Bachelorarbeit

Zur Erlangung des akademischen Grades
Bachelor of Science

Classification of tZ-Production Events Using an Artificial Neural Network

Autor:
Keno Goertz

Gutachter:
Prof. Dr. Thomas Lohse
Prof. Dr. Heiko Lacker

Eingereicht am Institut für Physik
der Humboldt-Universität zu Berlin am:
23. August 2019

Abstract

An Artificial Neural Network was trained to identify FCNC $gu \to tZ$ events using data from a Monte Carlo simulation. This network was then used to classify experimental data from the ATLAS detector, which was recorded at a collision energy of $\sqrt{s} = 13\,\mathrm{TeV}$ and an integrated luminosity of $\mathcal{L}_{int} = 140\,\mathrm{fb}^{-1}$. A 95% confidence upper limit on the cross section of the tZ-production process was obtained:

$$\sigma_{95\%} = 40.93\,\mathrm{fb}$$

Contents

1 Introduction

Particle physics (also known as *high energy physics*) is a field of physics studying the fundamental particles that make up radiation and matter. The best current understanding of particle physics is called the *Standard Model* (SM). While the SM describes a lot of observable interactions very precisely, it still fails to explain some observations. For this reason, new theoretical models have been introduced, some of which predict *Flavor Changing Neutral Currents* (FCNC), which are highly suppressed in the SM. So far, the theoretical suppression of FCNC is in high agreement with experimental observations. Thus, a discovery of FCNC would greatly influence constraints on theoretical model-building, which makes them an interesting research topic. This research looks for a hypothesized FCNC process in which proton-proton collision results in the production of a top quark and a Z-boson.

1.1 Particle physics

1.1.1 Particle accelerators and the LHC

The *Large Hadron Collider* (LHC) is the world's largest particle accelerator, with a circumference of 27 kilometers. It also holds the record for the highest collision energy, at $13\,\mathrm{TeV}$.[3] The LHC is mostly a ring of superconducting magnets, which have the purpose of keeping the particle beam centered in the ring. It also contains several accelerating structures, using electric fields to increase the energy of particles passing by. One type of collision that has had ongoing measurements in the LHC is proton-proton-collision (pp), with this research's process being a result of pp-collision at $13\,\mathrm{TeV}$. This collision energy, or *invariant mass* $\sqrt{s}$ is defined (in natural units) as

$$\sqrt{s} = \sqrt{\left(\sum_i E_i\right)^2 - \left(\sum_i \vec{p_i}\right)^2} \tag{1.1}$$

where E_i are the energies of the incoming particles, and $\vec{p_i}$ are their momenta.

The LHC features multiple detectors, with the data used in this research being from the *ATLAS* detector. It encases the place of collision and is made up of multiple detection elements. The first one is the *Inner Detector*, which tracks the paths of charged particles in a magnetic field, giving valuable information about particle charge, type and momentum. This is followed by two calorimeters, with the purpose of measuring the particles' energies through absorption in electromagnetic interactions. The *Electromagnetic Calorimeter* absorbs electrons and photons, but hadrons and heavier leptons pass through it. The hadrons are absorbed later in the *Hadronic Calorimeter*, but muons still pass through. Their properties are finally measured with the *Muon Spectrometer*. Neutrinos are not detected due to the extremely low likelihood of interaction with detector elements. They are simply registered as a "missing" energy and momentum.

After interpreting the raw detector data, the particles' types, energies and momenta are obtained. The three-dimensional momentum vector is usually represented in a coordinate system using the *pseudorapidity* η, which describes the angle of a particle's momentum relative to the beam axis. Its definition is

$$\eta = -\ln\left[\tan\left(\frac{\theta}{2}\right)\right]$$

where θ is the polar angle between the beam axis and the particle's momentum. The momentum's component that lies in the plane perpendicular to the beam is represented in polar coordinates, where p_T is the *transverse momentum* and ϕ is the polar angle in the transverse plane.

1.1.2 Detection of bottom- and top quarks

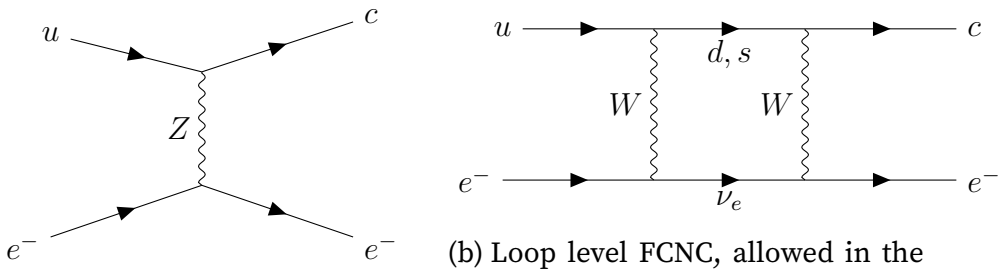
Quarks and gluons can not exist individually due to color confinement. In the SM, they thus combine with quarks and antiquarks spontaneously created from the vacuum and form hadrons. These hadrons are then detected in the hadronic calorimeter as *hadronic jets*. Quarks can not be directly measured in the detector, but their existence can be inferred from these jets. *b-tagging* is a method used to determine whether such a jet was caused by the hadronization of a bottom quark. b-jets have several unique features which can be measured in the detector, making it possible to identify bottom quarks as the source of a jet:

- b-mesons generally have longer lifetimes than lighter mesons, due to the small elements in the CKM matrix V_{ub} and V_{cb}. This longer lifetime enables the b-meson to travel a measurable distance in the detector before decaying into the lighter hadrons, creating a secondary vertex. b-tagging algorithms can try to trace the jet back, and in case a secondary vertex is found, infer that the jet is a b-jet.

- The high mass of the b-quark may lead to larger transverse momenta with respect to the jet axis of the decay products of a hadron containing a b-quark. This leads to a wider opening angle of the whole jet, so this property can be used for tagging algorithms.

It should be noted that the same principle can be applied to charm quarks, although the lower masses make it much less effective. Tagging algorithms for up-, down- and strange quarks do not exist.

The top quark presents a very different situation. It is much heavier than all other quarks, making it the only quark with a lifetime so short that it decays before hadronizing. In all observations so far, the top quark decays over weak interaction into a bottom quark, leaving behind a b-jet and the W^+-boson's decay products. [9]



(a) Tree level FCNC, forbidden in the SM

(b) Loop level FCNC, allowed in the SM but suppressed by the GIM mechanism

Figure 1.1: Example Feynman diagrams for FCNC

1.1.3 The Standard Model and flavor changing neutral currents

The *Standard Model* (SM) reflects the best current theoretical understanding of particle physics. However, it fails to explain a number of phenomena. Examples are gravity, dark matter and dark energy, matter-antimatter asymmetry, and neutrino masses. These problems have motivated the development of several theoretical models *beyond the Standard Model* (BSM). A number of these models predict *Flavor Changing Neutral Currents* (FCNC). These are hypothetical processes where the flavor of a fermion is changed without affecting the electric charge. In the SM, they are forbidden on tree level and highly suppressed on loop level by the GIM mechanism [10]. All measurements so far have been in agreement with this. Fig. 1.1 shows an example for a forbidden tree level FCNC process and an allowed, but suppressed, loop level process. The suppression of FCNC is an important constraint in model building and a discovery would be a direct indicator of new physics.

The current state of research by the LHC top-quark physics working group on FCNC processes involving top quarks is summarized in Fig. 1.2. No discovery has been made yet, so this plot summarizes the current upper limits on the processes' branching fractions.

