Machine Learning based on KERAS for Single Top Physics at ATLAS

Theodor Bettray

Bachelorarbeit in Physik
angefertigt im Physikalischen Institut

vorgelegt der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität
Bonn

September 2019

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie die Zitate kenntlich gemacht habe.

Bonn,	. . . .	. . . .
	Datum	Unterschrift

1. Gutachter:	Prof. Dr. Ian C. Brock
2. Gutachter:	Dr. Eckhard von Toerne

Acknowledgements

I would like to thank all members of the Brock group not only for their valuable inputs to this thesis, but, in particular, for the delightful working atmosphere.

Contents

CHAPTER 1

Introduction

Machine Learning (ML) is increasingly becoming a tool for analyzing complex patterns in physical sciences [1]. Complexity can stem from a high number of variables, non-linear dependencies, huge amounts of data, and high dimensional data. Data collected in particle detectors has all of those complexities, hence it is useful to apply ML techniques. The basis for this thesis is a Neural Network Code (*NNC*) skeleton developed within the tW-Group of ATLAS based on the KERAS package, an Open Source Deep-learning library initiated by Francois Chollet [2]. With its public availability it is highly desirable to further develop that skeleton into a framework interface for particle physics analyses. The main goals of this thesis were

- to debug and amend the code skeleton and harmonize the code syntax to work with python 3.7 and within the IT environment provided for in the Physikalische Institut (PI),
- to update all links to external libraries; KERAS in particular,
- to document the main functional elements of the code and identify data input interfaces, and
- to create a list of further improvements opportunities.

so that new users have a fast introduction to the code and save substantial time and effort in understanding and using the code.

The testing of the code has been done with data derived from the ATLAS single top-quark analyses [3]. The physics model, some main features of the used data structure and the main characteristics of neural networks will be briefly described in chapter 2, followed by a description of the amended NNC in chapter 3. The NNC is then optimized for a Di-lepton dataset. In a next step its performance is applied to a single-lepton dataset, which has previously been analyzed by an ML packgage called *NeuroBayes®*, This package is a commercial software distributed by Physics information technology.[1] Since this package is supposed to lose its comprehensive support for scientific research, an improved version of NNC might become the ML package for the tW-Group. Based on the code status and documentation provided by this thesis, it should allow fast access to further analyses and improvements.

[1] phi-t, based in Karlsruhe, Germany

CHAPTER 2

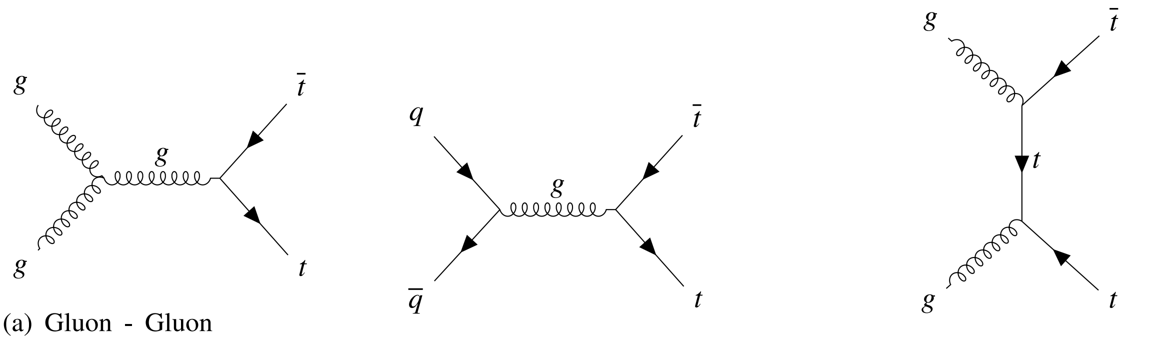
Foundations

2.1 Top-Quark Physics

Most of the background needed for this thesis is ML- and IT-related. Although the application of the code itself is agnostic to the meaning and/or interpretation of the data processed, a short characterization of the kind of data used for this thesis should be given. This chapter illustrates where the data used for the machine learning/training in the following chapters is "located" in the overall analyses within top-quark physics.

2.1.1 Top-Quark Production Processes

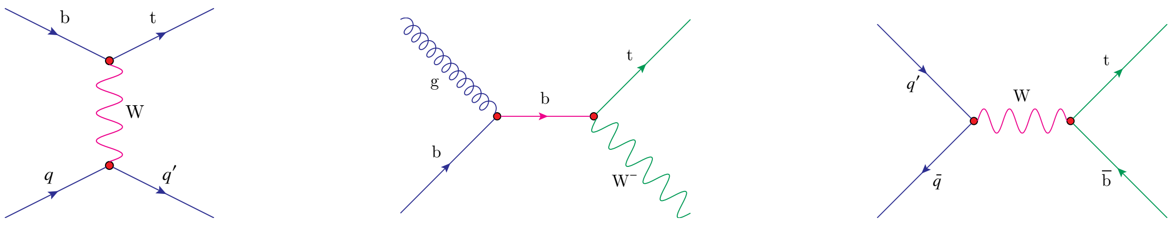
The top-quark is the heaviest of the six quarks in the standard model of particle physics [4] and for that reason especially interesting in high energy particle physics. The top-quark was postulated in the 1970's [5] and discovered much later in the mid-1990's by the D0- and CDF-collaborations at Tevatron ([6], [7]), which for the first time provided enough collision energy to produce top-quarks. At Tevatron protons and antiprotons collided and produced top-quark pairs, which happens via the strong interaction. In the proton and antiproton there are quarks that can annihilate into a gluon with enough energy to decay in two top-quarks. At LHC, which can achieve much higher energies,



(a) Gluon - Gluon fusion (b) Quark annihilation (c) Gluon - Gluon fusion

Figure 2.1: Feynman diagrams for single top production (leading order)

protons are brought to collision, which makes it more likely that two gluons collide. The relevant mechanism are shown in the Feynman diagrams in 2.1. Furthermore, top-quarks can be produced singly via electroweak interactions (see figure 2.2) within 3 main channels [8]. Due to their small cross-sections compared to top-quark-pair production, these channels were more difficult to detect at the center-of-mass-energy levels at Tevatron ($\sqrt{s}$ =1.96 TeV), because of their low cross-section compared to top-quark pair production. Evidence of the electroweak process was found in 2008/2009 [9], [10]. With the energy levels at LHC, evidence of the electroweak top-quark production was presented by the ATLAS and the CMS collaboration in 2012/2013.[8] It can be seen from the graphs



(a) t-channel (b) tW-channel (c) s-channel

Figure 2.2: Feynman diagrams for single top production (leading order)

that the W-Boson for the t- and the s-channel is linked via two vertices to four fermions, whereas for the tW-channel there is only one btW-vertex only. This allows an isolated study of that interaction, which might lead to new insights into physics beyond the standard model [8].

2.1.2 Top Quark Decay Channels

Due to its high mass, the lifetime of the top-quark is very short [4] and hence it does not hadronize. Hence its properties can only be reconstructed from its decay products which is a property unique to top-quarks. For the target region in this thesis with single lepton this results in the final states shown in figure 2.3 [11]. The two final states are identical in terms of resulting particles, hence it is difficult to separate the final states from single top-quark decay from the states of top-quark pair decay. The approach is to do this via large samples simulated by Monte-Carlo methods and reconstructed kinematic variables.

The task for the Machine Learning code in this thesis is to separate the single top decay channel (regarded as **signal**) from the $t\bar{t}$ -production (regarded as **background**).

