ETP-Bachelor-KA/2022-01

Karlsruhe Institute of Technology

Bayesian Optimization of Graph Neural Networks with Hypergraph Inputs in $t\bar{t} + b\bar{b}$ Events at the CMS Experiment

Bachelor Thesis

Clemens Wolter

At the Department of Physics
Institute of Experimental Particle Physics

Reviewer:	Prof. Dr. Ulrich Husemann
Second reviewer:	Dr. Michael Waßmer
Advisor:	Emanuel Pfeffer

Karlsruhe, 16. February 2022

This thesis has been accepted by the first reviewer of the bachelor thesis.

PLACE, DATE

.....

(**Prof. Dr. Ulrich Husemann**)

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

PLACE, DATE

.....

(**Clemens Wolter**)

Contents

1 Introduction

Understanding the intricate fabrics of our reality has always been the underlying goal, of not just physics, but all science. With the introduction of the Standard Model (SM), particle physics has come closer than ever before to achieving this goal. This theory allows to describe many elementary particles and three of the four known fundamental forces of the universe. The nature of physics is to always test such bold claims against experimental evidence, which has been done numerous times, with the most famous event being the prediction and discovery of the Higgs boson at CERN Large Hadron Collider [1] [2]. Testing our hypothesis remains of utmost importance as many phenomena still remain unexplained and the Standard Model still does not account for gravity, therefore we always look for ways to improve data analysis.

With computing power becoming cheaper each year, more costly methods of data analysis become viable. A semi recent development in data science is the adaption of machine learning methods. These algorithms can and have been used to solve complex tasks. Therefore, applying them in physics is just the logical next step. The applications of more computing power do not just stop there, because with enough computing power it becomes feasible to not only optimize the parameters of machine learning methods but also retrain these machines and optimize hyperparameters.

This thesis focuses on studying machine learning methods to analyze simulation data of $t\bar{t} + b\bar{b}$ events, where the two heaviest quarks, top and bottom are produced in a proton-proton collision. These $t\bar{t} + b\bar{b}$ events are of importance as firstly they make up a huge irreducible background in measurements of $t\bar{t}H$ production with $H \to b\bar{b}$ decays. Secondly, the $t\bar{t} + b\bar{b}$ process itself is particularly interesting from a theoretical and experimental perspective, because two QCD processes occur at different energy scales [3].

Previously Deep Deep Neural Networks (DNNs) [3] and Graph Neural Networks (GNNs) [4] have already been used to identify additional b jets in the $t\bar{t} + b\bar{b}$ process. The core quantity for performance measurements in these events is the 2/2 classifications rate, which measures the percentage of events where both additional b jets are classified correctly. Apart from finding ways to improve the 2/2 classification rate, the study aims at building upon the previous work and improving the machine learning methods and algorithms used within the mentioned studies.

2 Theoretical foundations

2.1 The Standard Model

The Standard Model of elementary particle physics describes all known elementary particles and includes the unified theory of the electroweak interaction and quantum chromodynamics. The only interaction not described with the Standard Model is gravity. Figure 2.1 illustrates many aspects of the Standard Model. Much of the first section is based on [5].

The elementary particles with a spin of $\frac{1}{2}$ are called fermions. Fermions make up the fundamental building blocks of all matter. They are split into two groups, the quarks and the leptons. These groups comprise six particles each that are split into three generations. The first generation contains the lightest and therefore most stable particles. The other generations contain heavier particles that are much less stable. For every fermion an antiparticle exists. This is a particle with the same mass but opposite charge.

The first generation of quarks consits of the up and down quarks, the second includes the charm and strange quarks and the third and final generation contains the top and bottom quark. Every quark is electrically charged with multiples of $\frac{1}{3}e$ and also carries a color charge. The QCD confinement implies that quarks cannot exist on their own, as only hadrons with neutral color charge are allowed. In a process called hadronisation, quarks and gluons form a colorless hadron. If a hadron collides with a high energy particle the quark can be separated from the gluon. In order to obey QCD confinement the particles create other colored objects around them, these however also have obey QCD confinement, creating even more colored objects etc. The created particles all tend to travel into the same direction and thus form a narrow bundle called jet. Including the antiparticles a total number of 12 quarks are known.

The charged leptons include, ordered by generation, the electron, the muon and the tauon. These three have a negative electric charge and rise in mass with their generation. Every aforementioned lepton also has a neutrino with no electric charge and a mass for which only the upper limit is known. Again including antiparticles yields a total number of 12 leptons.

Elementary particles for which the spin is an integer are bosons and convey all known interactions except for gravity.

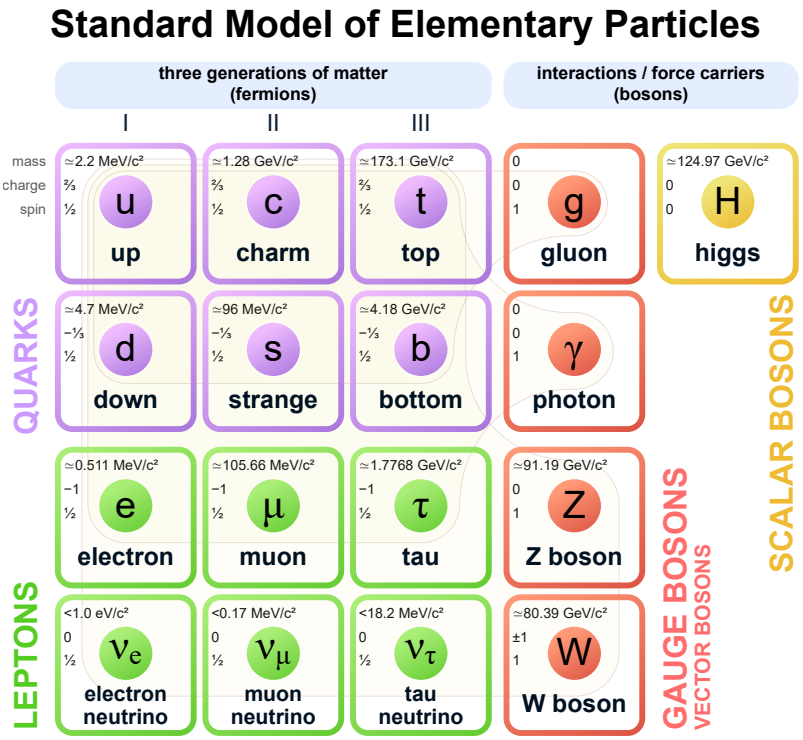

Figure 2.1: The Standard Model of elementary particle physics. The diagram shows all elementary particles. The fermions are on the left. The possible interactions for each fermion are illustrated through a connection in the background with the exchange boson on the right. The image is taken from [6].

2.2 LHC and CMS

The Large Hadron Collider (LHC) with a circumference of 27 km is the most powerful particle accelerator in the world. As illustrated in Figure 2.2 the LHC is the last accelerator in a chain of particle accelerators at CERN.

The Compact Muon Solenoid (CMS) experiment is a multi purpose detector located at the LHC and therefore needs to incorporate many different detection mechanisms as illustrated in Figure 2.3.

As the study is targeted to later on be used with real data, only data that is accessible from the CMS experiment can be used. The first quantities, the direction of the charged particles as well as the transverse momentum p_t are measured by the innermost layer, the silicon tracker. The second quantity, the energy E, is measured within the next two layers of the CMS detector. The first one, the electromagnetic calorimeter, is used to measure the energy of electrons, positrons and photons. The second one, the hadron calorimeter, is used to measure the energy of hadrons. Other quantites are also measured by combining the information of multiple layers, e.g. the mass of the particle M. Additionally, the jet charge q_J can also be calculated.

The coordinate system of the CMS detector has its origin in the center of the detector, the collision point of the particles. The x-axis points radially into the center of the LHC and the y-axis is perpendicular to it and points upward. The x-y-plane is also referred to as the transverse plane. The z-axis is perpendicular to the transverse plane and points counterclockwise along the beam direction. Cylindrical coordinates are used for the description of the particles, as CMS itself is built in a cylindrical shape. Hereby, ϕ denots

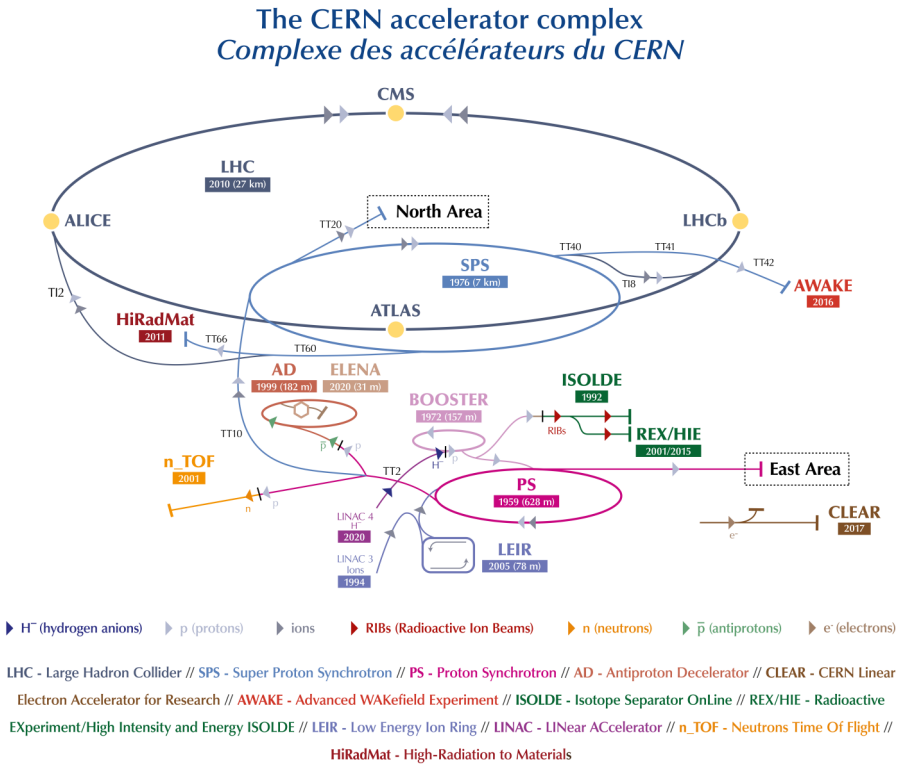

Figure 2.2: A representation of the different accelerators used in CERN. The LHC, which is the final accelerator in a chain of accelerators, is represented as the largest ring. The CMS, as well as other LHC experiments are shown as yellow dots. The picture is taken form [7].

