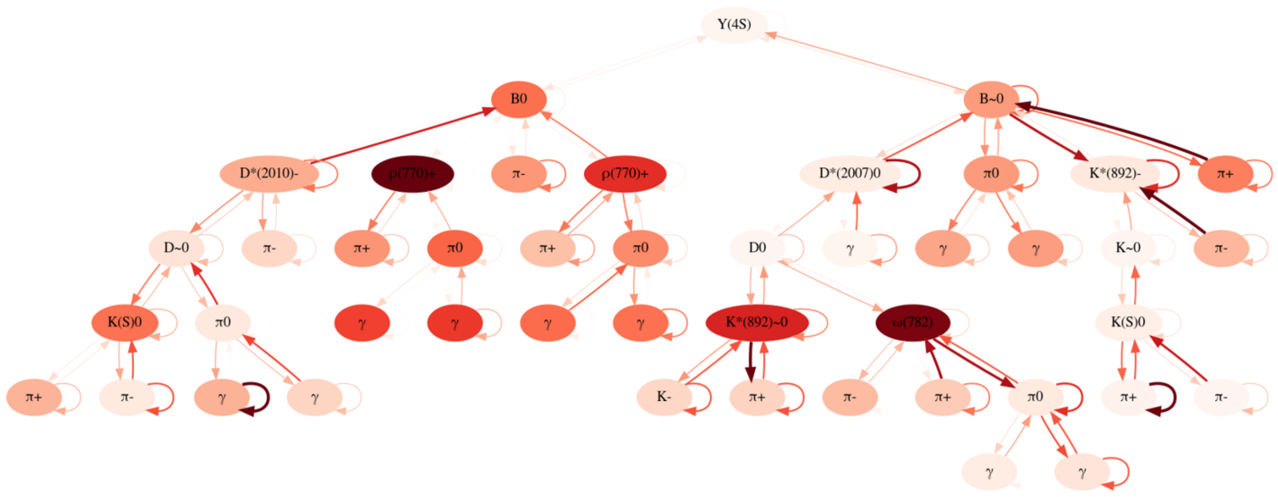


Figure 4.9: Visualisation of Attention Weights of one graph

The most important nodes in this step are ρ^+ and ω according to their color in dark red. Some important directed connections including self-loops can also be recognized. However, it is hard to interpret further information from this single graph. Hence the statuses of the same graph in different layers/modules and in different heads are listed together in figure 4.10. In this example, *the number of layers/modules* is 4 and *the number of heads* is 8. Vertically, the differences among layers can be noticed, meaning that in different runs the attention of the network is changing as well. But horizontally, the distributions of the graph attention from the same layer in different heads are almost the same, indicating that the multi-head-attention is not making much difference. This conclusion helps me to merely concentrate on small *number of heads* in the following tunings in order to save time and computing resources.

I also tried to study the evolution of the attention over the layers, aiming to analogise the dynamic of attention distributions to the propagation of information so that the design of the neural network model can be further improved through deeper understanding. However this idea is still at the stage of envisioning.

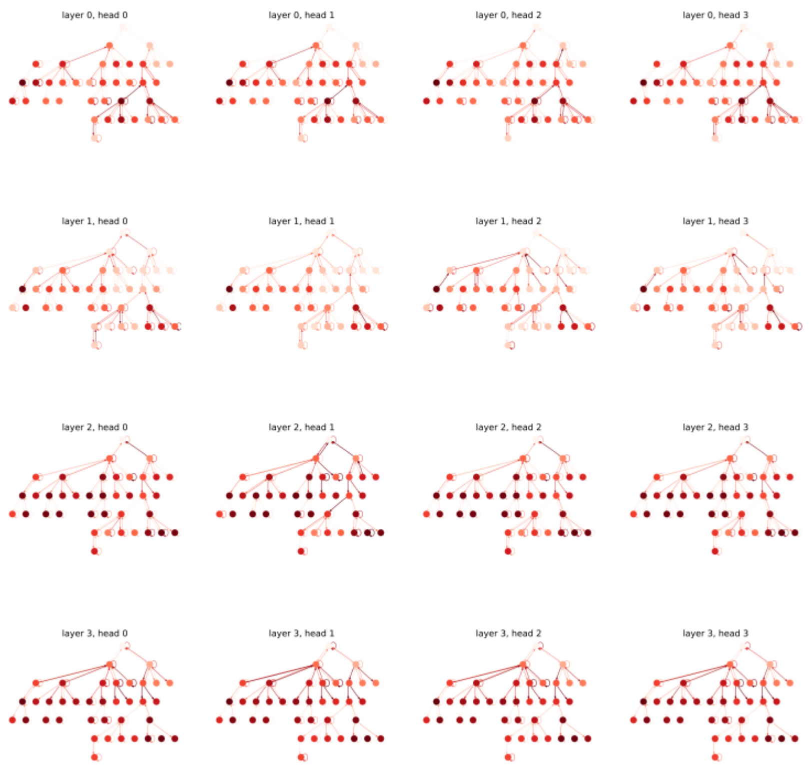
4.2.3 Tuning and Comparison

The total tuning process in search of the best model in this work consists of three branches:

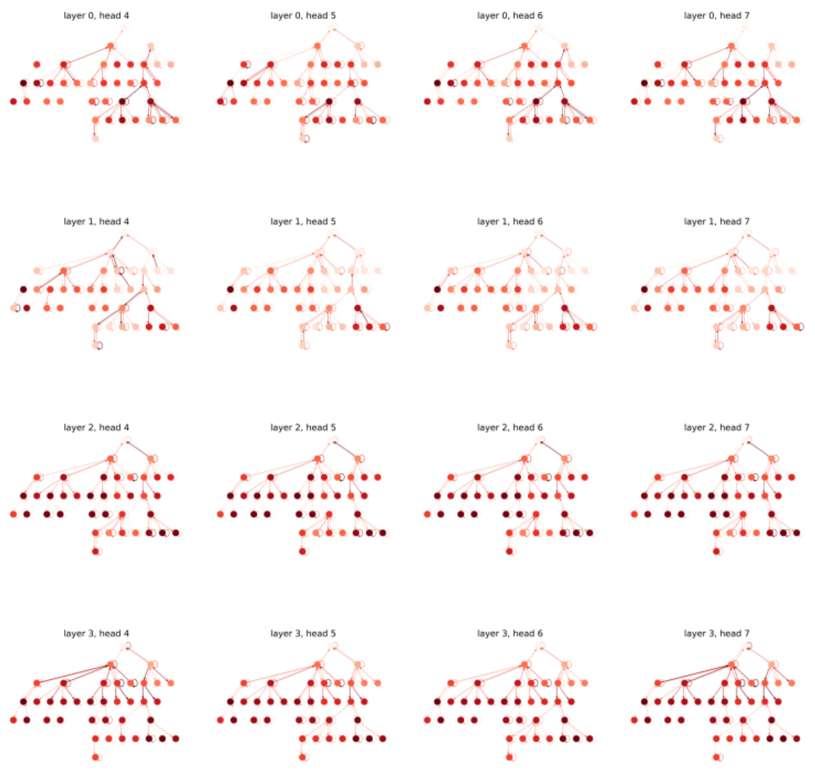
- **Kernel parameter tuning**: Looking for the best combination of *the number of heads* and *the number of layers/modules.*
- **Capacity tuning**: Learning the influence of balancing *the number of units* and *batchsize.*
- **Model comparing**: Choosing the best model among the four by comparing their performances.

In the first two tunings I assume GATGAP(gen) to be the best choice according to its highest complexity, and carry out the searches only based on this model. Finally all the models with the same configuration are compared, assuming that this configuration is the best one for all the models according to their structural similarity. During all the tunings and comparisons, cross entropy (See Chapter 3.3.4) is chosen as the loss function.

