UPPSALA
UNIVERSITET

Applying Neural Networks in Data Analysis

Tobias Nordahl, Tim Littau

Project in Computational Science: Report

January 2019

PROJECT REPORT

Institutionen för informationsteknologi

Abstract

This work investigates the efficiency of using neural networks in analysis of data obtained by the Beijing spectrometer. The Beijing spectrometer analyses a wide variety of hadronic processes produced in electron-positron collisions. In this particular project the focus lies in the analysis of the Λ-hyperon decay.

As in many other problems in this field of research, the discrimination of signal and background events is crucial to perform further analysis.
In this project the neural network model of choice, to achieve such discrimination, was the Bayesian neural network. In contrast to conventional neural networks the Bayesian approach uses distributions for its weights instead of fixed numbers. This makes the model more adaptive in the choice of optimization during the training. The results show the usage of neural network techniques as a tool to detect specific patterns within the data.
Especially the Bayesian approach seems to perform even better than conventional Neural Networks.

With further investigation of the algorithm and adjustments on the training, this technique could easily be applied for data other than the Λ-measurements.

Contents

1 Introduction

The Beijing spectrometer BESIII is located in Beijing, China and is designed to study the physics of charm, charmomium and light hadron decays [BES].

The BESIII group at Uppsala University focuses on hyperons to investigate strong interaction between quarks. The strong interaction is responsible for almost 99 % of the mass in the universe.
In order to pursue this research in a desirable manner, only the decay events of interest need to be investigated. Every event other than those are considered background events.

The aim of this project is to provide an algorithm that detects the patterns of signal events and filters out the background events as much as possible using neural networks.

1.1 Problem Formulation

To specify the scope of this project there are the following tasks to solve:

- Analysis of the provided data sets and setup of the final input
- Development of an appropriate neural network model
- Testing and improving the developed model

In this study data analysis does not have to be performed in real-time. Hence, there are no hard limitations on how fast the algorithm is supposed to be.

2 Theoretical Background

This section outlines the underlying Hadron-physics of interest for this project and provides a basic explanation of neural networks and the final model of choice.

2.1 BESIII

The provided data set is collected with the Beijing Spectrometre III (BESIII). As described [Asn08], BESIII performs its measurements at the Beijing Electron-Positron Collider II (BEPCII). The BEPCII operates within an energy region of 2.0 - 4.6 GeV. When an electron and a positron collide they annihilate each other and one resulting hadron is the J/Ψ-Meson which decays in almost every case via strong or electromagnetic interaction into other hadrons. One possible decay is resulting in a Λ and a $\bar{\Lambda}$-Hyperon.
In this project the focus is on the events with such decays and especially the Λ-Hyperon.

2.2 Λ-Hyperon

Hyperons are a special case of baryons with three quarks in total and at least one strange quark, for example (uds) which would be a Λ-hyperon. Specifically the Λ-hyperon has one strange (s), one up (u) and one down (d) quark. Λ is electrically neutral, it has a mass $M = (1115.683 \pm 0.006)$MeV and a life-time of $\tau = (2.632 \pm 0.020) \cdot 10^{-10}$s. In $(63.9 \pm 0.5)\%$ of the cases Λ decays into a proton p and a negative pion π^- [Bie] and with $(35.8 \pm 0.5)\%$ into a neutron n and a neutral pion π^0. Every other decay measured has chances of $(1.75 \pm 0.15) \cdot 10^-3$ or less to happen [Tan18]. Especially the decay into n and π^0 will not be detected since they are electrically neutral.
The Λ-Decay can be characterized by topology and kinematics. The topology is mainly defined by the decay vertices which are given as points in the three-dimensional space, whereas kinematics is entirely characterized by the four-dimensional momentum vector $P = (E, p_x, p_y, p_z)$. Using the four-vector to calculate invariant Mass as $M = \sqrt{E^2 - ||p||^2}$ provides a quick option to analyze a data set of decay events.

2.3 Neural Networks

A neural network in the subject of machine learning is inspired by the biological brain itself. In mathematical expression, each neuron in an artificial neural network acts as a non-linear function that can be associated with certain aspects of the input.
The usual way of creating a neural network that is supposed to optimize a specific task, is by training it with back propagation. Starting with a random state of the neurons, the connections weights between the neurons are adjusted as the network sees already known data.
Especially in recent years neural networks have proven to be a great tool for binary classification of signal and background events. Not only because they are easy to use , but also quite efficient in their analysis, neural networks are nowadays extensively used not only in particle physics [ZHL11].
In Figure 1 a schematic representation of a one-layer neural network is shown. The mathematical representation of this example of neural network is shown in equation 1.

$$Y = \mathrm{a}(W^{(2)\mathbf{T}} \cdot (W^{(1)\mathbf{T}} \cdot X + b^{(1)\mathbf{T}}) + b^{(2)\mathbf{T}}) \tag{1}$$

Here a is some activation function, $W^{(i)}$ are the weight-matrices, $b^{(i)}$ are the bias vectors, X the input vector and Y the output, in this case one number.

