AI-Lab-smart-background

Valentin Kiendl, Paul Kopold, Suchetana Chatterjee

May 2023

Contents

1 Introduction and Theoretical Background

1.1 The Belle II experiment : an introduction

The Belle II experiment is a high-energy, particle physics experiment located at the SuperKEKB particle accelerator, which is also referred to as a "B-factory". A B factory, also known as a B meson factory, is a type of particle physics experiment that is specifically designed to produce and study B mesons. Belle II's main objective is to investigate the minute disparities between matter and antimatter, known as CP violation. Charged particles like electrons, muons, and their antiparticles (positrons and antimuons) are picked up by the Belle II detector. These charged particles leave behind detector traces that can be detected and recreated. The Belle II detector is additionally indirectly sensitive to neutral particles via their decay byproducts. For instance, the electromagnetic calorimeter can pick up photons (light particles) from the disintegration of neutral mesons like the neutral pion or neutral kaon.

The experiment focuses on the physics of bottom quark and antiquark-based B mesons, which are unstable particles. The Belle II detector is intended to carefully measure the properties and interactions of B mesons, which are created in vast quantities in collisions at the SuperKEKB accelerator. This experiment is an electron-positron (e+/e-) collider experiment. This has an important advantage, as the initial condition of e+/e- collisions being particle-free considerably lowers the background noise in the detector. This is crucial for uncommon and delicate activities that can be hidden by the numerous strong interaction processes present in p/p collisions. The studies' sensitivity is increased by the higher signal-to-noise ratios made possible by the lower background. In our lab course, we will indeed be focusing on how to improve our studies more by filtering this background even more efficiently.

1.2 The goal of the AI lab experiment

Through the Belle II experiment, the investigation of extremely rare processes is made possible by the massive luminosity and associated large data flow. However, it also means that in order to enhance analysis methods and create statistical models of the data, we require huge simulated samples. The workflow of any HEP experiment can be expressed in the following flow chart, referenced from the Belle II software online book :

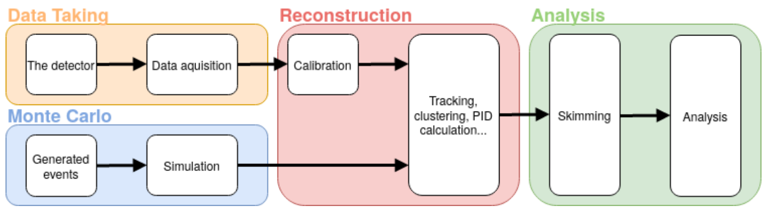


Figure 1: Workflow of the Belle II experiment

We ask ourselves, which steps is computationally more expensive, and what may we do to make it easier and more cost effective?

- Detector Simulation: Simulating the Belle II detector involves modeling the complex geometry, material interactions, and response of the detector components. As a result the detector simulation frequently requires a substantial investment of time and computing power.

- Reconstruction: Following the acquisition of the detector simulation's simulated signals, a process to rebuild the collision's particle's properties is carried out. Tracking charged particles, classifying particle kinds, measuring their energies and momenta, and reconstructing their decay products all need the use of sophisticated algorithms. Due to the significant amount of data and complex computations required, the reconstruction stage often demands significant processing resources.

As we see, reconstruction and analysis is one of the most demanding and expensive steps of the experiment. In this lab we address this specific area of computation. In order to deal with this problem, The strategy that will be addressed in this lab course is to train a neural network to anticipate which events correspond to ones that will be selected ultimately even before the simulation chain has even begun. By this, we will obtain an idea of the final results without requirement of the total analysis of the reconstructed events. As the background contains many processes irrelevant to the signal that we do not need to study, we train a "smart background filter" that helps us discard a large number of these events.

We have been provided with a labeled dataset that indicates whether or not an event passes the "Full Event Interpretation" Skim. The Full Event Interpretation, a method of analysis specific to B factories, proves crucial in measuring uncommon decays. By reconstructing one of the B mesons, this method can deduce significant constraints for the other B meson. The two B-mesons are referred to as "Tag-side" B-Meson : B_{Tag} and "Signal-side" B-Meson : B_{Sig} respectively. Reconstructing the particles and training the Multivariate Classifiers(MVCs) in a hierarchical manner are the basic concepts of the Full Event Interpretation. First, candidates for final-state particles are chosen, and related classification algorithms are trained using data from the detector. This is followed by the reconstruction of intermediate particle candidates and the training of a multivariate classifier for each used decay channel. A candidate's whole profile is condensed by the MVC into a single value called the signal-probability. Thus candidates from various decay channels can be treated equally in the subsequent stages of reconstruction. We have learned that only 5% of events survive the FEI skim filter, so it will be our goal to increase this efficiency even more through a smart pre - filtering. Due to the fact that everything is predicated on detector level information, it is challenging to forecast which events will be reconstructed only from generated particle data without detector contact. As a result, we will use machine learning to address the issue of smart background simulation.