In the **Kernel parameter tuning** I first limit *the number of heads* to be no larger than 4, considering the visualisation shown in the last section. The first scan of *the number of layers* ranges from 2 to 8. The resulted loss and accuracy are shown in the figure 4.11. For the case of 2 heads, more layers leads to better performance. But for 4 heads, 8 layers starts to show an overfitting. Therefore the model should not be more complex than 4 heads and 8 layers. Then I check the evolution of their AUC values and find that both 4 heads 6 layers and 2 heads 8 layers are the best combinations (Figure 4.12). After some repeated experiments I decide to use 4 heads 6 layers as the best configuration because it is more stable than the other one.



(a) Vertical: Layer 0-3, Horizontal: Head 0-3



(b) Vertical: Layer 0-3, Horizontal: Head 4-7

Figure 4.10: Visualisation of Attention Weights of one graph from all the 4 layers and 8 attention heads.

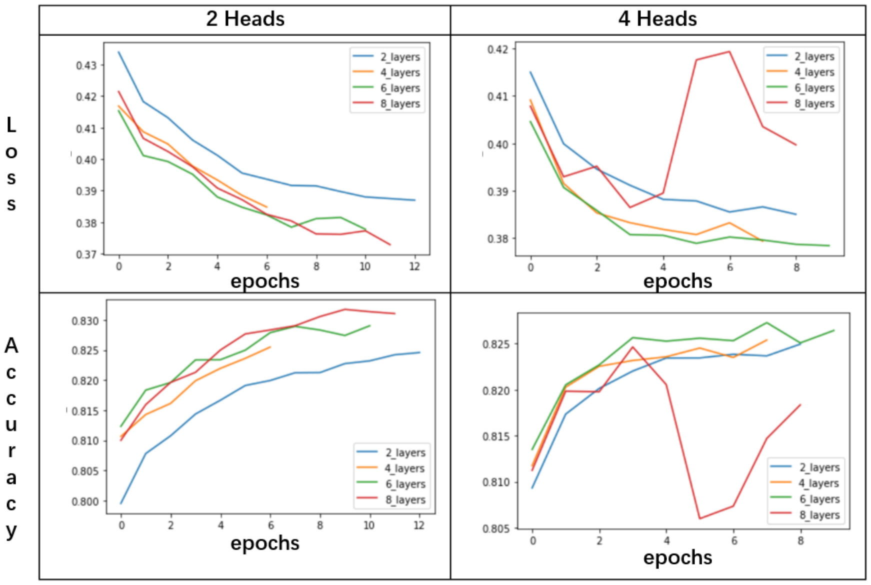


Figure 4.11: Kernel parameter tuning in terms of loss and accuracy

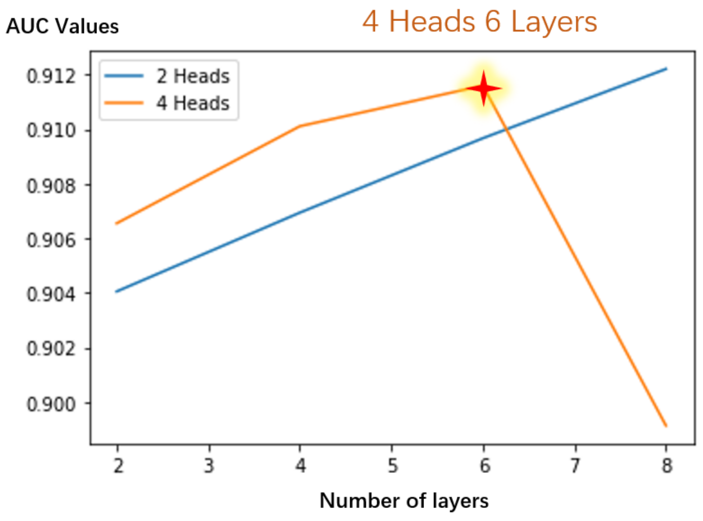
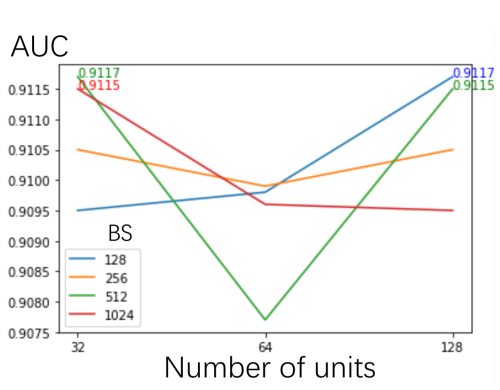


Figure 4.12: Kernel parameter tuning in terms of AUC values

Next is the **Capacity tuning** where I first do a grid search with

- *The number of units* in 32, 64, 128, 256, 512
- *Batchsize* in 128, 256, 512, 1024.

I find that small *number of units* are usually working better than the ones larger than 256. The area for the best combinations is shown in the figure 4.13 left. Then I further give less units a try. It turns out that 16 units works even better, but the training time is so long compared to the improvement that makes it not worth choosing (Figure 4.14).



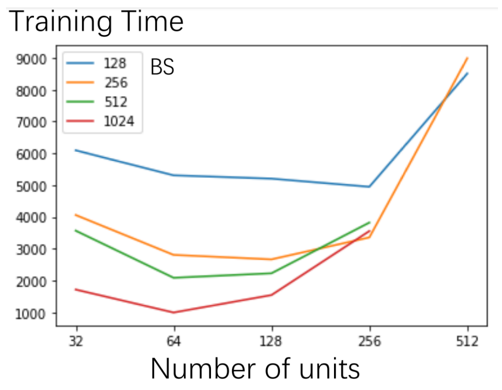
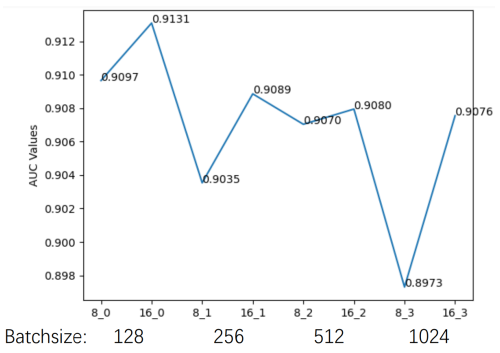


Figure 4.13: (left) AUC values for 32-128 units. (right) Training times.