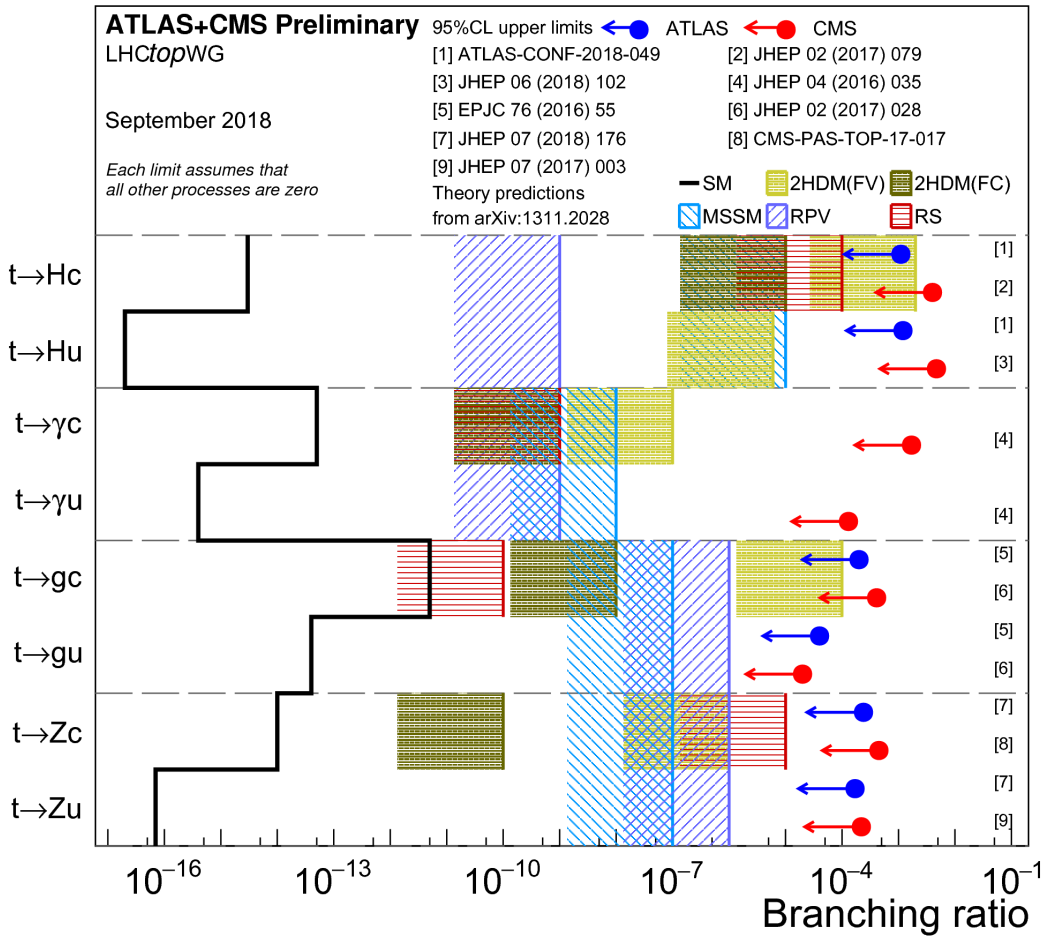


Figure 1.2: Current upper limits on FCNC processes involving top quarks by LHC*top*WG[2]

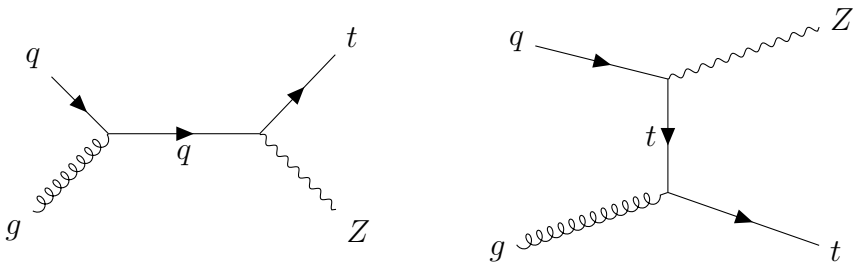
1.1.4 tZ-production

This research looks for tZ-production via FCNC. The two channels for this signal process are shown in Fig. 1.3.

After the production, there are several possible decay modes for the Z boson. The branching fractions of the most relevant decays are [9]

$$\begin{aligned}\mathcal{B}(Z \to \text{hadrons}) &= (69.911 \pm 0.056)\% \\ \mathcal{B}(Z \to \text{invisible}) &= (20.000 \pm 0.055)\% \\ \mathcal{B}(Z \to \ell\ell) &= (10.099 \pm 0.012)\%\end{aligned}$$

This research exclusively looks at events in which the Z boson decayed into a lepton pair. Even though these make up the smallest branching fraction, they provide more sensible data than hadronic and invisible decays. Invisible decays into neutrinos impose the obvious problem of the neutrinos being invisible to the detector, and hadronic jets impose several other problems such as flavor-tagging being impossible for light quarks. Another approach would be to look at $Z \to bb$ decays, which have a branching fraction of $(15.12 \pm 0.05)\%$ [9] and provide the benefit of being identifiable via b-tagging. However, this approach has already been taken by a previous thesis. [11]



(a) s-channel

(b) t-channel

Figure 1.3: Feynman diagrams of the studied tZ-production process

For the top quark, there is only one relevant decay mode, which is its weak decay into a bottom quark. The W-boson in this decay may then decay into a

lepton-neutrino pair or into a quark pair. The resulting branching fractions for the top quark's decay modes are [9]

$$\mathcal{B}(t \to q\bar{q}b) = (66.5 \pm 1.4)\%$$
$$\mathcal{B}(t \to b\ell\nu) = (33.8 \pm 1.1)\%$$

This research only looks at events in which the top quark decays with $t \to \ell\nu$.

Additional jets to the b-jet may be observed, caused for example by random gluon emissions. This research thus also takes into account events with an additional second jet. The complete Feynman diagram for the s-channel of the signal $gu \to tZ \to b\ell\ell\ell\nu$ is shown in Fig. 2.1. The branching fraction of this decay can be calculated by multiplying the relevant mentioned branching fractions:

$$\begin{aligned} \mathcal{B}(tZ \to b\ell\ell\ell\nu) &= \mathcal{B}(Z \to \ell\ell) \cdot \mathcal{B}(t \to b\ell\nu) \\ &= (3.41 \pm 0.12)\% \end{aligned} \tag{1.2}$$

In the Monte Carlo simulation that was used to train the network, the signal's weights were normalized so that the cross section of the entire process $gu \to tZ$ is at an arbitrarily chosen $1\,\mathrm{pb}$. This means that these branching fractions do not need to be considered when calculating the measured cross section for the signal later.

1.2 Artificial neural networks

Artificial Neural Networks (ANN) are a machine learning method inspired by the structure of biological neural networks. An ANN "learns" iteratively by performing mathematical operations on the input data, using a kind of *loss*

function to assess how its output compares to the desired output, and applying a *backpropagation* algorithm to adjust its parameters, in hopes of a better score on the loss function in the next iteration. Neural networks have been used for decades in particle physics, mainly for the tasks of event classification, function approximation and pattern recognition. [15] They have also found usage in the search for FCNC processes. [5,4]

1.2.1 Basic function and structure

The fundamental parts of an ANN are the artificial neurons. These neurons receive inputs x_i that are multiplied with the weights w_i and produce a single output y. They also have an activation function $A : X \to Y$. The purpose of this function is, most importantly, to introduce nonlinearities and, sometimes, to limit the possible outputs to the set Y (e.g. real numbers between zero and one). Finally, a neuron may also have a bias b independent of the inputs, in order to allow shifting of the activation function by a constant value. For a neuron with N inputs, its output is given by

$$y = A\left(\sum_{i=1}^{N} w_i x_i + b\right) \tag{1.3}$$

A neural network consists of multiple *layers* of neurons. Each layer may have a different number of neurons. The outputs of the first layer's neurons are connected with the inputs of the second, and so on. In a fully connected network, each output of layer l is connected to all inputs of layer $l + 1$. The first layer's inputs are the network's inputs, hence it is called the *input layer*. Conversely, the last layer's outputs are the network's outputs and it is called the *output layer*. The layers in between the input- and the output layer are called the *hidden layers*. Most problems only require one hidden layer, while more complicated problems like image recognition may require multiple hidden layers, which is called *deep learning*. While each neuron could have a different activation function, one generally sticks to the same activation function for all neurons, perhaps using a different function in the output layer in order to limit the network's possible outputs to a specific set. In

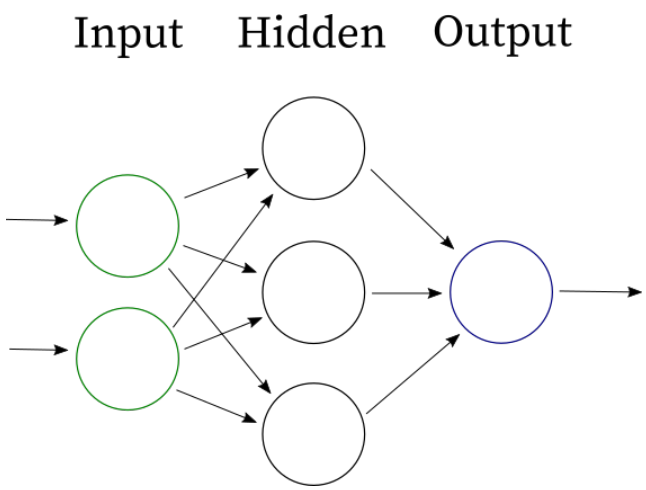


Figure 1.4: Example of a fully connected neural network with one hidden layer, two input neurons, three neurons in the hidden layer, and one output neuron

general, the output y_j^l of a neuron j in the layer l of a fully connected neural network with N_l neurons in the layer, with the bias b_j^l of the neuron and the single activation function A used for all neurons, is

$$y_j^l = A\left(\sum_{i=1}^{N_{l-1}} w_{ij}^l y_i^{l-1} + b_j^l\right) =: A(z_j^l) \tag{1.4}$$

where w_{ij}^l are the weights of the connections between neurons y_i^{l-1} and y_j^l. An example of such a fully connected ANN can be seen in Fig. 1.4. The learning process works by using a backpropagation algorithm to change the weights w_{ij}^l and the biases b_j^l.