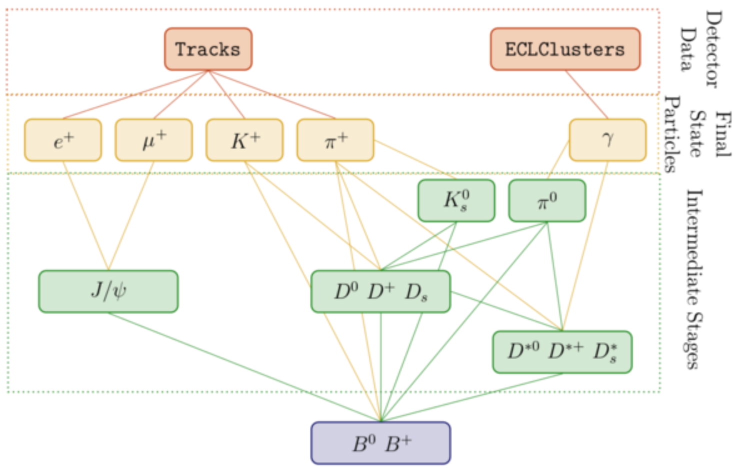


Figure 2: Hierarchical reconstruction obtained through FEI

1.3 Deep Sets

The data we will be analyzing consists of simulated collider events which in turn include the properties of all the created particles in that event. Since the ordering of the particles doesn't matter, we want our model to be invariant under exchange of two particles in an event. For this we can use Deep Sets. The idea of the architecture is to somewhere use a pooling function which aggregates the values of all particles together. The network will have the following general form

$$\rho(\sum_{p \in P} \phi(x_p)),$$

where P is the set of all particles and ρ and ϕ are suitable functions [2]. In our case, this means we use dense neural network layers acting on each particle individually in the same way, then use some pooling function and after that again use some dense layers. Some possible pooling functions are averaging over all values, taking the maximum, minimum and looking at its standard deviation. The most commonly used ones are the maximum and average pooling.

1.4 Graph Convolutional networks

In the deep set model we assume there is no structure to the particles at all, however the particles in our simulated events form a tree of subsequent decays of each of the particles. We can exploit that fact by using Graph Convolutional networks. To use the structure of the decay tree, we write down its adjacency matrix A. This matrix has rows and columns signifying the different particles and we set a matrix element A_{ij} to one if the particle number i is created by

the particle number j. Additionally we write down the degree matrix D which has only diagonal elements consisting of the number of particles the particle is either created by or is creating. Using these two matrices, we can formulate the update rule of a graph convolutional network.
Differing from the architecture of a fully connected network, where the forward propagation rule for a layer with feature matrix $H^{(l)}$ to the next layer with feature matrix $H^{(l+1)}$ is

$$H^{(l+1)} = \sigma\left(H^{(l)}W^{(l)}\right) \tag{1}$$

with weight matrix W and some activation function σ, in a graph convolutional network the update rule is [1]

$$H^{(l+1)} = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right). \tag{2}$$

The matrix $\hat{A}$ is the adjacency matrix of the graph with added self connections and $\hat{D}$ its degree matrix. If we would write the summation out, for each particle we only add up contributions of particles the respective particle has connections with, because all other elements of the adjacency matrix elements are zero. The self connection is added, such that the respective particle node can pass on information to the particle node of the same particle in the next layer. The terms $\hat{D}^{-\frac{1}{2}}$ are added in order to normalize the adjacency matrix, such that multiplying by the resulting matrix $\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$ doesn't change the order of magnitude of the values in the layer before

1.5 Overview of network architectures used

Utilizing Deep Sets and Graph convolutional Networks, one of the main goals of the experiment was to find the best performing model architecture. To be able to compare different model types, we first implemented a notebook with all the necessary pre-processing of the data, which included embedding, masking and calculating the adjacency matrix needed for graph layers. After that the model was constructed. The basis for those models were all the same, it started with three layers of input, two for the data features and one for the adjacency matrix. After concatenating the features they got passed into six layers of either dense networks or graph neural networks, where the arrangement of each type were chosen differently for every model. The next step is to use a pooling function, followed by a dropout layer and another three dense layers. For the final layer a dense layer with just a single node with sigmoid activation was used. We will interpret the output of this last layer as the probability with which the event should be discarded according to our model. The following picture shows one of those models as an example.