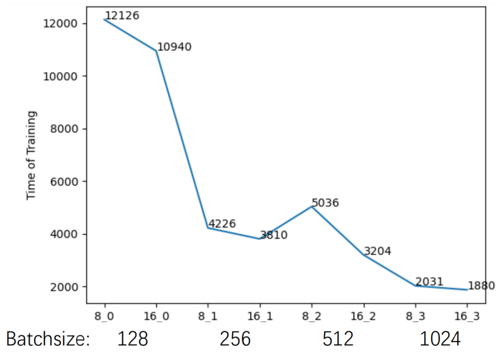


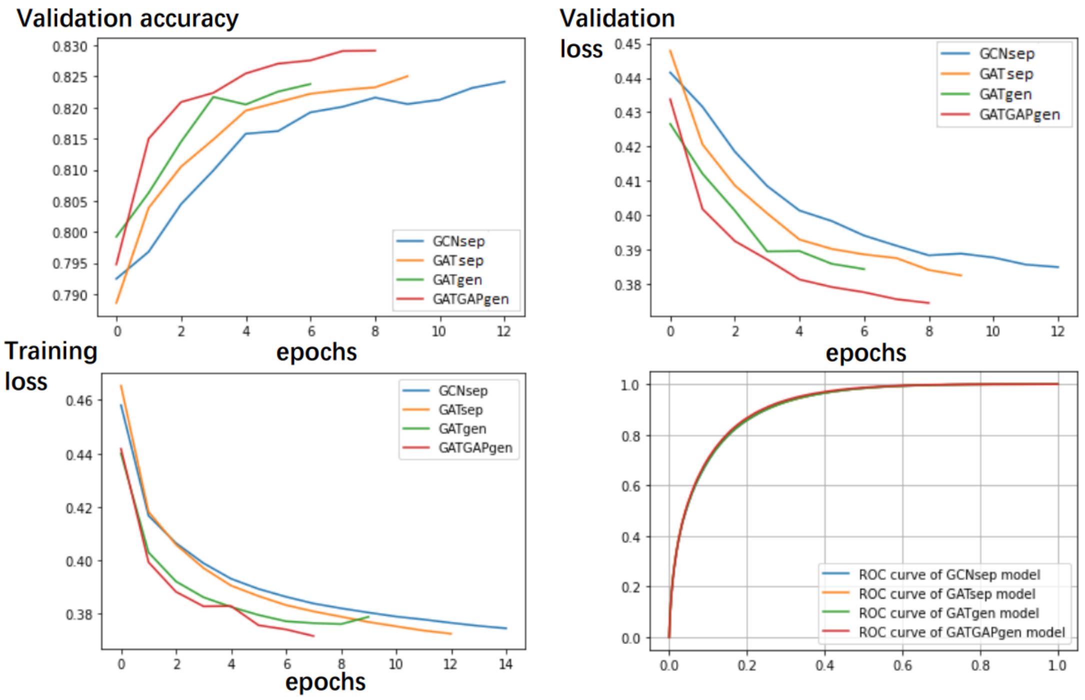
Figure 4.14: AUC values (left) and training times (right) for 8/16 units.

To conclusion, I list all the best combinations in the table 4.15 lest. For the later comparison of different models, I choose the combination of *batchsize* 128 and *units* 128, which has the second highest AUC and also shows strong stability, as the best configuration.

Batchsize	Number of Units	AUC	Training Time in s
128	16	0.9131	10940
512	32	0.9117	3568
128	128	0.9117	5205
1024	32	0.9115	1716
512	128	0.9115	2228
256	128	0.9105	2666
256	32	0.9105	4061

Number of Units	Number of Parameters
16	34,911
32	120,527
64	459,951
128	1,808,495

Figure 4.15: (left) Best combinations for the best AUC values. (right) Network sizes.



(a) Validation accuracy (upper left), Validation loss (upper right), Training loss (lower left) and ROC curves (lower right) of all the models

Model	AUC Value	Training Time
GCN(sep)	0.9083	3619
GAT(sep)	0.9094	4047
GAT(gen)	0.9089	3471
GATGAP(gen)	0.9122	5059

(b) AUC Values and Training Times of all the models

Figure 4.16: Comparison of the models according to their accuracy, loss, AUC and training time.

However, after taking the memory usage (Table 4.15 right) into consideration, I finally decide to take 1024 as *batchsize* and 32 as *the number of units* in the post-processing.

In the end, the abilities of different models to accomplish the task of the classification between background and signal events are compared through different criteria including accuracy on validation set, loss on validation and training set, training time on training set, ROC curve and AUC values on test set (Figure 4.16). Going through all the aspects it can be concluded that the learning ability of the GATGAP(gen) model is the strongest and therefore it performs always the best among the four models, except for the relative long training time, which is however acceptable and will not influence the final employment on test data set, once it gets well trained.

The final configuration after all the tunings and comparisons is shown in the table 4.4.

Model	GATGAP(gen)
The number of heads	4
The number of layers/modules	6
The number of units	32
Batchsize	1024

Table 4.4: Best configuration of the neural network filter.

Chapter 5

Event weighting methods

This chapter is the second part of the practice in this work. I will first derive two **metrics** used for sampling and reweighting method, which are the two major directions for the post-processing to weight events, and further explain their robustness for the sake of future generalization. Then comes the **post-processing** that shows the performances of one sampling process and two reweighting processes.

5.1 Speedup Metrics

The design of **Speedup** functions originates in the previous study [14] aiming to quantify the improvement of the speed to produce events with the help of neural network structures. In this work, taking the randomness of MC processes into account, the baseline of speedup metrics is chosen as **Statistical Uncertainty**, which describes the amplitude of variations around expectations generated by batches of samples. For weighted events, the **Effective Sample Size** is understood as the number of events that would yield the same statistical uncertainty in case of unweighted events. Hence the speedup is generally defined by the ratio between the time consumings of the whole work flow with and without NN filters for producing the same effective sample size.

In this work, I compare two ways of bias mitigation: sampling and reweighting. The origin of statistical uncertainties by both methods are not the same. Thus I am going to explain their derivations in the following chapters separately.

5.1.1 Speedup for the Sampling Method

The uncertainty comes directly from the random selection of the sampling method (See Chapter 3.1), where all the events take their NN scores p as probabilities to be kept. For each event an uncertainty $w = \frac{1}{p}$, the inverse probability, is introduced and equals the weight for sampling method,

$$w_i = \frac{1}{p_i} \tag{5.1}$$

For samples that pass the skim, taking the probabilities to appear into account, the total statistical uncertainty takes the form

$$S_{\text{total}} = \sqrt{\sum_{\{i|t_i=\text{Pass}\}} w_i^2 p_i} = \sqrt{\sum_{\{i|t_i=\text{Pass}\}} \frac{1}{p_i}} \tag{5.2}$$

with t the target or label for each event.

The relative statistical uncertainty is obtained by dividing by the total weighted uncertainty $\sum_{\{i|t_i=\text{Pass}\}} w_i p_i$,

$$S = \frac{S_{\text{total}}}{\sum_{\{i|t_i=\text{Pass}\}} w_i p_i} = \frac{\sqrt{\sum_{\{i|t_i=\text{Pass}\}} w_i^2 p_i}}{\sum_{\{i|t_i=\text{Pass}\}} w_i p_i} = \frac{1}{N_{\text{Pass}}}\sqrt{\sum_{\{i|t_i=\text{Pass}\}} w_i} \tag{5.3}$$

with the number of pass events N_{Pass}.

The effective sample size is the inverse of the squared relative statistical uncertainty

$$N_{\text{eff}} = \frac{1}{S^2} = \frac{(\sum_{\{i|t_i=\text{Pass}\}} w_i p_i)^2}{\sum_{\{i|t_i=\text{Pass}\}} w_i^2 p_i} = \frac{N_{\text{Pass}}^2}{\sum_{\{i|t_i=\text{Pass}\}} w_i} \tag{5.4}$$

On the other hand, the sample size can be represented directly with NN scores. The number of events that can pass the NN filter is given by

$$N_{\text{filter}} = \sum_i p_i \tag{5.5}$$

Since the metric is evaluated on a sample consisting of equal amounts of events that pass and fail the skim selection, the retention rate r (See Chapter 1.4) has to be taken into account in the equation 5.4 and 5.5

$$N'_{\text{eff}} = rN_{\text{eff}} \tag{5.6}$$

$$N_{\text{filter}} = rN_{\text{True-Positive}} + (1-r)N_{\text{False-Positive}} \tag{5.7}$$

$$= r\sum_{\{i|t_i=\text{Pass}\}} p_i + (1-r)\sum_{\{i|t_i=\text{Fail}\}} p_i \tag{5.8}$$

The effective sample size for the case without NN filter should be enlarged by a factor $\frac{1}{r}$ in order to compensate the retention rate. So the final effective sample size, to be understood as the number of events that are needed if there is no NN filter to achieve the same statistical uncertainty the same as if there is an NN filter should be

$$N_{\text{no-filter}} = \frac{1}{r}N'_{\text{eff}} = \frac{1}{r}rN_{\text{eff}} = N_{\text{eff}} \tag{5.9}$$

Combining the equation 5.5 with filter and 5.9 without filter, the effective speedup is calculated as the ratio

$$\text{Speedup}_{\text{eff}} = \frac{N_{\text{no-filter}}}{N_{\text{filter}}} \tag{5.10}$$

indicating the extra effort that has to be paid in the work flow without the neural network filter compared to advanced case. The larger the speedup, the better the filter works.