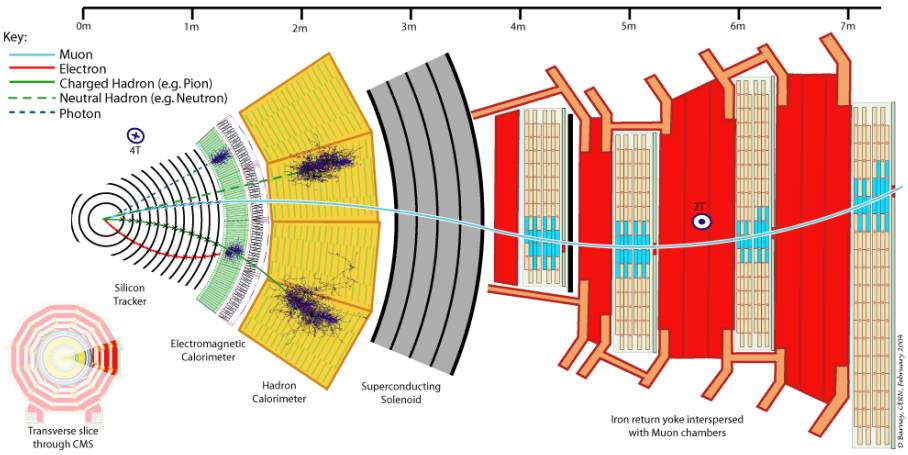

Figure 2.3: A transverse slice of the CMS experiment, taken from [8]. Particle collisions occur on the left side of the image. The particles then travel in the transverse different through the different detection layers

the azimuthal angle in the transverse plane and is measured from the x-axis. The polar angle θ is measured from the z-axis. The distance r is measured from the collision point.

Another important quantity is the pseudo rapidity η, as calculated in Equation 2.1. Small values of η describe a direction of flight perpendicular to the z-axis, whereas large values describe a flight in the direction of the z-axis

$$\eta = -\log \tan\left(\frac{\theta}{2}\right). \tag{2.1}$$

The spatial distance in the η,ϕ-plane of two particles i and j is denoted by ΔR

$$\Delta R_{i,j} = \sqrt{(\eta_i - \eta_j)^2 + (\phi_i - \phi_j)^2}. \tag{2.2}$$

None of the detectors in the CMS experiment can measure neutrinos directly. However, the transverse momentum of the particles before the collision is sufficiently small to be disregarded, thus p_T and E_T are assumed to be 0 before the collision. As both quantities are conserved they are 0 after the collision as well. Consequently, any difference to 0 in the total transverse momentum and energy means that some of the momenta and energies are not measured. Therefore, the missing transverse energy (MET) is recorded.

The b-tag value of any recorded jet is a calculated quantity using the DeepJet algorithm [9]. The value describes the pseudo probability of any jet to be a b jet.

The invariant mass combination of any two highly relativistic particles ($E \gg M$) i and j can be calculated using the formula in Equation 2.3

$$M_{i,j} = 2p_{\mathrm{T},i}p_{\mathrm{T},j}\left(\cosh\left(\eta_i - \eta_j\right) - \cos\left(\phi_i - \phi_j\right)\right). \tag{2.3}$$

2.3 $\mathrm{t\bar{t}} + \mathrm{b\bar{b}}$ process

The core of the study, the $\mathrm{t\bar{t}} + \mathrm{b\bar{b}}$ process, is part of a proton-proton collision. Two of the gluons then decay into a top quark and a top antiquark. Subsequently, the top quarks react into a W-boson which can then decay either leptonically into an electron and electron neutrino or hadronically into a quark and an antiquark. Additionally, this decay can radiate a gluon that then decays into a bottom quark and a bottom antiquark. These two "additional b quarks" are the focus of the study, as the goal is to identify the associated "additional b jets". The description only summarizes the leading order of the $\mathrm{t\bar{t}} + \mathrm{b\bar{b}}$ process, as corrections of higher order also allow for decay into multiple particles. This is summarized within the leading order Feynman diagram of the $\mathrm{t\bar{t}} + \mathrm{b\bar{b}}$ process in Figure 2.4.

2.4 Data Basis

As mentioned before, this study focuses on working with simulated data of $\mathrm{t\bar{t}} + \mathrm{b\bar{b}}$ events. The simulation of the data is split into three different levels as illustrated in Figure 2.5. The first level is the particle level where individual particles are simulated. The second level, the generator level simulates the actual proton-proton collisions and contains every quantity about the individual particles. The last level, the reconstruction level models what data would actually be measured within the detector and therefore includes unique detector effects. Thus, the information as to which measured jet is an additional b jet is lost in this level of the simulation.

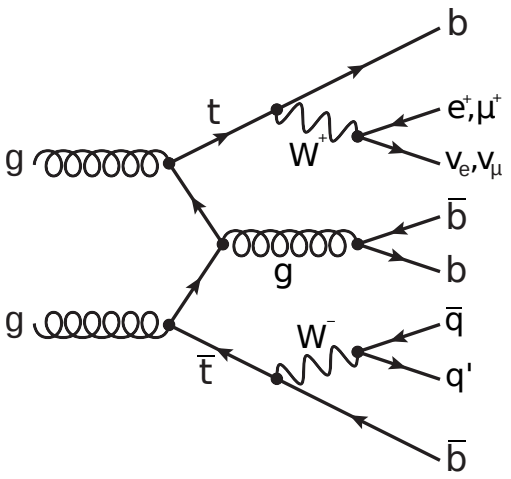

Figure 2.4: An example of a leading order Feynman diagram of the $t\bar{t} + b\bar{b}$ process, taken from [3].

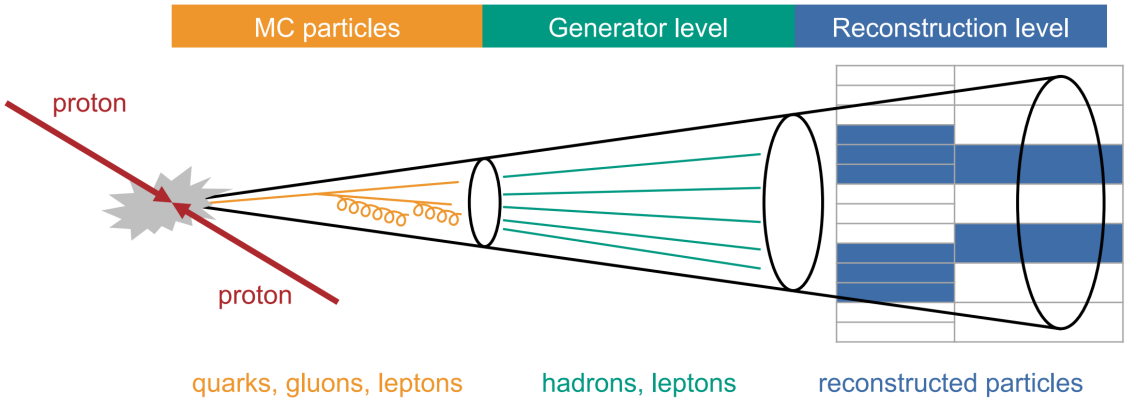

Figure 2.5: An illustration of the simulation levels of a jet. The middle shows the generator level, the reconstruction level is shown on the right side of the figure. Exemplary physics objects are listed at the bottom. Taken from [3]

However, the goal of the study is to classify additional b jets from reconstruction level data. To measure the performance of the classifier the ground truth about the jets must be accessed. Therefore, [3] suggests a matching algorithm to reassign the jet labels from generator level to the reconstruction level. The basic idea behind the matching algorithm is that the ΔR value between the same jet at generator level and reconstruction level tends to be close to 0.

From the whole simulation data only events with two additional b jets at generator level are used. The full set of simulation data comprises 30 007 $t\bar{t} + b\bar{b}$ events. Of those, for 28 425 events both additional b jets can be reassigned at reconstruction level.

3 Graph Neural Networks

3.1 Neural Networks

Artificial Neural Networks (NNs) have existed since the 20th century and have been formally defined many times. Thus, the article [10] is referred to for a formal definition.

3.1.1 Feedforward Calculation

Typically, Neural Networks are viewed as in Figure 3.1, with the main components: input layer, hidden layer and output layer. For this application of NNs it is sufficient to view the input as a vector of real numbers $\mathbf{x} \in \mathbb{R}^{l^0}$ with the length $l^0 \in \mathbb{N}$. Specifically, in the case of $\mathrm{t\bar{t}} + \mathrm{b\bar{b}}$ inputs this vector contains the information about each jet, i.e. all the kinematic information.

The hidden layer contains $L \in \mathbb{N}$ hidden states. The k-th hidden state $\mathbf{h}^k$, is similar to the input vector, a vector of real numbers of the length l^k: $\mathbf{h}^k \in \mathbb{R}^{l^k}$. The values of $\mathbf{h}^k$ are calculated within the feedforward calculation. They depend on four things:

- The previous hidden state: $\mathbf{h}^{k-1}$.
- The weight matrix of the state: $\theta^k \in \mathbb{R}^{l^{k-1} \times l^k}$. This matrix is multiplied with the previous hidden state $\mathbf{h}^{k-1}$, and therefore determines how much each feature of $\mathbf{h}^{k-1}$ is factored into the new hidden state $\mathbf{h}^k$. By choosing specific values for the weights different functions can be modeled. Tuning the weights to accomplish a task, is the underlying goal of optimizing a network. Algorithms for the weight tuning are called learning algorithms, accordingly the weights are called learnable parameters.
- The bias of the hidden layer: $b^k \in \mathbb{R}$. After the multiplication with the weight matrix a bias, which is the same for the entire layer, can be added to the intermediate result. The additional bias is also needed to model specific functions and thus is also called a learnable parameter.
- The differentiable, nonlinear activation function: $\sigma^k()$. The last step of the calculation is to apply $\sigma^k()$ to the result. This nonlinear function is necessary, otherwise the calculation of all hidden layers collapses into a single layer. This is proven in section A.1.