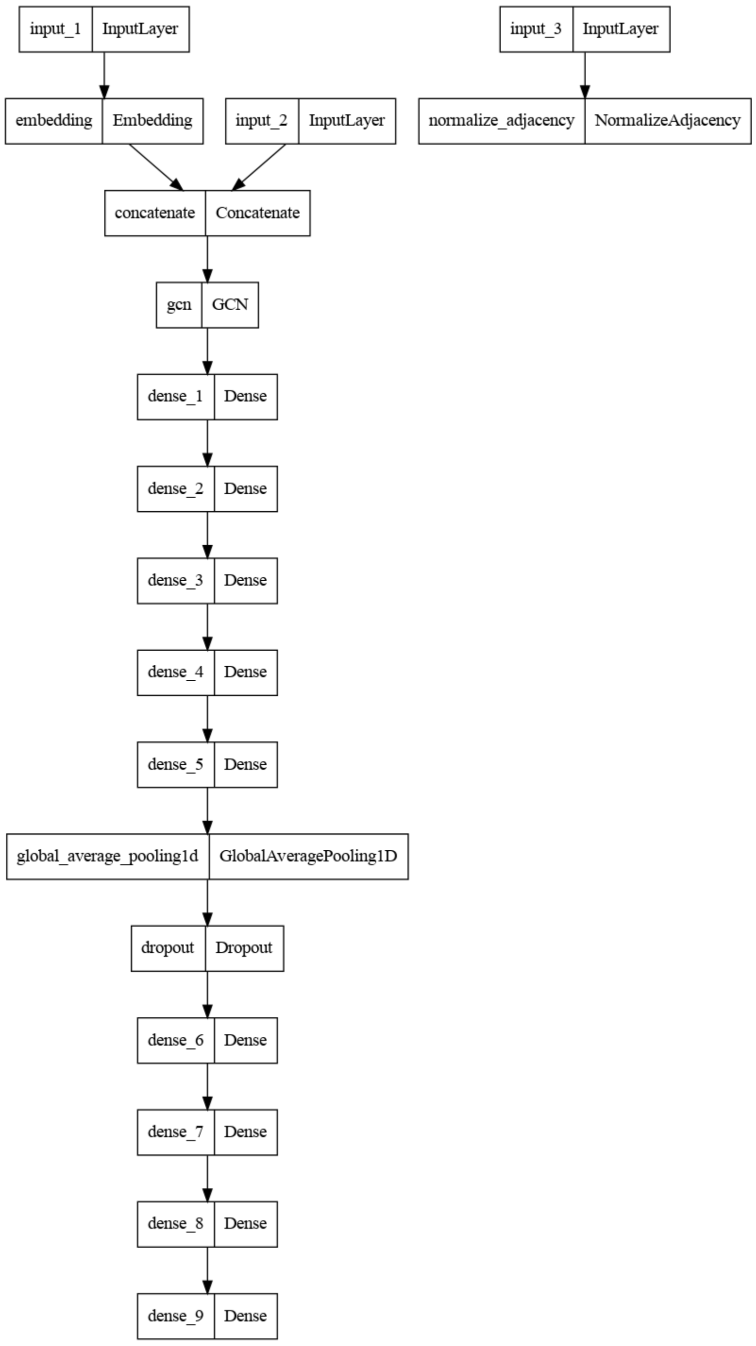


Figure 3: Model Example

2 Experiments

To evaluate the performance of the models, we first plotted the loss and accuracy on the training data and a smaller testing dataset. Both were taken from a dataset with 400k events, 40k of which were used for testing. For later evaluation of the model an additional dataset with 400k events was used. All the models evaluated were trained for 20 epochs and had the same number of hidden units in each layer to be able to get a good comparison of the architectures.

2.1 Network architecture

After testing several different model types we have chosen the three most promising. These models were:
- A model where the first three layers were a graph layer followed by three dense layers
- A model where the first and last layer were a graph layer
- A model where the layers got alternated starting with a graph layer

In the following figures one can clearly see that the difference in the three best performing models is quite marginal. The current top performer was the Alternating Model which showed an accuracy of 0.8150 therefore performing better then the second best by about 0.005.