The deviation above does not take additional time consumption for the NN inference into account and neglects the time for Monte carlo generation. A rough estimation of the execution times is taken from the previous study [14] and shown in the table 5.1.

Stage	Time (ms/event)
Monte Carlo generation t_{gen}	0.08
Neural network inference t_{NN}	0.63
Simulation and Reconstruction t_{SR}	97.04

Table 5.1: Rough estimation of the execution time in ms for a single event during each stage of the simulation.

The processing time of sampling is so short that can be ignored. However, the execution time for neural network inference can be quite different depending on how the neural network is built and what device is used. The time for MC generation as well as Simulation and Reconstruction can also vary due to a different choice of data set. I will therefore study how robust the speedup metrics are and show the result in section 5.1.3.

The calculation of speedup taking the execution times into account is similar to the derivation above. For this it has to be clarified which samples end up in which computation step. A summary is shown in Figure 5.1 left.

	t_{gen}	t_{NN}	t_{SR}
TP	✓	✓	✓
FP	✓	✓	✓
FN	✓	✓	
TN	✓	✓	

Prediction / Label	Positive	Negative
Pass	True-Positive (TP)	False-Negative (FN)
Fail	False-Positive (FP)	True-Negative (TN)

Figure 5.1: (Left) Execution times for different type of samples. (Right) Confusion/Error Matrix

Both TP and FP events have positive results by the filter and will be sent to the skimming, thus they experience t_{SR}. The difference between these two categories is their proportions: r and $1 - r$ accordingly. For FN and TN events it is impossible to get through the filter, thus no t_{SR}. Their numbers are counted by the rejections probabilities

$1-p$ instead of the score p comparing to the equation 5.5. Gathering all the information together, the time consumption with neural network is given by

$$\begin{aligned} t_{\text{filter}} =&(N_{\text{TP}} + N_{\text{FP}}) \cdot (t_{\text{gen}} + t_{\text{NN}} + t_{\text{SR}}) \\ &+ (N_{\text{FN}} + N_{\text{TN}}) \cdot (t_{\text{gen}} + t_{\text{NN}}) \\ =&(r \sum_{\{i|t_i=\text{Pass}\}} p_i + (1-r) \sum_{\{i|t_i=\text{Fail}\}} p_i) \cdot (t_{\text{gen}} + t_{\text{NN}} + t_{\text{SR}}) \\ &+ (r \sum_{\{i|t_i=\text{Pass}\}} (1-p_i) + (1-r) \sum_{\{i|t_i=\text{Fail}\}} (1-p_i)) \cdot (t_{\text{gen}} + t_{\text{NN}}) \end{aligned} \tag{5.11}$$

The time without neural network contains only Monte Carlo generation and Simulation and Reconstruction, with the effective sample size from equation 5.9. That is

$$\begin{aligned} t_{\text{no-filter}} &= N_{\text{no-filter}} \cdot (t_{\text{gen}} + t_{\text{SR}}) \\ &= \frac{N_{\text{Pass}}^2}{\sum_{\{i|t_i=\text{Pass}\}} w_i} \cdot (t_{\text{gen}} + t_{\text{SR}}) \end{aligned} \tag{5.12}$$

In the end, combining 5.11 and 5.12, the speedup for sampling method is expressed as

$$\text{Speedup}_{\text{Sampling}} = \frac{t_{\text{no-filter}}}{t_{\text{filter}}} \tag{5.13}$$

The speedup with sampling method can reach up to 2 in this work, meaning that the processing efficiency of the new work flow is two times better than the original one. For details see chapter 5.2.1.

5.1.2 Speedup for the Reweighting Method

With the reweighting method, only events whose scores are larger than the threshold are selected and given a weight which equals the inverse of the conditional probability $q_i \equiv p(\text{TP}|\text{P}, i)$ with P representing Positive prediction, P=TP+FN. The weight for every sample can be then written as

$$w_i = \begin{cases} 0, & \text{for } p_i < \text{threshold} \\ \dfrac{1}{q_i}, & \text{for } p_i \geq \text{threshold} \end{cases} \tag{5.14}$$

Because there is no more random selection, the probability of each event to appear is no longer the neural network score, but boolean 0 or 1:

$$b_i = \begin{cases} 0, & \text{for } p_i < \text{threshold} \\ 1, & \text{for } p_i \geq \text{threshold} \end{cases} \tag{5.15}$$

The following derivations are similar to the sampling speedup. From the equation 5.3, the relative statistical uncertainty is

$$S = \frac{\sqrt{\sum_{\{i|t_i=\text{Pass}\}} w_i^2 b_i}}{\sum_{\{i|t_i=\text{Pass}\}} w_i b_i} = \frac{\sqrt{\sum_{\{i|t_i=\text{Pass},p_i\geq\text{threshold}\}} w_i^2}}{\sum_{\{i|t_i=\text{Pass},p_i\geq\text{threshold}\}} w_i} = \frac{\sqrt{\sum_{\{i|\text{TP}\}} w_i^2}}{\sum_{\{i|\text{TP}\}} w_i} \tag{5.16}$$

The effective sample size is

$$N_{\text{eff}} = \frac{1}{S^2} = \frac{(\sum_{\{i|\text{TP}\}} w_i)^2}{\sum_{\{i|\text{TP}\}} w_i^2} \tag{5.17}$$

The number of each type of the events can be counted directly through the criteria for labels and scores:

$$N_{\text{TP}} = \sum_{\{i|t_i=\text{Pass},p_i\geq\text{threshold}\}} \cdot r \equiv r\Omega_{\text{TP}} \tag{5.18}$$

$$N_{\text{FP}} = \sum_{\{i|t_i=\text{Fail},p_i\geq\text{threshold}\}} \cdot(1-r) \equiv (1-r)\Omega_{\text{FP}} \tag{5.19}$$

$$N_{\text{FN}} = \sum_{\{i|t_i=\text{Pass},p_i<\text{threshold}\}} \cdot r \equiv r\Omega_{\text{FN}} \tag{5.20}$$

$$N_{\text{TN}} = \sum_{\{i|t_i=\text{Fail},p_i<\text{threshold}\}} \cdot(1-r) \equiv (1-r)\Omega_{\text{TN}} \tag{5.21}$$

Ignoring again the processing time of reweighting, following the equation 5.11, the whole execute time with neural network filter is

$$\begin{aligned} t_{\text{filter}} =& (N_{\text{TP}} + N_{\text{FP}}) \cdot (t_{\text{gen}} + t_{\text{NN}} + t_{\text{SR}}) \\ &+ (N_{\text{FN}} + N_{\text{TN}}) \cdot (t_{\text{gen}} + t_{\text{NN}}) \\ =& (r\Omega_{\text{TP}} + (1-r)\Omega_{\text{FP}}) \cdot (t_{\text{gen}} + t_{\text{NN}} + t_{\text{SR}}) \\ &+ (r\Omega_{\text{FN}} + (1-r)\Omega_{\text{TN}}) \cdot (t_{\text{gen}} + t_{\text{NN}}) \end{aligned} \tag{5.22}$$

and the time without filter is

$$t_{\text{no-filter}} = N_{\text{eff}} \cdot (t_{\text{gen}} + t_{\text{SR}}) = \frac{(\sum_{\{i|\text{TP}\}} w_i)^2}{\sum_{\{i|\text{TP}\}} w_i^2} \cdot (t_{\text{gen}} + t_{\text{SR}}) \tag{5.23}$$

The ratio of the above two equations gives the speedup for the reweighting method with different values but the same form as for the sampling method in equation 5.13

$$\text{Speedup}_{\text{Reweighting}} = \frac{t_{\text{no-filter}}}{t_{\text{filter}}} \tag{5.24}$$

The reweighting speedup above is suited to both *GBDT reweighting* and *Histogram reweighting*. The only difference between these two reweighting methods is the way to determine the weight, or the conditional probability q_i. The speedup for *reweighting* methods can reach 5 to 6.