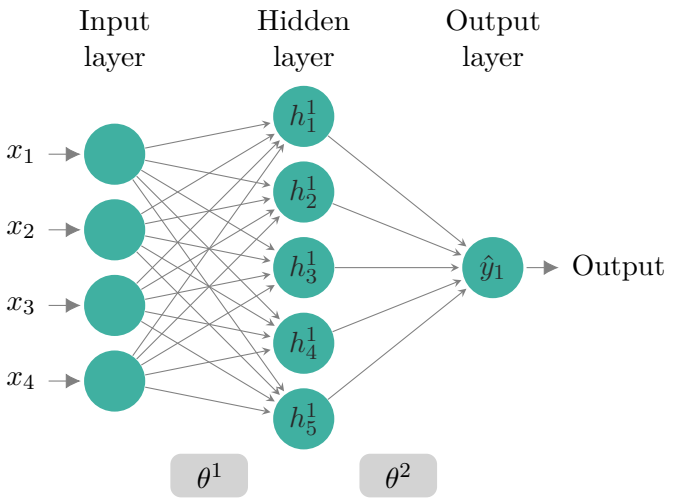

Figure 3.1: Sketch of a Neural Network with $L = 1$ hidden layer and a single node in the output layer. The input vector $\mathbf{x} = \mathbf{h^0}$ has $l^0 = 4$ features. The number of features in the hidden state $\mathbf{h^1}$ is $l^1 = 5$, thus the weight matrix $\theta^\mathbf{1}$ must be an element of $\mathbb{R}^{4\times5}$. The entries of the weight matrices are represented by the small arrows linking the nodes. For example the weight $\theta^1_{1,1}$ connects the node x_1 to the node h^1_1. The feature vector of the output layer $\hat{\mathbf{y}} = \mathbf{h^{L+1}} = \mathbf{h^2}$, has a length of $l^2 = 1$. This explains the dimension of the second weight matrix $\theta^\mathbf{2} \in \mathbb{R}^{5\times1}$.

Lastly, the output layer is similar to any hidden state but contains the output state: $\hat{\mathbf{y}} \in \mathbb{R}^{l^{L+1}}$, which is a vector of the length $l^{L+1} \in \mathbb{N}$.

Assuming that $\mathbf{x} = \mathbf{h}^0$ and $\hat{\mathbf{y}} = \mathbf{h}^{L+1}$ makes it possible to express the whole feedforward calculation in Equation 3.1, which shows how to calculate the i-th component of the feature vector of the hidden state $\mathbf{h}^k_i$, by summing over the activations

$$h^k_i = \sum_{j=1}^{l^{k-1}} \sigma^k \left(h^{k-1}_j \cdot \theta^k_{i,j} + b^k \right). \tag{3.1}$$

3.1.2 Loss Function

The loss function is a measure of performance for a Neural Network under a given task. So the main goal of optimizing the network is to optimize the loss function. In the most basic case the loss function depends on the output of the Neural Network $\hat{\mathbf{y}}$ and the ground truth $\mathbf{y}$. One of the most common examples of a loss function is the Mean Squared Error (MSE) which is shown in Equation 3.2.

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{m} \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2 \qquad \text{m: number of samples} \tag{3.2}$$

As the main goal of the study is to identify additional b jets in $\mathrm{t\bar{t}}+\mathrm{b\bar{b}}$ events, it is reasonable to use a loss function suitable for a binary classification task. This implies that the Neural Network can only classify additional b jets and nothing else. The most common function in this case is binary cross entropy as defined in Equation 3.3. This loss function assumes

$\hat{\mathbf{y}}$ to be a pseudo probability, meaning $\hat{\mathbf{y}} \in (0, 1)$. In the case of $\mathrm{t\bar{t}} + \mathrm{b\bar{b}}$, this is the pseudo probability of a particle being an additional b jet

$$L(\mathbf{y}, \hat{\mathbf{y}}) = -\left(\mathbf{y} \log\left(\hat{\mathbf{y}}\right) + (1 - \mathbf{y}) \log\left(1 - \hat{\mathbf{y}}\right)\right). \tag{3.3}$$

In binary cross entropy true positives are weighted the same as true negatives within the loss function. However, if most of the samples have the same label, the network can already achieve a low loss by classifying every node with the same label. This is specifically the case in $\mathrm{t\bar{t}} + \mathrm{b\bar{b}}$, as most particles are not additional b jets. Therefore, the labels of additional b jets must be weighted heavier in the loss function than other labels. The weight can however, also not be too high because then, every node can be classified as an additional b jet while achieving a relatively low loss. Therefore this weight has to optimized, but as the weight directly scales the loss function, it cannot be optimized like standard parameters. This weight is a hyperparameter that is further referred to as the "class weight scalar". The optimization is discussed within chapter 4.

3.1.3 Backpropagation

By just using random weights and biases for every layer in the initial state, the feedforward calculation from Equation 3.1 can be applied and the network will produce an output $\hat{\mathbf{y}}$. Consequently, this output is random and for a classification task the labels are just as good as randomly assigned labels. Following this, the loss can be calculated by using the outputs. The loss function is differentiable towards the parameters $\theta^{\mathbf{k}}$ and b^k, as the loss function itself is just a composition of differentiable functions. Therefore, the gradient descent method can be applied: the gradient always yields the direction of steepest ascend, as proven in section A.2. Accordingly, the negative gradient always yields the direction of steepest descent. For the sake of simplification, all learnable parameters are part of the vector ψ. This theoretical vector can be constructed by taking all entries from the matrices $\theta^{\mathbf{k}}$ and the scalars b^k and listing them as entries of the vector ψ. The greatest improvement to the loss function can then be achieved by adjusting the parameters ψ infinitesimally in the negative direction of the gradient. In real world applications this infinitesimal step is replaced by a small step whose size is determined by the learning rate $\gamma \in (0, 1)$, as formalized in Equation 3.4

$$\psi' = \psi - \gamma \operatorname{grad}_{\psi}\left(L(\hat{\mathbf{y}}(\psi))\right). \tag{3.4}$$

Using this knowledge, an iterative algorithm can be formulated for parameters to converge into a local optimum of the loss function:

- Evaluate the network output according to Equation 3.1 with the current parameters ψ and calculate the loss.
- Adjust the parameters according to Equation 3.4.

3.1.4 Data sets

With the backpropagation algorithm the network can be trained to locally minimize the total loss on training data. This does, however, not guarantee that the network performance generalizes to unseen data. To measure the performance on unseen data, the original data set is split into two smaller sets: Firstly, the training set whose data is used for the training of the network and secondly, the test set which is only used to measure the performance on unseen data.

One possible issue that prevents the network from generalizing is overfitting. Hereby, the network approximates the test data very closely, i.e. it also learns the underlying noise, which is entirely different in the test data and therefore results in a bad test loss. Networks with a huge number of trainable parameters are susceptible to overfitting, as in contrast to small networks these larger networks have the capacity to store the information needed for overfitting. However, historically Deep Neural Networks, as in networks with many layers and learnable parameters, have performed better and the trend continues to shift towards even deeper networks with e.g. ResNet that contains 1000 layers [11]. To resolve overfitting while still using Deep Neural Networks the loss function can be modified. As shown in Equation 3.5 the L^2-norm of the weight matrices θ^k is added to the loss function, which penalizes large weights

$$L(\mathbf{y}, \hat{\mathbf{y}}, \theta) = L_{\text{original}}(\mathbf{y}, \hat{\mathbf{y}}) + \alpha \sum_{\{i,j,k\}} \left(\theta^k_{i,j}\right)^2 . \tag{3.5}$$

The penalty to the loss function also introduces the weight decay rate α. This is an example of a hyperparameter, as the value of α is constant during training and cannot be optimized with backpropagation. In order to find the best possible value of α the performance of the fully trained networks has to be compared. However, the best value of α might not generalize to unseen data. Thus, the data set has to be split again. The final three data sets are:

- The training set, which is used to optimize network parameters.
- The validation set, whose purpose is to optimize hyperparamters.
- The test set, on which the final network performance is evaluated.

The value of α has direct impact on the loss function and can therefore not be optimized according to the modified loss function. Consequently, a different measure has to be used for α. After training, the original loss function without α can be used again, as weights are not being optimized anymore. The strategy used for hyperparameter optimization is further discussed in chapter 4.

3.1.5 Representation of $t\bar{t} + b\bar{b}$ events in a Neural Network Input

Every $t\bar{t} + b\bar{b}$ event consists of multiple particles, where each of them contains multiple features, e.g. the kinematic quantities of the particle. To allow this to be represented in a Neural Network, each particle i has to have its own feature vector ν_i as illustrated in Figure 3.2. These can be passed separately to the Neural Network, to allow for additional b jet classification.

However, passing jets individually into a Neural Network, only allows calculations on these single jets. Therefore implementing kinematic quantities like the invariant mass of multiple jets into a Neural Network, as in [3], requires complex algorithms. But there is strong physics motivation to use these combined features, thus using an elegant representation of these quantities is encouraged.

3.2 Graphs

Having defined Neural Networks and the background calculations in section 3.1, it becomes clear that a simple vector structure of the input does not easily allow to represent relations between different inputs. Therefore, an expanded representation of the input is suggested: Graphs.

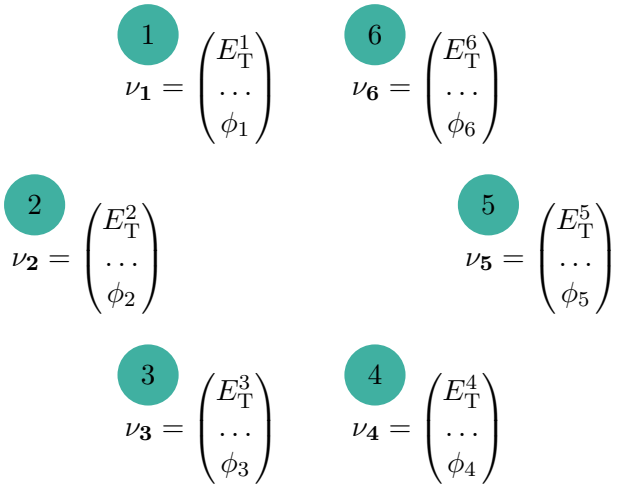

Figure 3.2: Sketch of the data representation of a single $t\bar{t} + b\bar{b}$ event for a Neural Network. Each particle i has its own feature vector ν_i. The feature vectors are passed singularly into the Neural Network. Therefore, when passing the data of a particle into the Neural Network, it cannot factor in information of other particles.

3.2.1 Formal Definition

This study uses parts of the definitions from [12]:

- A Graph is an object consisting of two sets called its vertex set and its edge set. The vertex set is a finite nonempty set. The edge set may be empty but otherwise its elements are two-element subsets of the vertex set.
- The elements of the vertex set of a Graph are called vertices or nodes and the elements of the edge set are called edges. The number of vertices is denoted by "v" and the number of edges by "e".
- If $\{X, Y\}$ is an edge of a Graph, $\{X, Y\}$ joins or connects the vertices X and Y. X and Y are adjacent to one another. The edge $\{X, Y\}$ is incident to each of X and Y, and each of X and Y is incident to $\{X, Y\}$. Two edges incident to the same vertex are called adjacent edges. A vertex incident to no edges at all is isolated.
- Two Graphs are equal if they have equal vertex sets and equal edge sets.