1.2.2 Training process

The training of a neural network works via gradient descent of a loss function $\Lambda(\vec{y}^L, \vec{o})$, where L is the number of layers (so $\vec{y}^L$ is the network's out-

put vector) and $\vec{o}$ are the expected outputs of the network. This method requires knowledge of the expected outputs, which means each entry in the input dataset needs to be labeled with a corresponding output vector. This is called *supervised learning*. The loss function should be minimal for $\vec{o} = \vec{y}^L$ and increase with increased difference between the expected outputs and the network's actual outputs. Since the network's outputs are entirely dependent on its weights, biases and the inputs, the loss function can also be written as $\Lambda(\vec{x}, \underline{w}, \underline{b}, \vec{o})$, where $\vec{x}$ is the input vector, $\underline{w}$ is the three-dimensional weight matrix and $\underline{b}$ is the two-dimensional bias matrix. Then, the average $\overline{\Lambda}(\underline{w}, \underline{b})$ of the loss function over all input vectors in the training dataset is calculated. This average is independent of the individual input and output vectors, and thus solely depends on the weights and biases of the network.

The idea of gradient descent is that the negative gradient of this function provides the steepest descent to a local minimum. So the weights and biases can be adjusted in each iteration like:

$$(w_{ij}^l)' = w_{ij}^l - \alpha \frac{\partial \overline{\Lambda}}{\partial w_{ij}^l} \tag{1.5}$$

$$(b_j^l)' = b_j^l - \alpha \frac{\partial \overline{\Lambda}}{\partial b_j^l} \tag{1.6}$$

where α is the step size. Several different optimization algorithms have been created that adjust this step size in some way to help escape local minima in hopes of finding the global minimum or at least a deeper local minimum. The weights and biases are usually initialized randomly before the training is started.

Now, the only task left is to determine the gradient. This is where backpropagation comes into play. Mathematically, it is a method that applies the chain rule in order to calculate the derivatives of the loss function. For each input vector, the gradient $\nabla_{(\underline{w},\underline{b})}\Lambda$ is calculated. After this has been done for the entire dataset, the average is calculated to obtain $\nabla\overline{\Lambda}$ and Eq. (1.6) is applied to adjust the weights and biases. A period in which the network has seen the entire training dataset is called an *epoch*. In practice, the gradient descent is usually done in mini-batches from the training dataset, meaning

that there are many gradient descent iterations in one epoch. This means that the calculated gradient is less accurate, but it is good enough to reach the local minimum and this technique saves computational time.

Calculating the derivative for a weight connecting a neuron in the last hidden layer to another neuron in the output layer works as follows (using Eq. (1.4)):

$$\begin{aligned}\frac{\partial \Lambda}{\partial w_{ij}^L} &= \frac{\partial \Lambda}{\partial y_j^L} \cdot \frac{\partial y_j^L}{z_j^L} \cdot \frac{\partial z_j^L}{\partial w_{ij}^L} \\ &= \frac{\partial \Lambda}{\partial y_j^L} \cdot \frac{\partial y_j^L}{z_j^L} \cdot y_i^{L-1}\end{aligned} \tag{1.7}$$

The first and second term can be calculated analytically from the loss function and the activation function. The same can be done for a weight connecting a neuron in layer $L-2$ to one in layer $L-1$. This only introduces more terms with the chain rule, taking one further back in the network, hence the name backpropagation.

$$\begin{aligned}\frac{\partial \Lambda}{\partial y_j^{L-1}} &= \sum_{k=1}^{N_L} \left(\frac{\partial \Lambda}{\partial y_k^L} \cdot \frac{\partial y_k^L}{\partial z_k^L} \cdot \frac{\partial z_k^L}{\partial y_j^{L-1}} \right) \\ &= \sum_{k=1}^{N_L} \left(\frac{\partial \Lambda}{\partial y_k^L} \cdot \frac{\partial y_k^L}{\partial z_k^L} \cdot w_{jk}^L \right)\end{aligned} \tag{1.8}$$

$$\begin{aligned}\frac{\partial \Lambda}{\partial w_{ij}^{L-1}} &= \frac{\partial \Lambda}{\partial y_j^{L-1}} \cdot \frac{\partial y_j^{L-1}}{\partial z_j^{L-1}} \cdot \frac{\partial z_j^{L-1}}{\partial w_{ij}^{L-1}} \\ &= \frac{\partial \Lambda}{\partial y_j^{L-1}} \cdot \frac{\partial y_j^{L-1}}{\partial z_j^{L-1}} \cdot y_i^{L-2}\end{aligned} \tag{1.9}$$

This process can be repeated until the inputs of the network are reached. Applying the same technique to the biases leads to the following equations:

$$\frac{\partial \Lambda}{\partial b_j^L} = \frac{\partial \Lambda}{\partial y_j^L} \cdot \frac{\partial y_j^L}{\partial z_j^L} \tag{1.10}$$

$$\frac{\partial \Lambda}{\partial b_j^{L-1}} = \frac{\partial \Lambda}{\partial y_j^{L-1}} \cdot \frac{\partial y_j^{L-1}}{\partial z_j^{L-1}} \tag{1.11}$$

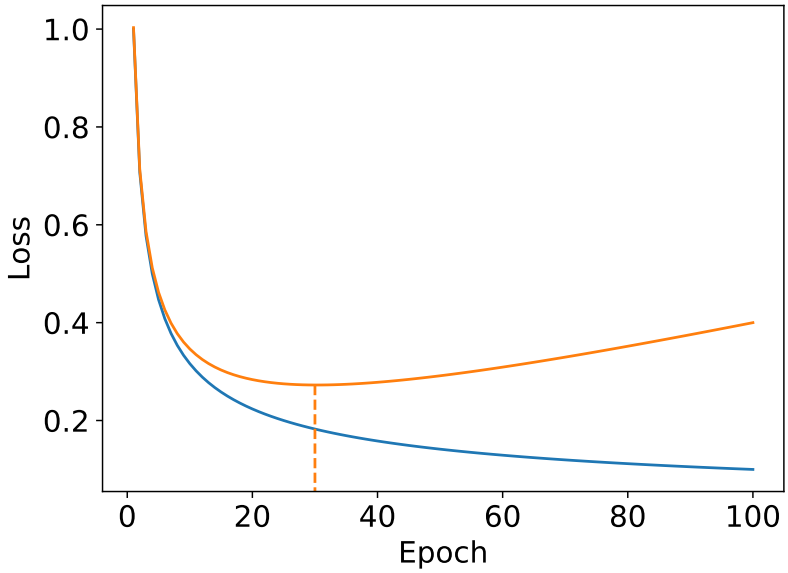


Figure 1.5: Visualization of overfitting, the dashed line marks when the validation loss becomes minimal and training should be stopped.

1.2.3 Validation process and overfitting

Neural networks are prone to a problem called *overfitting*. This is when the network starts to learn patterns specific to the training dataset that don't generalize well to data that has not been used for training. It is similar to how a higher-order polynomial may fit to some random noise instead of fitting the underlying function. Overfitting becomes especially problematic with

insufficient training data or an overly complicated network architecture, for example unnecessarily many hidden neurons. Therefore, it is crucial to have sufficient training data and to optimize the network's architecture, for example adding neurons until it begins to overfit.

In order to check that the network generalizes well, the input data should be split into a training and a *validation* dataset. The latter is not used to train the network, meaning it does not contribute to the loss function's gradient. Instead, the loss of the validation dataset is calculated after each gradient descent iteration and can be compared to the training loss to assess the amount of overfitting. This monitoring is especially useful to determine when the optimal weights have been reached, which is when further training decreases the training loss but increases the validation loss. Fig. 1.5 visualizes what overfitting may look like and marks what would be the optimal point to stop training.