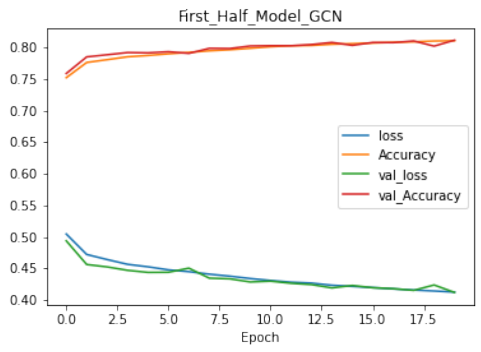


Figure 4: First Half Model

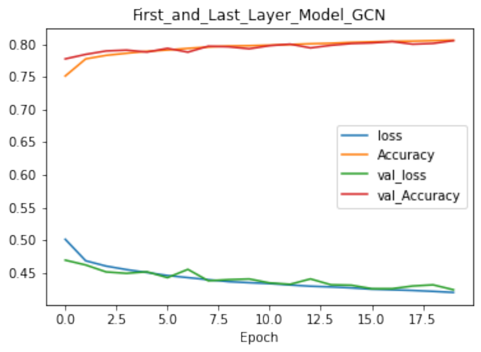


Figure 5: First and Last

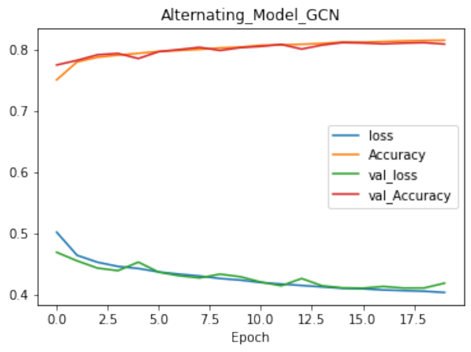


Figure 6: Alternating

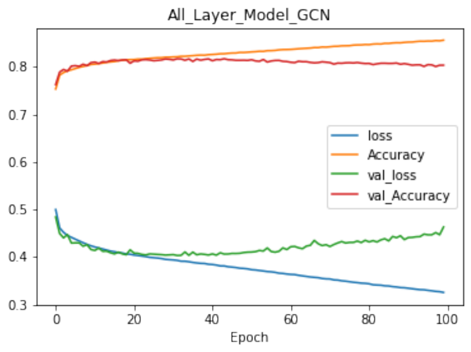


Figure 7: Epoch Evaluation

Training for more epochs than 20 resulted in overfitting, which can be seen in Figure 7.
In order to compare graph neural networks against deep set models, one can see a direct comparison between the deep set model, where no graph layers were used, against our best performing graph neural network.

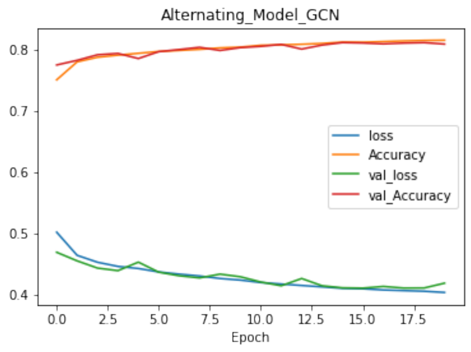


Figure 8: Alternating Model

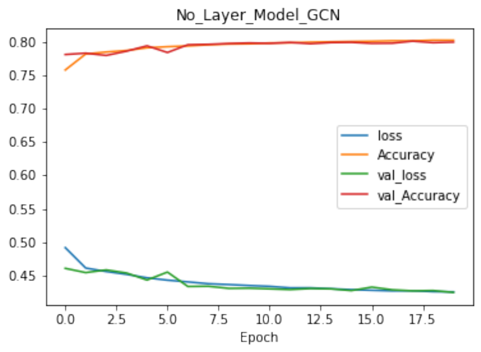


Figure 9: No Layer Model

In the following figure we see the ROC curves of the two best performing models and a model with no graph layers.

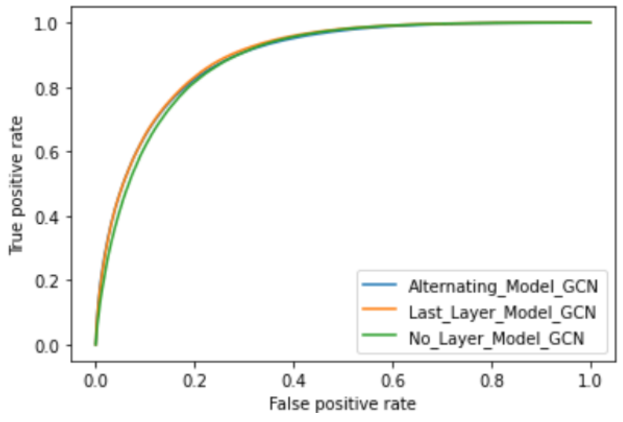


Figure 10: model comparison

The exact values for the performances of the models can be found in the appendix.