For the purpose of using Graphs with Neural Networks additional definitions are useful:

- Each edge also has an edge weight $\omega \in (0, 1)$ associated with it, which determines the "strength" of the connection.
- Each vertex is assigned a feature vector ν_i, which, in this Thesis, contains the aforementioned features of the jet i.

Another useful simplification for computer science applications is, instead of allowing anything to represent a vertex, just enumerate vertices starting from 1. This makes it possible to construct the adjacency Matrix $A \in \{0, 1\}^{v \times v}$, where every entry $a_{i,j}$ contains a 1 when the vertices i and j are connected and 0 otherwise.

3.2.2 Examples of Graphs

As the formal definition of a Graph doesn't really provide intuition as to what a Graph is, several examples are provided.

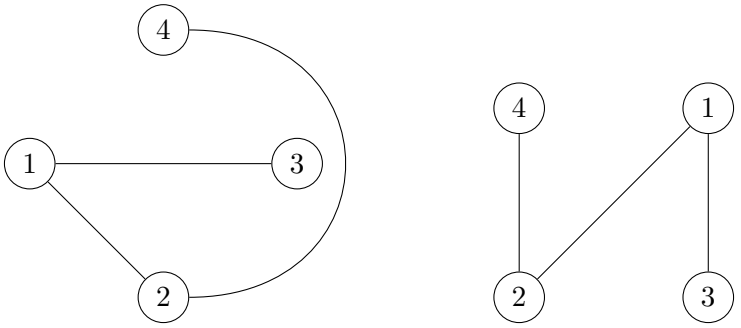

Figure 3.3: An example of two representations of the same Graph

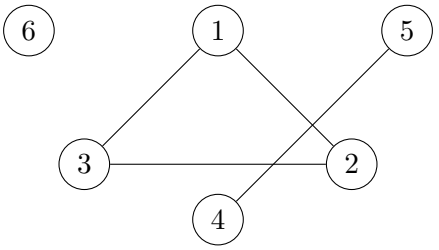

Figure 3.4: A representation of a Graph

The Graph G_1 in Figure 3.3 has the vertex set $\{1, 2, 3, 4\}$ and the edge set $\{\{1, 2\}, \{1, 3\}, \{4, 2\}\}$. The associated adjacency matrix is shown in Equation 3.6:

$$\mathbf{A}_{\mathrm{G1}} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}. \tag{3.6}$$

As illustrated in Figure 3.3, representations of Graphs can look very different, but still be mathematically equivalent.

The Graph G_2 in Figure 3.4 has the vertex set $\{1, 2, 3, 4, 5, 6\}$ and the edge set $\{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{4, 5\}\}$. Figure 3.4 depicts that the vertices 1,2 and 3 are adjacent as well as the vertices 4 and 5. The overlapping of the edge $\{4, 5\}$ with the edges $\{1, 2\}$ and $\{2, 3\}$ has no meaning, as this could just be drawn differently. Lastly, vertex 6 is isolated. The adjacency matrix of G_2 is shown in Equation 3.7:

$$\mathbf{A}_{\mathrm{G2}} = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{3.7}$$

3.2.3 Graph Input Data

To represent any $\mathrm{t\bar{t}} + \mathrm{b\bar{b}}$ event as a Graph, each particle i can be assigned to a vertex of a Graph. The feature vectors ν_i of each vertex i are determined by the data of the particle, which is similar to the input data in section 3.1. However, different to the previous design there are also edges which connect pairs of vertices. To construct the edges, different approaches can be taken.

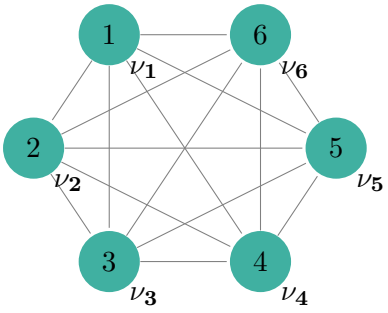

Figure 3.5: Implementation of a Graph without edge weights. Each node i still has its feature vector ν_i.

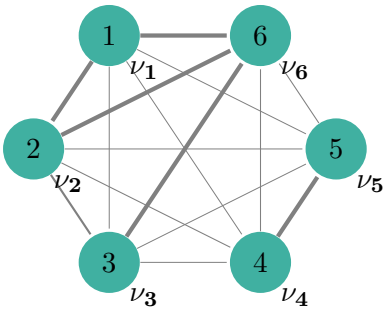

Figure 3.6: Implementation of a Graph with edge weights, as in the ΔR design, where the edge weights are scaled according to the ΔR value of the connected nodes. The edge weights are represented as the line thickness of the connections.

A simple approach to building the connections in the Graph is to just construct every possible edge and setting all associated edge weights to 1. These connections stress that all jets are dependent on each other and no redundant information is passed into the Neural Network. This approach is depicted in Figure 3.5.

The weights of the edges can also be scaled according to a function depending on the features of the connected vertices as illustrated in Figure 3.6. For example, the ΔR value, as introduced in section 2.2, of each pair of jets be can used to scale the connection weight. This is physically motivated as additional b jets, which are closer together in the η,ϕ-plane, i.e. they have a low ΔR value, tend to stem from the same mother-particle. It should be noted, however, that if the values of η and ϕ of each jet are still passed to the network from within the feature vector, this implementation adds redundant information to the network inputs, as the GNN could also calculate this information itself. But the internal calculation of ΔR is highly unlikely, as it involves multiple calculation steps which by themselves do not benefit the optimization of the loss function. That means gradient descent does not explore this direction and the possible redundancy can be ignored.

The ΔR design was successfully implemented in [4]. The function used to scale the weights is linear, meaning the highest ΔR value from the data set is mapped to 1 and the lowest to 0, all values in between are scaled linearly. This will further be referred to as "linear ΔR design". It is not intuitive, as the vertices of the jets, which are the furthest apart from each other, are connected with the strongest edge weight. Therefore, the "multiplicative inverse ΔR design" is implemented. In this design the values of $1/\Delta R$ are mapped linearly between 0 and 1.

Apart from changing the scaling function of the ΔR design, it is also possible to leave specific edges out of the edge set, which is represented in Figure 3.7. The specific edges to

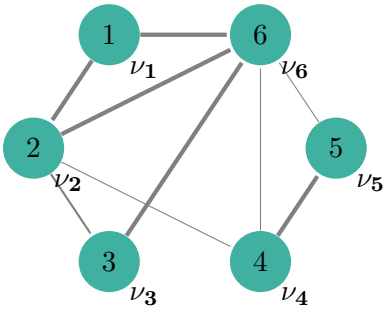

Figure 3.7: Implementation of a Graph with edge weights and only selected edges. This is resemblant of the lower ΔR threshold design with edge weights.

leave out are based on the ΔR value. According to a ΔR threshold, connections with a higher ΔR value will be left out, which will later be referred to as the "higher ΔR threshold design". As it can be useful to leave out ΔR values below the ΔR threshold, this is coined the "lower ΔR threshold design".

3.3 Graph Neural Networks

Having defined the inputs for a Graph structure in section 3.2, the Neural Networks to operate on such a structure are yet to be defined. Many different approaches for Graph Neural Networks exist. Most of these stem from Message Passing Neural Networks (MPNNs), therefore this layer is introduced first.

3.3.1 Message Passing Neural Network

The feedforward calculation of MPNNs is generally divided into three parts. The first two steps can be repeated, every iteration is called a time step t. The first part is the aggregation of the message in Equation 3.8. In this step, for each vertex i a message is calculated by summing the message function $\mathbf{M}_t$ of this time step t over every node $\mathcal{N}(i)$ that is adjacent to i. The message function $\mathbf{M}_t$ can be any differentiable function and can depend on the features of the nodes i and j at the current time step t, as well as the weight $\omega_{i,j}$ of the edge connecting the nodes i and j

$$\mathbf{m}_i^{t+1} = \sum_{j \in \mathcal{N}(i)} \mathbf{M}_t \left(\nu_i^t, \nu_j^t, \omega_{j,i} \right). \tag{3.8}$$

The second part of the MPNN calculation in Equation 3.9 is called the update function $\mathbf{U}_t$ of the time step t, which again can be any differentiable function. The only inputs $\mathbf{U}_t$ depends on are the previous feature vector ν_i^t and the message $\mathbf{m}_i^{t+1}$ for the node i. This information is then used to update the features of the node i

$$\nu_i^{t+1} = \mathbf{U}_t \left(\nu_i^t, \mathbf{m}_i^{t+1} \right). \tag{3.9}$$

The first two steps can be done an arbitrary number of times, afterwards the third step can be applied. The calculation in Equation 3.10 is called the readout step, the function $\mathcal{R}$ takes in the feature vectors of all nodes in the Graph $\mathcal{G}$ and lists all features in a single vector $\mathbf{g}$. This vector $\mathbf{g}$ can then be used as an input for a Neural Network to perform e.g. a Graph level classification, see Equation 3.11

$$\mathbf{g} = \mathcal{R} \left(\{ \nu_i^{\text{final}} \mid i \in \mathcal{G} \} \right) \tag{3.10}$$

$$\hat{\mathbf{y}} = \text{NN}(\mathbf{g}). \tag{3.11}$$

An important difference to the usual application of the MPNN algorithm in the application of assigning additional b jets in $t\bar{t} + b\bar{b}$ events, is that the readout step from Equation 3.10 does not apply in this case. In the readout step the Graph structure is normally lost and therefore no node classification is possible after this step. However, the calculation can also be done just by skipping Equation 3.10 and using the same Neural Network for every node to get a single value that represents the pseudo probability for the node to describe an additional b jet.