2 Construction and training of the neural network

2.1 Input data

The data used for the training of the ANN was created in a Monte Carlo simulation. It includes the signal process that this research is looking for and a number of different background processes. There are 26,174 entries for signal events and 192,422 entries for background events in the dataset. The signal process is $gu \rightarrow tZ$, and its Feynman diagram is shown in Fig. 2.1. The background processes include:

- Standard model tZ-production: 71,175 entries
- Top pair production: 1,969 entries
- Top pair production in association with a boson (ttZ or ttW): 35,918 entries
- WZ-production: 43,269 entries
- ZZ-production: 35,411 entries
- Z+jets: 1,898 entries
- tWZ-production: 2,782 entries

As will be explained in the next section, only the WZ and the ZZ background were used in the optimization process of the network's structure. However, all backgrounds were used to train the final network. The dataset consists of weighted entries that provide values for:

- Lepton flavors and charges

- The number of jets (1 or 2)
- Jet b-tagging scores
- Angle ϕ, transverse momentum p_T, and pseudorapidity η, for all decay products
- Masses for all decay products

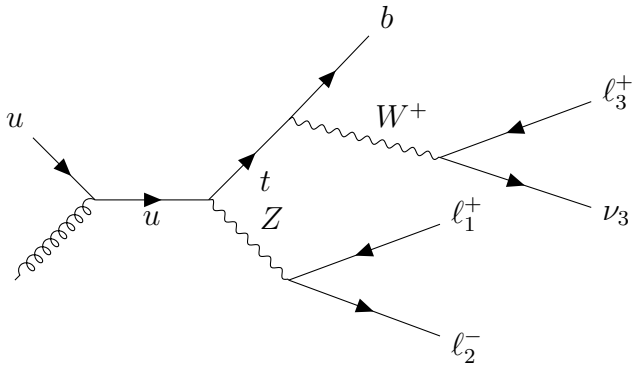


Figure 2.1: Feynman diagram of the FCNC signal process $gu \to tZ \to b\ell\ell\ell\nu$

As can be seen in Fig. 2.1, the decay products are a lepton pair produced by Z-boson decay in the signal, a lepton and a neutrino produced by W-boson decay in the signal, and a jet produced by the bottom quark. Instead of using the pseudorapidities, polar angles and momenta of each lepton in the pair resulting from Z-boson decay, their values can be used to calculate the pseudorapidity η_Z, polar angle ϕ_Z and the transverse momentum $p_{T,Z}$ of the Z boson:

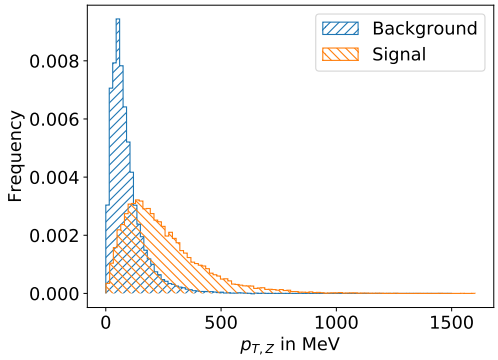
$$p_{T,Z} = \sqrt{p_{T,1}^2 + p_{T,2}^2 + 2p_{T,1}p_{T,2}\cos(\phi_1 - \phi_2)} \tag{2.1}$$

$$\phi_Z = \arctan\frac{p_{T,1}\sin\phi_1 + p_{T,2}\sin\phi_2}{p_{T,1}\cos\phi_1 + p_{T,2}\cos\phi_2} \tag{2.2}$$

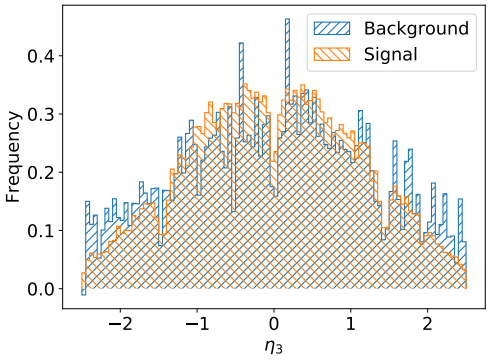
$$\eta_Z = \operatorname{arcsinh}\frac{p_{T,1}\sinh\eta_1 + p_{T,2}\sinh\eta_2}{p_{T,Z}} \tag{2.3}$$

Histograms of the variables in the dataset give an initial idea on which variables might be inconsequential to the training of a neural network, and con-

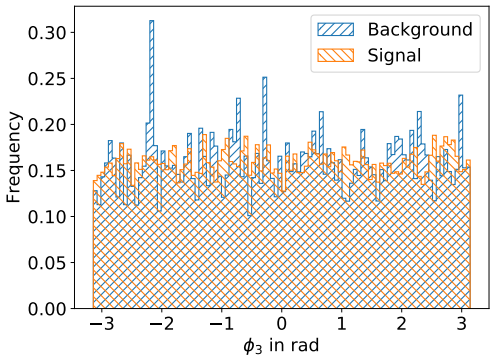
versely, which might have a big impact. When the background and the signal show a strong separation in a histogram, this is a good indicator that the variable should be included in the network inputs. However, missing separation does not mean that we can simply discard the variable, because a neural network might detect patterns that are not obvious by simply looking at the statistics of individual variables. Fig. 2.2 shows three examples with different levels of separation.



(a) Z boson transverse momentum, showing strong separation.



(b) ℓ_3 pseudorapidity, showing weak separation.



(c) ℓ_3 polar angle, not showing separation.

Figure 2.2: Histograms from the dataset showing varying levels of separation between signal and background

The amount of separation between the signal and background distributions can be described by the *Earth Mover's Distance* (EMD) metric. If U and V are

the cumulative distribution functions (CDFs) of distributions u and v, then the EMD of u and v is given by[16]

$$\mathrm{EMD}(u(x), v(x)) = \int_{-\infty}^{+\infty} |U(x) - V(x)| dx \tag{2.4}$$

This means that the EMD between a signal and background distribution can be computed using the empirical CDFs $C_S(x), C_B(x)$ of signal and background, with $x \in \{x_1, \ldots, x_N\}$ and calculating:

$$\mathrm{EMD}(S, B) = \sum_{i=1}^{N} |C_S(x_i) - C_B(x_i)| \tag{2.5}$$

In order to ensure comparability of the resulting EMDs associated with different variables of the dataset, the data first needs to be standardized. For each variable in the dataset, signal and background data are first combined to calculate the total standard deviation. Then, this standard deviation is used to standardize both the signal and the background by dividing all values with it. Finally, the EMD between standardized signal and background is calculated. The result can be seen in Fig. 2.3.

2.2 Network inputs

In addition to the analysis that was done in the previous section, the choice of network inputs will be aided by a different approach here: At first, a network is trained with *all* 20 variables as inputs. This network has a single hidden layer with 30 neurons. The inputs are standardized so that for each input, its mean in the training dataset becomes zero and its standard deviation becomes one. The standardization constants are saved to make the network usable on other datasets than the training dataset. The network is then trained for 100 epochs. After training, its weights are analyzed in order

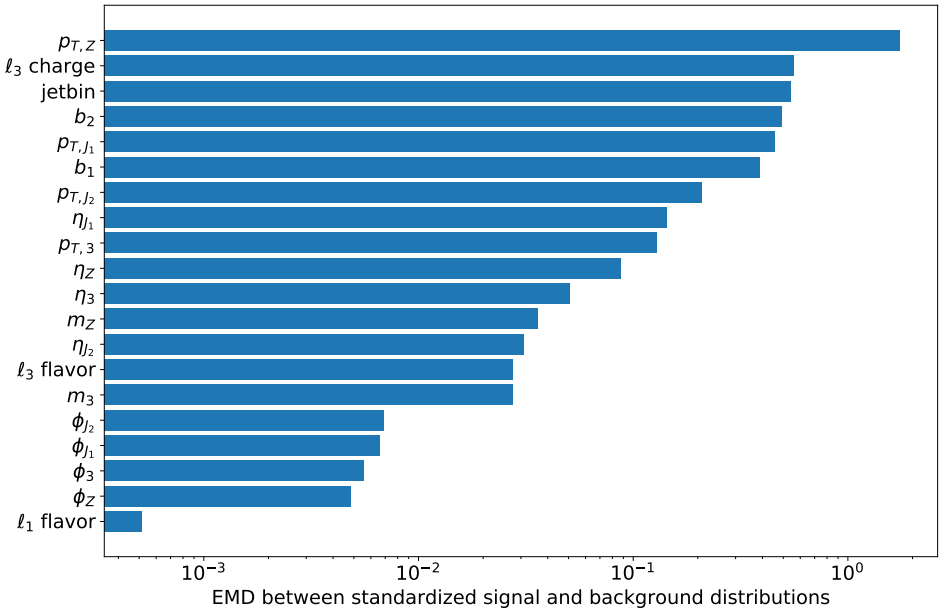


Figure 2.3: Earth Mover's Distance (EMD) between signal and background for each variable in the dataset. A greater value means stronger separation.

to gain an understanding of each input's contribution to the network's output. It can then be concluded that the inputs that contribute more are more important to the functioning of the network.

The weights are analyzed by taking the average of the absolute values of all weights between each input and the hidden layer:

$$\overline{w}_i = \sum_{i=1}^{N} |w_{ij}| \tag{2.6}$$

where $\overline{w}_i$ is the average of input neuron i's absolute weights, and w_{ij} is the weight on the connection between input neuron i and neuron j in the hidden layer. The results can be seen in Fig. 2.4.