2.2 Importance of masking

For this experiment in particular a special kind of pre-processing was necessary. Since the data set included features that had varying length, which is not compatible with different kind of layers that do require a fixed length input, we had to implement zero-value padding. A problem of padding is that the extra added values to the data set skew the underlying data distribution. Whether or not this is a problem in this case was tested by performing the same model fitting for the best performing model, the Alternating Model, one time with and one time without adding a mask. Figure 10 and 11 include a direct comparison for those two cases. Again there seems to be little to no difference between masking and no masking. This result is confirmed by comparing the best accuracy and loss value where the masked alternating model has an accuracy of 0.8150 and the non-masked one has a value of 0.8149.

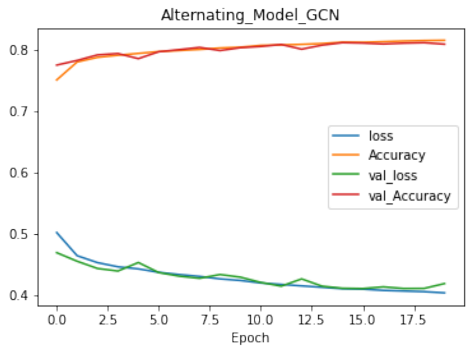


Figure 11: Alternating Model

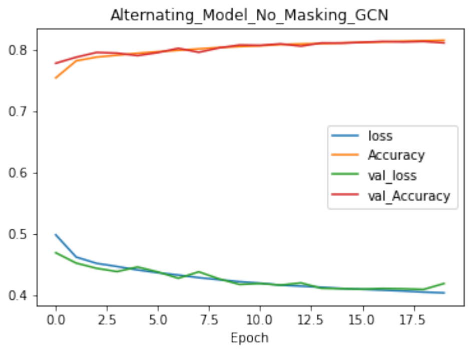
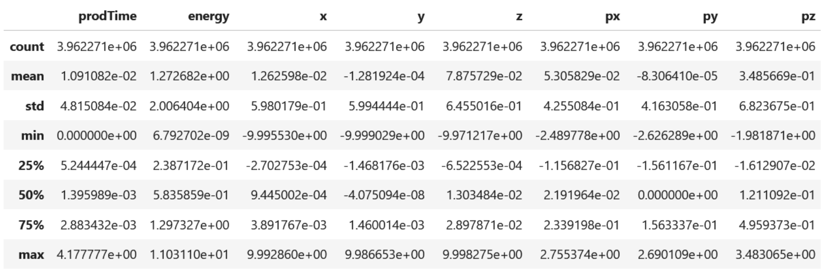


Figure 12: Alternating Model without masking

One reason for masking being not needed here could be that the underlying data features are highly centered around the value zero already, meaning that adding some zeros does not really change the distribution in any meaningful way. A second reason might be that the neural network simply learns that the huge amount of zero values have no significance in the overall model and are therefore added with a minimal weight.

2.3 Feature scaling

In order to decide, whether feature scaling is necessary, we first look at the data. As can be seen in figure 13 all of the features are in a reasonable range, so our assumptions are that feature scaling might not be necessary and probably doesn't improve the accuracy.



	prodTime	energy	x	y	z	px	py	pz
count	3.962271e+06	3.962271e+06	3.962271e+06	3.962271e+06	3.962271e+06	3.962271e+06	3.962271e+06	3.962271e+06
mean	1.091082e-02	1.272682e+00	1.262598e-02	-1.281924e-04	7.875729e-02	5.305829e-02	-8.306410e-05	3.485669e-01
std	4.815084e-02	2.006404e+00	5.980179e-01	5.994444e-01	6.455016e-01	4.255084e-01	4.163058e-01	6.823675e-01
min	0.000000e+00	6.792702e-09	-9.995530e+00	-9.999029e+00	-9.971217e+00	-2.489778e+00	-2.626289e+00	-1.981871e+00
25%	5.244447e-04	2.387172e-01	-2.702753e-04	-1.468176e-03	-6.522553e-04	-1.156827e-01	-1.561167e-01	-1.612907e-02
50%	1.395989e-03	5.835859e-01	9.445002e-04	-4.075094e-08	1.303484e-02	2.191964e-02	0.000000e+00	1.211092e-01
75%	2.883432e-03	1.297327e+00	3.891767e-03	1.460014e-03	2.897871e-02	2.339198e-01	1.563337e-01	4.959373e-01
max	4.177777e+00	1.103110e+01	9.992860e+00	9.986653e+00	9.998275e+00	2.755374e+00	2.690109e+00	3.483065e+00

Figure 13: data overview

Upon comparing the alternating Model with and without feature scaling, where we normalized every feature separately to a mean value of zero and a standard deviation of one, we observe that the model with feature scaling performs worse than the model without, see figure 14. This finding is somewhat surprising and we couldn't find an explanation for it.