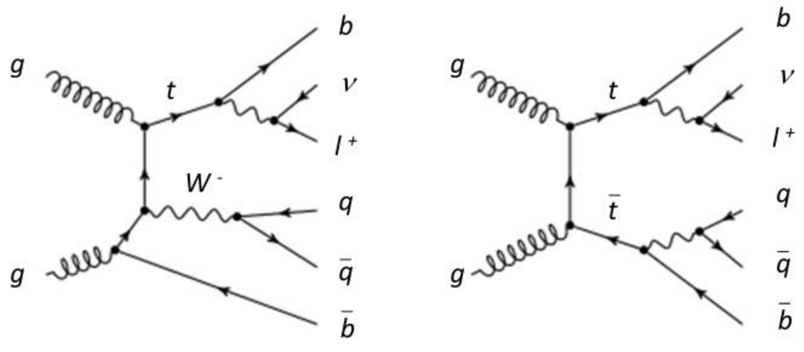


Figure 2.3: Top-quark decay channels (next to leading order)

2.2 Characteristics of Neural Networks

2.2.1 Introduction

Neural Networks (NN) are a set of algorithms that mimic a biological neural network. They are a technique under the umbrella that is machine learning. Machine learning "can be characterized as a subcategory of Artificial Intelligence (AI) which is capable of identifying patterns in large data samples and/or deriving new insights which are not immediately obvious to humans." Because those insights are not a-priori known, there is no way to develop explicit program code with fixed logic structure to arrive at a desired result. The system is learning patterns from examples and can generalize them to unknown examples [12]. Learning strategies are typically [13] classified as:

- **supervised learning**: Learning based on a set of **training** examples where for a given set of variables it is known, whether the result must be labeled signal or background. The learning algorithm will then be applied to a second set of **test** examples (with known labeling). The Neural Network will label the results for these test examples as signal or background based on its structure optimized for the test examples. The quality of the algorithm can then be described by the number of correct identifications of the expected result (true positives) versus the number of incorrect identifications of the expected result (false positives).

- **Unsupervised learning**: The algorithm only receives input data without known labels. It should find distributions within the set of data and deliver new insights about the data.

- **Reinforcement learning**: The development of the algorithm is based on a reward function incentivizing successful learning strategies.

In particle physics, where ML can be used, supervised learning is often the strategy of choice and it is the strategy chosen for the NNC in this thesis.

2.2.2 Structures of Neural Networks

The concept of Neural Networks has been developed as an analogy to the biological structure of the human neural system eventually organized by the human brain (see figure 2.4). Similar to the reception and processing of external impulses in the human neural system, an artificial neural system, i.e. the Neural Network, can be modeled with a tailored software algorithm. The network is characterized by the following elements:

- Input nodes, which represent numerical values for given variables. In this thesis, they represent various kinematic variables, which are measured in particle physics experiments.

- Connections combining the inputs with different weights into one node. In a simple approach this can just be the weighted average.

- An activation mechanism that translates the numerical result of the node into a logical statement like in this case true/false. There is many different mathematical functions [14], which allow transformation of a continuous result into digital step-function.

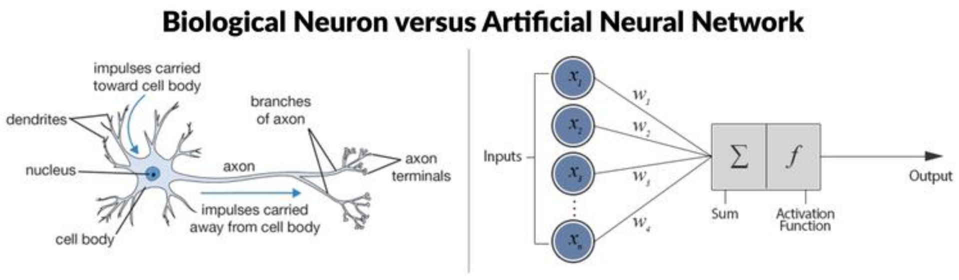


Figure 2.4: Comparison of biological and artificial neural networks [2]

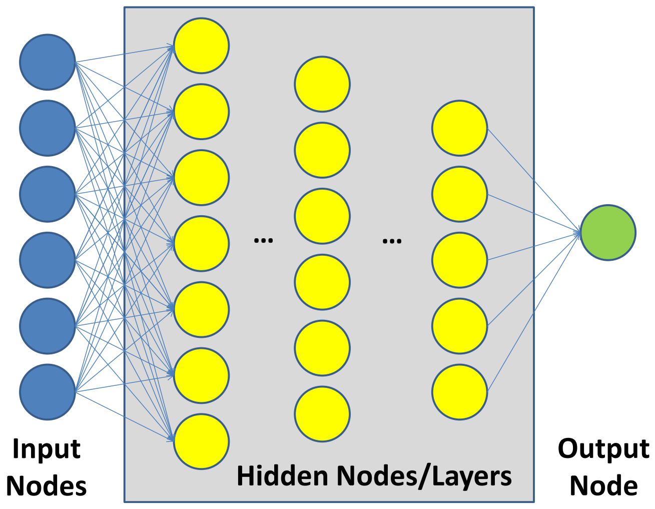


Figure 2.5: Deep Neural Network consisting of more than one layer in the Neural Network

The neural network base structure can be enhanced by putting more layers between the input nodes and the output node as shown in figure 2.5. Increasing the number of nodes and layers allows the network to create a more granular, complex model for the underlying dataset, which then in principle allows to derive more concise statements. Granularity and complexity obviously come with increasing demands on computing time and resources, which are the limiting factors for the complexity. Furthermore, the chosen complexity of the model should be adequate for the complexity of the object model. If the model is too complex it might overlearn and create just a copy of the training dataset and not a good estimator for unknown objects [12].

2.2.3 Machine Learning Process

A typical machine learning process consists of two major steps:

- a pre-processing step, which transforms the available knowledge of the physics problem into relevant selected inputs
- the learning process with those selected inputs within the defined Neural Network Structure.

For the classification task in particle physics pre-processing could consist e.g. of selection of relevant variables, and determination of variable correlation, which avoids over-emphazing individual factors in the learning process [15]. The latter step is schematically shown in figure 2.6. The pre-selected

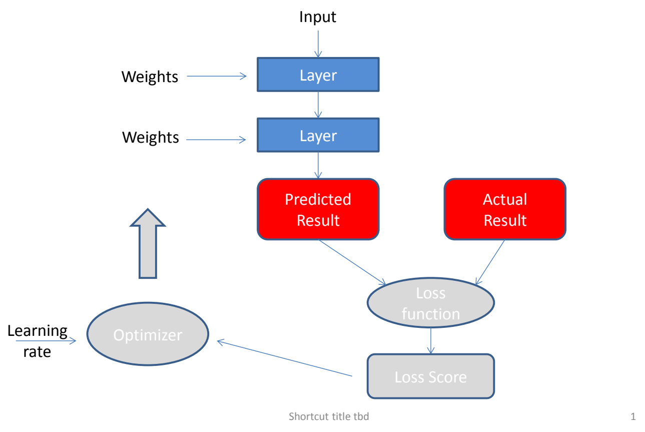


Figure 2.6: Neural Network Dynamics

inputs are fed into the Neural Network layer structure with initially random weights. This will result in predictions for the samples of data which are given to the Network. These predictions will be compared with the actual expected results given. The metric used by the network to gauge its learning is called a loss function. The aim of the network is to minimize the value of the loss function which takes as arguments the predicted label and the given label per event. This task is taken over by Optimizers, which change the weights of the connections between the nodes, so that with the next iteration of the learning process (called *epoch*) the loss score is reduced. This process is controlled by the step width the optimizer is allowed to take for changing the weights (*learning rate, momentum*) and the number of epochs given to the process. This could be designed as a self-controlling mechanism with stopping the iterations, if the change in loss score falls under a given threshold.