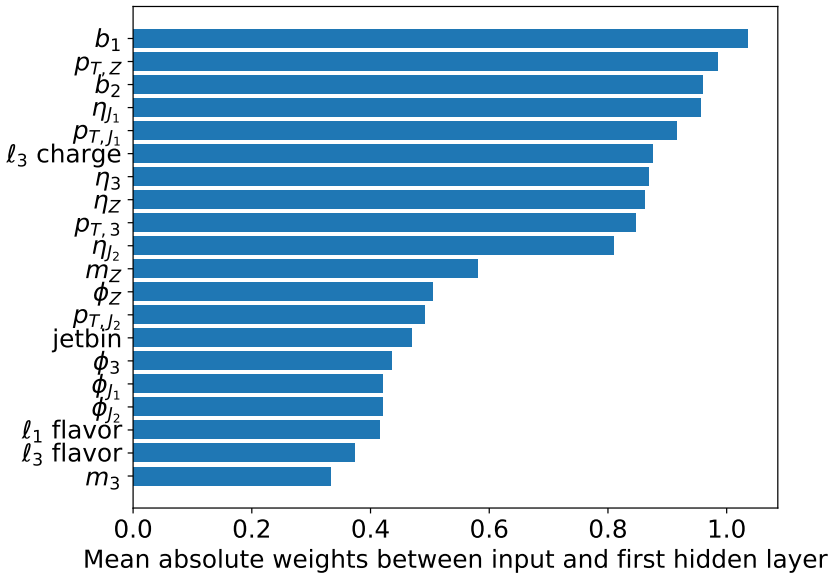


Figure 2.4: Average of the absolute values of the weights between each input variable and the hidden layer (30 neurons) after 100 epochs

This can be understood as a ranking of how important the input variables are. The next step is to determine how many of these inputs can be left out completely without affecting the performance of the network. This has been done by subsequently removing the input variable with the lowest ranking, and evaluating the network's performance on the validation dataset. For each input vector, the network has been trained 10 times for 50 epochs, with the division of the dataset into training and validation data being randomized each time. The mean and standard error were calculated for these measurements of the network's accuracy on the validation data. The results can be seen in Fig. 2.5.

It should be noted that, in this and all following network optimization sections, only the WZ and the ZZ background were used for both training and validation, because datasets for the less important backgrounds were not available at the time. Also, the data was trimmed so that the number of background events match the number of signal events, which tries to ensure that

the training does not get stuck in a local minimum by classifying all events as background. Since the network is trained a lot of times, this type of analysis is also very computationally expensive, so a smaller dataset is preferable to ensure sane execution times.

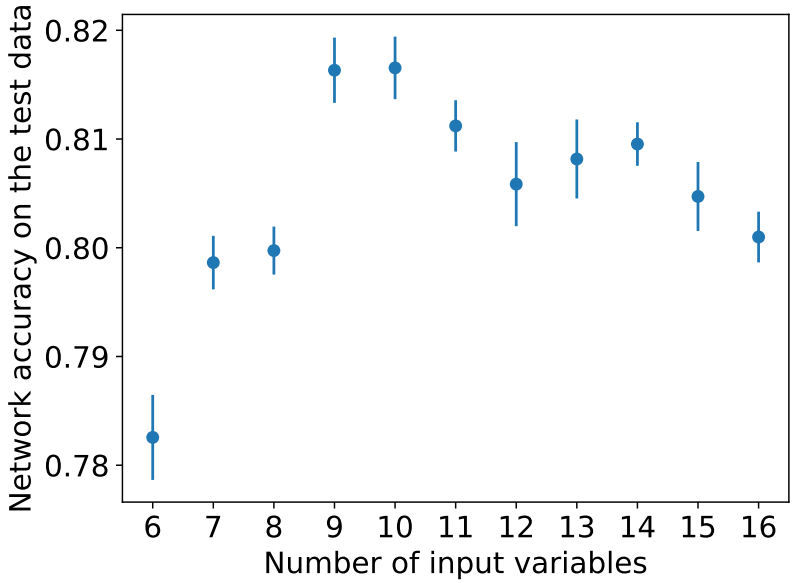


Figure 2.5: Network's performance after 50 epochs on the validation dataset, depending on the number of input neurons

The best accuracy is reached when taking the 9 highest-scoring variables from Fig. 2.4. Further inputs do not improve network performance, and consequently are not used.

2.3 Network structure

2.3.1 Number of hidden neurons

Using the optimal inputs determined in the previous section, a similar process was done to obtain the optimal number of neurons in one hidden layer.

Neurons were added one after another, and the network was again trained 10 times for 50 epochs in each step. The results are shown in Fig. 2.6 and display the selected optimal value of 29 neurons in the hidden layer.

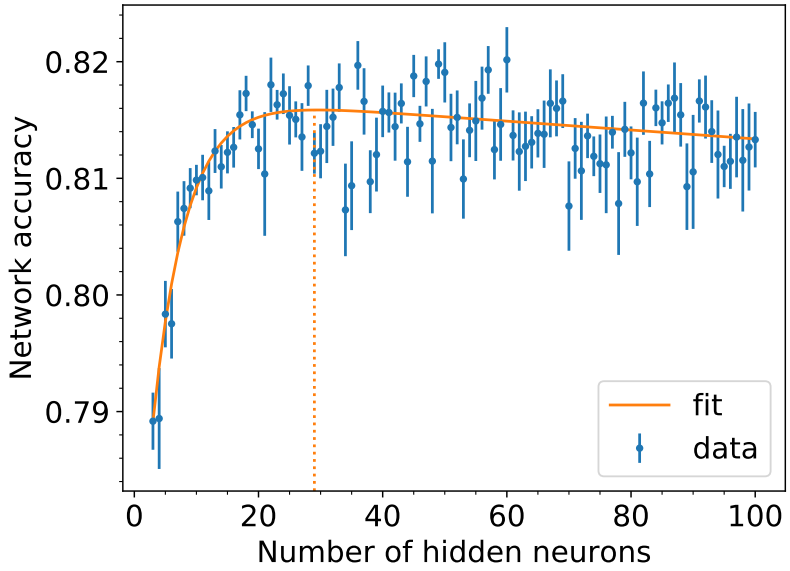


Figure 2.6: Network's performance on the validation dataset after 50 training epochs, depending on the number of neurons in its single hidden layer, with the selected optimal value of 29 neurons highlighted in orange

The value was selected by fitting with a custom fit function. There are two main effects that can be observed when adding neurons: At first, the accuracy of the network on the validation data increases dramatically with each added neuron. However, this increase in accuracy becomes less until the maximum possible accuracy on the validation dataset is reached. After that, further neurons *decrease* the performance of the network on the validation data, while the performance on the training data still increases. This is due to overfitting, where the network begins to learn small patterns in the training dataset that do not generalize well and lead to a worse performance on the validation data. The initial effect of performance increase appears proportional to the negative of an exponential decay. The effect of overfitting

Table 2.1: Results of the fit shown in Fig. 2.6, Eq. (2.7)

Fit parameter	value $\pm$ standard deviation
a	0.8172 ± 0.0008
b	$(3.8 \pm 1.3) \cdot 10^{-5}$
c	$(49 \pm 6) \cdot 10^{-3}$
d	0.186 ± 0.023

seems to be linear. Combining these two effects leads to the following expression, which was used for fitting:

$$A(N) = a - b \cdot N - c \cdot e^{-d \cdot N} \tag{2.7}$$

Here, A is the accuracy on the validation dataset, N is the number of hidden neurons, and a, b, c, d are fit parameters. a is mostly related to the maximum network accuracy, b is related to the severity of overfitting when adding neurons, c is related to how much of a benefit is gained by adding neurons in the beginning, and d is related to how long one can add neurons before the effect of overfitting starts to take over. The results of the fit can be seen in Tab. 2.1

The position of the maximum can be determined analytically from the fit parameters by setting the derivative of Eq. (2.7) to zero:

$$\begin{aligned} N_{max} &= -\frac{1}{d} \ln\left(\frac{b}{cd}\right) \\ &= 29 \pm 2 \end{aligned} \tag{2.8}$$

2.3.2 Number of hidden layers

It was also determined whether increasing the number of hidden layers benefited network accuracy. This was done by subsequently adding fully

connected layers with 29 neurons in each layer and repeating the process of training 10 times for 50 epochs, like in the previous sections. As can be seen in Fig. 2.7, additional hidden layers do not increase the network's performance. This problem does thus not benefit from a deep learning approach.

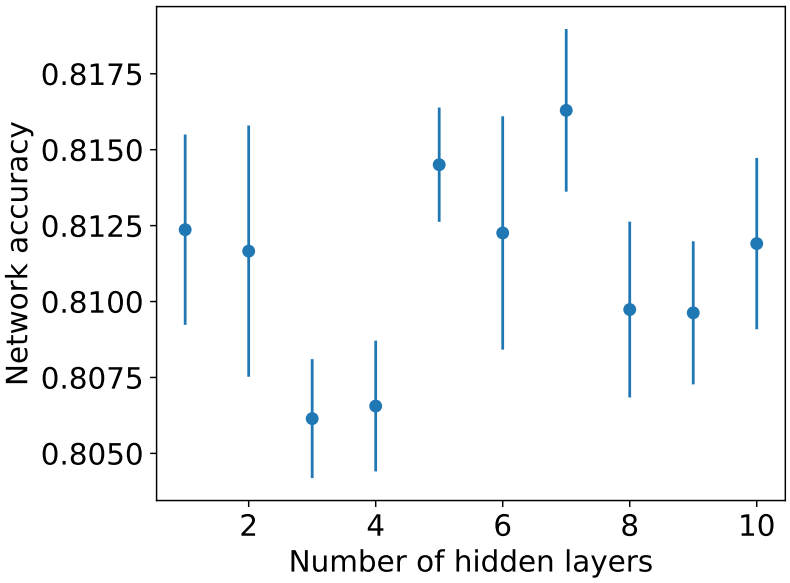


Figure 2.7: Network's performance on the validation dataset after 50 training epochs, depending on the number of hidden layers with 29 neurons each

2.3.3 Choice of activation functions

Finally, the optimal choice of activation functions in the hidden layers was determined. For the output layer, the sigmoid function was used as the activation, to ensure an output between zero and one. For the hidden layers, several activation functions have been tested. The best result was obtained with the *Softmax* function, as can be seen in Fig. 2.8.