2.4 Bayesian Neural Network

To train a neural network with a supervised learning approach it is necessary to provide it with a labeled training data set. In this specific kind of problems such labeled training data is usually obtained by Monte-Carlo simulations of the corresponding experiment. When a conventional neural network is trained with such training data, one has to be additionally cautious with over-fitting.

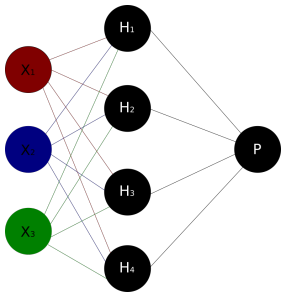

Figure 1: Schematic representation of a neural network

Since reality rarely results in the exact same outcome as a simulation, the trained neural network should offer more flexibility to predict unlabeled data obtained in real measurements.

Bayesian neural networks (BNNs) follow the bayesian inference theorem.
The weights in the BNN are prior distributions instead of fixed values like in the conventional neural networks. Rather than adjusting the weights step by step, the BNN is adjusted based on a collection of networks among which the most optimized one is chosen, based on the loss-function. The advantage of BNNs is they are less prone to over-fitting and their size can actually be smaller compared to conventional neural networks [CBBP06].
Therefore, they offer a higher flexibility and are therefore expected to be better suited for problems, as the one presented in this work.

3 Interfaces and Frameworks

For this project, two available options of interfaces have been found. On one hand, ROOT with TMVA is a quite common tool already used in physics for data analysis. On the other hand Tensorflow is a viable alternative, since it is a state-of-the-art machine learning framework.

3.1 ROOT and TMVA

ROOT is written in C++, but there are also Python bindings available. The development was initiated in 1994 by scientists at CERN, so ROOT is now over 20 years in usage [oM]. The TMVA library adds additional possibilities to perform multivariate analysis on the desired data set. But not only does it perform multivariate analysis, it also does some necessary preprocessing of the data and displays this analysis of the input set [CER].
In this project TMVA is especially used for the input analysis to easily obtain information of the input by just using the original data sets given in ROOT-files. One example is the correlation of data within the training data set split for each, the signal labeled data and the background labeled data.

3.2 TensorFlow

TensorFlow is a state of the art machine learning framework. It is mainly developed for python.
Additionally to TensorFlow, the built-in API Keras is used to ease the implementation of neural networks. TensorFlow itself offers an additional library called TensorFlow Probability, which can be used to implement a BNN as described above [ea15]. Below is a short example of how a BNN would be implemented with Keras and Tensorflow Probability.

```
with tf.name_scope("bayesian_neural_net", values=[input]):
  neural_net = tf.keras.Sequential([
      tfp.layers.DenseFlipout(22, activation=tf.nn.relu),
      tfp.layers.DenseFlipout(2)
  ])
```

Keras provides the functions to create the basic structure of the neural network while TensorFlow Probability is used for the special distributional weight layers. Further preprocessing of the data is done with common Python libraries such as Numpy. In the final implementation the ROOT library for Python is used with an extra library called Root_Numpy to implement a ROOT-importer and exporter for even more simplicity.
Due to simplicity of implementation, Tensorflow with Keras are the interface of choice for the final neural network implementation. The provided data set is already in ROOT-format, so for input analysis ROOT and TMVA are easier to use and these are the tools of choice for the Data Mining task.

4 Data Mining

When implementing a neural network as for many other machine learning techniques it is important to know what kind of data we are analysing. For instance it would be useless to train the model on an observable that does not tell at all whether the output should be a signal or not. So when training the neural network, first of all the observables of most interest have to be chosen as input when training the neural network. Furthermore, the observables have to be checked for correlation. If they are correlated, it is advised to apply some decorrelation technique on the data. Since the neural network implemented in this project is based on supervised learning, there has to be a training data set which is completely labeled. This training data set has to be created manually from the provided data sets.

4.1 Labeling of the Data

The provided data itself is not labeled. But in fact there was an experimental data set, containing close to 90% background events. The signal-labeled data originated from a Monte-Carlo simulation.
To determine the amount of background in the background-data set,

the histogram of the invariant mass is investigated. The invariant mass of a particle can be calculated using the energy and the momentum of that particle. The invariant mass varies for different particles. If the invariant mass of one observable from the data set is far away from the invariant mass of a lambda particle it is likely not to be a lambda particle and this can be seen by plotting a histogram of the invariant mass as in Figure 2.The first idea is to take this data set exclusively and label the events in the signal area of the invariant mass as signal and the rest as background. But here in fact the background and the signal are overlapping, so the signal labeled data set would be highly contaminated with background events.
As already mentioned, the final training data set consists of the Monte-Carlo simulation data labeled as signal and the experimental measured data labeled as background.
Even though there are signal-like events in the background labeled data set, the additional flexibility of the model is expected to most probably ignore the signal events which are labeled as background and rather "remember" the real background-like patterns.

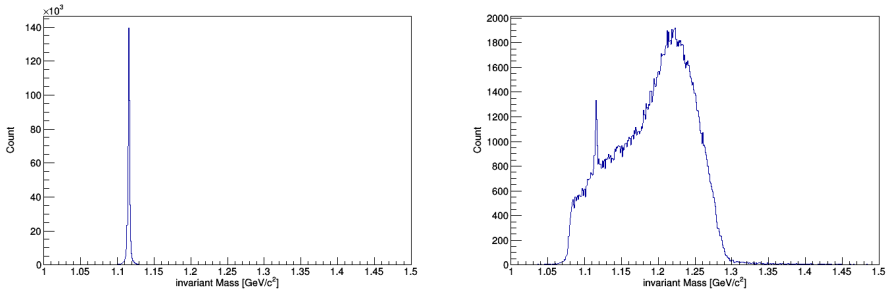

Figure 2: Histogram of the invariant mass from experiment(right) and Monte Carlo simulation(left).

4.2 Properties of the Data Set

The performance of the neural network depends on which observables are included. It is unnecessary to include observables that do not give any information whether a particle is a signal or not in the BNN and even worse the weights of the BNN can be trained badly since the BNN will base its predictions on information that does not tell if a particle is a signal or not. Therefore, before training it is important to first analyze the data set.
The observables from the data set that are provided can be divided into kinematic observables and topological observables. The kinematic observables are hyperon momentum and energy of the Λ-hyperon. The topological observables are decay length, track parameters and vertex coordinates. The next step is to decide which observables to include in the training. Since the observables previously mentioned should indicate whether an event is a signal event or background event, the final decision after some analysis is to include both kinematics and topology when constructing the training data set. Now after deciding which

observables to include in the training, these have to be checked for correlation. If there is correlation between the chosen observables, they should be decorrelated in some manner. In this project, if the data is correlated Principal Component Analysis (PCA) would be the method of choice.

4.2.1 Principal Component Analysis

Principal component analysis is a mathematical method which can be used to decorrelate the variables and can also be used to reduce the variable space. To find the principal component, which can be thought of as to be the structure of the data, the eigenvalues and the corresponding eigenvectors are calculated. This means that if one is using for instance five observables one gets five eigenvectors and eigenvalues. Each eigenvalue corresponds to one observable. Once the eigenvalues and the eigenvectors are calculated the eigenvector with the largest eigenvalue has to be found, since this will be the principal component. The eigenvectors can now be used to represent the data set instead of the regular observables and if this one eigenvalue is close to zero it would mean this observable does not contribute much to the structure of the dataset and is more or less useless. Then that eigenvector corresponding to the small eigenvalue is removed and the whole data set can still be completely represented without losing too much information since the removed eigenvector does not tell much wether an event is a signal or not. [Shl14]

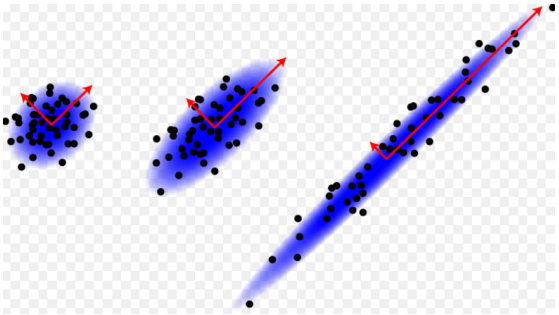

Figure 3: Demonstration of two principal components (i.e. two eigenvectors from the data set). The data set to the right has one large eigenvector and one small. This means that the large one has a larger eigenvalue and is therefore more relevant to the structure of the data set. [PCA]