3.3.2 Graph Convolutional Neural Network

The following layer from [13] uses the MPNN algorithm combined into a single line in Equation 3.12. For this application the v feature vectors ν_i, which each contain f features, are combined into the matrix $\mathbf{X} \in \mathbb{R}^{v \times f}$. The matrix $\hat{\mathbf{A}} \in \mathbb{R}^{v \times v}$ is an adjusted variant of the adjacency matrix. As shown in Equation 3.13, only an identity matrix is added. From the definition of the adjacency matrix in section 3.2.1 follows that adding an identity matrix represents adding self loops to the Graph. The diagonal matrix $\hat{\mathbf{D}} \in \mathbb{R}^{v \times v}$ is the vertex degree of the adjusted adjacency matrix, the entry $\hat{D}_{i,i}$ stores how many nodes are incident to the vertex i. The learnable parameters in this case are the matrices $\mathbf{\Theta}^t$, and no readout step takes place. It should be noted that the exponent in Equation 3.12 is to be understood elementwise

$$\mathbf{X}^{t+1} = \sigma\left(\hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}^t\mathbf{\Theta}^t\right) \tag{3.12}$$

$$\hat{\mathbf{A}} = \mathbf{A} + \mathbb{I} \tag{3.13}$$

$$\hat{D}_{i,i} = \sum_j \hat{A}_{i,j} \qquad \forall i, j : i \neq j \longrightarrow \hat{D}_{i,j} = 0. \tag{3.14}$$

3.3.3 Gated Graph Sequence Neural Networks

The best contender for classifying additional b jets with a GNN from the study [4] are Gated Graph Sequence Neural Networks (GGSNNs). As the study already contains an explanation of this layer, it will not be explained any further.

3.4 Hypergraphs

Using Graphs as an input type for a Neural Network makes it possible to input kinematic features that can only be calculated by combining the information of pairs of nodes. While the network is capable of calculating features that require combining the information of even more nodes, it is still very unlikely that the network constructs complicated physical quantities, as the network only converges into a local optimum of the loss function. Some of these quantities have, however, shown great performance in jet classification tasks [3]. Therefore strong motivation exists to allow the input of more complicated quantities involving multiple nodes. As Hypergraphs can have connections that link any number of vertices, they are a solution to this problem.

3.4.1 Formal Definition

Hypergraphs are a generalization of Graphs, thus the definition from section 3.2.1 can be used with small adjustments:

- A Hypergraph is an object consisting of two sets called its vertex set V and its hyperedge set E. The vertex set is a finite nonempty set. The hyperedge set is a subset of the power set of the vertex set: $E \subseteq \mathcal{P}(V)$. The power set $\mathcal{P}$ of any set A

contains the set of all subsets of A which includes the empty set and A. As the vertex set is unchanged from Graphs to Hypergraphs and the edge set of Graphs are also a subset of the power set of the vertices, every Graph is also a Hypergraph.

- The elements of the vertex set of a Graph are called vertices or nodes and the elements of the hyperedge set are called hyperedges. The number of vertices is denoted by "v" and the number of hyperedges by "e".
- If $\{X_1, X_2, \ldots, X_N\}$ is a hyperedge of a Hypergraph, $\{X_1, X_2, \ldots, X_N\}$ joins or connects all the vertices within the hyperedge, they are adjacent to each another. The edge $\{X_1, X_2, \ldots, X_N\}$ is incident to each of vertex from the set. A vertex incident to no hyperedges at all is isolated.
- Two Hypergraphs are equal if they have equal vertex sets and equal hyperedge sets.

For the usage of Hypergraphs within machine learning applications, additional definitions and simplifications that stray from standard notion of Hypergraphs also make sense. Parts of these are taken from [14]:

- Vertices are simplified so they are only allowed to be represented by natural numbers starting from 1 without leaving gaps. The same simplification is applied to hyperedges.
- The Hypergraph can be represented by the incidence matrix $\mathbf{H} \in \{0,1\}^{v \times e}$. $h_{i,j}$ is 1 if the vertex i is part of the hyperedge j and 0 otherwise.
- Each hyperedge also has an hyperedge weight $\omega \in \mathbb{R}$ associated with it, which determines the "strength" of the connection. For the sake of simplification of calculations these hyperedge weights are stored within the diagonal matrix $\mathbf{W} \in (0,1)^{e \times e}$.
- The vertex degree $\mathbf{D} \in \mathbb{R}^{v \times v}$ is a diagonal matrix that stores the cumulative edge weight impacting the node i: $D_{i,i} = \sum_{j=1}^{e} W_{j,j} H_{i,j}$.
- The edge degree $\mathbf{B} \in \mathbb{N}^{e \times e}$ is a diagonal matrix that stores the number of nodes the hyperedge j is connected to: $B_{j,j} = \sum_{i=1}^{v} H_{i,j}$.
- Each vertex i is assigned a feature vector ν_i, which contains the aforementioned f features of the jet i. All v feature vectors ν_i are further represented in the matrix $\mathbf{X} \in \mathbb{R}^{v \times f}$.

3.4.2 Examples of Hypergraphs

The Hypergraph $\mathcal{H}_1$, in Figure 3.8, has the vertex set $\{1,2,3,4,5,6\}$, the hyperedge set $\{\{1,2,3\},\{1,2,4\},\{3,4,5\}\}$ and the hyperedge weights $\mathbf{W} = \mathrm{diag}\,(0.2,\ 0.8,\ 0.4)$. Using this information, the incidence matrix, the vertex degree and the edge degree can be calculated.

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \qquad \mathbf{D} = \begin{pmatrix} 1.0 & & & & & \\ & 1.0 & & & & \\ & & 0.6 & & & \\ & & & 1.2 & & \\ & & & & 0.4 & \\ & & & & & 0.0 \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} 3 & & \\ & 3 & \\ & & 3 \end{pmatrix} \tag{3.15}$$

3.4.3 Hypergraph Input Data

The underlying motivation to implement the Hypergraph structure can be explained when looking at the Feynman diagram of the leading order of a $\mathrm{t\bar{t}} + \mathrm{b\bar{b}}$ event in section 2.3. Three particles each are generated by two top quarks. Due to conservation of energy the invariant

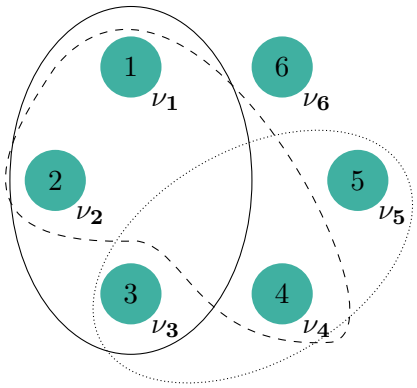

Figure 3.8: Example of a Hypergraph with 6 vertices and 3 hyperedges. The hyperedge with the dashed line connectes the vertices 1,2 and 4. The vertices 3,4 and 5 are connected by the hyperedge represented by the dotted line. The solid line represents the hyperedge connecting the nodes 1, 2 and 3.

mass of the respective three particles is still the same as the invariant mass of the top quark. In principle, from calculating the invariant masses of pairs of three particles, the Neural Network could rule out six particles, as these stem from a top quark and can therefore not be additional b jets. Reality however, is not that easy, as the number of combinations of three jets grows by $\binom{n}{3} \in \mathcal{O}\left(n^3\right)$, where n is the total number of particles. For this reason, it is possible for multiple combinations of three particles to have an invariant mass close to the top mass considering its natural width as well as the reconstruction uncertainty. Hence, this cannot be used as the only information to classify additional b jets. However, this still adds information to the network that may help to improve the classification rate.

The Hypergraph information is not redundant, even though the network has the information to construct the invariant masses, there is no probable way for the network to learn the construction of invariant masses. This stems from the fact that the invariant mass construction has many intermediate steps that do not improve the loss function. Therefore, gradient descent does not explore this direction.

The actual calculation of the invariant mass with Hypergraphs is analogous to the implementation of the ΔR design:

1. The invariant mass of all combinations of three particles is calculated for every event.
2. From every calculated invariant mass the top mass of $173\,\mathrm{GeV/c^2}$ [5] is subtracted and the absolute value of this difference is taken.
3. An hyperedge is created between any combination of three jets for which the absolute invariant mass difference is lower than a specified threshold, the "lower Δm threshold".
4. The hyperedge weights are calculated from the absolute invariant mass difference according to a scaling function that output numbers between 0 and 1.

It is possible to use a "higher Δm threshold" in step 3, where only hyperedeges are created with an absolute invariant mass difference higher than the threshold.

Information can be used redundantly in this implementation, because hyperedges are chosen according to the Δm value, and subsequently, the scaling of the hyperedge weights

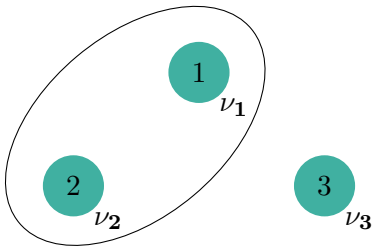

Figure 3.9: An example of a Hypergraph with a single hyperedge connecting the nodes 1 and 2. Node 3 is isolated. Each node i contains a feature vector ν_i.

is determined by the Δm value as well. Both options are presented as the evaluation and optimization are discussed in chapter 4 and all possibilities are explored.

3.5 Hypergraph Neural Networks

While the inputs for the Hypergraph structure are defined, the Neural Networks that will be used to operate on such a structure are yet to be specified. In the case of Hypergraphs the pytorch geometric implementation also only has one layer that can accept Hypergraph inputs [14].

3.5.1 Hypergraph Convolution

The "Hypergraph Conv" class from [14] uses the Message Passing Neural Network algorithm compressed into the single Equation 3.16

$$\mathbf{X}^{t+1} = \sigma\left(\mathbf{D}^{-\frac{1}{2}}\mathbf{H}\mathbf{W}\mathbf{B}^{-1}\mathbf{H}^{\intercal}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}^{t}\mathbf{\Theta}\right). \tag{3.16}$$

The matrix $\mathbf{X}^0$ contains the v nodes, each with f_0 features of the Hypergraph mentioned in section 3.4.1. The matrices $\mathbf{X}^t \in \mathbb{R}^{v\times f_t}$ for $t > 0$ store the updated state, with v nodes and each f_t features, for each time step t.