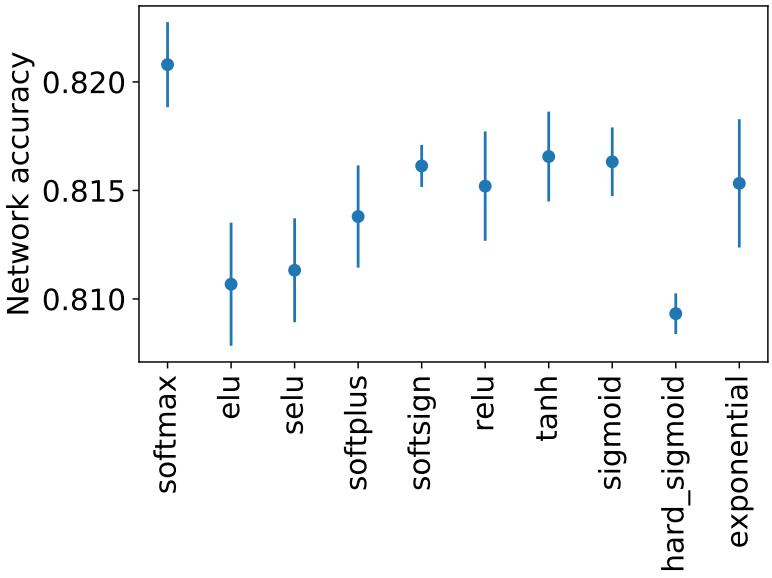


Figure 2.8: Network's performance on the validation dataset after 50 training epochs, for several different activation functions applied after the input- and the hidden layer

2.4 Training process

From the calculations done in the previous sections, the following network structure was chosen:

- **Input neurons:** 9
- **Hidden neurons:** 29
- **Output neurons:** 1
- **Activation function:** Softmax after hidden neurons, Sigmoid after output neuron

Now, this network is finally trained and validated using the entire dataset, including all background processes. The dataset was split into a training set and a validation set, in a 70:30 ratio. This finally results in a training dataset

with 153,017 entries and a validation dataset with 65,579 entries. In order to ensure that the network does not overfit on the background processes, the weights were adjusted so that the sum of all weights for background and signal were equal.

Binary cross-entropy was chosen as the loss function. The cross-entropy of two distributions $f(x)$ and $g(x)$ with $x \in X$ is given by:

$$L(f, g) = -\sum_{x \in X} f(x) \log g(x) \tag{2.9}$$

In our case of a classification network, $X = \{0, 1\}$, where background events are labeled with a 0, and signal events are labeled with a 1. For a specific event in the dataset with the label $e \in X$, $f_e(x)$ then simply becomes one for $x = e$ and zero for $x \neq e$. $g_e(x)$ is the network's prediction of e being x. The network's output o is interpreted as a likelihood of the event in question being a signal event. Consequently, $1 - o$ is the likelihood of the event being a background event. This leads to the following expressions for $f_e(x)$ and $g_e(x)$:

$$f_e(x) = \begin{cases} 1 & x = e \\ 0 & x \neq e \end{cases} = \begin{cases} e & x = 1 \\ 1 - e & x = 0 \end{cases} \tag{2.10}$$

$$g_e(x) = \begin{cases} o & x = 1 \\ 1 - o & x = 0 \end{cases} \tag{2.11}$$

Finally, the expression for binary cross-entropy is obtained:

$$L(e, o) = -(e \cdot \log o + (1 - e) \cdot \log(1 - o)) \tag{2.12}$$

It is important to note that the loss function was weighted according to the weights in the dataset stemming from the Monte Carlo simulation, which takes the cross sections of the individual background process into account.

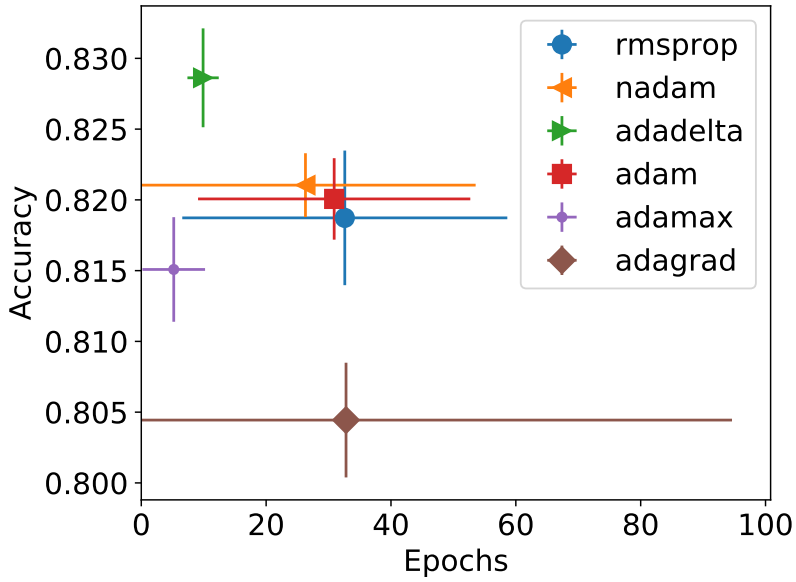


Figure 2.9: Maximum accuracy on the validation data and number of epochs required to obtain it for different optimizers, when the training is stopped after 50 epochs without improvement of the validation accuracy. Displayed are the means and standard deviations of 10 training runs

One epoch describes a period of time over which the network has seen all entries in the training dataset. After an epoch is complete, the weighted average over all the losses in that epoch is calculated to determine the epoch's loss. The validation loss is also weighted, however, it is calculated only after the epoch is finished. This explains why, in Fig. 2.12b, the validation loss is initially lower than the training loss, when it is expected to be greater than or equal to the training loss because of overfitting.

Besides the loss function, there is another important metric to assess the network's performance, which is the *accuracy*. It is calculated by labeling all outputs greater than 0.5 as signal events, and all less than or equal to 0.5 as background. Then, the percentage of correct classifications is determined. The loss function is important for training, while the accuracy is more im-

portant to assess the network's performance, since the accuracy is calculated in the way the network is intended to be used after training. Accuracy and binary cross-entropy are not closely related, because cross-entropy can provide any real number between zero and one for a single event, while the accuracy of a single event is either exactly zero or exactly one. While the network is trained, its accuracy on the validation dataset is monitored. When this accuracy has not improved for 50 epochs, the training is stopped, and the network weights that resulted in the best validation accuracy are restored. The accuracy was chosen as the metric for this early stopping, instead of the loss, because of the aforementioned reasons.

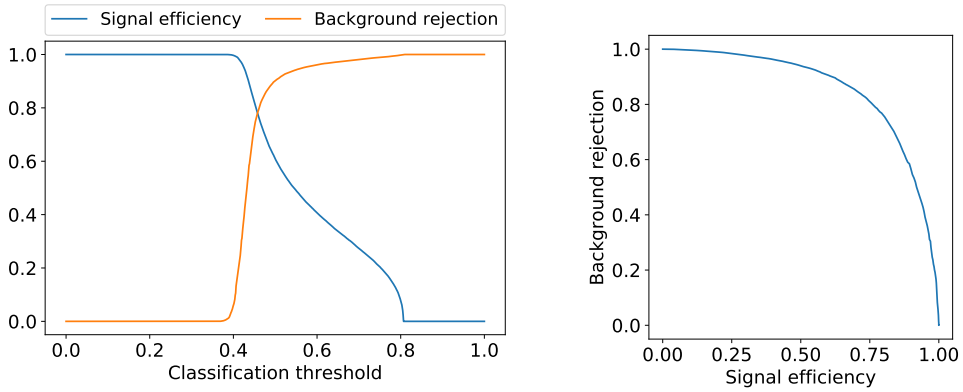
The optimizer was chosen by training the network like this with different optimizers for 10 times each, observing the maximum accuracy and the number of epochs until it was reached. Then, the mean and standard deviation of this maximum accuracy and the number of epochs were calculated, and they were plotted against each other, as can be seen in Fig. 2.9. Obviously, a higher accuracy and a smaller number of epochs required to obtain it are desirable. The optimizers tested were rmsprop[12], AdaGrad[7], ADADELTA[17], Adam[14], AdaMax[13] and Nadam[6]. It can be seen that ADADELTA provides the best results, so it was chosen as the optimizer.