5.1.3 Robustness

Robustness reflects the ability of a system to cope with perturbations without changing its initial configurations [37]. In the language of this chapter, it refers to the ability of the speedup metrics to be tolerant to different computing powers or the complexities of the NN architectures and to different data sets from other skims or other constructions.

In this section, I quantitatively test the evolution of speedup with the change of three parameters $t_{\mathrm{gen}}, t_{\mathrm{NN}}$ and t_{SR}, particularly, the change of the ratio between t_{NN} and t_{gen} to study device dependency and of the ratio between t_{SR} and t_{gen} to study data set dependency. Due to the same dependency on the three times of both sampling and reweighting speedup, I only choose the one for the sampling method without loss of generality.

First I train the network (See Chapter 4.2) to get a reference value of speedup with the initial configuration as shown in Table 5.1. Then to simulate device dependency, which will only make a difference in the time for neural network execution, I linearly change the ratio $\frac{t_{\mathrm{NN}}}{t_{\mathrm{gen}}}$, which is 7.785 initially, from 0 to 100, with t_{gen} staying unchanged. The result is shown in the figure 5.2 left. The speedup develops linearly with the ratio from 0.47 to 0.62, indicating a change of less than 30% if execution time is 12 times longer, and less than 2% if the computation is 8 times faster. This allows a running of the code on CPU instead of GPU or further optimizations of the NN model or the use of some even more complex structures, etc..

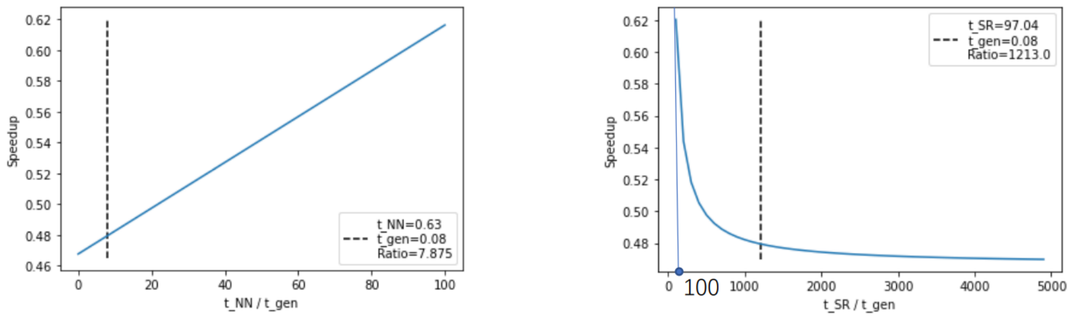


Figure 5.2: (Left) Filter efficiency dependency of speedup for sampling method. (Right) Data set dependency of speedup for sampling method.

Next I test the influence of changing data set which will provide different time of Simulation and Reconstruction as well as MC generation. This time the ratio $\frac{t_{\mathrm{SR}}}{t_{\mathrm{gen}}}$, which is 1213 initially, is linearly changed. As shown in the figure 5.2 right, the speedup grows exponentially for small ratios. However for 100, which is already 1/12 of the initial value, the speedup only reaches 0.62. For large ratio, which is tested up to 10 times of the initial value but cutted off for a better view, the speedup even converges to some value larger than 0.46.

To summary, the speedup metrics are robust against a change of the three parameters by a factor of at least 10.

5.2 Post-processing

After the neural network filter, each event will get a score, originally indicating the probability of this event to pass the following skims. The trivial way to make use of these values is to select the events whose score are larger than a certain number, known as the threshold, for further work flow. However, this selection will reject some samples that have been able to pass the skims, known as the false negative events, and cause the deviations, known as the bias, in the following analysis. The goal of this part of the work is to find out and minimize the influences of the bias with the help of some statistical tools.

The first step is to position the bias. Following the previous studies, 14 physics observables attached to each event are selected (see section 4.1.1). All these variables do not take part into the training of the neural network and are only used in the post-processing. In the experiment, the histogram of each variable of all the *Pass* labeled events is compared with that of all the *True-Pass* events, whose scores are required to be larger than the threshold. In the figure 5.3 as an example, I show the deviations of 8 variables before any post-processing, which will also be used as trainers in the reweighting method (see section 5.2.2). In this example, the threshold is set as 0.85. All the histograms on the left are normalized, with the *True* distributions colored in blue and the *True-Positive* distributions colored in orange. The relative deviations between each pair of histograms are shown on the right. These deviations are expected to be eliminated during the operations in the following sections.

5.2.1 Importance Sampling

The first idea to deal with the bias is the sampling method. As introduced in section 3.1, the events are randomly selected according to their scores and a threshold is no longer needed. On the other hand, the sampling is based on the statistical uncertainty instead of the accuracy, therefore the loss function during the training of the neural network filter is also chosen to be the speedup metric (see section 5.1.1) instead of the cross entropy. By maximizing the speedup function, the network is expected to give the best sampling weight to each event. No further tuning is needed in this method but the performance is satisfying (Figure 5.4 upper right, M_{bc} as an example). The deviations can be hardly recognized by eyes for all the variables. However, even though the neural network is trained with speedup as loss function in this case, the final speedup can only reach to 2, meaning that half of the production time can be saved for each event, without any bias originated.