For an intuitive understanding of the Hypergraph convolution the calculation is applied to the Hypergraph in Figure 3.9, with the parameters in Equation 3.17

$$\mathbf{X}^0 = \begin{pmatrix} 0.1 & 1.3 & 0.5 & 3 \\ 0.9 & 1 & 0 & 0.2 \\ 0.3 & -2 & 1 & 0.9 \end{pmatrix} \qquad \mathbf{H} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \qquad \mathbf{W} = \begin{pmatrix} 0.9 \end{pmatrix}. \tag{3.17}$$

The update calculation starts with the evaluation of the matrix $\mathbf{A}_* = \mathbf{D}^{-\frac{1}{2}}\mathbf{H}\mathbf{W}\mathbf{B}^{-1}\mathbf{H}^{\intercal}\mathbf{D}^{-\frac{1}{2}} \in \mathbb{R}^{v\times v}$, which is constant for the hypergraph and has the dimension of an adjacency matrix. The application to the example in Equation 3.18 shows immediate problems with isolated nodes, as each of these leads to an empty row and column in the matrix $\mathbf{D}$, which makes $\mathbf{D}$ not invertible. In these cases only the submatrix that is invertible is actually inverted and the rest stays the same. The pseudo adjacency matrix $\mathbf{A}_*$ in Equation 3.19 shows important features of the "Hypergraph Conv" calculation, the first rows of the matrix are equal to one another, as the nodes 1 and 2 are part of the same hyperedge(s). The third row of $\mathbf{A}_*$ only contains zeros because the third node is not connected to any hyperedge

$$\mathbf{B} = \mathbf{B}^{-1} = \begin{pmatrix} 1 \end{pmatrix} \qquad \mathbf{D} = \begin{pmatrix} 0.9 & 0 & 0 \\ 0 & 0.9 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \longrightarrow \quad \mathbf{D}^{-\frac{1}{2}} \approx \begin{pmatrix} 0.9^{-\frac{1}{2}} & 0 & 0 \\ 0 & 0.9^{-\frac{1}{2}} & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{3.18}$$

$$\mathbf{A}_* = \mathbf{D}^{-\frac{1}{2}}\mathbf{H}\mathbf{W}\mathbf{B}^{-1}\mathbf{H}^\intercal\mathbf{D}^{-\frac{1}{2}} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{3.19}$$

The next step in the calculation is to multiply the matrix $\mathbf{A}_*$ with $\mathbf{X}^0$ in Equation 3.20. This step further improves the interpretability of $\mathbf{A}_*$ as an adjacency matrix. This becomes evident when focusing on the multiplication of the first row of $\mathbf{A}_*$ with the first column of $\mathbf{X}^0$, the result is a weighted sum of the first node features and thus defines how to calculate the first feature of the first node from the first feature of all nodes

$$\mathbf{A}_*\mathbf{X}^0 = \begin{pmatrix} 1 & 2.3 & 0.5 & 3.2 \\ 1 & 2.3 & 0.5 & 3.2 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{3.20}$$

The second to last step of the calcuation is to multiply with the weight matrix $\Theta \in \mathbb{R}^{f_t \times f_{t+1}}$. The number of features in the next step has to be defined before the calculation, in the example it is $f_1 = 2$. The entries of $\mathbf{\Theta}$ are the actual learnable part, in the example they are arbitrarily chosen. A detailed look into the multiplication with $\mathbf{\Theta}$ reveals that only features of the same nodes are recombined in this step of the calculation. Therefore the weight matrix can only indirectly, from the previous calculation, factor in the information of the Hypergraph

$$\Theta = \begin{pmatrix} 0.1 & 0.4 \\ 0.5 & 0.3 \\ 0.3 & 1 \\ 1 & 0.5 \end{pmatrix} \qquad \mathbf{A}_*\mathbf{X}^0\Theta = \begin{pmatrix} 4.6 & 3.19 \\ 4.6 & 3.19 \\ 0 & 0 \end{pmatrix}. \tag{3.21}$$

The final step is to apply an activation function. The authors of [14] use the leaky ReLu activation function. Applying leaky ReLu in the result of Equation 3.21, does not change anything because only positive numbers are involved. Therefore this matrix is already the final result of the example.

The result of Equation 3.21 shows two important effects of the calculation. The first effect is oversmoothing. The first two nodes are connected to exactly the same hyperedges and therefore receive exactly the same information, thus the result for both nodes is the same. The second effect of the calculation is that the information of the third node is disregarded entirely because the node has no connections. This feature can have a positive impact on the additional b jet classification rate, because in combination with the Δm design for the hypergraph, the network can ideally know which node describes the additional b jet, as this node is less likely to be a part of hyperedges, due to the construction of the hypergraph in section 3.4.3.

3.5.2 Adaptation Algorithm

Looking back at the Graph Convolutional Neural Network in section 3.3.2, the update step of the Message Passing Neural Network looks very similar to the Hypergraph Convolutional

Neural Network. This becomes clear from the structure of the contents of the activation function in Equation 3.16: Some non-learnable matrices that handle the Graph structure and weights, then the matrix $\mathbf{X}$ which contains all nodes and features and finally the learnable matrix $\mathbf{\Theta}$.

The similarities between the discussed Graph- and Hypergraph layer raise the question whether any layer that uses Graphs as inputs can be modified in the same way to allow for Hypergraph inputs. Therefore, a general transformation that maps any Hypergraph to a Graph is needed. Without any specifications there are many ways to construct such a transformation. As the Hypergraph structure is specifically introduced to be able to represent more information, the goal of such a transformation, for this study, is to keep all properties of the Hypergraph. In section A.3, an algorithm is suggested to achieve this goal.

Using the transformation algorithm the Δm design can be implemented in the previously best performing layer the Gated Graph Sequence Neural Network (GGSNN). The new design will further be referred to as Gated Hypergraph Sequence Neural Network (GHGSNN), to distinguish it from the previous one.

3.6 Network Merging

As the results from [4] already showed improvements to the classification rate in [3], it is not straightforward to make further improvements. Because the most promising quantity ΔR is already used and the Δm design is not expected to perform better than the previously used layer, the question emerges as to how the different networks can be merged.

3.6.1 Addition Design

The first approach, the "addition design" in Figure 3.10, first evaluates the outputs of all networks that shall be merged. Thus, every network generates a single feature for every node that represents the pseudo probability for the node to represent an additional b jet. The results of every network for a node are then combined into a single new feature vector. Every node is then passed singularly into a Neural Network to generate the final output, which is a single value that represents the additional b likeness. As the information of different networks is combined with a learnable layer, the importance of the different networks can be learned.

3.6.2 Combination Design

The second approach, the "combination design" in Figure 3.11, first evaluates the "helper networks", which is in this case the GHGSNN, and generates a single output for each node. These outputs are appended to the original feature vectors. Then all information is passed into the GGSNN that generates the final output. The main benefit in this design is that, at the time of final evaluation, the network still has access to all feature information and therefore is capable of deciding which network outputs are more important under the given circumstances.

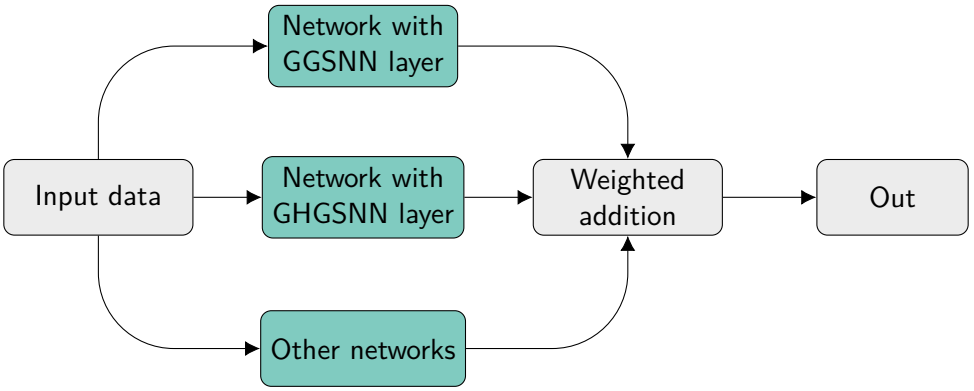

Figure 3.10: Sketch of the addition design for the merging of different networks. The input data contains all node features, the Graph data and the Hypergraph data. This data is evaluated singularly in the different networks which each generate a single output for every node that represents the additional b likeness. The node information is then recombined to a single feature vector for each node. In the sketch, before the weighted addition, every node has 3 features. Every node is then singularly passed into the learnable addition layer, which adds the node features each multiplied with a weight. After rescaling to $(0, 1)$ this represents the final pseudo probability for the node to be an additional b jet.

$$\nu^{(i)} = \left(p_{\mathrm{t}}^{(i)}, \eta^{(i)}, ...\right) \longrightarrow \nu^{(i)} = \left(h_{\mathrm{res}}^{(i)}, p_{\mathrm{t}}^{(i)}, \eta^{(i)}, ...\right)$$

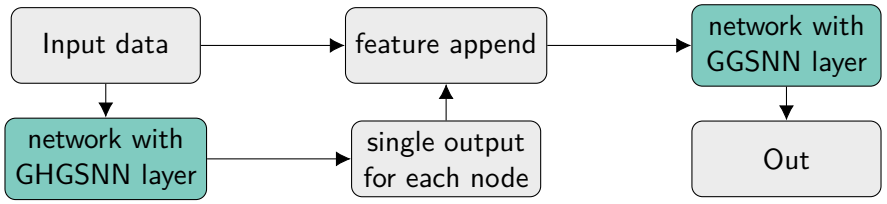

Figure 3.11: Sketch of the combination design for the merging of different networks. The input data contains all node features, the Graph data and the Hypergraph data. Firstly, all "helper" networks are evaluated, in this case this is only the GHGSNN. All "helper" networks generate a single output $h_{\mathrm{res}}^{(i)}$ for each node i, which represents the additional b likeness. These new features are then appended to the original node features. Finally, the nodes with additional features are passed into the previously best performing layer, the GGSNN. This generates the final output.

4 Bayesian Optimization

Previous studies have already analysed different architectures of Neural Networks for the additional b jet classification in $t\bar{t} + b\bar{b}$, namely Deep Neural Networks [3] and Graph Neural Networks [4]. This study continues to use GNNs. Formerly, hyperparameters have been optimized either manually or by applying a grid search. Bayesian optimization, on the other hand, can have a number of advantages, as illustrated in Figure 4.1 from [15].

4.1 Motivation

Unlike model parameters, hyperparameters are set before training by the programmer and cannot be optimized by backpropagation like the standard parameters. Additionally, some hyperparameters might only be well defined for specific values of other parameters. For example a hyperparameter can be whether or not to include a fully connected layer at the end of the network. Then the number of nodes in this layer is a hyperparameter that is only required when the fully connected layer is used. To account for such possibilities the hyperparameter space ξ is considered to be Graph-structured.

For this consideration, training and validation is simplified into a single function that takes in hyperparameters and outputs the validation loss. The optimization can be formalized as finding the argument $\mathbf{x}^*$ from the parameter space ξ that minimizes the objective function, which is in this case the validation loss $L(\mathbf{x})$, as shown in Equation 4.1

$$\mathbf{x}^* = \arg\min_{\mathbf{x} \in \xi} L(\mathbf{x}). \tag{4.1}$$

A simple optimization method is to create a grid of hyperparameter combinations and select the one with the lowest error on the validation set. This is called “grid search” and has the disadvantage of testing unnecessary points, such as a point where all surrounding points evaluate to a high validation error. Additionally, training and validating the network is very costly. Consequently, a smart data point selection is needed.