2.5 Results

The network was trained using the structure and the training process described in the previous section. The maximum accuracy a on the validation data and the number of epochs n_{opt} that were needed to obtain it are:

$$a = 83.81\%$$
$$n_{opt} = 8$$

The network's performance can be assessed better than simply using the accuracy, using the *Receiver Operator Characteristic* (ROC) curve. To obtain it, the classification threshold is varied from zero to one. The classification



(a) Plotted against the classification threshold

(b) Plotted against each other (ROC)

Figure 2.10: Signal efficiency and background rejection

threshold is the threshold above which the network's output is intrepreted as a signal classification. While varying this threshold, the *background rejection* and *signal efficiency* can be recorded, as seen in Fig. 2.10a. The background rejection is the weighted fraction of background events that were correctly classified. Its ideal value is one, and as it decreases, the number of false positives increases. The signal efficiency is the weighted fraction of signal events that were correctly classified. As it decreases, the number of false negatives increases. When these two values are plotted against each other, the ROC is obtained, shown in Fig. 2.10b.

It is evident that the network has learned to classify the events. However, in Fig. 2.10a, it can be observed that the network does not use the full classification range from zero to one, but stays mostly within outputs between 0.4 and 0.8. This is a result of the few number of training epochs until the optimal accuracy was reached. As seen in Fig. 2.12, the accuracy is at its maximum when the loss is still relatively high. Continuing training decreases the loss, but also decreases the accuracy, which is the more important metric to evaluate the performance of the network. A test run with 2,000 training epochs confirmed that the accuracy will not start increasing again after such an amount of training time, so the choice of stopping training after just eight epochs is reasonable.

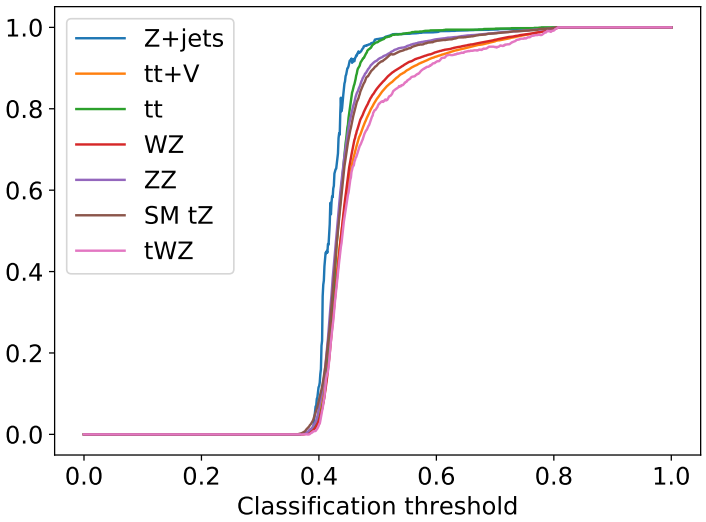
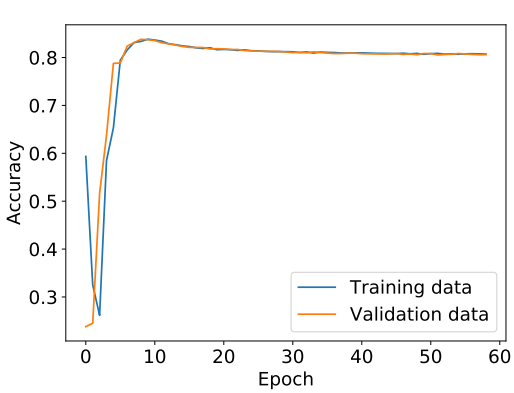


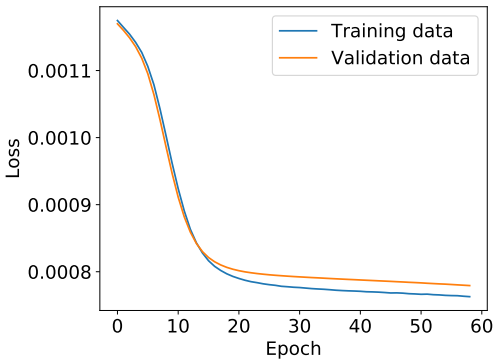
Figure 2.11: Background rejections for the different background processes in the validation dataset

It is also possible to record the background rejections for the individual background processes. As can be seen in Fig. 2.11, the network is best at rejecting Z+jets events and worst at rejecting tWZ-production events.

The loss and accuracy of training and validation datasets plotted over epochs can be seen in Fig. 2.12.



(a) Accuracy plotted over epochs



(b) Binary cross-entropy plotted over epochs

Figure 2.12: Training progress

3 Evaluation of experimental data

3.1 Dataset

The experimental data was obtained from ATLAS measurements during the years 2015–2018, at an invariant mass of $\sqrt{s} = 13\,\mathrm{TeV}$ and an integrated luminosity of $\mathcal{L}_{int} = 140\,\mathrm{fb}^{-1}$. It contains a total of 746 preselected detector events. The preselection criteria are:

- There are exactly three charged leptons (electrons or muons) with transverse momenta of at least $15\,\mathrm{GeV}$ and pseudorapidities $|\eta| < 2.5$
- One of the leptons has a transverse momentum of at least $27\,\mathrm{GeV}$ and has triggered the ATLAS detector
- Another one of the leptons has a transverse momentum of at least $25\,\mathrm{GeV}$
- There are two candidates for decay products of the Z boson, meaning two charged leptons of the same flavor with opposite charges. The invariant mass of this lepton pair is $81.2\,\mathrm{GeV} < \sqrt{s} < 101.2\,\mathrm{GeV}$. If there are multiple candidates, the pair with the invariant mass closest to $91.2\,\mathrm{GeV}$ is labeled as coming from Z decay.
- There are one or two jets with transverse momenta of at least $30\,\mathrm{GeV}$ and a pseudorapidity $|\eta| < 4.5$.
- One jet has a b-tagging score of at least 4 (corresponding to 70% of b-jets being tagged with such a score)

The weights in the simulated data, which has been introduced in Section 2.1, take into account the cross section of the various background processes and the integrated luminosity of the experiment. For the signal, an arbitrary

cross section of $1\,\mathrm{pb}$ was chosen in the simulation. This means that the number of detected events can be directly compared between the experimental dataset and the full simulated dataset, without needing to adjust weights to match the integrated luminosity. As seen in Fig. 2.12a, the accuracies obtained for the validation and the training dataset are not different, so it is possible to merge them back together and evaluate the network's response on the entire dataset. The sum of the background weights in the simulated data is 733.45, which is slightly less than the counts obtained in the experiment. In the following sections, the network's response will be used to cut both datasets further, in a way that maximizes signal significance, and the counts will be compared.

3.2 Optimization of signal and background statistics

In order to achieve a cut with high signal significance, the classification threshold of the network is optimized as to maximize a *Figure of Merit* (FOM). A commonly used FOM for discoveries is

$$\mathrm{FOM} = \frac{S}{\sqrt{B}} \tag{3.1}$$

where S is the number of signal events and B is the number of background events. While varying the classification threshold, the sum of the weights of background and signal events that were classified as signal events is kept track of. These are the B and S of Eq. 3.1. Then, the classification threshold that maximizes this FOM is chosen. Fig. 3.1 shows this relation of the FOM to the classification threshold.

The classification threshold t_{max} that maximizes this FOM, the sum of the wrongly classified background events' weights B, and that of the correctly classified signal events' weights S, when using this threshold t_{max} for classification, are:

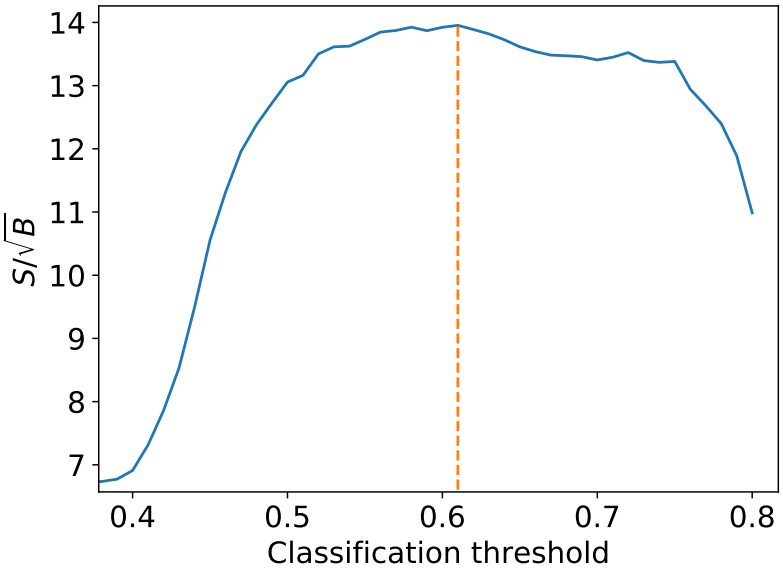


Figure 3.1: The FOM $S/\sqrt{B}$ while varying the classification threshold.