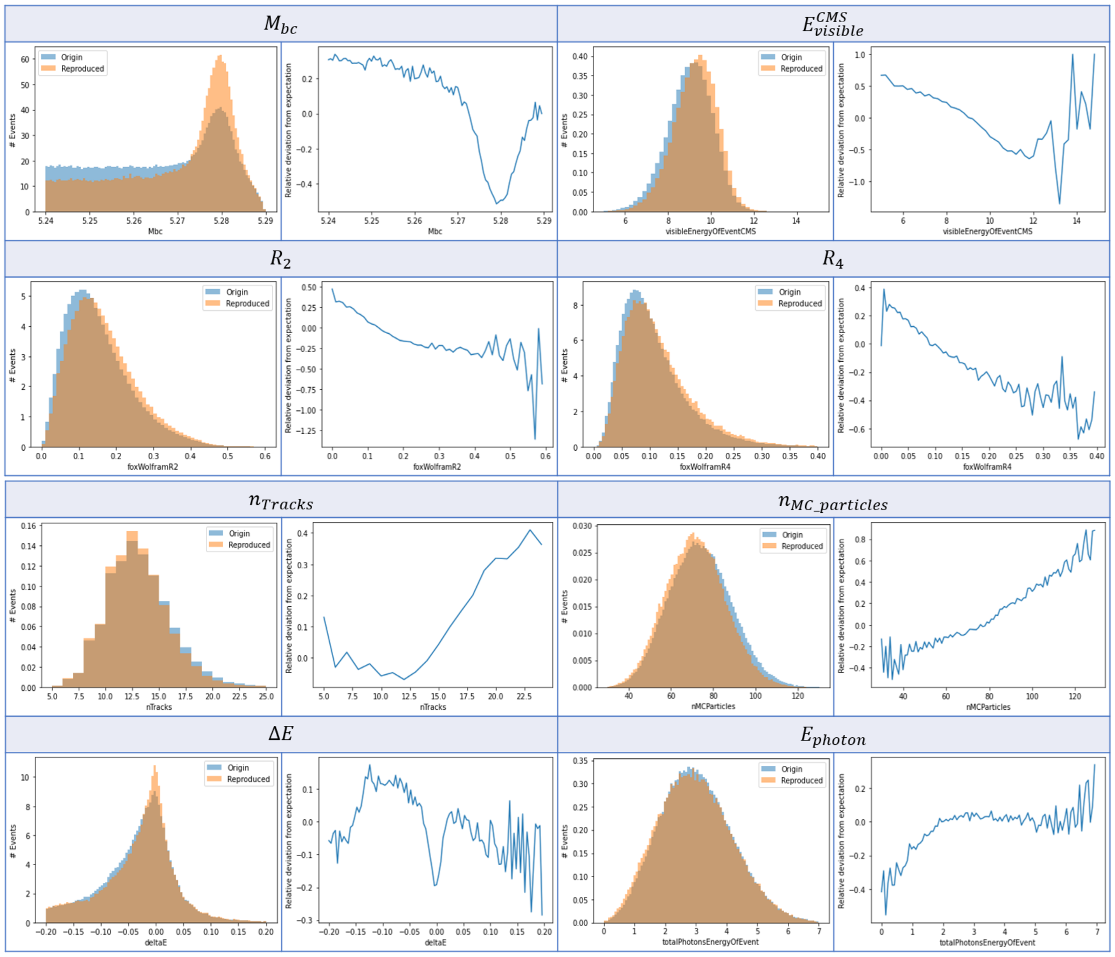


Figure 5.3: Variables to be used as trainers.

5.2.2 GBDT Reweighting

In search of a higher speedup, the reweighting method is studied, with the speedup metric alternated as well. The neural network is again trained with cross entropy to predict the probability of an event to pass. For the determination of the weights, or the inverse conditional probability of *True-Positive*, two methods, as introduced in section 3.2, are invested.

The first one is GBDT Reweighting, where a classifier depending on Gradient Boosting Decision Trees determines the weights directly. The training set is therefore built as a combination of *True-Positive* and *False-Negative* events with label 1 and 0 separately.

Similar to the tuning process for the neural network filter, the GBDT classifier has also to be well configured according to the task. To begin with, three *physics observables*: M_{bc}, $E_{\text{visible}}^{\text{CMS}}$ and R_2 are selected as trainers because of their strong bias among all the variables. With the help of these trainers, following grid-scans are finished:

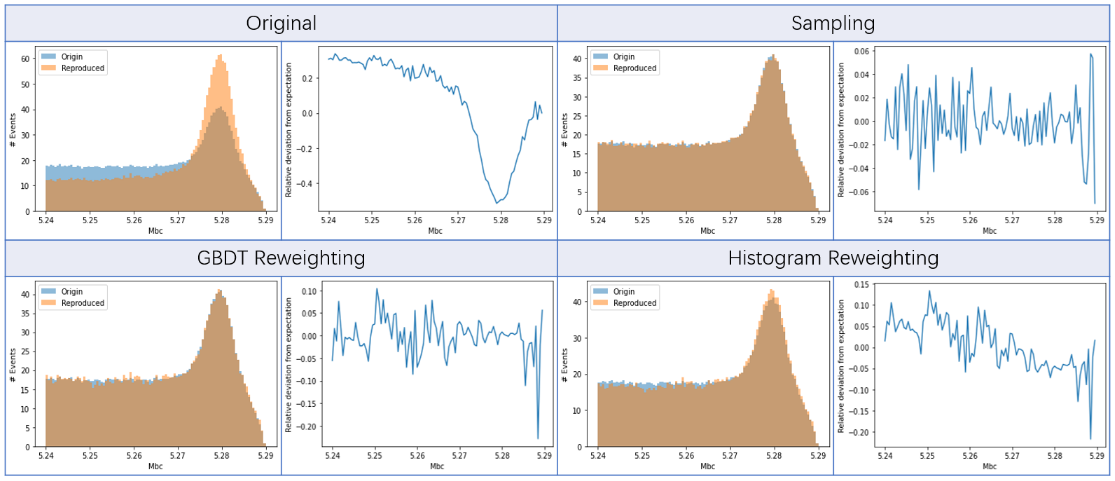


Figure 5.4: Reproduced M_{bc} with original selection (upper left), sampling (upper right), GBDT reweighting (lower left) and Histogram reweighting (lower right).

- Scan 1:

  Number of estimators in 50, 100, 250, 500, 750, 1000, 1250, 1500, 1750
  Learning rate in 0.15, 0.1, 0.05, 0.01, 0.005, 0.001

  With the help of the Calibration Curves (see Chapter 3.3.1) and the Kolmogorov-Smirnov Test (see Chapter 3.3.6), two best combinations are found in table 5.2:

Number of estimators	750	1500
Learning rate	0.1	0.15

Table 5.2: Best combinations after scan 1.

- Scan 2:

  Maximum depth in 3, 5, 7, 9, 11, 13, 15
  Minimum samples split in 100, 300, 500, 700
  Two best combinations after scan 1 as in table 5.2

  There remains only one best setting this time:

  Number of estimators = 750, Learning rate = 0.1
  Maximum depth = 3, Minimum samples = 100

- Scan 3: Subsample in 0.5, 0.6, 0.7, 0.8, 0.9, 1. The best choice turns out to be 0.5.

Number of estimators	750
Learning rate	0.1
Maximum depth	3
Minimum samples	100
Subsample	0.5

Table 5.3: Best configuration for the GBDT classifier.

Another important parameter to be decided is the threshold with which the positive events are selected against negative. To look for the best threshold, I scan over the range (0,1) and choose the threshold that corresponds the maximum of speedup (Figure 5.5 as an example). For GBDT reweightings the best threshold is usually around 0.85 and the maximum speedup is around 5.5.

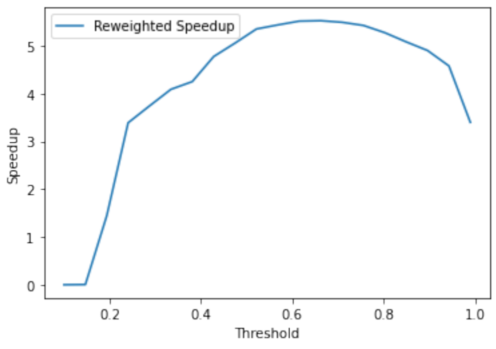


Figure 5.5: An example of speedup against threshold.

After the tunings above, the best configuration is found as shown in table 5.3 with the threshold equaling 0.85. With these settings, I train the classifier once again, but this time with all the variables used as trainers. Then I output the feature importance of each variable, showing the contribution of each character to the classification task (Figure 5.6). With the help of the feature importance function [8] as well as the histogram deviations, I finally choose the 8 features: $\mathbf{M}_{\rm bc}$, $\mathbf{E}_{\rm visible}^{\rm CMS}$, $\mathbf{R}_2$, $\mathbf{R}_4$, **No. tracks**, **No. MC Particles**, $\mathbf{\Delta E}$ and $\mathbf{E}_{\rm photon}$ as trainers.

With all the tuning processes ready, the performance of the final GBDT classifier is shown in the figure 5.4 (lower left). I also show the comparison of the reweighted histograms with the original ones for 6 of the training variables (Figure 5.7 left), who has the largest bias among the 8, except for $M_{\rm bc}$ which is already shown, and all the 6 other variables that are not included in the group of trainers.

It is impossible to recognize which method works better merely through the comparison of the histograms from figure 5.4 and figure 5.7. Thus the statistic tools introduced in

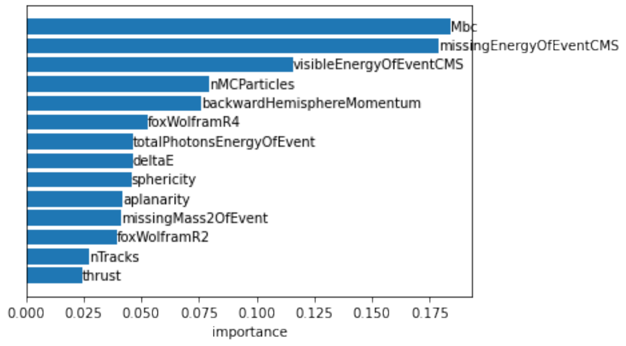


Figure 5.6: Feature importance of all the variables.

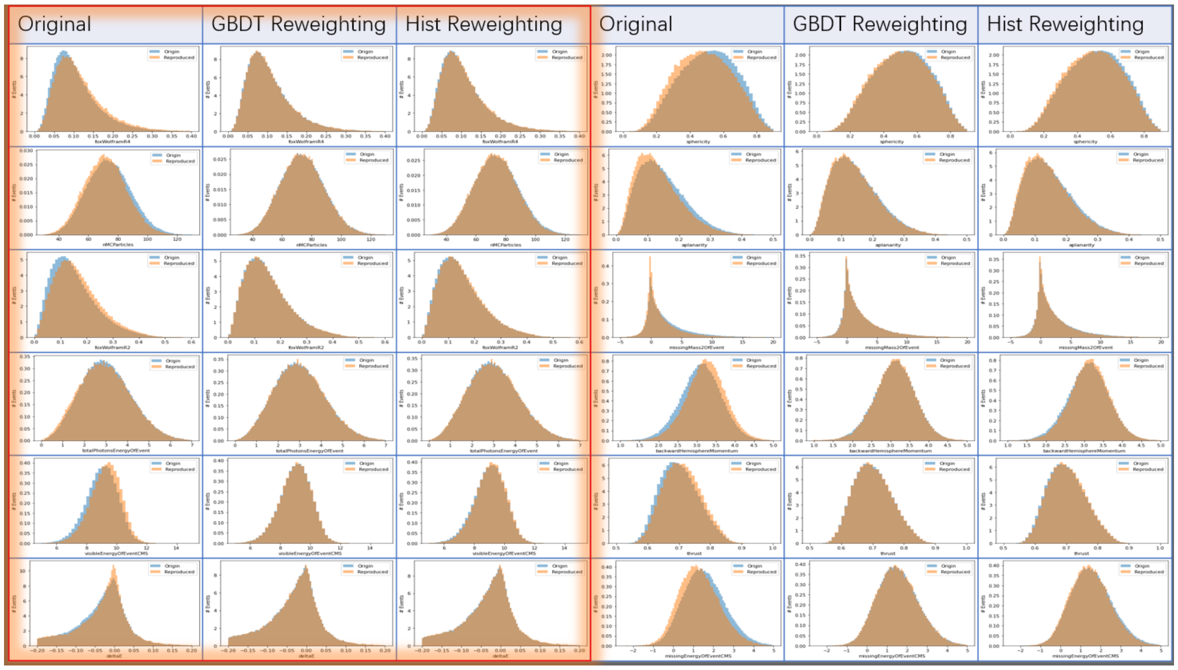


Figure 5.7: Performances of GBDT and Histogram Reweightings on trainers (left) and non-trainers (right).

Chapter 3.3 are studied and invested in the comparison among the methods as well as among the variables under each method. An example of KL Divergence and KS Test of all the variables for each method is shown in the figure 5.8. If only consider the reproduced variables under the GBDT reweighting, it can be concluded that all the variables used in the training are well reweighted, with relative low z-scores and KL divergences. The non-trainer variables have in general higher deviations of up to 8σ.

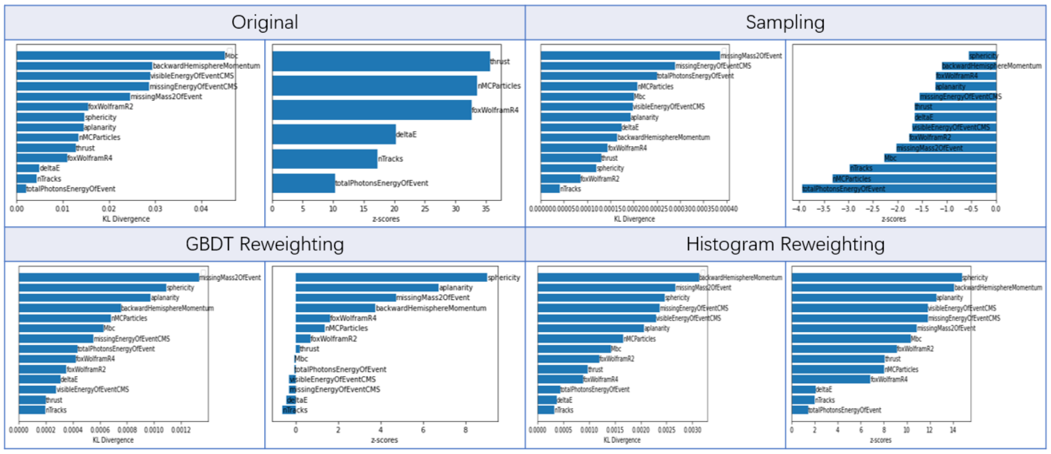


Figure 5.8: KL Divergence and KS Test.

5.2.3 Histogram Reweighting

The histogram reweighting in this work also bases on the GBDT classifiers with the same training targets: *True-Positive* and *False-Negative* events. Therefore it is not necessary to tune the classifier used in the GBDT reweighting once again. The best threshold of the histogram reweighting also lies around 0.85 but the speedup can reach up to 6.5 which is almost 20% higher than the GBDT reweighting and 2.2 times higher than the sampling method. However, the histogram reweighting also suffers a higher bias compared to the former two methods. The ability of the reweighting is already shown in the figures 5.4, 5.7 and 5.8.

As a conclusion, the sampling method has the strongest ability of bias mitigation but the lowest speedup, in contrast the histogram reweighting has the highest speedup but the worst quality of reproduction. A trade off between the speedup and the bias minimizing is shown in the table 5.4.

Method	Best Speedup	Bias (KS Test)
Sampling	2.0	~ 0
GBDT Reweighting	5.5	$\sim 8\sigma$
Histogram Reweighting	6.5	$\sim 15\sigma$

Table 5.4: Conclusion of the different weighting methods.

Chapter 6

Summary

6.1 Results

In this work, I follow the achievement of James [25] and Yannick [14]. First I improve the performance of the graph neural network filter by using the attention mechanism and integrating the training of the node features and the global features into the convolution module. Then in order to avoid the bias generated by the discard of *False-Negative* events, several methods including importance sampling, GBDT reweighting and Histogram reweighting based on statistical uncertainty are introduced so that the selected events, either randomly or through a threshold, will have higher weights for compensation.

6.2 Outlook

According to the study of the clustering method in the preprocessing and the robustness of the speedup metrics, the achievements in this work are expected to be generalized on other data sets for different skims or even from other projects.