2.2.4 Summary of required inputs for a machine learning process

Picking up the parameters described in the previous sections the required inputs can be summarized as shown in table 2.1. All these items are available in the code (called `tWaml code`) presented in chapter 3 which adapts to various input parameters and the data samples.

Base structure	**Depth of network**	**Dynamics**
Input (variables)	Number of hidden layers	Loss function
Test and Training Data	Number of hidden nodes	Optimizer
Activation function		Learning rate, momentum
Connection weights		Number of epochs

Table 2.1: General parameters for Neural Network architectures

For the test runs (see chapter 4), one common set of data samples was used. It was sliced into several parts. The number *n* of the parts is called *n-fold factor* [16]. Training can potentially be improved by selecting 1 subset of data as test data and the remaining *n*-1 subsets as training data. Repeating this approach by selecting each of the subsets as a test data sample results can be cross-validated and the overall result can be improved.

2.3 File structure type HDF5

The data in this physics context typically stems from a Root-file format, which is a scientific software framework developed at CERN for data analysis in high energy physics.[1] One element of the scope of this thesis was to ensure that the NNC can also work with data input from HDF5-files, because this data file format may potentially become a preferred file format in particle physics.[2]

The HDF5-format supports n-dimensional datasets and each element in the dataset may be a complex object. It allows to keep the metadata with the data, hence streamlining data life cycles and pipelines. It claims to allow high performance input/output allowing access time and space optimizations.

HDF5 consists of a **file format** for storing HDF5 data, a **data model** for logically organizing and accessing HDF5 data from an application, and the **software** (libraries, language interfaces, and tools) for working with this format.[3] The file format specifies the bit-level organization of an HDF5 file on storage media. The data model consists of two primary objects, groups and datasets. Groups and

[1] For documentation see https://root.cern.ch/documentation, here status August 2019.

[2] All descriptions with regard to HDF5.file data structures are based on **H**ierarchical **D**ata **F**ile Group web site descriptions. [17]

[3] https://portal.hdfgroup.org/display/HDF5/Introduction+to+HDF5

links between the groups organize the data objects, whereas datasets contain the “raw” data values. In addition to the data itself, datasets consist of metadata that describes the data.

CHAPTER 3

Neural Network Functionality

The Neural Network used in this thesis and within the computing facility BAF2 of PI is a Python-based code using KERAS with a TensorFlow backend. This chapter generally describes the functionality of the code by module and the relevant Input-/Output interfaces. The steps to provide the respective Python, KERAS and TensorFlow modules in the PI BAF environment are documented in appendix B. A supervised learning strategy is used, hence a dataset for signal (single-top-production) and a dataset for background (top-pair production) are provided to the network with labels that tell it what the classification goals are. The draft code also contains an option to launch an adversarial network training, which is out of scope of this thesis.

3.1 Code Structure Overview

The code is structured into six Python-sub codes as shown in figure 3.1. The description of those

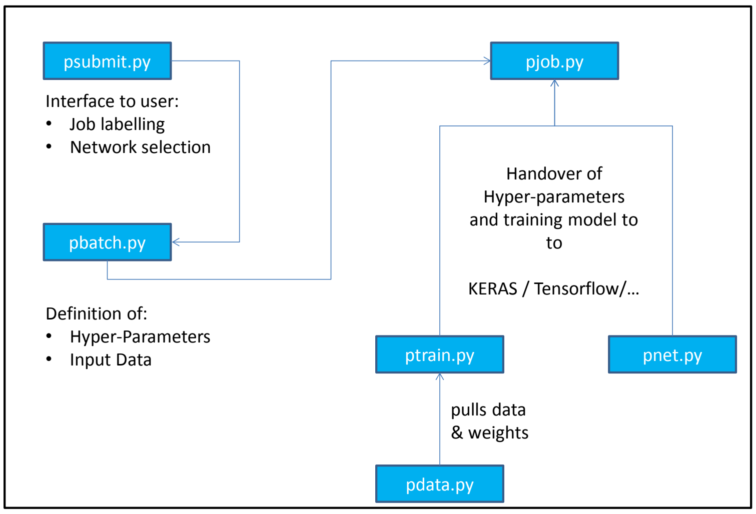


Figure 3.1: Overview of code structure and main functionality

modules is following the sequence of execution for a "simple" machine learning job run. Jobs can be started with the file **psubmit.py**, which allows labeling the job with a name[1], the selection of program run options and manual input of hyper-parameters for the network. These options can be specified within the command line starting the job.

The program run options are `_run` for starting a job on worker nodes and HTCondor submissions, `htc` for creating a batch of executable jobs.

As required, the code allows to specify the paths for the signal and the background datasets (both in HDF5 format) as separate input files and the set of variables, which are to be used by the network to separate the two samples.[2]

For creating a batch of executable files one can further specify input arrays for the following hyper-parameters: number of hidden layers and nodes, learning rate and activation function.

The file **pbatch.py** contains the class **Batch**, which has three main functions:

- offers a default dataset (see details in appendix) for the neural network training which gets updated by the inputs into psubmit.py
- offers a default dataset for the adversarial component of the training which gets updated by the inputs of submit.py
- definition of additional hyper-parameters for neural network and adversarial network training; including number of epochs, n-fold factor and momentum.
- furthermore, it includes a function **create_jdl** to combine multiple submissions with different hyper-parameters (more details in appendix A).

Eventually the module then calls the class **Job** in the module **pjob.py**.

The file pjob.py has two segments/classes, one for the Neural Network and one for the adversarial network. The function **describe** ensures that the parameter input is handed over to KERAS/TensorFlow in the right format, whereas the **function** `run` does start the training by call the classes in the file **ptrain.py**.

Within ptrain.py the dataset for training is picked via the supporting classes in **pdata.py** which allow for various data input structures. The data together with the parameters is handed over to the **class** `Deepnet` in the file **pnet.py** to run the training. The standard results are plotted automatically using matplotlib functionalities.

3.2 Data Structure

Figure 3.2 shows a typical structure based on the datasets used in this thesis. Figure 3.2 a) shows a file for the tW-signal only, consisting of a signal tree and a dedicated tree for the weights. Figure 3.2 ab) shows signal and background tree in one file with the weights being one item in the block items. Figure 3.2 c) shows a tabular structure with the table being a matrix of variables and events. The program offers three input formats for the data file: pytables, HDF5-file, Root data. The pytable

[1] Job name should be submitted as an array done typically as a HDF5-file.

[2] As discussed below it was tried to pull signal and background from one HDF5-data file. This lead to confusing reductions in the number of events of the respective signal and background files, probably stemming from overwriting processes in the creation of the HDF5-file structure from event tables.