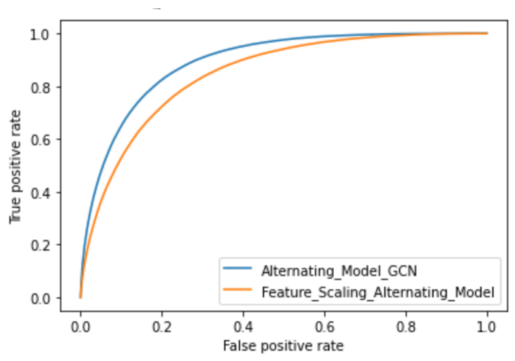


Figure 14: Alternating Model

2.4 Aggregation function

The aggregation function used in all the other models we evaluated was an average. Now we also look at another aggregation function, which is a combination of 4 different ones. The output of the pooling layer in this case are four layers, one with the average value, one with the maximum, one with the minimum and one with the variation of the previous layer. In the following plots we see the comparison of the ROC curves of the Alternating model architecture and of the Last Layer architecture as seen in chapter 2.1, once with an average and once with the new aggregation function. The models with the new pooling function will be called 'Alternating_Model_GCN_Pooling' and 'Last_Layer_Model_GCN_Pooling'

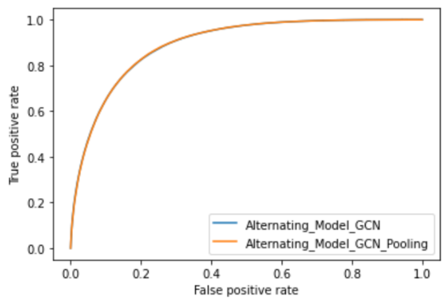


Figure 15: Alternating Model comparison of with different pooling layers

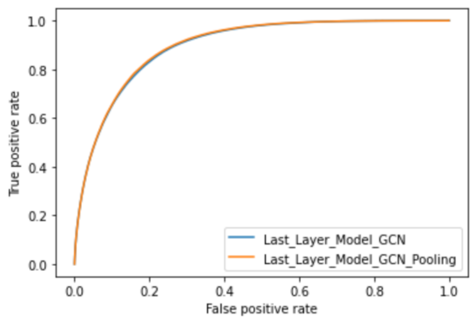


Figure 16: Last Layer Model comparison of with different pooling layers

As can be seen in the evaluation of the models overview in the appendix, the alternating model performs slightly better with just the average pooling function. For the last layer model, the one with the four different pooling functions performed slightly better.

2.5 Information retention of actual size of event

In our model implementation, we are using the GlobalAveragePooling1D function. Global pooling reduces each channel in the feature map to a single value through averaging. In a CNN, it is meant to take the place of completely connected layers. The plan is to create a feature map for each category that corresponds to the last layer's classification task. But taking the average over all hidden states we may lose information on the size of the event (number of particles), so the question arises : how can we include this information?

In order to deal with this question, we have implemented a model where we included the event size information and compared it to our best performing model to see how it measures up. The architecture of this model is made of alternating graph and dense layers, so as to resemble our best performing model. We introduce the event size information as an input within the concatenate function after the pooling is done, so as to preserve the total event size.