In the future, the extensions of this work can be expected in the following directions:

- The graph neural network structure with attention mechanism can be further tuned. But according to the robustness of the speedup metrics, the improvement has to be large enough to make a difference.
- The sampling method can be improved by a better choice of the loss function in the training of the neural network, or some changing in the speedup function for the sampling method, in order to reach a higher speedup without scarifying the performance.
- Other classifiers can be used in the reweighting methods to balance speedup and bias.
- More studies are required about the histogram reweighting. Possible directions include new methods as classifiers, finer binnings or inhomogeneous binnings to build

histograms to minimize the influence of the fluctuations generated by low number of events in some of the areas, more advanced way to map the *True-Positive* histogram with the *True* histogram, etc..

- The statistical metrics can be further studied in order to better evaluate the performances of the weightings.

Bibliography

[1] Belle 2 software documentation–sphinx, 2021. https://software.belle2.org/development/sphinx/index.html.

[2] A concise explanation of learning algorithms with the mitchell paradigm, 2021. https://www.kdnuggets.com/2018/10/mitchell-paradigm-concise-explanation-learning-algorithms.html.

[3] Deep graph library tutorials and documentation, 2021. https://docs.dgl.ai/.

[4] Ibm cloud education - neural networks, 2021. https://www.ibm.com/cloud/learn/neural-networks.

[5] Khan academy - sampling methods review, 2021. https://www.khanacademy.org/math/statistics-probability/designing-studies/sampling-methods-stats/a/sampling-methods-review.

[6] Mit: Introduction to deep learning, 2021. http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf.

[7] Pytorch documentation, 2021. pytorch.org/docs.

[8] Scikit-learn library, 2021. scikit-learn.org.

[9] Scipy documentation, 2021. docs.scipy.org/doc/scipy/.

[10] Superkekb and belle ii, 2021. https://www.belle2.org/project/super_kekb_and_belle_ii.

[11] S. Agostinelli et al. Geant4a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 506(3):250–303, 2003. https://www.sciencedirect.com/science/article/pii/S0168900203013688.

[12] Kazunori Akai, Kazuro Furukawa, and Haruyo Koiso. Superkekb collider. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 907:188–199, 2018. Advances in Instrumentation and Experimental Methods (Special Issue in Honour of Kai Siegbahn), https://www.sciencedirect.com/science/article/pii/S0168900218309616.

[13] Caterina Biscari. Accelerators r&d. *Proceedings of Science*, pages 202009–019, 2009. https://doi.org/10.15161/oar.it/1448873081.77.

[14] Jannick Bross. Bias Mitigation in Selective Background Monte Carlo Simulation at Belle II. Master's thesis, Ludwig-Maximilians-Universität München, 10 2020.

[15] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, et al. Machine learning and the physical sciences. *Rev. Mod. Phys.*, 91:045002, Dec 2019. https://link.aps.org/doi/10.1103/RevModPhys.91.045002.

[16] The Belle II collaboration. Belle ii technical design report, 2010. https://arxiv.org/abs/1011.0352.

[17] Thomas M Cover and Joy A Thomas. Elements of information theory second edition solutions to problems. *Internet Access*, 2006. https://onlinelibrary.wiley.com/doi/book/10.1002/047174882X.

[18] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006. ROC Analysis in Pattern Recognition.

[19] Geoffrey C. Fox and Stephen Wolfram. Observables for the analysis of event shapes in e^+e^- annihilation and other processes. *Phys. Rev. Lett.*, 41:1581–1585, Dec 1978.

[20] Karl Pearson F.R.S. X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900. https://doi.org/10.1080/14786440009463897.

[21] Karl Pearson F.R.S. X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175, 1900. https://doi.org/10.1080/14786440009463897.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[23] Particle Data Group. Review of particle physics. *Phys. Rev. D*, 98:030001, Aug 2018. https://link.aps.org/doi/10.1103/PhysRevD.98.030001.

[24] J. L. Hodges. The significance probability of the smirnov two-sample test. *Arkiv för Matematik*, 3:469–486, 1958. https://projecteuclid.org/journals/arkiv-for-matematik/volume-3/issue-5/The-significance-probability-of-the-smirnov-two-sample-test/10.1007/BF02589501.full.

[25] James Meier Samuel Kahn. *Hadronic tag sensitivity study of B → K(*)v?v and selective background Monte Carlo Simulation at Belle II.* PhD thesis, April 2019. `http://nbn-resolving.de/urn:nbn:de:bvb:19-240131`.

[26] T. Keck et al. The full event interpretation. *Computing and Software for Big Science*, 3(1), Feb 2019. `http://dx.doi.org/10.1007/s41781-019-0021-8`.

[27] Thomas Keck. The Full Event Interpretation for Belle II. Master's thesis, KIT, Karlsruhe, 11 2014. Presented on 11 11 2014, `https://inspirehep.net/literature/1514166`.

[28] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017. `https://arxiv.org/abs/1609.02907`.

[29] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951. `https://doi.org/10.1214/aoms/1177729694`.

[30] David J. Lange. The evtgen particle decay simulation package. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 462(1):152–155, 2001. BEAUTY2000, Proceedings of the 7th Int. Conf. on B-Physics at Hadron Machines, `https://www.sciencedirect.com/science/article/pii/S0168900201000894`.

[31] Jun Liu. *Monte Carlo Strategies in Scientic Computing.* 02 2009. `https://www.springer.com/gp/book/9780387763699`.

[32] Martin N Marshall. Sampling for qualitative research. *Family practice*, 13(6):522–526, 1996. `https://doi.org/10.1093/fampra/13.6.522`.

[33] Tom M Mitchell et al. *Machine learning.* McGraw-hill New York, 1997. `http://www.cs.cmu.edu/~tom/mlbook.html`.

[34] Franco Scarselli et al. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. `https://ieeexplore.ieee.org/document/4700287`.

[35] Torbjrn Sjstrand et al. An introduction to pythia 8.2. *Computer Physics Communications*, 191:159177, Jun 2015. `http://dx.doi.org/10.1016/j.cpc.2015.01.024`.

[36] Petar Velikovi, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Li, and Yoshua Bengio. Graph attention networks, 2018. `https://arxiv.org/abs/1710.10903`.

[37] Andreas Wieland and Carl Marcus Wallenburg. Dealing with supply chain risks: Linking risk management practices and strategies to performance. *International journal of physical distribution & logistics management*, 2012. `https://www.depositonce.tu-berlin.de/bitstream/11303/7030/1/wieland_wallenburg.pdf`.

Acknowledgements

First I want to thank my supervisor, Professor Thomas Kuhr who gave me the chance to take part in this project, provided me many opportunities to show and discuss our achievements in the meetings and also gave me lots of support during my writing and presentations.

I am really grateful for the guiding, encouraging and supplying from Dr. Nikolai Hartmann. Without his ideas and teachings it is impossible for me to carry out this work. I learned a lot from him not only in the field of physics, programming and data processing, but also the experiences in the using of Linux and working on remote servers. I also thank to his carefully correction of my thesis and his great patience on my poor English writing skills. He makes me feel so warm especially during this hard time of Corona situations.

Further I would like to thank all the members in the AG-kuhr group who gave me the chance to know about their advanced physical researches in group meetings and conferences. Their introductions enriched my horizon and their efforts encouraged me to keep on moving.

Finally I want to thank my family for giving me emotional as well as financial support and encouragements.

Erklärung/Declaration

Hiermit erkläre ich, die vorliegende Arbeit selbständig verfasst zu haben und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel benutzt zu haben.

I hereby declare that this thesis is my own work, and that I have not used any sources and aids other than those stated in the thesis.

München, 21.09.2021
Boyang Yu