4.3 Methods of Preprocessing

All the data sets are provided as ROOT-files. The neural network is built in TensorFlow, to use the data within Python the file is converted from root to a comma separated value (CSV) file. The coding language for root is C++ so to convert the ROOT-file to a CSV file a simple C++ script is used, which is compiled and executed in ROOT to read in a ROOT-file and produce a CSV-file. In Figures 4 and 5 the correlation matrices of the signal and the background data set are shown. As the analysis on those data sets does not indicate that the

data is correlated, the rest of the data as not been subjected to decorrelation. Applying PCA or any method of decorrelation on an already uncorrelated data set usually delivers no improvement or in worst-case a degradation of the input. High correlation means either a proportional behaviour for positive or inverse proportional behaviour for negative correlation between two variables.
There is actually a slight correlation between the vertex coordinates and the momentum in the signal data. An explanation would be that the further a Λ-hyperon travels before decaying, the higher its momentum becomes. This can however not be seen in the background labeled data and therefore PCA is not chosen to be applied.

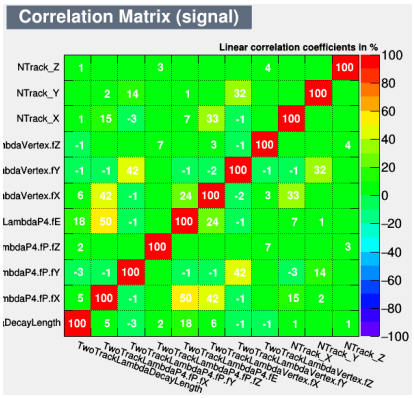

Figure 4: Correlation Matrix of the signal data set

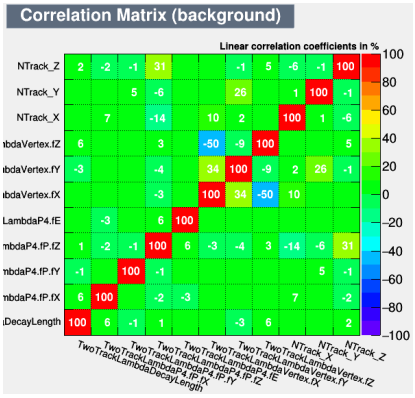

Figure 5: Correlation Matrix of the background data set

4.4 Final Input Data Set

The final decision is to use both kinematics and topology for training, since the kinematic observables contain crucial information to detect patterns of signal events. The training data set should contain an almost equal amount of signal and background events to reduce bias. So in this case the size of the whole training data set is defined by two times the size of the smallest of those two (signal or background) data sets. As for many machine learning models the labeled data set is not used entirely for training. Instead, a part of the labeled data is left out as testing data so the model can be tested on some labeled data. There is no exact rule on how the whole data set should be split into training and testing data, but a guideline is to use two thirds or more for training and the rest as testing data. In this project the size of training data is 75% of the labeled data set, while the remaining 25% of it is taken as testing data.

Total number of labeled events:	400 000
Training events:	300 000
Testing events:	100 000
Observables used:	• Lambdavertex • Trackparameters • Momentum • Energy

Table 1: Total number of events in each splitted labeled data set and final choice of input variables

5 Implementation

For the final implementation of BNN, there are overall ten input variables. Vertex Coordinates, Track Coordinates, 4-Vector and the decay length. The input variables define the input space and therefore also the number of input nodes.
BNN should remain simple with not more than 100 nodes in order to keep training performance and efficiency optimal. In this work BNN with one hidden layer is used, as also mentioned in [PEB07] for a similar problem. After testing different amounts of hidden nodes, the final number is 22 hidden nodes. Especially for BNN, too many hidden nodes would unnecessarily increase computation time but not improve any results on predictions of other data than the training data [PEB07].
The final network expressed in mathematical form, reads

$$Y = \text{sig}(W^{(2)\mathbf{T}} \cdot (W^{(1)\mathbf{T}} \cdot X + b^{(1)\mathbf{T}}) + b^{(2)\mathbf{T}}) \tag{2}$$

$$W^{(1)} \in \mathbb{R}^{(10x22)}, b^{(1)} \in \mathbb{R}^{22} \tag{3}$$

$$W^{(2)} \in \mathbb{R}^{(22x1)}, b^{(2)} \in \mathbb{R} \tag{4}$$

$$Y \in (0, 1). \tag{5}$$

Here $W^{(i)}$ are the weight-matrices, $b^{(i)}$ are the biases, Y is the output, X is the input and sig() is the Sigmoid function which guarantees a result in range (0, 1). The layers are specified as DenseFlipout layers of TensorFlow Probability, which have prior normal distribution for the weights before the training. They also apply a built-in loss function with the Kullback-Leibler divergence, which is necessary since these are probability distributions. The Kullback-Leibler divergence as in [MKH18], is especially used when looking at variational inference and a better choice to optimize the weight distributions. Furthermore, the Adam Optimizer is used. [KB14] shows the Adam Optimizer is an alternative to the classical stochastic gradient descent procedure to update the network weights. It is computationally more efficient and already a popular choice for deep learning solutions. Both methods are included by the TensorFlow API.
For every predicted event the algorithm performs 50 predictions, which can slightly differ due to the nature of BNN and take the mean of them as the prediction probability of the event being a signal.

6 Model Results and Performance

In this section the results of the neural network when applied to the unlabeled data set are presented. Also the results from the training session are shown and discussed to demonstrate the choices on amount of training iterations and final decision cut. Finally there is a discussion about training of the neural network and ways to investigate whether the neural network is over-trained or not.

6.1 Training Session

While training, for each iteration the neural network is provided with one batch. According to the results the weights of the neural network are then optimized by reducing the loss-function. When increasing the number of iterations during the training session the weights eventually get optimized in a way that the neural network labels all training data correctly. Accuracy as a measure of quality for prediction is defined as the fraction of correctly labeled events in a batch. One might think it would be good to have a number of iterations as high as possible to get an accuracy close to 100% for the training data. But this is not recommended because then the network becomes over-trained. This means the weight distributions are optimal to predict the training data only. Any data unknown to the neural network would then most probably be predicted wrong, since the network is heavily biased towards the training data. One wants the model to be flexible to new data the neural network has not yet worked on and if the network is over-trained this flexibility is reduced. A good way to choose the right number of iterations is by plotting the training accuracy versus the number of iterations and investigate at which number of iterations the accuracy starts to saturate.

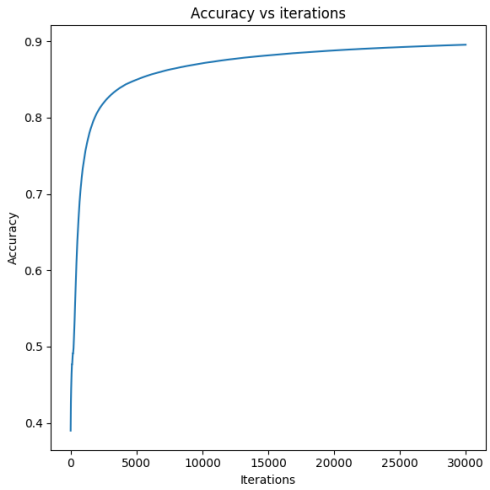

Figure 6: Accuracy vs number of iterations during the training phase. The accuracy is plotted for every tenth iteration.

Figure 6 shows that the accuracy starts to saturate at around 2500 iterations and is almost saturated at 5000 iterations so the number of iterations used should be somewhere in between 2500-5000. This can similarly be done for the batch size. Finally, 3000 iterations are used for the training of the neural network. The next step is to look at how well the neural network manages to separate the testing data by plotting a histogram of its predictions. This plot gives a hint whether the neural network is over-trained and is shown in Figure 7.

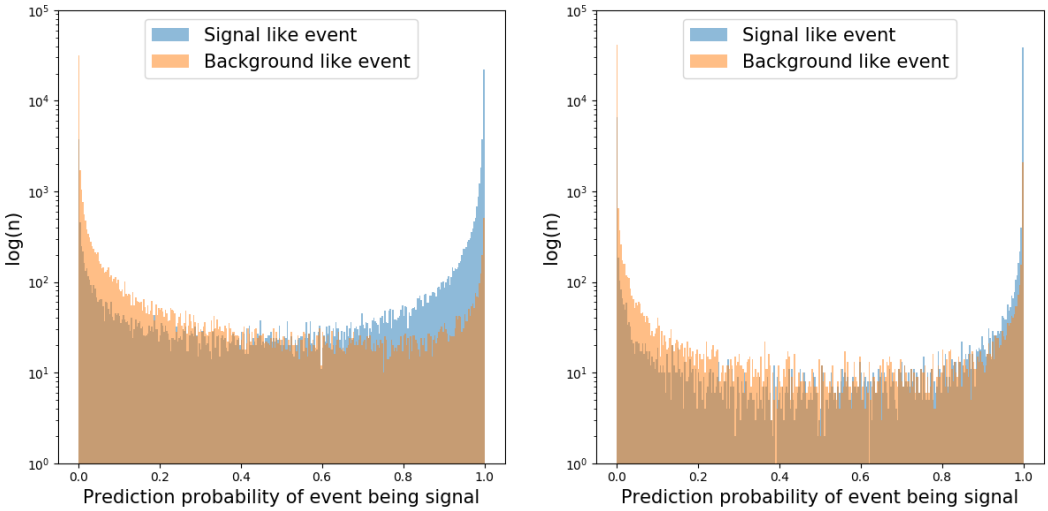

Figure 7: Prediction probability of an event being signal for the testing data during the training phase for 3000 iterations(Left) and 50000 iterations(Right)

The left histogram in Figure 7 shows the neural network when trained for 3000 iterations and the right histogram when trained for 50000 iterations. The blue histogram shows the signal like events and the orange histogram the background like events. On the x axis the prediction probability that an event is a signal is shown. The closer a prediction is to one the more confident the neural network is that this is a signal event. The brown histogram shows the overlap of the two histograms or so to speak the contamination, which means these are background events predicted as a signal event and vice versa. The total contamination for both cases is around 12 % but the difference is that for the over-trained network the network is more certain about its predictions even though the prediction might be wrong. When the neural network gets overtrained it becomes harder to label new data correctly as previously mentioned.
The output of the neural network is given in probabilities of how certain it is that an event is a signal i.e. a value close to zero is more likely to be a background event and a value close to one is more likely to be a signal event. To specify the classification even further, a decision cut is implemented. The cut is used to label the data and since the output is given in probabilities one has to decide at which probability should events be labeled as signals. To decide at what probability the final decision cut should be, the true positive rate and the false

positive rate are investigated. Labeling a true signal as a signal is called a true positive, while labeling a background event as a signal event is called a false positive. For the final cut the number of true positives should be maximized while at the same time the number of false positives should remain small. For the final decision cut the true positive rate and the false positive rate are plotted versus the decision cut as shown in Figure 8.

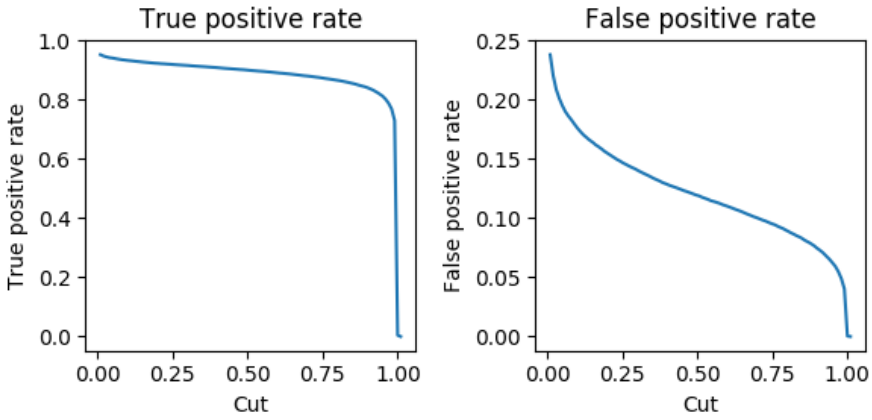

Figure 8: True positive rate (left) and False positive rate (right) plotted over decision cut

Based on Figure 8 the decision cut is set to 0.75 to achieve a low false positive rate at 10% and a high true positive rate at around 88% to keep as many signal events as possible.
In Table 2 the final set of parameters is shown.

Parameters	
Iterations:	3000
Batch Size:	100
Decision cut:	75%

Table 2: Final parameters for the neural network

6.2 Signal and Background Discrimination

After training the neural network it is time to predict unlabeled data. Once the neural network is applied to the data, the amount of signal events in the signal area and the amount of background events in the signal area are investigated to calculate the level of background. To count the amount of signal events in the signal area one plots a histogram of the invariant mass against the momentum as presented in Figure 9 where the invariant mass is plotted on the X-axis and the momentum is plotted on the Y-axis. Figure 9 is then divided into three regions of the same size, such that there are two background areas with one signal area

in between. The next step is to count the amount of events in the three different areas and estimate the amount of signal events as in equation 6 where S is the estimation of signal events in the signal area, N is the number of events in the signal area, B_1 and B_2 is the number of events in the two background areas. The background areas are typically called sideband regions [WLSZ14].

$$S = N - \frac{B_1 + B_2}{2} \tag{6}$$

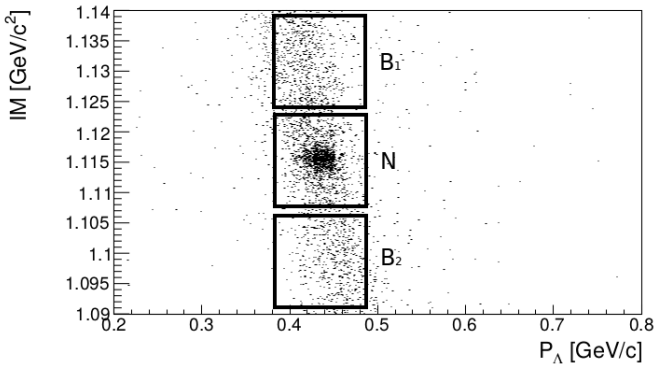

Figure 9: Two background areas and the signal area in between. Invariant mass on the X-axis and the momentum on the Y-axis

In Table 3 the level of signal events and the background contamination before and after applying the neural network are shown. As shown, the amount of signal events is reduced by almost 29.8%, however the level of background is significantly reduced from 35% to 7.8%. Finally, the overall accuracy on the unlabeled data set is 85%.

Before Neural Network	
Signal events:	1094
Level of background:	35%
After Neural Network	
Signal events:	768
Level of background:	7.8%
Accuracy	
	85%

Table 3: Final result from the unlabeled data in the signal area

Figures 10 and 11 show a comparison between the entire unlabeled data set and the signal predicted events. Figure 10 depicts the invariant mass calculated as

defined in Chapter 2.2. The left histogram in Figure 10 shows the distribution of the entire data set and the right histogram in Figure 10 shows the distribution of events predicted as signal. In Figure 10 a significant reduction of background events an be observed. In Figure 11 2D-histograms of the vertices in X and Y direction are shown. The left histogram is the entire data set and the right histogram the signal predicted events. Figure 11 shows the reduction of background especially at the outer circular region which comes from events hitting the beam pipe.

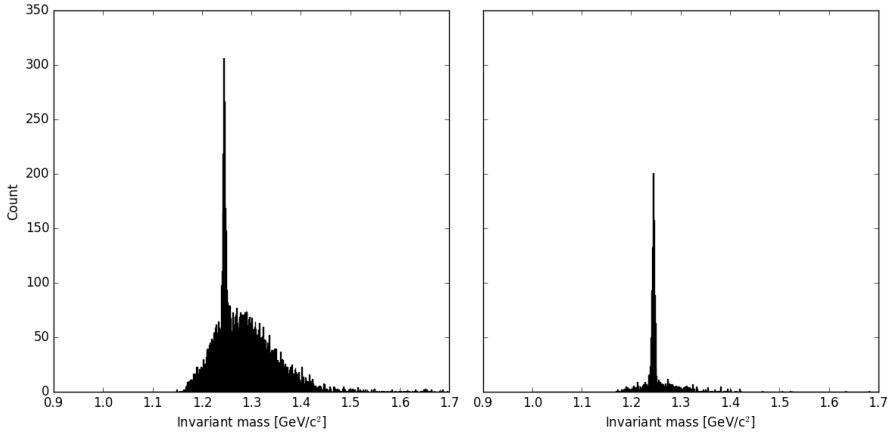

Figure 10: Invariant mass before the neural network is to the left and after is to the right.

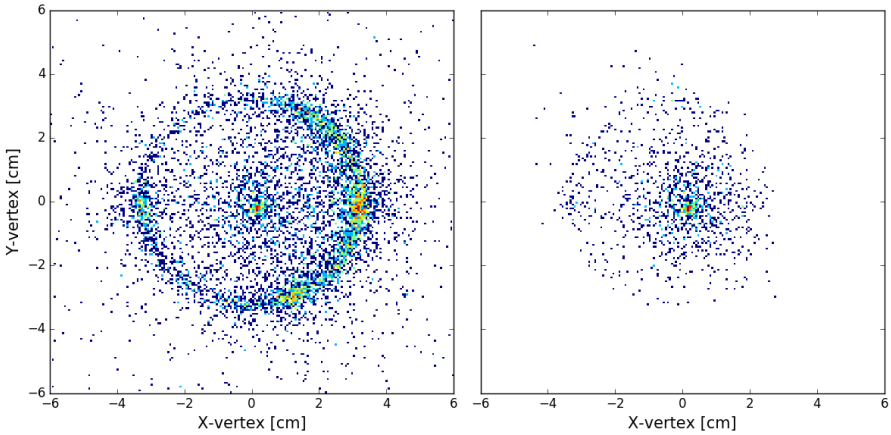

Figure 11: 2D-Histogram of the Λ vertices in X and Y direction before the neural network is to the left and after is to the right.

7 Discussion

Even though the results are quite satisfying, the loss of signal data after applying the neural network could be reduced by using a more realistic signal simulation. In the following paragraphs, the problems and advantages of the work are pointed out.

7.1 Labeled Data

In general, a constraint for supervised learning is the reliance on the training data set. If the training data set leaves out important information which should be used for future applications, the network will have problems to correctly predict those events.
A usual procedure for tasks as presented in this project is to use Monte-Carlo simulations as labeled data for both the background and the signal events. In this project the background simulation has not yet been performed. Therefore an experimental data set with almost only background events was used instead. This of course can lead to some problems concerning the detection of signal events, since there are still some signal events in the background labeled training data.
In theory the use of a BNN instead of a conventional neural network is supposed to at least reduce this source of error due to its higher flexibility.

Additionally, even though the Monte-Carlo simulation data set is supposed to be only simulated signal-events, it had a small fraction of background events included.

Considering this set up labeled training data set, the results are in fact astonishing and proved the capability of the Bayesian approach to be very adaptable and flexible.

7.2 TMVA

Implementing TMVA was quite a challenge. Even though it was used for input analysis, the implementation of a multi-layer perceptron (MLP) was never fully realized with comparable results.
Therefore the implementation was dropped for the sake of improvements on the BNN implementation.

8 Conclusion

To conclude, this project is finished with an algorithm to perform a discrimination of signal and background events for measurement data of the Λ-decay. The results of such discrimination with a reduction of background from originally 35% to 7.8% were astonishingly satisfying.

The algorithm is written in a modular way in order to offer the possibility of adaptation for other analysis than the Λ-decay.
Possible improvements of the results could be achieved by using more precise Monte-Carlo simulations which are completely labeled. Or in general a labeled data set with even less or none contamination in the labels.

8.1 Outlook

Focusing on the Bayesian approach, there was recently an implementation of a BNN in ROOT, which could be adjusted to match the BNN presented in this project [ZHL11].
Future research could include the application of BNN developed within this project on other decays. Additionally, an investigation of computation-time was not performed in the scope of this project. This could be done to offer an option of predicting the events in real-time on site.

References

[Asn08] D. M. et al. Asner. Physics at BES-III. *arXiv e-prints*, September 2008.

[BES] Beijing spectrometre (besiii) experiment. `http://bes3.ihep.ac.cn/`. Accessed: 2018-01-09.

[Bie] Ph.D. Jacek Biernat. Hyperon structure with besiii. `https://www.jlab.org/conferences/hyp2018/talks/thu/session-a/pm-a2/07_biernat.pdf`. Accessed: 2018-12-11.

[CBBP06] PUSHPALATHA C. BHAT and HARRISON B. PROSPER. Bayesian neural networks. *Statistical Problems in Particle Physics, Astrophysics and Cosmology - Proceedings of PHYSTAT 2005*, pages 151–, 05 2006.

[CER] CERN. TMVA. `https://root.cern.ch/tmva`. Accessed: 20108-12-11.

[ea15] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[MKH18] Vikram Mullachery, Aniruddh Khera, and Amir Husain. Bayesian neural networks. *CoRR*, abs/1801.07710, 2018.

[oM] University of Minnesota. What is ROOT? `https://zzz.physics.umn.edu/computing/contrib/root/localdoc#recommendations_to_root_users`. Accessed: 20108-12-11.

[PCA] Shape of data principal component analysis. https://shapeofdata.wordpress.com/2013/04/09/principle-component-analysis/. Accessed: 2019-01-09.

[PEB07] M. Pogwizd, L. J. Elgass, and P. C. Bhat. Bayesian Learning of Neural Networks for Signal/Background Discrimination in Particle Physics. *ArXiv e-prints*, July 2007.

[Shl14] Jonathon Shlens. A tutorial on principal component analysis. *CoRR*, abs/1404.1100, 2014.

[Tan18] M. et. al. Tanabashi. Review of particle physics. *Phys. Rev. D*, 98:030001, Aug 2018.

[WLSZ14] Yadi Wang, Beijiang Liu, Xiaoyan Shen, and Ziping Zhang. Background subtraction using probabilistic event weights. *CoRR*, abs/1401.6813, 2014.

[ZHL11] J. Zhong, R.-S. Huang, and S.-C. Lee. A program for the Bayesian Neural Network in the ROOT framework. *Computer Physics Communications*, 182:2655–2660, December 2011.