Sequential Model-based Global Optimization (SMBO) algorithms have been used in many applications where evaluation of the loss function is expensive [16]. As the true objective function is costly, a surrogate S, which is much cheaper to evaluate than the objective function, is constructed to approximate the fitness of the parameter space. The surrogate S

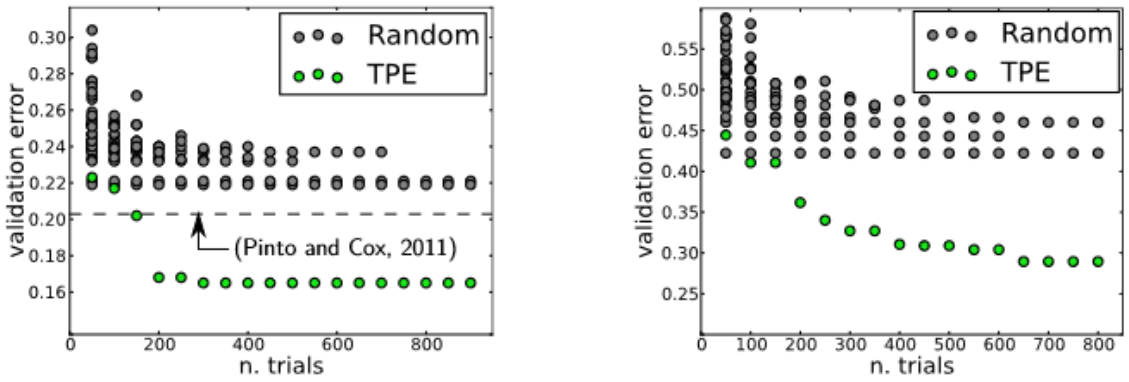

Figure 4.1: Example of advantages of Bayesian optimization over random search on two different image classification datasets, taken from [15]. Validation error is plotted against the number of trials. The Bayesian optimization using the Tree-structured Parzen Estimator (TPE), which is explained in section 4.2, reaches a much lower validation error much faster.

can either be a model M_t of the objective function or any transformation of M_t. The transformation of M_t is called acquisition function and defines "the usefulness" of a point for the underlying optimization task. The surrogate can then be numerically optimized to suggest the next test point $\mathbf{x}^*$. The proposed point is then evaluated in the objective function. All new information is appended to $\mathcal{P}$, the history of observations and associated parameters. The surrogate can subsequently be recalculated with the history $\mathcal{P}$. All steps are iteratively repeated until the maximum number of evaluations is reached. The pseudo-code for this is shown in algorithm 1. As this process is graphically intuitive, a depiction is provided in Figure 4.2.

Algorithm 1: The pseudo-code of a generic Sequential Model-based Global Optimization. The algorithm is taken from [16] but slightly adjusted.

Function `SMBO`(L, M_0, T, S)**:**
- $\mathcal{P} \longleftarrow \emptyset$
- **for** $t \longleftarrow 1$ **to** T **do**
  - $\mathbf{x}^* \longleftarrow \arg\min_{\mathbf{x}} S(\mathbf{x}, M_{t-1})$
  - Evaluate $L(x^*)$ \\this step is expensive
  - $\mathcal{P} \longleftarrow \mathcal{P} \cup (\mathbf{x}^*, L(x'))$
  - Fit a new surrogate model M_t to $\mathcal{P}$
- **end**
- **return** $\mathcal{P}$

Sequential Model-based Global Optimization accelerates hyperparameter optimization in comparison to the aforementioned methods. But the curse of dimensionality still applies, i.e. adding parameters to the search space exponentially increases the number of possible parameter combinations in the search space. Even though the optimizer can theoretically "recognize" if parameters are uncorrelated and optimize the associated parameters independently, optimizing many parameters simultaneously still worsens performance. Therefore, any parameters for which no causal link is to be expected, should not be optimized simultaneously but rather iteratively, which drastically reduces the impact of the curse of dimensionality.

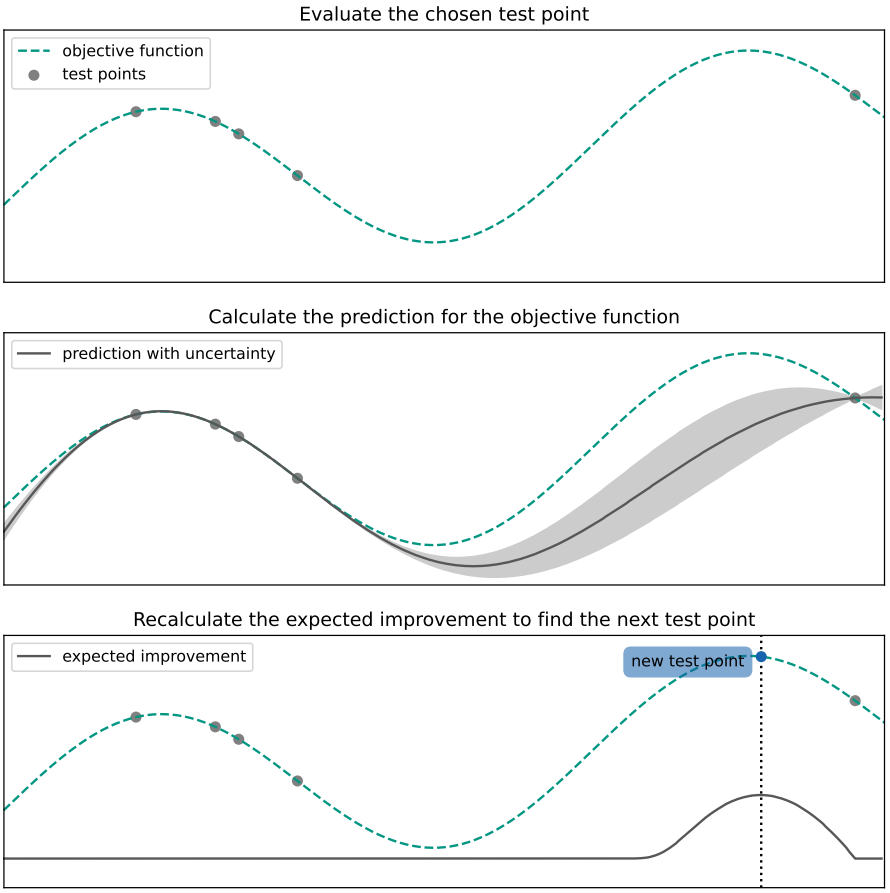

Figure 4.2: An example of the iterative update algorithm for hyperparameter optimization. Scalings and axes are left out for the introduction. The first step is to evaluate the objective function for the current test point, as shown in the upper plot. This new information can then be used to calculate the prediction with uncertainties of the objective function for the whole parameter space, which is depicted in the middle plot. From the updated prediction, the expected improvement of the loss can also be recalculated. Finally, the argument which maximizes the expected improvement can be located and used as the new test point.

4.2 Loss Function Model

To implement Sequential Model-based Global Optimization, an algorithm to model the loss function based on the history of parameters and observations is needed. The authors of [16] suggest using a Gaussian Process (GP) approach or a Tree-structured Parzen Estimator (TPE), with the side note that TPE is built for "hyperparameter optimization tasks that mean high dimensions and small fitness evaluation budgets". The difference in the results of GP and TPE is that a GP models the probability of a loss function value under the assumption of x having a specific value: $p(L \mid x)$. The TPE, however, models two probabilities. Firstly, the probability of a value x under the assumption of a specific loss function value: $p(x \mid L)$. Secondly, the probability for any value of the loss function $p(L)$. As this study aims to optimize many hyperparameters, TPE will be used to model the loss function.

4.3 Acquisition Function

4.3.1 Upper Confidence Bound

Having defined a model for the loss function makes it already possible to implement Sequential Model-based Global Optimization, by just adding the expected value of the loss $\mu_L(\mathbf{x})$ to the associated standard deviation $\sigma_L(\mathbf{x})$ multiplied with a coefficient β for any given parameter. This is the so called upper confidence bound acquisition function $u(\mathbf{x})$ and is shown in Equation 4.2

$$u\left(\mathbf{x}\right) = \mu_L(\mathbf{x}) + \beta \cdot \sigma_L(\mathbf{x}). \tag{4.2}$$

As the argument of the maximum of the acquisition function is proposed as the next point of evaluation of the objective function, the value of β in Equation 4.2 balances "exploration vs. exploitation":

- A low value of β means that the maximum of $u(\mathbf{x})$ is very likely to be the maximum of $\mu_L(\mathbf{x})$, therefore a low β favours exploitation of a promising region over exploration of a new one.
- A high value of β on the other hand, weighs the standard deviation more than the mean and thus favours exploration over exploitation.

However, the upper confidence bound acquisition function introduces another parameter during hyperparameter optimization. Therefore, another acquisition function that intrinsically balances exploration vs. exploitation is most commonly used, the expected improvement.

4.3.2 Expected Improvement

The improvement $I(\mathbf{x})$ in Equation 4.3 quantifies how useful evaluating each point of the parameters space is. The difference between the best observed loss so far L^* and the loss of every point of the parameters space $L(\mathbf{x})$ is taken. Thus any point of the parameter space with a loss lower than the current best observed loss, yields a positive valued improvement. Points with worse losses, i.e. a negative difference, are replaced with zeros by the max() function, as depicted in Figure 4.3

$$\mathrm{I}(\mathbf{x}) = \max\left(L^* - L(\mathbf{x}), 0\right). \tag{4.3}$$

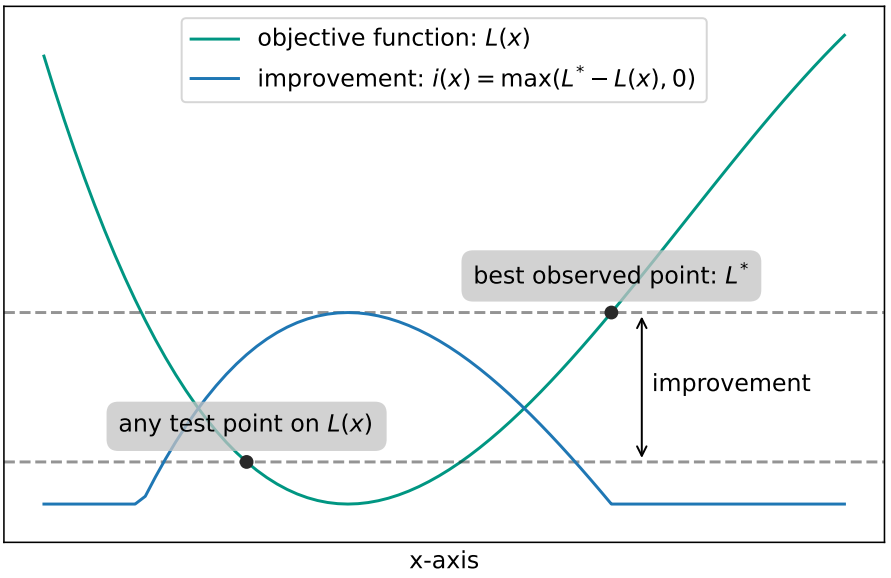

Figure 4.3: Illustrative example of the improvement function. The improvement of a single test point is visualized between the two dashed lines. The example also shows that the improvement function is zero anywhere where the loss is higher than the best observed loss.