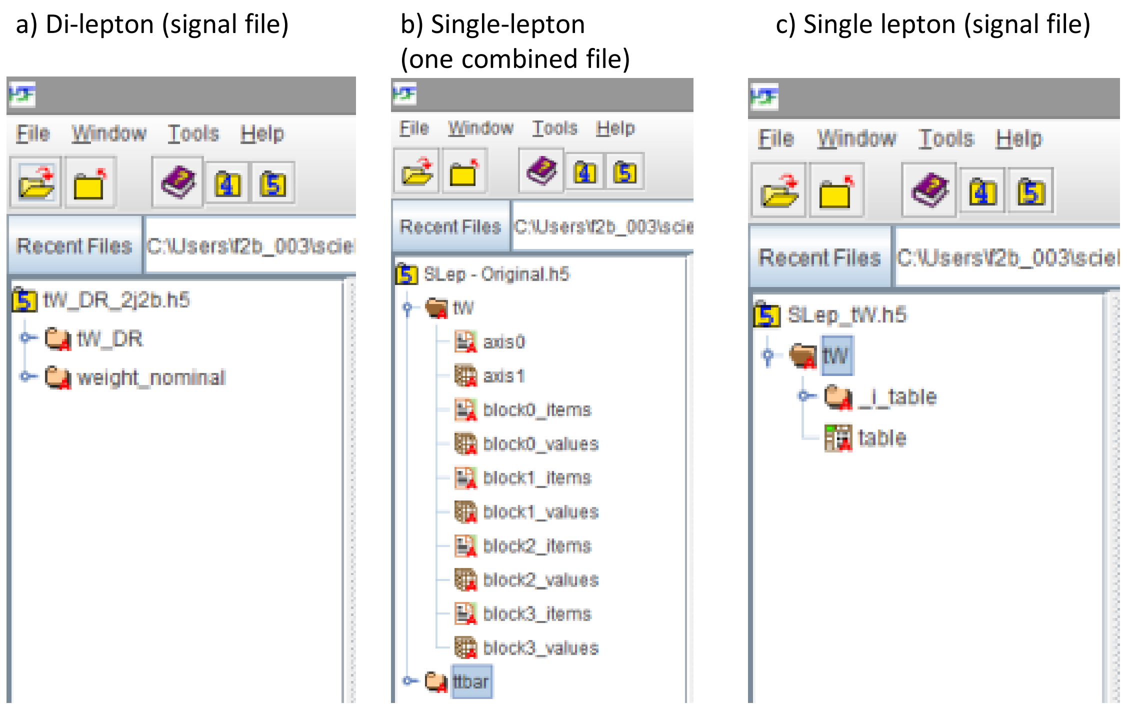


Figure 3.2: Analyzed Group structures of HDF5-files

format and an HDF5-format have been tested. Both formats read the data correctly as long as the underlying dataset structure is a table, where the column heads can be identified as variable names and a respective path of the group object can be found. The data group structure itself appears to be more a question of convenience or analysis chain of the individual situation, in which the dataset is created. This may partially be an element of the personal workflow and hence ultimately not be harmonized by a universe of rules and definitions.

3.3 Input-Output Interfaces

From the previous section, it is obvious that the major effort in getting the NNC started-up has not been a genuine ML task, but rather an approach to organize the interface from data patterns tailored for particle physics and the generic ML modules KERAS/TensorFlow and the connection to the BAF2 computing site.

3.3.1 Input-Output to User

As already mention the file psubmit.py allows two input options for model parameters. The first option is to specify the input within the program code listing A.1. The location of those inputs is documented in appendix A. The second option is to specify the input for the hyper-parameters in an hard-coded array, also documented in A. This particular section is used to build a multitude of learning jobs combining all values for all parameters. The third option is to input parameters in the command line, when starting the program. This is predominantly used when running batches of learning jobs. The structure is given in Appendix B.

3.3.2 Input-Output to KERAS/TensorFlow

The input to KERAS/TensorFlow is hard-coded in the files ptrain.py and pnet.py. The program will generate ROC- and loss-curves as output (see chapter 4) as well as accuracy and response plots. The latter will not be used in the thesis and not discussed. Furthermore, the layer set-up will be plotted. All files will be stored in a newly created file directory at the directory, from which python got started.[3]

3.3.3 Input-Output-Alternatives

One straight-forward question after the description of the code above is, whether one could streamline the semi-hierarchical structure of the code given by the various modules. One approach to do this would be to create one single program code accompanied by one single configuration file containing all required data for the physics and network model. The main advantage would be to have a single common data entry point. As will become obvious in the following chapter, a subset of data is subject to continuous changes during testing of the network (e.g. network structure parameter, learning rate,...), whereas other subsets are unchanged (e.g. data sample files). Furthermore, the actual data sample structure can differ and needs to be almost manually (hard-coded) be adapted to. This could all be done with a user interface allowing many input options, but most likely lead to a input hierarchy similar to the program module hierarchy chosen here.

[3] As a practical note: when working on worker nodes there is some time delay before the directory is visible as a directory.

CHAPTER 4

Neural Network Testing

4.1 Optimization Di-Lepton data

The first two challenges for this thesis were the debugging of the program code and the proper handover of data to the learning modules KERAS and TensorFlow. For those tasks a Di-lepton dataset compiled by the tW-group was given and used, because the learning performance of KERAS/TensorFlow for that dataset had already been proven in a different IT-setup (hard- and software).

4.1.1 Preparatory Work

The first task for this thesis had been to analyze the code and debug it. Main bugs corrected have been:

- adapting the code to python 3.x syntax modifying all program lines which would not work under python 3,
- adapting all external file links and ensure proper addressing of all internal classes with the right interfaces;
- adapting the data input format, so that it can be processed by python 3 code (e.g. string in utf-8),
- ensuring that the code is up to date with the modules it relies on,[1] and
- checking whether the output files are in line with the expected result structures.

The respective external links and the walk through the code hierarchy/classes are documented in appendix A.

4.1.2 Selection of Hyper-parameters and Batch creation

As already explained, there is no deterministic way of selecting the "right" ML model a-priori. The selection of hyper-parameters started with an initial set of parameters[2] then expanded by combinations of the parameters given by table 4.1. There is numerous choices for activation rules in KERAS [2]. In

[1] This will be an ongoing taks for the life time of this code, but based on compiler warnings it should be very well specified, where necessary changes have to be included.

[2] In fact, his could have been any set. An inital interval found in the code was used to start with. As figure 4.2 shows hyper-parameters beyond the given intervals were explored.

	Minimum	**Maximum**	**Increment**
Epochs	200	1000	200
Hidden layers	10	20	5
Hidden nodes	10	20	10
Learning rate	0.01	0.1	0.01
Momentum	0.1	0.9	0.2

Table 4.1: Design items for Neural networks

this thesis `elu`, `relu`, and `tanh` were used. In all sample runs for this thesis the impact of change in activation for the results was minor. Like the loss function (*binary_crossentropy*) and the optimizer (*Stochastich Gradient Descent*) used here, these are more internal parameters of KERAS and variation was out of scope.[3]

The data sample consists of approximately 2 million events and four kinematic variables already pre-selected by other analyses had been given as being most indicative of the distinction between signal and background.

4.1.3 Results and Learnings

Major output for the supervised training is the so called ROC-curve[4]. An example is shown in figure 4.1 The two axes indicate the true positive and false positive decisions by the machine learning system.

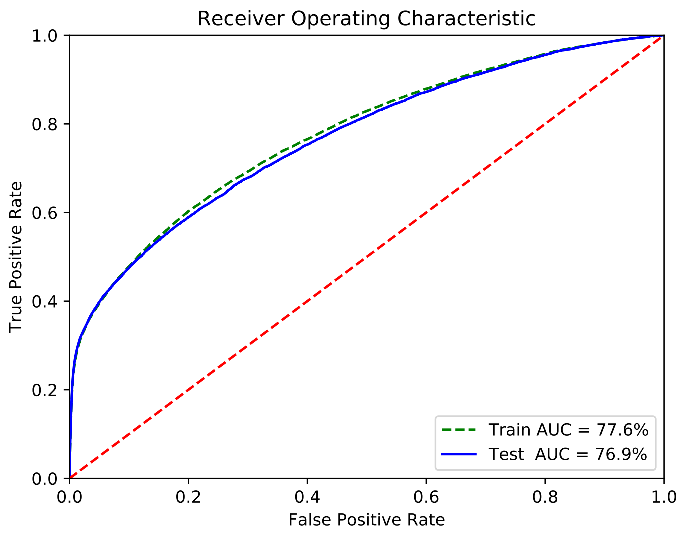


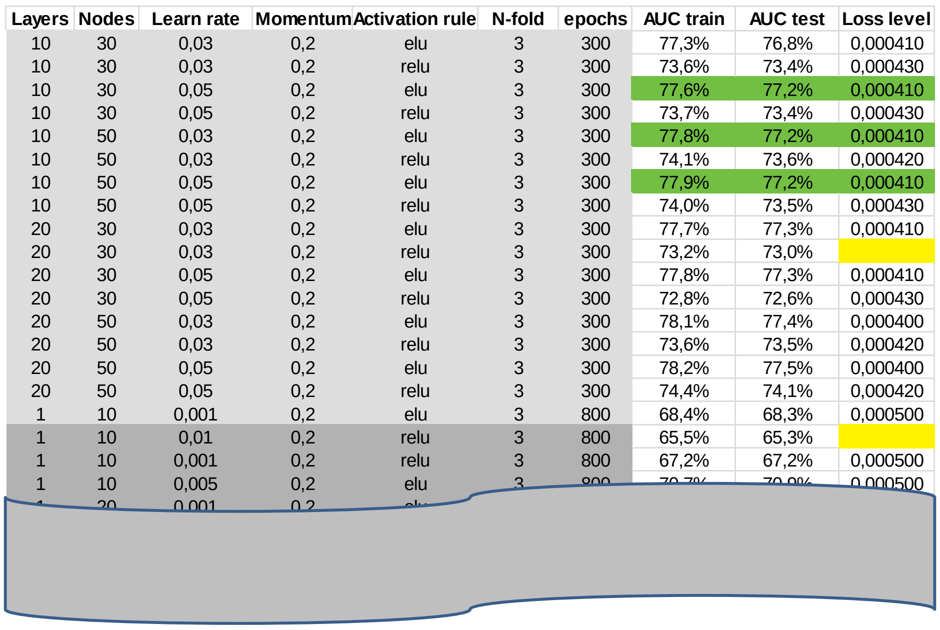
Figure 4.1: Example of ROC curve

For the case of random guessing these rates should be equal, which is indicated by the red dotted diagonal line. A measure of the quality of the learning process is the surface under the curves for

[3] For detailed descriptions of those parameters and the numerous options see [2]

[4] ROC = Receiver Operating Characteristic

the applied training and test data sample sets, which is called AUC (for Area Under the Curve). Maximizing AUC can be done by variation of the features of the network. To explore the influence of the individual hyper-parameter combinations have been tested. The principal set-up and results can be derived from the overview in figure 4.2



Layers	Nodes	Learn rate	Momentum	Activation rule	N-fold	epochs	AUC train	AUC test	Loss level
10	30	0,03	0,2	elu	3	300	77,3%	76,8%	0,000410
10	30	0,03	0,2	relu	3	300	73,6%	73,4%	0,000430
10	30	0,05	0,2	elu	3	300	77,6%	77,2%	0,000410
10	30	0,05	0,2	relu	3	300	73,7%	73,4%	0,000430
10	50	0,03	0,2	elu	3	300	77,8%	77,2%	0,000410
10	50	0,03	0,2	relu	3	300	74,1%	73,6%	0,000420
10	50	0,05	0,2	elu	3	300	77,9%	77,2%	0,000410
10	50	0,05	0,2	relu	3	300	74,0%	73,5%	0,000430
20	30	0,03	0,2	elu	3	300	77,7%	77,3%	0,000410
20	30	0,03	0,2	relu	3	300	73,2%	73,0%	
20	30	0,05	0,2	elu	3	300	77,8%	77,3%	0,000410
20	30	0,05	0,2	relu	3	300	72,8%	72,6%	0,000430
20	50	0,03	0,2	elu	3	300	78,1%	77,4%	0,000400
20	50	0,03	0,2	relu	3	300	73,6%	73,5%	0,000420
20	50	0,05	0,2	elu	3	300	78,2%	77,5%	0,000400
20	50	0,05	0,2	relu	3	300	74,4%	74,1%	0,000420
1	10	0,001	0,2	elu	3	800	68,4%	68,3%	0,000500
1	10	0,01	0,2	relu	3	800	65,5%	65,3%	
1	10	0,001	0,2	relu	3	800	67,2%	67,2%	0,000500
1	10	0,005	0,2	elu	3	800	70,7%	70,9%	0,000500
1	20	0,001	0,2	elu					

Figure 4.2: Testing set-up for hyper-parameters

Nevertheless, the maximum value is influenced by the structure of the problem. For this problem an AUC of almost 77% has been achieved after some optimization. Due to its handling of large complex data structures, ML almost acts like a black box and it results in probabilistic rather than general statements.

Taking the AUC values as an initial indicator for the ability of the NNC to make the distinction between signal and background the following trends were found:

- An increase in the number of layers does not significantly improve the training result, despite the fact that it comes at a cost in terms of computational resource.
- An increase in the number of nodes per layer slightly improves the training results.
- An optimum was found at learning rates of 0.03. Decreasing the learning rate by a factor of 10 which is equivalent to smaller steps in changing the model weights did worsen the quality of the training.
- The activation rule “elu” consistently produced better training results than the activation rule “relu”.

Figure 4.2 also shows that in addition to the AUC-value an approximate look at the resulting loss levels was done. The NNC provides this output in the format shown in figure 4.3. The loss curve indicates the dynamics of the laerning succes over the epochs and as expected shows reducing losses with increasing number of epochs.

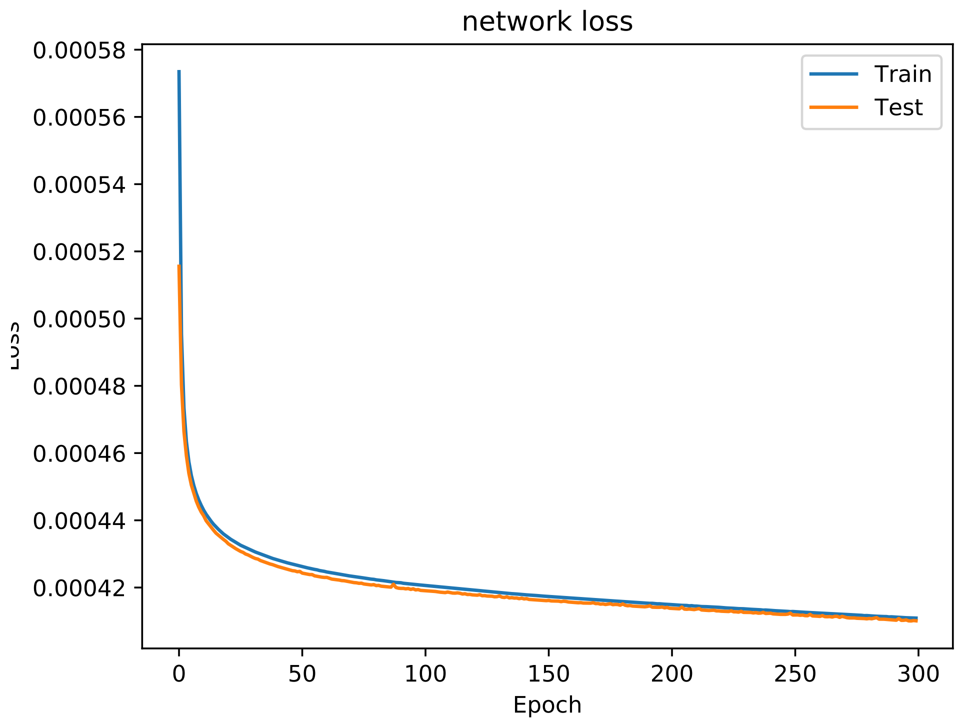


Figure 4.3: Typical loss curve pattern

4.2 Single Lepton

As outlined in chapter 2, the separation of signal and background is tough due to the same final states for single-top-quark and top-quark-pair production at NLO level. Hence this situation can be used as a stress test scenario for the current status of the NNC.

4.2.1 Preparatory Work

The ML training runs are based on a data sample of events of the single-lepton decay mode which have been processed with the Neuro-Bayes package before [15]. A detailed description of how the dataset has been created is given there as well.

In a first attempt, the handover had been naively organized as receiving an HDF5 file including data samples for signal and background. As shown in 3.2 b) one single file was given with one tree for signal data and one tree for background data. It turned out the net data sample number in this file reduced to 169 samples, due to overwriting of data in the creation of the HDF5-file.

The second attempt worked with separate files and could successfully pick all data samples for the training. There is two prerequisites which were found out during the data adaptation process. Within the HDF5 file there needs to be a data object which has the form of a table. The current setup expects variables in the columns with the heads of the columns being the variable names and the events in the rows. The other small but significant hurdle to overcome was the string format of the variable names. Python 3 encodes strings to bytes upon storing and therefore needs to be addressed when reading. The di-lepton samples were most likely created using python 2 and there the previous read structure did not need to be changed. The relevant spots in the code are given in appendix A. It has not been investigated whether there is even more efficient ways of storing data in HDF5 file format and what could be a stringent uniform process to transfer ATLAS data into HDF5 format.

4.2.2 Results and Learnings

As a first step, the hyper-parameters were chosen similar to the parameter intervals for the di-lepton case. The variable selection was based on the analysis in [15] according to their importance. This did not lead to satisfying training results. Hence the hyper-parameters were on the one hand chosen in a wider range (for layers and nodes) and on the other chosen more granular (for learning rate). To adapt to the smaller sample size of approximately 500,000 samples the n-fold factor was varied. Additional activation functions were tested.

The network has been able to process the given data without any problem, but it does not arrive at results which are useful. For all parameter settings analyzed, no single training run achieved an AUC-value significantly higher than 60%. This is not far away from a random classification with a 50% AUC value. It appears that the variation of the hyper-parameters alone is not yet powerful enough to provide significant labeling of signal and background in the challenging single-lepton situation.

4.3 Comparison to NeuroBayes

The performance of the current NNC does not allow at this stage a comprehensive comparison to NeuroBayes. The performance improvements attempted in this thesis were focused on finding an optimal set of hyper-parameters for NNC performance. The two additional conceptual options, on the one hand pre-processing[5] the raw physical data into a more optimized input for the NNC and and, on the other hand using more advanced options of the KERAS/TensorFlow package have not been investigated due to time constraints. This would be a logical follow-up for further analyses. For NeuroBayes pre-processing capabilities have been investigated and documented in [15].

[5] For NeuroBayes pre-processing capabilities have been investigated and documented in [15].

CHAPTER 5

Conclusion and Outlook

In this thesis a machine learning code developed by the tW-group has been analyzed, made workable and used for an example for the top-quark single-lepton decay region.

The main tasks of understanding the initial code skeleton, debugging and documenting it have been accomplished. There is a straight-forward approach to test a multitude of hyper-parameter combinations with an easy batch creation method. With that, the learning effort and time for anyone wanting to work with the NNC should have been reduced significantly.

The code has been used for two physical cases. For both cases it worked in terms of picking-up input data correctly and producing technically correct outputs. For the di-lepton case it showed significant ability to separate signal from background events, whereas for single-lepton it did not.

Further areas to improve the performance lie in

- optimizing the tailoring for the set of inputs given to the code (pre-processing), and
- addressing more sophisticated methods within KERAS/TensorFlow.

Both directions would be logical follow-ups based on the fast access to the code provided by this thesis.

Bibliography

[1] G. Carleo et al., *Machine learning and the physical sciences*, 2019, arXiv: 1903.10563 [physics.comp-ph].

[2] F. Chollet et al., *Keras*, https://keras.io, 2015.

[3] R. Moles-Valls, “Single top-quark production in the ATLAS and CMS Experiments”, *Proceedings, 52nd Rencontres de Moriond on QCD and High Energy Interactions: La Thuile, Italy, March 25-April 1, 2017*, CERN, CERN, 2017 177.

[4] M. Thomson, *Modern particle physics*, Cambridge University Press, 2013, ISBN: 9781107034266.

[5] M. Kobayashi and T. Maskawa, *CP-Violation in the Renormalizable Theory of Weak Interaction*, Progress of Theoretical Physics **49** (1973) 652, ISSN: 0033-068X, URL: https://doi.org/10.1143/PTP.49.652.

[6] S. Abachi et al., *Observation of the top quark*, Phys. Rev. Lett. **74** (1995) 2632, arXiv: hep-ex/9503003 [hep-ex].

[7] F. Abe et al., *Observation of top quark production in $\bar{p}p$ collisions*, Phys. Rev. Lett. **74** (1995) 2626, arXiv: hep-ex/9503002 [hep-ex].

[8] M. Aaboud et al., *Measurement of the cross-section for producing a W boson in association with a single top quark in pp collisions at $\sqrt{s} = 13$ TeV with ATLAS*, JHEP **01** (2018) 063, arXiv: 1612.07231 [hep-ex].

[9] V. M. Abazov et al., *Evidence for production of single top quarks*, Phys. Rev. **D78** (2008) 012005, arXiv: 0803.0739 [hep-ex].

[10] T. Aaltonen et al., *First Observation of Electroweak Single Top Quark Production*, Phys. Rev. Lett. **103** (2009) 092002, arXiv: 0903.0885 [hep-ex].

[11] Sebastian Mergelmeyer, *Measurement of the Associated Production of a Single Top-Quark and a W Boson in Single-Lepton Events with the ATLAS-Detector*, Bonn 2016.

[12] Schwaiger, Roland and others, *Neuronale Netze programmieren mit Python*, Bonn 2019.

[13] F. Chollet, *Deep Learning mit Python und KERAS*, Frechen 2018.

[14] C. Kirfel, *Hyperparameter Optimisation of an Adversarial Neural Network in the tW channel at 13 TeV with ATLAS*, master thesis Bonn 2019.

[15] F. G. Diaz Capriles, *Measurement of the Single Top tW-Channel Inclusive Cross Section in the SIngle Lepton Final State at 13 TeV with ATLAS*, master thesis Bonn 2019.

[16] Raschka, Sebastian and others, *Machine Learning mit Python und Scikit-learn und TensorFlow*, 2. Auflage, Frechen 2018.

[17] The HDF Group, *Hierarchical Data Format, version 5*, http://www.hdfgroup.org/HDF5/, 1997-2019.

APPENDIX A

Documentation of code and instructions

A.1 The Python Code

The following sections follow the code structure depicted in figure 3.1.

A.1.1 User interface psubmit.py

This file starts with a call to the batch in the file structure and loads some system modules.

Listing A.1: Initial loading of Input-Output modules

```
#Call Batch function which defines data sources and Model hyperparameters
from pbatch import Batch

#Module argparse introduces command line interface
import argparse
#system specific parameters and functions, os...portable way of using system
    dependent functionality
import sys, os
```

The parser function is being used to determine, whether the code runs automatically with one given set of data or is creating a job wrapper to be used under HT condor or is fully variable.

```
def parse_options(args):
#Description of the program, prog showing program name in help messages
parser = argparse.ArgumentParser(description='Neural network training', prog=
    'submit')
#dest...Name of the attribute under which subcommand will be stored
subcommands = parser.add_subparsers(dest='command')

_run = subcommands.add_parser('_run', help='Run training with current setting
    ')
_run.add_argument('_run', nargs='*')

htc = subcommands.add_parser('htc', help='Generate HTCondor scripts')
run = subcommands.add_parser('all', help='Run trainings locally')

return parser.parse_args(args)
```

After that the main program execution starts with

- giving the job a name (hard-coded),
- connecting input files and variables to process, and
- picking up the jobname from the command line.

```
if __name__ == '__main__':
base_directory = os.getcwd()
#DataFrame Input
inputs = {'name': '2j2b',
'signal_h5': '/cephfs/user/rzhang/Wtr21/run/v28/h5files/tW_DR_2j2b.h5',
'backgd_h5': '/cephfs/user/rzhang/Wtr21/run/v28/h5files/ttbar_2j2b.h5',
'syssig_h5': '/cephfs/user/rzhang/Wtr21/run/v28/h5files/tW_DS_2j2b.h5',
'variables': ['mass_lep1jet2', 'mass_lep1jet1', 'deltaR_lep1_jet1', '
    mass_lep2jet1', 'pTsys_lep1lep2met', 'pT_jet2', 'mass_lep2jet2'],
}

jobname = sys.argv[1]
```

The following call of the class BATCH will update the default hard-coded input with the inputs defined in the submit file. (see A.1.2).

```
pbatch = Batch(jobname, base_directory, inputs)
args = parse_options(sys.argv[2:])
```

If the _run option has been selected the function _run in the file pbatch.py gets triggered by

```
pbatch._run(args._run)
```

with the arguments being defined by

```
_run = subcommands.add_parser('_run', help='Run training with current setting
    ')
_run.add_argument('_run', nargs='*')
```

with commands from the command line.
The remainder of the submit.py file is used for job wrapping, etc. and will not be described here.

A.1.2 Data structuring file pbatch.py

Within the batch file there is some default connections to data input (probably to already populate a full structure for the training) which then via the update function gets adapted to the input defined by psubmit.py.

```
def update_dict(orig_dict, new_dict):
for k, v in new_dict.items():
if k in orig_dict:
orig_dict[k] = v

class Batch(object):
def __init__(self, jobname, base_directory, inputs):
self.jobname = jobname
```

```python

self.base_directory = base_directory + '/'
self.para_train_sim = {'name': '2j2b',
'signal_h5': 'tW_DR_2j2b.h5',
'signal_name': 'tW_DR',
'signal_tree': 'wt_DR_nominal',
'backgd_h5': 'ttbar_2j2b.h5',
'backgd_name': 'ttbar',
'backgd_tree': 'tt_nominal',
'weight_name': 'weight_nominal',
'variables': ['mass_lep1jet2'],
}
update_dict(self.para_train_sim, inputs)

self.para_net_sim = {
'name': 'simple',
'nfold': 3,
'train_fold': 0,
'epochs': 500,
'hidden_Nlayer': 10,
'hidden_Nnode': 100,
'lr': 0.03,
'momentum': 0.8,
'output': None,
'activation': 'elu',
}
update_dict(self.para_net_sim, inputs)

self.para_train_Adv = {**self.para_train_sim,
'name': 'NP',
'no_syssig': False,
'syssig_h5': '/Users/zhangrui/Work/Code/ML/ANN/h5files/tW_DS_2j2b.h5',
'syssig_name': 'tW_DS',
'syssig_tree': 'wt_DS',
}
update_dict(self.para_train_Adv, inputs)

self.para_net_Adv = {**self.para_net_sim,
'name': 'ANN',
'epochs': 2,
'hidden_auxNlayer': 2,
'hidden_auxNnode': 5,
'preTrain_epochs': 20,
'n_iteraction': 500,
'lam': 10,
}
update_dict(self.para_net_Adv, inputs)
```

In the initial given set-up not all variables get overwritten by the input in psubmit.py. The job gets started with the following command

```python
def _run(self, param):
setting = {param[i]: param[i+1] for i in range(0, len(param), 2)}
if self.jobname == 'ANN':
update_dict(self.para_net_Adv, setting)
```

```
pjob = JobAdv(**self.para_net_Adv, para_train = self.para_train_Adv)
pjob.run()
else:
update_dict(self.para_net_sim, setting)
pjob = Job(**self.para_net_sim, para_train = self.para_train_sim)
pjob.run()
```

calling the run function in pjob.py.

The remainder of the pbatch.py file is for job wrapping and will not be further discussed here.

A.1.3 Job configuration file pjob.py

The run-function in pjob.py assigns the data chosen for the training to the class Train in the file train.py and includes the output of the training itself.

```
def run(self):

''' An instance of Train for data handling '''
self.trainer = Train(**self.para_train)
self.trainer.split(nfold = self.nfold)

''' An instance of DeepNet for network construction and pass it to Train '''
self.deepnet = DeepNet(name = self.name, build_dis = False, hidden_Nlayer =
    self.hidden_Nlayer, hidden_Nnode = self.hidden_Nnode, hidden_activation =
     self.activation)
self.deepnet.build(input_dimension = self.trainer.shape, lr = self.lr,
    momentum = self.momentum)
self.deepnet.plot(base_directory = self.output)
self.trainer.setNetwork(self.deepnet.generator)

''' Run the training '''
self.result = self.trainer.train(mode = 0, epochs = self.epochs, fold = self.
    train_fold)
self.trainer.evaluate()
self.trainer.plotLoss(self.result)
self.trainer.plotResults()
```

A.1.4 Training file ptrain.py

This file initializes the training by import modules from the KERAS environment. Important here are the references to pdata, which contains the links to the actually used data files.

```
import numpy as np
from sklearn.model_selection import KFold
from sklearn.metrics import roc_auc_score
from pdata import dataset
from pdata import scale_weight_sum
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import os
```

The datasets themselves are pulled by the class train in line 12 and 13 from the routines in file pdata.py.

```python
1  class Train(object):
2  def describe(self): return self.__class__.__name__
3  def __init__(self, name = '3j1b', base_directory = './', signal_h5 = 'SLep_tW.h5', signal_name = 'tW_3j1b', signal_tree = 'tW', signal_latex = r'$tW$',
4  backgd_h5 = 'SLep_ttbar.h5', backgd_name = 'ttbar_3j1b', backgd_tree = 'ttbar', backgd_latex =  r'$t\bar{t}$', weight_name = 'weight_nominal',
5  variables = ['pt_balanceWWb', 'mT_jet2met', 'deltapT_lep1_jet1', 'mT_lep1met', 'cent_lep1jet2', 'pT_lep1', 'mass_WWb', 'deltaphi_lep1_jet1', 'mT_jet3met', 'aeta_jet1', 'mT_jet2met', 'deltaR_lep1_jet2', 'pT_bjet1', 'deltapT_lep1_jet2'],
6  no_syssig = True, syssig_h5 = 'tW_DS_2j2b.h5', syssig_name = 'tW_DS_2j2b', syssig_tree = 'tW_DS', syssig_latex = r'$tW$ DS',
7  ):
8  self.name = name
9  variables = [i.encode("utf-8") for i in variables]
10 self.signal_label, self.backgd_label, self.center_label, self.syssig_label = 1, 0, 1, 0
11 self.signal_latex, self.backgd_latex = signal_latex, backgd_latex
12 self.signal = dataset.from_h5(signal_h5, signal_name, tree_name = signal_tree, weight_name = weight_name, label = self.signal_label, auxlabel = self.center_label)
13 self.backgd = dataset.from_h5(backgd_h5, backgd_name, tree_name = backgd_tree, weight_name = weight_name, label = self.backgd_label, auxlabel = self.center_label)
14 self.signal.keep_columns(variables)
15 self.backgd.keep_columns(variables)
16 self.no_syssig = no_syssig
17 self.syssig_latex = None if self.no_syssig else syssig_latex
18 self.losses_test = {'L_gen': [], 'L_dis': [], 'L_diff': []}
19 self.losses_train = {'L_gen': [], 'L_dis': [], 'L_diff': []}
```

A.1.5 Training file pdata.py

The file pdata.py allows referencing data sets from Root-files, pytables and HDF5 format. As pointed out in the core thesis one needs to find a way through the tree to the data table. This is shown in the following listing for h5-structured data.

```python
def from_h5(
file_name: str,
name: str,
#columns: List[str],
tree_name: str = "WtLoop_nominal",
weight_name: str = "weight_name",
label: Optional[int] = None,
auxlabel: Optional[int] = None,
) -> "dataset":
"""Create a dataset from generic h5 input (loosely expected to be from
the ATLAS Analysis Release utility ``ttree2hdf5``

The name of the HDF5 dataset inside the file is assumed to be
```

```
``tree_name``. The ``name`` argument is something *you
choose*.

Parameters
----------
file_name: str
Name of h5 file containing the payload
name: str
Name of the dataset you would like to define
columns: List[str]
Names of columns (branches) to include in payload
tree_name: str
Name of tree dataset originates from (HDF5 dataset name)
weight_name: str
Name of the weight array inside the h5 file
label: Optional[int]
Give the dataset an integer label
auxlabel: Optional[int]
Give the dataset an integer auxiliary label

Examples
--------

>>> ds = dataset.from_h5('file.h5', 'dsname', tree_name='
    WtLoop_EG_RESOLUTION_ALL__1up')

"""
#print (columns)
ds = dataset()
ds._init(
[file_name],
name=name,
weight_name=weight_name,
#columns = columns,
tree_name=tree_name,
label=label,
auxlabel=auxlabel,
)

f = pd.read_hdf(file_name, mode="r", key=tree_name)
#print (f.shape)
full_ds = f
#print (full_ds.shape)
w_array = f[weight_name.encode('utf-8')]
#print (w_array)
#print (w_array.shape)
#coldict = {}
#for col in columns:
#    coldict[col] = full_ds[col]
#frame = pd.DataFrame(f)
ds._set_df_and_weights(full_ds, w_array)
return ds
```

A.1.6 Training file pnet.py

This file loads KERAS and TensorFlow and takes the output back. Note that even if it looks like all of the lines 12-15 are updated to a newer version of TensorFlow one receives warnings. Presumably, this is the case, because some internal TensorFlow commands are addressed inside the module and one has to update TensorFlow as such to a new version.

```
1  import tensorflow
2  from tensorflow import keras
3  # import keras.backend as K
4  from keras.layers import Input, Dense
5  from keras.models import Model
6  from keras.optimizers import SGD
7  from keras.utils.vis_utils import plot_model
8  # from keras.layers.advanced_activations import LeakyReLU
9  import keras.backend as K
10 import os
11
12 session_conf = tensorflow.compat.v1.ConfigProto(intra_op_parallelism_threads
       =6, inter_op_parallelism_threads=6)
13 tensorflow.compat.v1.set_random_seed(1)
14 session = tensorflow.compat.v1.Session(graph=tensorflow.compat.v1.
       get_default_graph(), config=session_conf)
15 tensorflow.compat.v1.keras.backend.set_session(session)
```

APPENDIX B

Transfer to BAF

B.1 BAF2 Environment

B.1.1 Connecting via terminal

Starting either via terminal icon on desktop or via putty from my laptop. The .bashrc-file will check the required command to start an interactive session on BAF2.

B.1.2 Starting an Interactive session

There is a setup-routine defined with the batch job *OS7_interactive.jdl*

```
Transfer\_input\_files    = .ssh
+ContainerOS = "CentOS7"
Request\_cpus = 1
Request\_memory = 2 GB
Request\_disk = 10 GB
Queue
```

Those commands are in line with the guidance given on the BAF2 webpage.
This can be submitted from the personal home directory by

```
condor_submit -interactive  OS7_interactive.jdl
```

When I do this; get error message

```
/usr/bin/xauth: file /jwd/.Xauthority does not exist
```

and a path name.

```
username@wn002 (SL6) /pool/condor/dir_879
```

As informed by IT-support the error message should not matter, because it relates to graphics files, which in any event cannot be displayed durign a batch session.

I do not get the xauthority-error message, when I login via putty from the outside, presumably due to graphics availability on my PC. When I do this the following commands are executed automatically :

```
cat ~/.bash_profile
source ~/.bashrc
```

Doing then

```
condor_submit -interactive  OS7_interactive.jdl
```

works without a problem.

B.1.3 Setting the environment

Following the BAF2 instructions I connect via

```
source /etc/profile
$BUDDY .bashrc_OS7
```

and get

```
/cephfs/user/s6thbett: is a directory
```

Next thing to check is the available python version via

```
ls /usr/bin/python*
```

In my initial login I only have version 2.6 available, hence the next step is to set up python 3.7. To do this I set-up a job in the HTCondor cluster workload management software environment:

```
conda create -y  -p ~/project_venv} python=3.7 anaconda
```

```
conda activate /jwd/project_venv}
```

To start a run enter

```
python /cephfs/user/s6thbett/projectdefinition/psubmit.py 2j2b_0.h5 _run
```

with 2j2b_0.h5 as job name and _run as execution option. As starting directory I used the home directory of my cephfs storage facility and all files relevant were stored in a subdirectory projectdefinition.

All of this is summarized in the file `setup_os7_anyenvironment.sh`, which, in addition allows to create a tar-ball to be unpacked for BAF2. Just un-comment the last two lines.

B.2 Creating batch files for BAF usage

As mentioned in the thesis the NNC can create batch files for use on BAF2. To create a batch file go to a worker node and start NNC with the htc option:

```
python /cephfs/user/s6thbett/projectdefinition/pubmit.py 2j2b_0.h5 htc
```

This will create a file like

```
1  Executable                = /cephfs/user/s6thbett/projby.sh
2  Universe                  = vanilla
3  Transfer_executable       = True
4  Request_memory            = 30 GB
5  Request_cpus              = 6
6  # Disk space in kiB, if no unit is specified!
7  Request_disk              = 4 GB
8  # Specify job input and output
9  Error                     = log/err.$(ClusterId).$(Process)
10 Output                    = log/out.$(ClusterId).$(Process)
11 Log                       = log/log.$(ClusterId).$(Process)
12 # Additional job requirements (note the plus signs)
13 # Choose OS (options: "SL6", "CentOS7", "Ubuntu1804")
14 +ContainerOS              = "SL6"
15
16 # Submit 1 job
17 arguments                 = hidden_Nlayer 60 hidden_Nnode 40 lr 0.0001
       activation elu
18 Queue 1
19
20 # Submit 1 job
21 arguments                 = hidden_Nlayer 60 hidden_Nnode 40 lr 0.0.0005
       activation elu
22 Queue 1
```

which is fitted to run on BAF2 as per Intranet instruction.
This file has in line 1 a reference to the file proj.by.

```
source /etc/profile
cd /jwd
module load anaconda/5.3.0-py37
tar xf $BUDDY/project_venv.tar.gz
source activate /jwd/project_venv
python /cephfs/user/s6thbett/projectdefinition/psubmit.py 3j1b_0.h5 _run $*
dest=/cephfs/user/s6thbett/run
\cp -r job__* $dest/
unset dest
```

One can see that this is loading anaconda into the virtual environment directory jwd and un-tars the project environment.
It starts python and defines in the next line where the run data needs to be stored. It is mandatory for BAF to finally copy the run data to a directory available outside BAF. This worked without a problem just with some time delay.
Please note that in every case you need to create a log-directory in the directory you start the job from, here my home directory on cephfs.

List of Figures

List of Tables