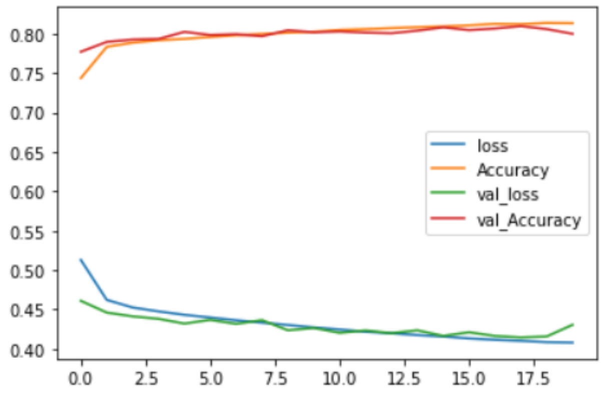


Figure 17: Performance of the model with the event size information included

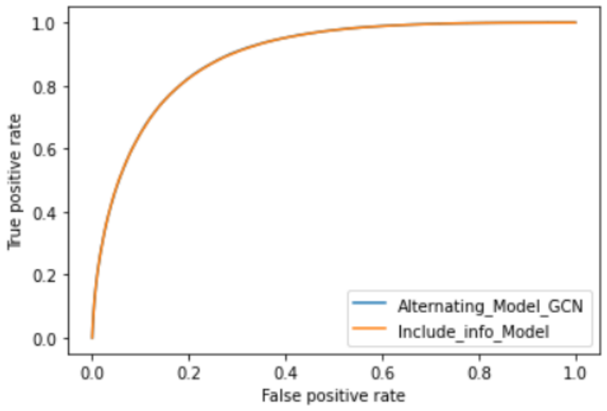


Figure 18: Roc curves of our best performing model and model with event information included

As we can see from the roc curves, the performance of our new model with inclusion of event size information is almost the same as our best performing alternating model. We have observed the speedup to be slightly more in case of the event information being included (around the threshold of 0.005 improvement) and the accuracy has improved by a factor of 0.001. As we are dealing with training several models with very small changes in overall performances, we think this is an important observation with respect to the experiment.

2.6 Dealing with overfitting

2.6.1 Early stopping

Early stopping involves stopping the training of the model when the performance of the validation set stops improving. This helps to prevent the model from continuing to improve on the training set at the expense of the overall generalization performance. When we observe the validation accuracy becoming equal to the accuracy of the model, it is best to stop the training as otherwise it will inevitably lead to overfitting. We have attempted in our plots and training to early stop the model before overfitting can happen. We will include an example plot of what happens when the model is left to train too long (in this case, 100 epochs when only 20 was necessary for optimal training)

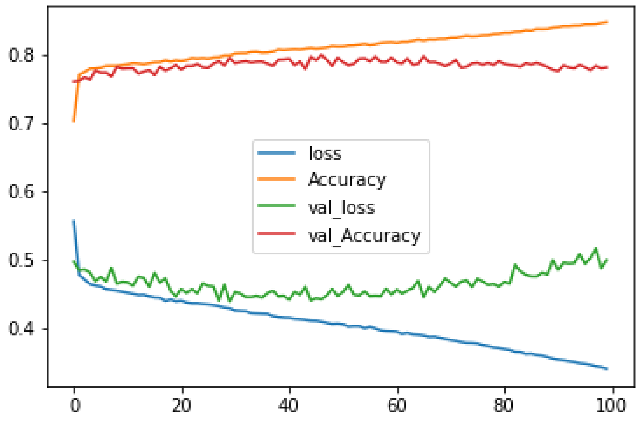
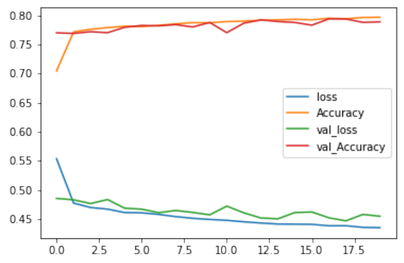
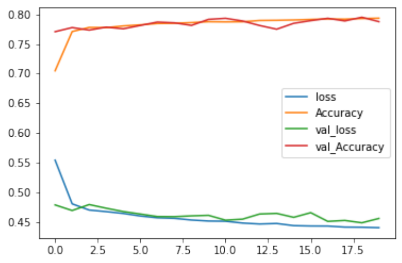


Figure 19: Example of overfitting when model is left to train for too long

2.6.2 Dropout

Dropout is the method through which certain percentage of neurons are randomly chosen to be dropped out during training. This helps us regularize the model as the model is prevented from always relying on a small number of neurons. We ran our model for 20 epochs, for three different rates of dropout : 0.05, 0.1, and 0.2. As we can see the performance of the fit and accuracy increased slightly as we increased the dropout rate. By increasing dropout, we can encourage the network to rely on multiple pathways and reduce the chances of over-reliance on specific features or connections, thus reducing overfitting.
We found out that the model with dropout set to 0.2 performed best with accuracy coming to 0.81 (approximately), followed by the models with dropout set to 0.1 and 0.05 respectively. This is consistent with our knowledge on how





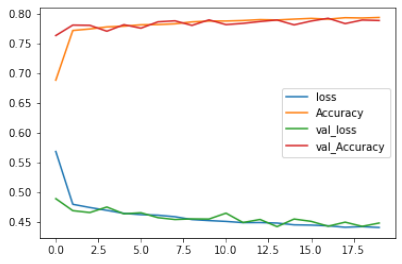


Figure 20: Minor but noticeable changes in model performance, with dropout set to 0.05, 0.1, 0.2 respectively

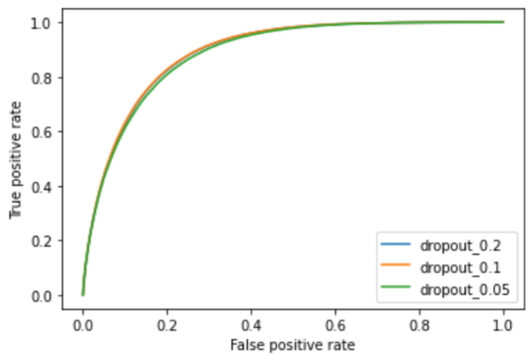


Figure 21: Roc curves of models with different dropout rates, for added clarity

increasing the dropout affects model performance. The threshold of maximal speedup was 0.914, 0.850 and 0.881 for models with dropouts 0.2, 0.1 and 0.05 respectively. With our observations, we can say the model with dropout 0.2 is easily the most optimized one with respect to our working data.

3 Discussion

The improvements made to the model by the different changes in the architecture were mostly small in terms of the accuracy or the AUC. However small changes in those two metrics can lead to quite significant changes in the maximal speedup the model can produce. The calculation of the maximal speedup can be found in the Appendix. Since the last layer of the model is a single sigmoid function, the output of the model will be a number between zero and one. We can then freely select at which threshhold we want to discard events and calculate the speedup for every possible choice. This leads us to the maximal speedup, which might be higher for models with a slightly lower AUC value, as can be seen in the comparison of the alternating and the last layer model, see appendix.

The general approach of discarding some events before the simulation leads to a quite significant speedup, the best performing models had, under our assumptions about the relative time of event simulation and generation a speededup factor around 6.87. However there might be reasons why we wouldn't want to discard events before we know for sure if the event is one we want to use in our experiment. The model will always discard some events incorrectly. This might introduce an error for example in calculating the decay rate of the B meson or in the branching fractions of the particles created. This is due to the fact that the model will discard events with certain characteristics. The complexity of the model doesn't allow us to completely determine which characteristics are used to discard events and therefore we do not have a way of correcting for the bias introduced.

A Estimation of speedup

In order to calculate the speedup resulting from discarding some events by the neural network, one can compare the time it takes for the full simulation with neural network to the time it takes without one.
The ratio between those times is

$$\frac{T_{nn}}{T_{wo}},$$

where T_{nn} and T_{wo} are given by

$$T_{nn} = \frac{(t_{sim}(N_0 f_p + N_1 t_p) + t_{gen}(N_0 + N_1)}{N_1 * t_p}$$

$$T_{wo} = \frac{(t_{sim} + t_{gen})}{0.05}.$$

N_0 and N_1 are the number of passed and failed events when not using a neural network respectively. t_p and f_p are the true positive and false positive rate. Assuming $\frac{N_0}{N_0+N_1} = 0.05$, $t_{sim} = 100 \cdot t_{gen}$ one arrives at the final speedup ratio

$$\frac{T_{nn}}{T_{wo}} = \frac{101 t_p}{100(0.95 f_p + 0.05 t_p) + 1}.$$

One can now use the values for t_p and f_p provided by the ROC-curve to calculate different speedups for different thresholds.

B Overview of model performances

Model	Loss	Accuracy	AUC	Maximal speedup	Threshhold*
Alternating_GCN	0.411734	0.811120	0.893008	6.873146	0.926112
Last_Layer_GCN	0.401771	0.816695	0.896407	6.740350	0.901285
No_Layer_GCN	0.416353	0.808420	0.887182	5.834084	0.882226
Last_Layer_GCN_Pooling	0.395405	0.820592	0.898588	6.744168	0.908284
Alternating_GCN_Pooling	0.414614	0.812010	0.893148	6.750855	0.938821
Include_info	0.408662	0.812290	0.892707	6.878151	0.911553
Feature_Scaling_Alternating	0.483545	0.767370	0.846796	5.272123	0.880613
dropout_0.2	0.407415	0.813990	0.891173	6.054597	0.914436
dropout_0.1	0.411071	0.812185	0.892124	6.029696	0.850808
dropout_0.05	0.417075	0.806802	0.885654	5.974291	0.881039

*: Threshhold used to obtain the maximal speedup

References

[1] Thomas N. Kipf. *Graph convolutional networks*. URL: `https://tkipf.github.io/graph-convolutional-networks/`.

[2] Manzil Zaheer et al. *Deep Sets*. arXiv: `1703.06114`.