$$
\begin{aligned}
t_{max} &= 0.61 \\
B &= 26.14 \\
S &= 71.34 \\
\frac{S}{\sqrt{B}} &= 13.95
\end{aligned}
$$

When applying the same cut to the experimental data, B is the expected number of background events. The S obtained here assumes the cross section that was arbitrarily chosen in the simulation as $1\,\mathrm{pb}$, and thus does not correspond to an expected number of signal events. The resulting signal efficiency ϵ_S and background rejection ρ_B are

$$
\begin{aligned}
\epsilon_S &= 39.2\% \\
\rho_B &= 96.4\%
\end{aligned}
$$

3.3 Calculating the cross section

Using the optimal classification threshold $t_{max} = 0.61$ determined in the previous section, the number of events N in the experimental data that were classified as signal events is:

$$N = 20$$

Since this is less than the number of expected background events $B = 26.14$, no discovery was made and the experimental data supports the background-only hypothesis. In this section, an upper limit on the signal's cross section is calculated. The counts follow Poissonian statistics, so the likelihood of observing a number of counts N, with the expected number of counts being λ, is:

$$P_\lambda(N) = e^{-\lambda}\frac{\lambda^N}{N!} \tag{3.2}$$

For the 95% confidence upper limit on the number of expected counts $\lambda_{95\%}$, given the measured counts N, the combined likelihood of counting less than or equal to N events needs to be:

$$P_{\lambda_{95\%}}(n \leq N) = \sum_{n=0}^{N} P_{\lambda_{95\%}}(N) = 0.05 \tag{3.3}$$

The value for $\lambda_{95\%}$, given $N = 20$, was calculated numerically as:

$$\lambda_{95\%} = 29.06$$

Finally, using the arbitrary signal cross section $\sigma_S = 1\,\mathrm{pb}$ from the simulation, the number of expected signal events S given that cross section and

the number of expected background events B, the upper limit on the signal's cross section $\sigma_{95\%}$ can be calculated:

$$\sigma_{95\%} = \frac{\lambda_{95\%} - B}{S}\sigma_S = 40.93\,\mathrm{fb} \tag{3.4}$$

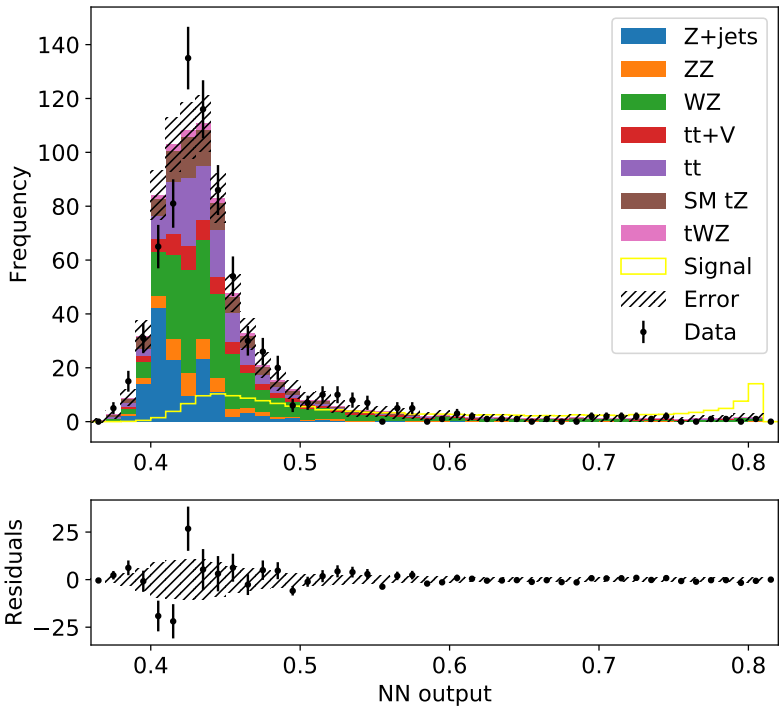


Figure 3.2: Histograms of the network response on the various simulated backgrounds, the simulated signal and the experimental data

Fig. 3.2 shows the histograms of the network's responses on the simulated background, on the simulated signal with an arbitrarily chosen cross section of $1\,\mathrm{pb}$ and on the experimental data. It can be seen that the data supports the background-only hypothesis.

4 Conclusion

A neural network has been trained to identify tZ-production events via FCNC. It has then been used to identify such events in experimental data, and an upper limit on the cross section has been calculated. The results are shown in Tab. 4.1

Table 4.1: Summarized results of this research

Quantity		Value
Signal efficiency	ϵ_S	39.2%
Background rejection	ρ_B	96.4%
Number of training epochs	n_{epochs}	8
95% UL on signal cross section	$\sigma_{95\%}$	$40.93\,\mathrm{fb}$

The signal efficiency that maximizes the network's signal significance is quite low. Also, the number of training epochs until optimal accuracy has been achieved is low, because while the loss of binary cross-entropy would continue decreasing with further training, the accuracy would decrease, too. Usage of a loss function that directly takes into account signal significance could lead to better results, as one could optimize purely using the loss. Such a loss function has already been proposed by previous research. [8] Optimization using the accuracy generally makes sense for classification networks, but may lead to problems here since the network is clearly unable to identify a large portion of signal events that seem to be indistinguishable to the background. This means that the neural network could be learning to give an output closer to zero for background events, which may in turn also give some signal events lower scores, decreasing network accuracy while also decreasing the binary cross-entropy loss.

Such improvements might improve the obtained upper limit on the cross

section. However, the biggest limiting factor is the amount of available data. More data would significantly improve the upper limits on the cross section. The amount of available data is mostly limited on current experimental constraints, for example the luminosity of the LHC, which is already planned to be upgraded with the High-Luminosity LHC project, with plans of becoming operational by 2026. [1]

Bibliography

[1] High-Luminosity LHC | CERN, Aug 2019. URL https://home.cern/science/accelerators/high-luminosity-lhc. [Online; accessed 21. Aug. 2019].

[2] LHCTopWG Summary Plots, Aug 2019. URL https://twiki.cern.ch/twiki/bin/view/LHCPhysics/LHCTopWGSummaryPlots. [Online; accessed 19. Aug. 2019].

[3] CERN. The Large Hadron Collider. URL https://home.cern/science/accelerators/large-hadron-collider. [Online; accessed 11. Aug. 2019].

[4] ATLAS Collaboration. Search for FCNC single top-quark production at s=7 TeV with the ATLAS detector. *Phys. Lett. B*, 712(4):351–369, Jun 2012. ISSN 0370-2693. doi: 10.1016/j.physletb.2012.05.022.

[5] D Collaboration. Search for flavor changing neutral currents via quark–gluon couplings in single top quark production using 2.3 fb-1 of $p\bar{p}$ collisions. *Phys. Lett. B*, 693(2):81–87, Sep 2010. ISSN 0370-2693. doi: 10.1016/j.physletb.2010.08.011.

[6] Timothy Dozat. Incorporating Nesterov Momentum into Adam, Feb 2016. URL https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ. [Online; accessed 11. Aug. 2019].

[7] John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011. ISSN 1533-7928. URL http://www.jmlr.org/papers/v12/duchi11a.html.

[8] Adam Elwood and Dirk Krücker. Direct optimisation of the discovery significance when training neural networks to search for new physics in particle colliders. *arXiv,* Jun 2018. URL https://arxiv.org/abs/1806.00322.

[9] Tanabashi, M. et. al (Particle Data Group). Review of Particle Physics. *Phys. Rev. D*, 98(3):030001, Aug 2018. ISSN 2470-0029. doi: 10.1103/PhysRevD.98.030001.

[10] S. L. Glashow, J. Iliopoulos, and L. Maiani. Weak Interactions with Lepton-Hadron Symmetry. *Phys. Rev. D*, 2(7):1285–1292, Oct 1970. ISSN 2470-0029. doi: 10.1103/PhysRevD.2.1285.

[11] Sascha Gleiß. Studien über die Sensitivität bei der assoziierten tZ-FCNC-Produktion. Jan 2017.

[12] Geoffrey Hinton. Neural Networks for Machine Learning. URL http://www.cs.toronto.edu/tijmen/csc321/slides/lecture_slides_lec6.pdf. [Online; accessed 11. Aug. 2019].

[13] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks, 2016. URL http://papers.nips.cc/paper/6573-binarized-neural-networks. [Online; accessed 11. Aug. 2019].

[14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv*, Dec 2014. URL https://arxiv.org/abs/1412.6980.

[15] Hermann Kolanoski. Application of Artificial Neural Networks in Particle Physics. *SpringerLink*, pages 1–14, Jul 1996. doi: 10.1007/3-540-61510-5_1.

[16] S. S. Vallender. Calculation of the Wasserstein Distance Between Probability Distributions on the Line. *Theory of Probability & Its Applications*, Jul 2006. URL https://epubs.siam.org/doi/abs/10.1137/1118101.

[17] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv,* Dec 2012. URL https://arxiv.org/abs/1212.5701.

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin,