Title of the Thesis

Author's name

Masterarbeit in Physik
angefertigt im Physikalischen Institut

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

MMM 2016

I hereby declare that this thesis was formulated by myself and that no sources or tools other than those cited were used.

Bonn, ...
Date Signature

1. Gutachter: Prof. Dr. John Smith
2. Gutachterin: Prof. Dr. Anne Jones

Contents

CHAPTER 1

Introduction

The introduction usually gives a few pages of introduction to the whole subject, maybe even starting with the Greeks.

For more information on LaTeX and the packages that are available see for example the books of Kopka [1] and Goossens et al [2].

A lot of useful information on particle physics can be found in the "Particle Data Book" [**pdg2010**].

I have resisted the temptation to put a lot of definitions into the file `thesis_defs.sty`, as everyone has their own taste as to what scheme they want to use for names. However, a few examples are included to help you get started:

- cross-sections are measured in pb and integrated luminosity in pb^{-1};
- the K_S^0 is an interesting particle;
- the missing transverse momentum, $p_{\text{T}}^{\text{miss}}$, is often called missing transverse energy, even though it is calculated using a vector sum.

Note that the examples of units assume that you are using the siunitx package.

It also is probably a good idea to include a few well formatted references in the thesis skeleton. More detailed suggestions on what citation types to use can be found in the thesis guide [**thesis-guide**]:

- articles in refereed journals[**pdg2010**, **Aad:2010ey**];
- a book [**Halzen:1984mc**];
- a PhD thesis [**tlodd:2012**] and a Diplom thesis [**mergelmeyer:2011**];
- a collection of articles [**lhc:vol1**];
- a conference note [**ATLAS-CONF-2011-008**];
- a preprint [**atlas:perf:2009**] (you can also use `@online` or `@booklet`for such things;
- something that is only available online [**thesis-guide**].

At the end of the introduction it is normal to say briefly what comes in the following chapters.

The lines at the end of this file are used by AUCTeX to specify which is the master LaTeX file, so that you can compile your thesis directly within `emacs`.

Acknowledgements

I would like to thank ...

You should probably use `\chapter*` for acknowledgements at the beginning of a thesis and `\chapter` for the end.

CHAPTER 2

The Standard Model of Particle Physics

Originally in an attempt do unify the electromagnetic, weak and strong force under one theory the standard model of particle physics summarizes the status quo of elementary particles and their interactions. Although the model has several weaknesses it stands very strong to tests so far and

Das STandardmodell ist nicht ohne Schwächen, hält aber experimentellen Tests in vielen Bereiche mit beeindruckneder Genauigkit stand. It is in no way a perfect and complete theoretical model but nontheless a very powerful basis for experimental predicitons.

2.1 Interactions

2.2 Particles

2.3 Open questions

CHAPTER 3

The LHC and the ATLAS detector

3.1 The LHC

Due to the statistical nature of Particle Physics it does not only require sufficiently high energies but also an exceedingly large amount of data to allow significant results. The LHC provides both with an energy of and an integrated luminosity of. The analysis in this work used data(?) based on the ATLAS detector, on of the four detectors at the LHC.

This chapter introduces the LHC, followed by the most important aspects of the ATLAS detector. In the next ch

3.2 The ATLAS detector

3.2.1 Tracking Detectors

3.2.2 Calorimeter (split)

3.2.3 Muon Spectrometer

3.2.4 How the detector works

3.2.5 The LHC

The Large Hadron Collider ("LHC") located at the facilities of the European Organization of Nuclear Research ("CERN") close to Geneva was built to extend the frontiers of modern particle physics by delivering high luminosities and reaching unprecedented high energies. The LHC is designed to collide bunches of up to 10^{11} protons at a luminosity of $10^{34}\,\mathrm{cm}^{-2}\,\mathrm{s}^{-1}$. The beams collide at four points where the four main experiments of the LHC are located. Two of these are special-purpose detectors, namely LHCb and ALICE while the other two, ATLAS and CMS, are general-purpose detectors. This experiment uses ATLAS data and Monte Carlo. Figure 3.1 shows the LHC, the four detectors and its general location.

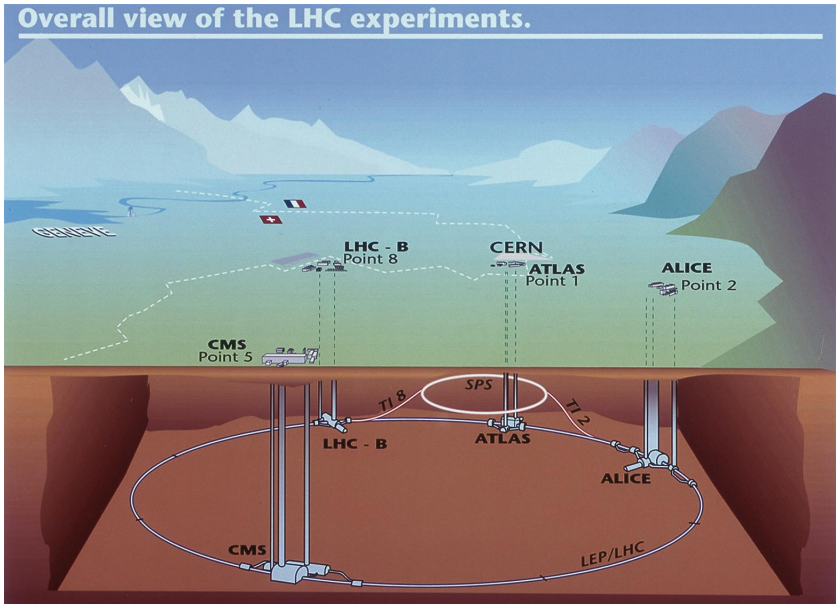


Figure 3.1: Sketch of the LHC ring, the position of the experiments and the surrounding countryside. The four big LHC experiments are indicated(ATLAS, CMS, LHC-B and ALICE)along with their injection lines(Point 1, 2, 4, 8)[**atlas_pictures**]

3.2.6 The ATLAS Detector

The ATLAS detector was developed to study the physic processes in a broad energy range available at the LHC. This enables the observation of highly massive particles that lower energy accelerators were not able to create and that would deliver new physics theory beyond the standard model of particle physics. It was designed to cover the maximum number of final states being a so called general purpose-detector.

3.2.7 The ATLAS coordinate system

The ATLAS coordinate system is defined by the beam direction with the z-axis pointing along the LHC's beam pipe. The corresponding transverse plane is defined by the x-axis pointing towards the ring's centre while the y-axis points upwards. The origin of the system is defined by the nominal point of interaction. The polar angle θ, is the angle between the z-axis and the x-y-plane and the azimuthal angle ϕ is the angle between the x- and the y-axis.

The coordinates used in this analysis are usually the azimuthal angle ϕ, the pseudo-rapidity η, and the transverse momentum pT. The pseudo-rapidity replaces the polar angle and is defined as

$$\eta = \frac{1}{2} ln \left[tan \left(\frac{\theta}{2} \right) \right]. \tag{3.1}$$

The transverse momentum is defined by

$$pT = \sqrt{p_x^2 + p_y^2} \tag{3.2}$$

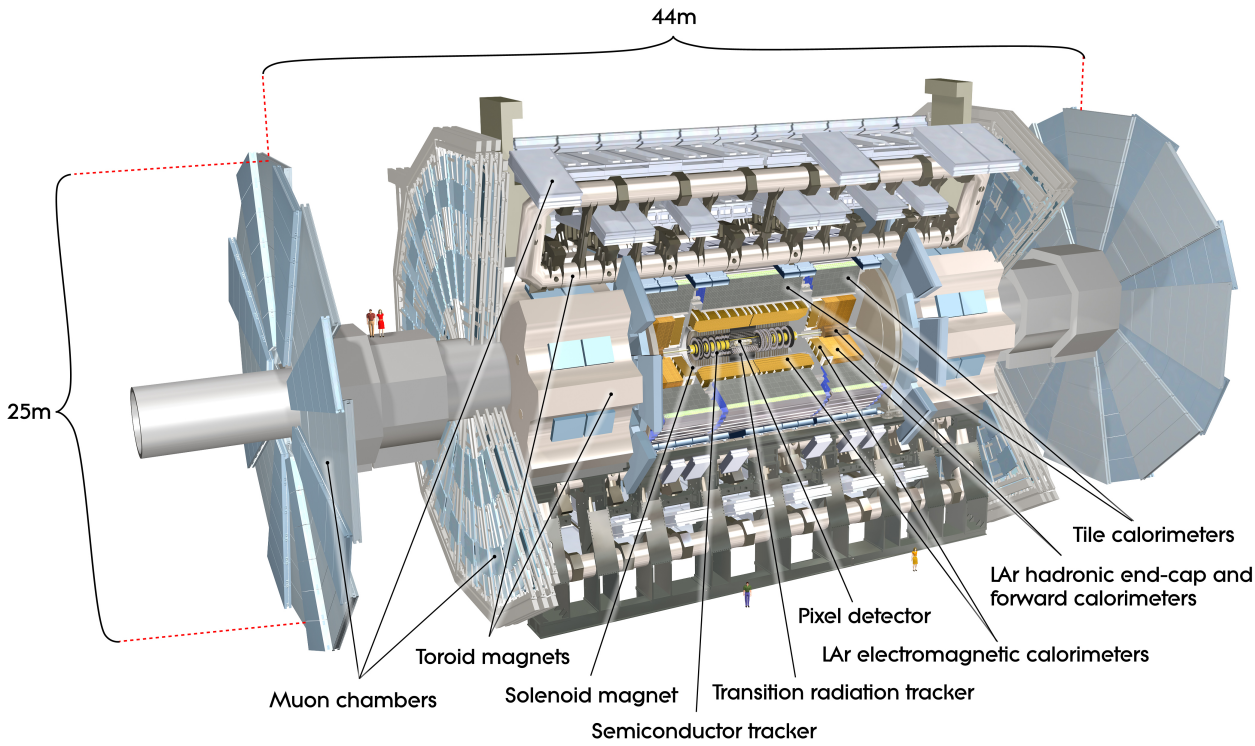


Figure 3.2: Sketch of the ATLAS detector

where p_x and p_y are the momenta along the corresponding axes.

The angular variables are defined within

$$\eta \in [-\infty, \infty], \phi \in [-\pi, \pi]. \tag{3.3}$$

3.2.8 Tracking detectors

To measure momentum, trajectory and charge of charged particles usually tracking detectors are used.

There are two main categories of tracking detectors following the same general principle, gaseous detectors and semiconductor detectors. If ionizing radiation passes any given medium it will create electron-hole pairs. These charge carriers can then be collected by an electric field. Depending on the detector the signal caused by the charge carriers can be related to the coordinate of ionization in space and time.

- Gaseous detectors: Gaseous detectors are based on the greater mobility of ions and electrons in the gas. The basis of the detector is usually a chamber filled with a proper gas. The gas contains an area of wires to which a strong electric field is applied. If gas atoms get ionized the charge carriers (electrons and ions) will drift to the wires and create a

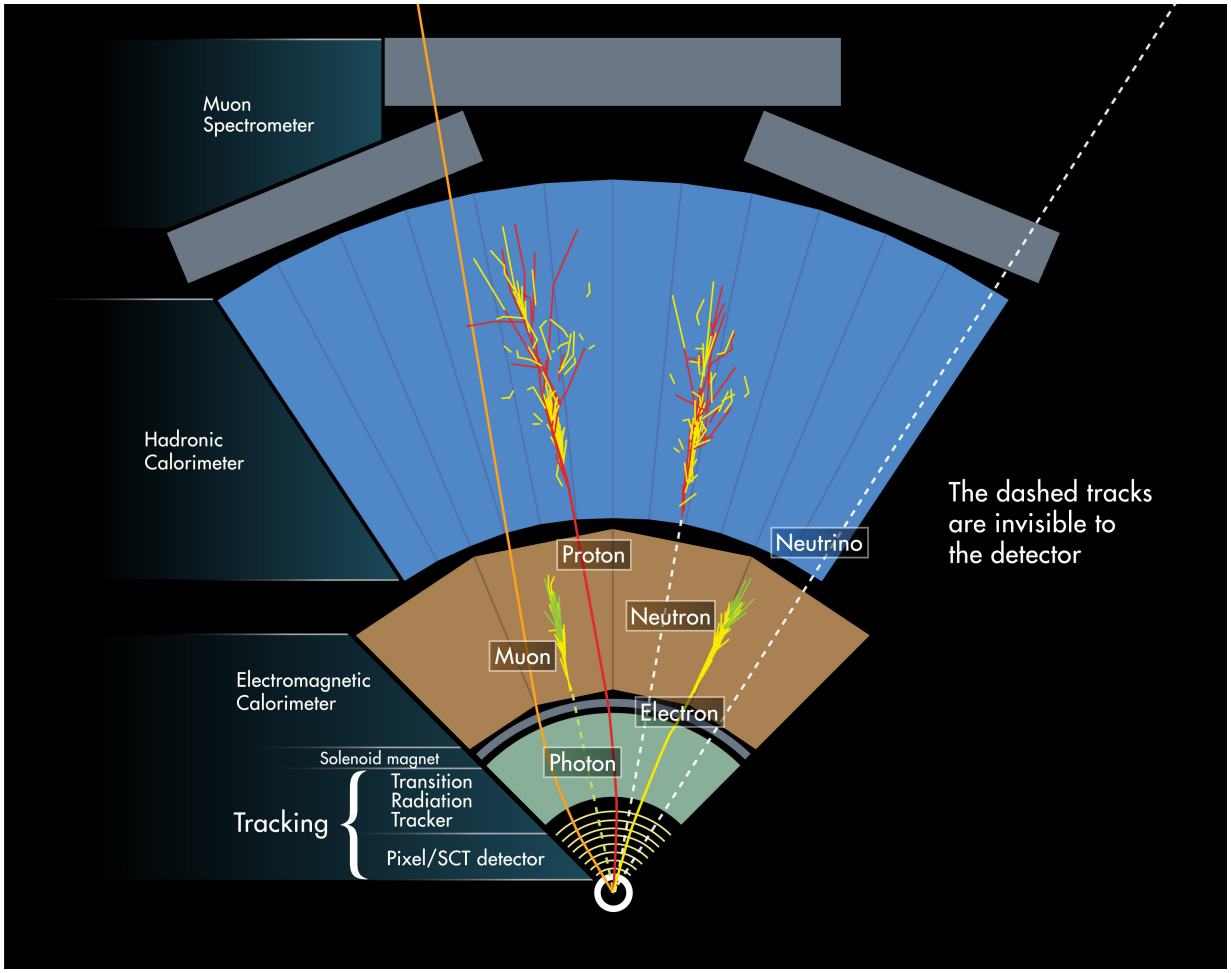


Figure 3.3: Scheme of the ATLAS-detector with examples of tyical particle detections. **[atlas_pictures]**

detectable signal. The wires give a rough estimation of space which is normally improved by calculating the exact ionization location from the drift time.

- Semiconductor detectors: Semiconductor detectors are as their name implies based on crystalline semiconductor material such as silicon and germanium. Their working principle is quite similar to that of gaseous detectors but the gas is exchanged by solid semiconducting material. In semiconducting material ionizing radiation will create electron-hole pairs instead of electron ion pairs that then can travel in a strong electric field to be detected. The big advantage of semiconductor detectors over gaseous detectors is that the energy required to create an electron-hole pair is about 10 times lower than the energy needed to ionize gas atoms. These detectors are commonly structured into wafers or pixels that allow a determination of the space.

Usually a magnetic field surrounds tracking detectors to bend the track and that way be able to compute the particles momentum and charge based on the curvature.**[leo]**

In the ATLAS detector both the inner detector and the muon spectrometer are tracking

detectors.

The Inner Detector

The innermost part of the ATLAS detector is called the Inner Detector, which consists of three sub-components, the Pixel detector (Pixel), the Semi-Conductor Tracker (SCT) and a Transition Radiation Tracker (TRT). Each of these sub-detectors is divided into the so called barrel part and two end-caps. The Inner Detector covers a region of $|\eta| < 2.5$.

The Muon spectrometer

The second tracking detector of ATLAS is the muon spectrometer which is the outermost part of the detector. The task of the spectrometer is to detect charged particles transversing the calorimeter without being stopped or deploying their complete energy, and to do both trigger and tracking to measure their momentum. Due to these two tasks the spectrometer is bifid with the first part being the trigger chamber covering a range of $|\eta| < 2.4$, followed by the high-precision chamber with a range of $|\eta| < 2.7$. The main detector's support feet cause a further gap at about $\phi = 300°$ and $\phi = 270°$.

3.2.9 Calorimeters

In particle physics a calorimeter is a device to measure first and foremost the total energy of a particle. Most of the time some positional information is taken additionally. The idea is that most particles loose all their momentum while crossing the calorimeter. Measuring the energy deposited this way gives a value for the particle's energy. Usually a particle deposits its energy by initiating a particle shower, the energy of which is then collected and measured. Calorimeters are distinguished by the main interaction of the particles one aims to detect.

Electromagnetic Calorimeters

Electromagnetic calorimeters are designed to detect charged particles that primarily interact via the electromagnetic interaction and to measure their total energy. Usually these particles are electrons and photons. There are various methods to construct these detectors. An example would be the usage of inorganic scintillators. These scintillators should be optically transparent and have a short radiation length to contain the shower in a compact region. The detection can then be followed by photon detectors with photo-multipliers which measure the emitted light being proportional to the detected particle's energy.**[leo]**

The electromagnetic calorimeter at ATLAS is a high-resolution and high-granularity liquid-argon sampling calorimeter using lead as absorber material. The calorimeter consists of two half-barrels which are only separated by a small gap at the interaction point. The end-caps at each side are segmented into two coaxial wheels to cover different polar angles.

Hadronic calorimeters

Hadronic calorimeters are used to obtain the energies of hadronic particles. Due to the relatively large distance between interactions these calorimeters occupy a significantly large volume in the detector.[**leo**]

A common technique to construct these calorimeters is a sandwich-like structure of alternating layers of high density absorber material and active material. The absorbers are used to develop the particle showers which then hit the active material and deposit their energy there. The determination of the particle belonging to the deployed energy relies on tracker information as is sketched in figure 3.3. Energy in the hadronic calorimeter without a track implies a neutral hadron, for example a neutron. A single track paired with a energy deposition means that the particle was a charged hadron like a proton and if many tracks belong to a deposition a jet has been the most likely origin of the energy deposition.

For more information about the ATLAS detector see the ATLAS design report [**atlas_report**] and for more information about the standard model particles see [**griffith**].

CHAPTER 4

The tW Channel

CHAPTER 5

Machine Learning

5.1 The concept of machine learning

For many decades computers have been an integral aspect of science, handling large amounts of data, completing tedious calculations and controlling experiments. For the most part the machines were assigned discrete tasks. They step-wise followed commands previously designed by human users and with non statistical outcomes. For particle physics in particular computers were used to select interesting data from large samples when given proper instructions making it possible to process amounts of data that would have exceeded human capacities. However the selection rules had always to be generated by the user, therefore requiring an in depth analysis of the underlying system. Artificial intelligence presents a way for a program to establish its own decision rules and improve these over several iterations and thereby learning to solve the problem all by itself.
There has been a great effort over the last decades trying to implement a way for machines to learn from experience and thereby enable them to analyse complex tasks in a wide range of applications.
Often times the human understanding of the task and the work done by the machine itself are in equilibrium as great knowledge of an assignment is required to find the most efficient way to have a machine not only work on it but work on it efficiently. This means fine tuning the machine's capabilities and degrees of freedom to the requirements and is called hyperparameter optimisation.
The task of machine learning can be simplified by comparing what a living being needs to be able to learn solving new problems, which can be split into understanding a system, evaluating a decision step and generating new decision steps.
Understanding a system means when given a task having a way to observe all features and possibilities relevant to work on the task. Humans have their senses to easily break down their observations into useful features and concepts that can then be processed for decision making. A computer has no senses built in and for most tasks this means that the step of filtering information for a relevant subset of features has still to be done by humans or a good preprocessing algorithm.
Once a system has been converted to a subset of features usable by a computer the step of mak-

ing own decisions has to be implemented. This can be done by weighting and interconnecting the information using structures often times inspired by the structure of the neurons in the human brain.
This process of decision making at the basis is somewhat random and so far no way of evaluating the success of a decision rule has been introduced. This is where a cost function comes into play.It pairs a generated decision rule with an estimator of its quality. Combined with an optimiser which suggests further steps, this function is a basis for a network to independently approach a good decision rule for a somewhat un-investigated topic.
A very commonly used machine learning algorithm is the artificial neural network which on its own forms a quite broad field that builds the base of this work. The most important concepts of machine learning will be explained using the example of a neural network. For the explanation of input, decision making process and the generation of a gradient the example of a neural network will be used.

5.2 Neural Networks

The artificial neural network, or just neural network, is the most commonly known approach to machine learning. Its structure is inspired by the neurons forming the human brain which is also where it gets its name from.
Instead of neurons a neural network consists of numerous very simple processors, called nodes. These nodes are usually structured into several layers. As presented in fig (todo). Also there is several ways to structure and connect these nodes often times matching a certain problem in this explanation only the most commonly way is explained. In that case each node of a layer is getting input from each node in the last layer and is outputting to each node of the following layer. Such a network structure is shown in figure 5.1. It describes the step between a node and the previous layer. The math will be explained in detail later.

The input layer

To understand a task and draw reasonable conclusions first the underlying system has to be understood. Its features need to be found and summarised. The human brain is capable of investigate unknown systems and learn the features that are the most unique or interesting. For that it has its senses to explore the system and process them later. To allow a machine to do something similar the unknown system has to be represented in a way that it is clear for the network what to look out for. This is usually the task that needs the most preprocessing by the user. The simplest case is to submit a list of variables to the network. In particle physics this could be kinematic variables of the particles in an event. The input to a neural network is just given to the input layer of nodes and then processed through each following layer. For different tasks different layers might deal with different parts of the information but in this work the linear way of giving all information used to an input layer and then processing it is used.

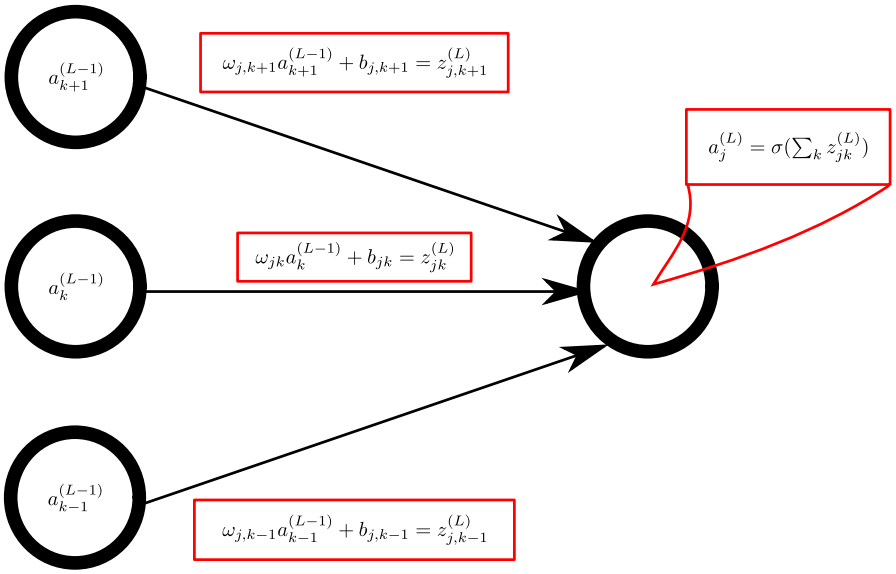


Figure 5.1: Network propagation from layer $(L-1)$ to layer L

Decision making process

Computers being not much more than very powerful calculators, excel at performing high numbers of clearly defined calculations. The trick is that the task has to be defined clearly. There usually is no fuzziness.
This is very different for the human brain. We rely on a certain fuzziness when processing information through a net of neurons where the output of every neuron is taken as input for the surrounding neurons. The challenge of machine learning is to represent this fuzziness by many somewhat discrete calculations. In the neural network the neurons and their fuzzy interaction is represented by the nodes which are very simple processors. Like neurons each node can use input from many other nodes to create a new output signal. Thereby the input information can be linked to each other in numerous ways. Combined with weights this allows for were clear decisions made from a complex set of input information.
The input of every node is the weighted output of all previous nodes as shown in equation (5.1). z_j^L is the input to the j-th in the L-th layer. ω_{jk}^L is the weight from the k-th node in the previous layer to the node, a_k^{L-1} is the nodes output and b_k the relevant bias. The sum indicates that all k previous node input to the j-th node.

$$z_j^L = \sum_{k=0}^{N} \omega_{jk}^L a_k^{L-1} + b_k \tag{5.1}$$

The weight allows to predict which variables are linked too each other or allow for better decision rules when combined but also easily to weight the importance of features. Furthermore the output of each node is non binary allowing to combine the input signals to a single output. This is called the activation function of a node. A common choice is the sigmoid function as

presented in equation (5.2)

$$a_j^L = \sigma(z_j^L) = \frac{1}{1 + e^{z_j^L}} \tag{5.2}$$

The sigmoid function has output between 0 and 1 which is often times what one wants for nodes especially for the final layer as we are looking for predictions of outcome probabilities. There is several other commonly used activation functions as shown in table y.
For this thesis especially the exponential linear unit or *elu* and the rectified linear unit or *relu* were tested.
elu is good for converging the cost to zero rather fast and has the possibility of negative output. *relu* allows for the same benefit as sigmoid while requiring less computational power for the simply linear output for positive values.

Of course the network does still not know the task assigned to it but if we add a cost function to estimate the quality of a decision rule created by a certain combination of the input we can easily make the network approach a better decision rule and hopefully the solution of the problem.
In supervised learning the network is trained with a set of known data. Each event of the training set is tagged with its outcome. Comparing this truth information to the network output makes it very easy to calculate the loss. A possible loss function is the crossentropy or just binary crossentropy for a binary output result. As in this work the result is binary, signal and background, the binary crossentropy is the natural choice. Equation (5.3) shows the underlying equation where p is the estimated probability for the outcome $\hat{y}$ and y is the truth tag for a correct or incorrect guess.

$$C = -(y \log p + (1 - y) \log(1 - p)) \tag{5.3}$$

Optimizers - Choosing the next step

The probability interaction of the nodes combined with the cost function enables a network not only to create a decision rule but also to evaluate it. The last missing part is an algorithm that can estimate a step to a decision rule that further minimises the step. One could certainly do this randomly until the network finds a very low costed decision rule if infinite computational power given but that would not be a learning process as desired.
The output of each node in the final layer is defined by the weighted and biased information of the previous nodes and lastly the activation function. For one connection therefor there are three variables that affect the input to the cost function. Summarising this for all nodes in a vector defines the gradient for the cost function. Building the derivative of this gradient allows to define which connection had a strong impact on the deviation and how to change it. This is called backpropagation. The algorithm works as follows:

1. A certain set of input variables is iterated through all layers of a network resulting in an estimator $\hat{y}$ at each output node.

2. The sum of deviations from the true value at all output nodes is determined as the cost C of the setup. The cost just defines how high our error is.

3. This error is propagated back through the network changing the weights and parameters at each connection according to their impact on the overall cost. Following the previous notation for node output this can be described as:

$$\frac{\partial C}{\partial a_k^{L-1}} = \sum_{j=1}^{N} \frac{\partial z_j^L}{\partial a_k^{L-1}} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial C}{\partial a_j^L}$$

This backpropagation algorithm is the backbone of neural networks and also variable by different cost or loss functions usually follows these steps over the learning process. The decision step based on the gradient defined above is defined by the networks optimizer and deserves a bit more attention. There is different choices of optimizers trying to accomodate different problems as well as some parameters important to understand and tune for an effective training. First we define the gradient g in a more general way. The batch size is m, f is the network for configuration θ and output $\hat{y}$. The target is y.

$$g = \frac{1}{m} \nabla_\theta \sum_j L(f(\hat{y}^j; \theta), y^i) \tag{5.4}$$

The configuration θ is then updated using the gradient and a constant η called the learning rate as it determines the step size for each update.

$$\theta\prime = \theta - \epsilon g \tag{5.5}$$

This process of optimization is called stochastic gradient descent and can be called the basis of all optimizers. Its only hyperparamter is the learning rate. A good learning rate should be small enough to avoid oscillations but high enough to approach a minimum efficiently fast. A good estimate is given by the Robbins Monro condition:

$$\sum_k \epsilon_k = \infty \tag{5.6}$$

$$\sum_k \epsilon_k^2 < \infty \tag{5.7}$$

Often times a second hyperparameter of an optimizer as SGD can be the momentum ν [4]. Momentum makes sure that each step is scaled by how aligned previous steps were. That means it will allow avoiding local minima by enlarging steps at the beginning of the training but also will slow down at the end of the training when the steps become shorter. It promises to speed up the training with less risk of large oscillations as a large learning rate would probably result in. Momentum also takes a single scaling hyperparamter α and is updated each step

following:

$$\nu' = \alpha\nu - \epsilon\frac{1}{m}\nabla_\theta \sum_j L(f(\hat{y}^j;\theta), y^j) \tag{5.8}$$

$$\theta' = \theta + \nu \tag{5.9}$$

Alternatively one can use Nesterov momentum [4] and update the step after applying the gradient:

$$\nu' = \alpha\nu - \epsilon\frac{1}{m}\nabla_\theta \sum_j L(f(\hat{y}^j;\theta + \alpha \times \nu), y^j) \tag{5.10}$$

$$\theta' = \theta + \nu \tag{5.11}$$

Lastly it can be helpful to decrease the learning rate of the network stepwise while approaching a minimum to avoid oscillations or leaving the minimum in general further. This can be accomplished by the hyperparameter of learning rate decay. It just updates the learning rate in each iteration t by a small hyperparamter ϕ

$$\epsilon' = \frac{\epsilon}{1 + \phi t} \tag{5.12}$$

Another commonly used optimiser is Adam. The name is derived from adaptive moments.
https://jamesmccaffrey.wordpress.com/2017/06/06/neural-network-momentum/

5.3 Regularisation and Optimisation

Fluctuations and noise in the training sample can be a big problem for a model trained on the sample as a neural network might pick noise and random fluctuations up as features for the decision rule.
This basically is the process of overfitting. The network sees way more features to work with than those actually present in reality or even in a different test sample. The most extreme scenario is that your network is large and deep enough to pick up every single feature in the training sample. If that happens the training error becomes very low and indicates a very good decision rule. For a different sample this decision rule is at most very unreliable but probably strictly wrong resulting in a high test sample error. The network just picked up and remembered every single feature in the training sample instead of general correlations. It becomes a mask of the sample.
A possible way to solve this issue is to stop the training early or to find a good way when to stop the training. This way noisy features might not yet have been picked up by the training. Then again this might also lead to a suboptimal result of the overall training as one cannot be sure that the correct features always get picked up first. Especially when large systematic uncertainties have to be dealt with.

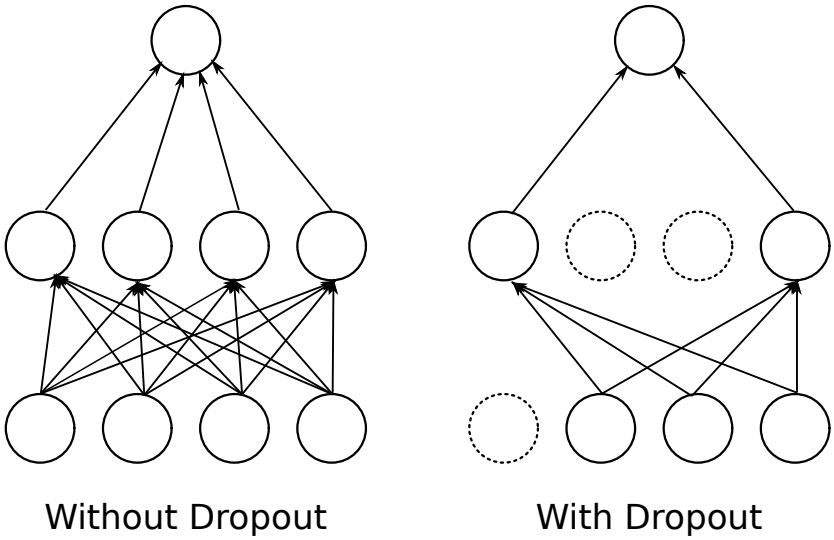


Figure 5.2: Sketch of network before and after the inclusion of dropout. On the left hand side dropout is not applied and all nodes are connected. On the right hand side the dashed circles are nodes excluded by dropout and therefor not connected to the other nodes.

More sophisticated approaches are called regularisations of a neural network. The most commonly used solution is a so called Dropout layer described in the subsection 5.3.1. Additionally a batch normalisation can have an effect of regularisation and is therefor introduced in this section as well.

5.3.1 Dropout

Dropout is attempting to keep the network from relying on subdominant features too much by removing different nodes in each iteration. This forces the network to build models that are not based on strong correlations between nodes. The weights become less interdependent. To simplify it a lot it means training several neural networks depending on which nodes are turned on during a training epoch. It keeps the training in motion for a large number of epochs. Dropout is added to each layer of a network and can also be restricted to a subset of layers too. It slows down the training as the additional motion slows down the process of finding a minimum but it also accelerates each epoch slightly as it simplifies the network architecture.

5.3.2 Batch normalization

In supervised learning the training result is heavily dependent on the set of data the network is trained on. This means that the performance might change a lot when the test data is very different. Imagine a classifier distinguishing between pictures that show cars and pictures that do not show cars. If the training set contains predominantly green cars the colour green might end up as a strong indicator for the classification car. In general the colour green will not be as dominant and the network will perform slightly worse when trying to classify cars of a different colour. Formally such a change of input is called a covariance shift.
A way to reduce the effect of covariance shift is batch normalization. The output of nodes in

general and the weight of a connection in a neural network is not necessarily limited allowing for certain connections to be really dominant and overshadowing less dominant features. We want to avoid this as the dominance of some features might just be present in the training sample. This can be achieved by normalizing the output of each layer in the network to the total output and thereby minimising the effects of strongly overrepresented features. This is done by normalizing each output to the mini-batch mean μ_B and the mini-batch standard deviation σ_B^2.

$$\mu_B = \frac{1}{m}\sum_i x_i \tag{5.13}$$

$$\sigma_B^2 = \frac{1}{m}\sum_i (x_i - \mu_B)^2 \tag{5.14}$$

$$x_{i,norm} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{5.15}$$

5.4 Adversarial Neural Networks

This main part of this work of the examination and training of an Adversarial Neural Network. An adversarial neural network consists of a classifier and a second network that tries to regularise the output of the first network.
The first section of this chapter motivates this setup in general and on the basis of particle physics. The mathematical basis is then introduced as well as the setup of the network. [3]

5.4.1 The adversarial neural network

Neural networks have been very efficient for classifying tasks but less successful for generative tasks. This was the original problem that gave birth to the idea of a generative adversarial network. Generative networks often times have output that is very easy to distinguish from real samples. The solution suggested is adding a classifier that tries to distinguish between generated samples and real samples. As long this adversary is able to accomplish this task the first network fails at its generative task. Training the two networks against each other should disincentive the generative network from using the features not dominant in real samples.
In this work the first network is not a generator but a classifier separating signal events from background events in a Monte Carlo simulation. These simulations contain systematic uncertainties. The classifier should not be too dependent on variables with high systematic uncertainties. If the classifier has these strong dependencies on systematic uncertainties it might lead to a high co-variance shift. Instead of a generated sample and a truth sample a so called nominal sample and up and down samples are used as input for the second network. Up and down samples have slightly different distributions than the original samples because of changes to the variables with the systematic uncertainties. Training the second network on determining whether it is looking at a nominal or a systematic sample allows to estimate how dependent the model is on variables with high systematic uncertainties. Training the classifier against the adversarial network promises to reduce the effect of systematic uncertainties on the

model. Mathematically this comes down to a minimax decision rule or a competition between two neural networks. Let us call the Classifier $Net1$ and the Adversary $Net2$ then the problem becomes:

$$\min_{Net1} \max_{Net2} V(Net1, Net2) = \mathbb{E}_{x \sim \rho_{data}}[logNet1(x)] + \mathbb{E}_{z \sim \rho_{sys}}[log(1 - Net2(z))] \tag{5.16}$$

$V(Net1, Net2)$ is the combined value function for the two adversary networks. The first network is trained to be an optimal classifier represented by $logNet1(x)$ while the second network is trained to distinguish between the nominal and systematics distribution $z \sim \rho_{sys}$ represented by $log(1 - Net2(z))$. In theory the first classifier should be trained slowly and kept close to its optimum while the second network slowly learns and allows the first network to adapt to it. This is achieved by training the two networks successively over multiple iterations using a combined value function. As this value function a combined loss function is used. It is just the difference between the two separate loss functions with a hyperparameter λ to control the impact of the adversary:

$$\mathcal{L} = L_{net1} - \lambda L_{net2} \tag{5.17}$$

In the first step of each iteration the first network is trained using the combined loss function $\mathcal{L}$. In the second step the second network is trained using its simple loss function λL_{net2}. Each of the networks has the usual set of hyperparameters to optimise explained in detail in ref missing.

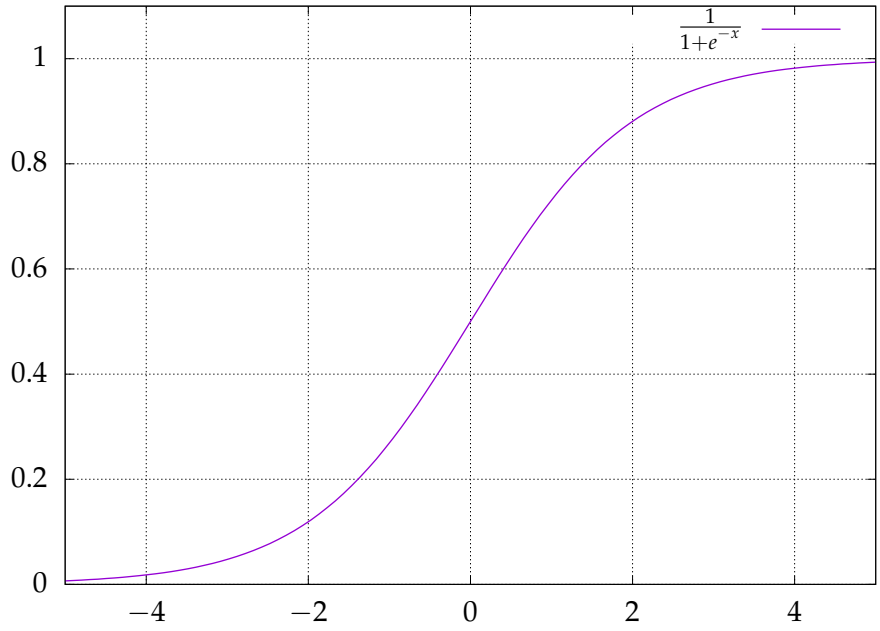
CHAPTER 6

Tuning and Results of a simple NN

what about backend, variable choice

6.1 Choice of nodes and layers

6.2 Choice of the Optimizer


$\frac{1}{1+e^{-x}}$
1
0.8
0.6
0.4
0.2
0
−4
−2
0
2
4

CHAPTER 7

Adversarial Neural Network

7.1 Setup of the first network

7.2 Setup of the second network

CHAPTER 8

Conclusions

CHAPTER 9

Machine Learning Tools

This chapter introduces the tools used for the implementation of the neural network. The main tool is the Keras python library. Keras is toolset for deep learning networks developed by google that is able to run on a variety of backends. It summarizes the vector calculations needed for a neural network in modules making it perfect for fast and easy usage.
The backend used in this work is tensorflow.

9.1 Keras

Keras is an application programming interface. It is written in python and able to run on Tensorflow, CNTK or Theano. It was developed by google and summarizes the necessary calculations for running a deep neural network training in fast and easy modules.
[4]

9.2 Tensorflow

Bibliography

[1] H. Kopka and P. W. Daly, *Guide to LaTeX*, 4th ed., Addison-Wesley, 2004 (cit. on p. 1).

[2] F. Mittelbach and M. Goossens, *The LaTeX Companion*, 2nd ed., Addison-Wesley, 2004 (cit. on p. 1).

[3] I. J. Goodfellow et al., *Generative Adversarial Networks*, arXiv e-prints, arXiv:1406.2661 (2014) arXiv:1406.2661, arXiv: 1406.2661 [stat.ML] (cit. on p. 22).

[4] F. Chollet et al., *Keras*, https://keras.io, 2015 (cit. on pp. 20, 31).

APPENDIX A

Useful information

In the appendix you usually include extra information that should be documented in your thesis, but not interrupt the flow.

List of Figures

List of Tables