However, the improvement is only a theoretical quantity because the objective function is not known. Nevertheless, the expected improvement (EI) in Equation 4.6, which is the expected value of the improvement, can be used as an acquisition function without the introduction of another parameter

$$\begin{aligned} \mathrm{EI}(\mathbf{x}) &= \mathbb{E}\left(\mathrm{I}(\mathbf{x})\right) && (4.4)\\ &= \mathbb{E}\left(\max\left(L^* - L(\mathbf{x}), 0\right)\right) && (4.5)\\ &= \int_{-\infty}^{\infty} \max(L^* - L, 0)\, p\left(L \mid \mathbf{x}\right) \mathrm{d}L. && (4.6) \end{aligned}$$

Justified by the central limit theorem, the Gaussian Process and the Tree-structured Parzen Estimator both model the loss function with a Gaussian distribution. This allows several simplifications to Equation 4.6, which yields Equation 4.7. The function ϕ is the Gaussian probability density function (PDF) with 0 mean and variance 1 and Φ the corresponding cumulative density function (CDF)

$$\mathrm{EI}(\mathbf{x}) = \sigma_L(\mathbf{x})\left(a \cdot \Phi(a) + \phi(a)\right) \qquad a = \frac{L^* - \mu_L(\mathbf{x})}{\sigma_L(\mathbf{x})}. \tag{4.7}$$

4.4 Implementation

4.4.1 General approach and Challenges

The original idea to implement automated hyperparameter optimization arose from the consideration that hyperparameter optimization can be used to optimize many more variables than usual. Therefore many different hyperparameters are explored. For example, the number of fully connected layers in a Neural Network can be used as a hyperparameter.

The approach of considering every possible hyperparameter and trying to optimize them has big challenges associated with it. Firstly, the optimizer only finds a local optimum and not a global one. Therefore, if a parameter has a logical reason to have a specific value, this choice is in most cases better than the optimized result. Additionally, optimizing multiple parameters at the same time can lead to problems as dependencies of specific parameters to the loss function can be suppressed by the change of a different parameter. In combination with the aforementioned curse of dimensionality, the optimizer cannot find a satisfying optimum within a reasonable number of trials when optimizing all possible hyperparameters at once.

This problem can be avoided by only optimizing small groups of hyperparameters at once and fixing the other hyperparameters. The small groups mostly contain parameters whose impact on the loss function also depends on the other parameters and therefore these parameters need to be optimized simultaneously.

4.4.2 Optimization Results

The first of the following hyperparameters are mostly independent of the optimized Neural Network and are thus explained without any model in mind.

Batch Size

The batch size is a natural number and controls how many graphs are processed within backpropagation in a single iteration. The batch sizes are initially drawn from a loguniform distribution between 50 and 5000. The plot in Figure 4.4 shows all 50 evaluated batch sizes and the corresponding 2/2 percentages as well as the number of epochs needed to reach peak performance. A strong negative correlation between the batch size and the 2/2 percentage is shown. However, for batch sizes between 50 and 500 the correlation vanishes. Other than that a slight correlation between the batch size and the number of epochs until peak performance is reached is indicated. As no correlation for the lower batch sizes has been found, the original batch size of 200 from [4] is used.

Jet Charge

This hyperparameter decides on whether or not to include the jet charge from section 2.4 within each node. The impact of the jet charge is very small. Therefore, most of the measured performance data is noise. However, more often than not the loss was lower when leaving the jet charge out.

Missing Transverse Energy

The missing transverse energy, also from section 2.4, is a node within the graph. Neutrinos cannot be directly measured within the CMS detector, and leading order $t\bar{t} + b\bar{b}$ events contain a neutrino in the final state. Therefore, MET is proportional to the transversal energy of the neutrino. The hyperparameter in this case is whether or not to include MET within the network. Generally a slight positive correlation between the usage of MET and a higher 2/2 classification rate was observed as shown in Table 4.1. Especially within the Hypergraph designs MET had a positive impact on the loss function. Thus, MET was always included.

Normalization method of the features

The features of all nodes can either normalized linearly or with Gaussian normalization. No difference in performance was detected for either option.

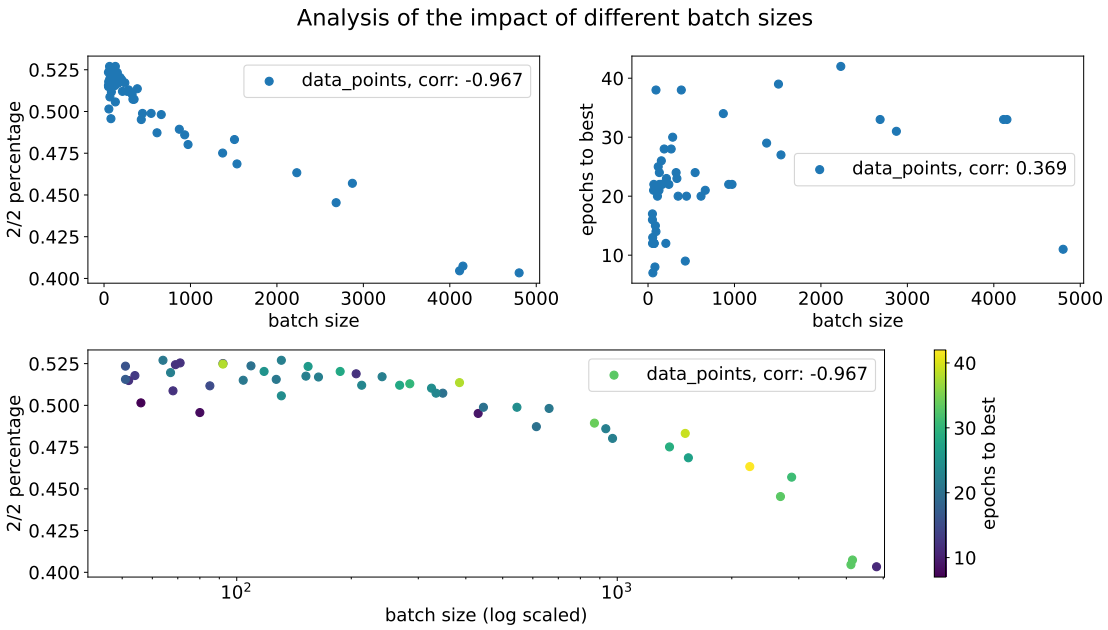

Figure 4.4: Result of the batch size analysis. The different evaluated batch sizes are chosen with TPE. The upper left plot shows the 2/2 percentage over different batch sizes. A strong negative correlation between the batch size and the 2/2 percentage is indicated. The upper right plot compares the number of epochs until peak performance is reached to the batch sizes. A weak correlation between the two parameters is also shown. The lower plot shows the 2/2 percentage over the log scaled batch size. The color of the points is based on the number of epochs until peak performance is reached. With the log scaled axis a plateau between batch sizes of 50 to about 500 is revealed.

Network	include MET	mean 2/2 percentage
GGSNN	True	52.25
	False	51.35
GHGSNN	True	42.89
	False	35.72

Table 4.1: Mean performance of the GGSNN and GGSNN for the two MET options. Shown is the network, whether or not MET is included and the mean 2/2 percentage.

Class Weight Scalar

The class weight scalar, as introduced in section 3.1.2, describes how much the nodes of additional b jets have to be weighted within the loss function. Initially the class weight scalar is used, because the number of nodes representing additional b jets is much lower than the number of other nodes. Thus, a relatively good loss can already be achieved by classifying all nodes as non-additional b jets. By upscaling the additional b jets weights in the loss to the same weight as all other nodes, this issue is no longer existent. This is the approach that has already been implemented within [4].

Many different values of the class weight scalar are possible and have different implications on the loss function. Additionally, only optimizing the loss function leads to scenarios where the loss far outperforms other networks, but the 2/2 classification rate is worse. Therefore, the class weight scalar is optimized with Bayesian optimization. The main optimization task is to find the class weight scalar that yields the best final 2/2 classification rate of a GGSNN. The secondary optimization goal is to find a class weight scalar that takes the smallest number of epochs to reach peak performance. The plots in Figure 4.5 show all 90 evaluated class weight scalars and the corresponding 2/2 percentages as well as the epochs needed to reach peak performance. The main result of the analysis is that the additional b jets do not have to be weighted, as the best class weight scalar is very close to one. This result is surprising and reasons are speculative. However, one possible root cause might be that the nodes which do not describe an additional b jet are scaled down further, leading to less confusion. The analysis also shows that, even without a large class weight for the additional b jets, the GNN does not classify all nodes as non-additional b jets. As the backpropagation algorithm optimizes according to the loss function and the main goal of the study is to achieve a high 2/2 percentage, the class weight scalar was fixed to the best value of 0.984 for every following training.

ΔR Design

The first decision of the ΔR design defined in section 3.2.3, is whether to use the higher or lower threshold design. Then the threshold also has to be chosen from any number between the lowest ΔR of any event 0.0011 and the highest according ΔR of 5.6257. The optimizer converged to always keeping most Graph connections, i.e. all connections with a ΔR value higher than 0.0011 or all connections with a ΔR value lower than 5.6257. The second optimization of the ΔR values is the normalization. The options in this case are either a linear normalization of ΔR values or a linear normalization of the values of $1/\Delta R$. Neither of the two designs nor the normalizations had an impact on the performance.

ΔM Design

The Δm design defined in section 3.4.3 can be implemented with either the higher or lower threshold. The lower threshold design is meant to only build connections between nodes that stem from a top quark. The threshold in this case is drawn from a uniform distribution between 2 GeV and 70 GeV. The higher threshold design draws the threshold from a loguniform distribution between 30 GeV and 500 GeV. The optimal choice in this case is to use the higher threshold design with a threshold of 56 GeV.

Data Input

The original node features contain the seven values in Equation 4.8. In multiple runs of optimization no difference in performance was detected when not using ϕ, E and q_{J} from

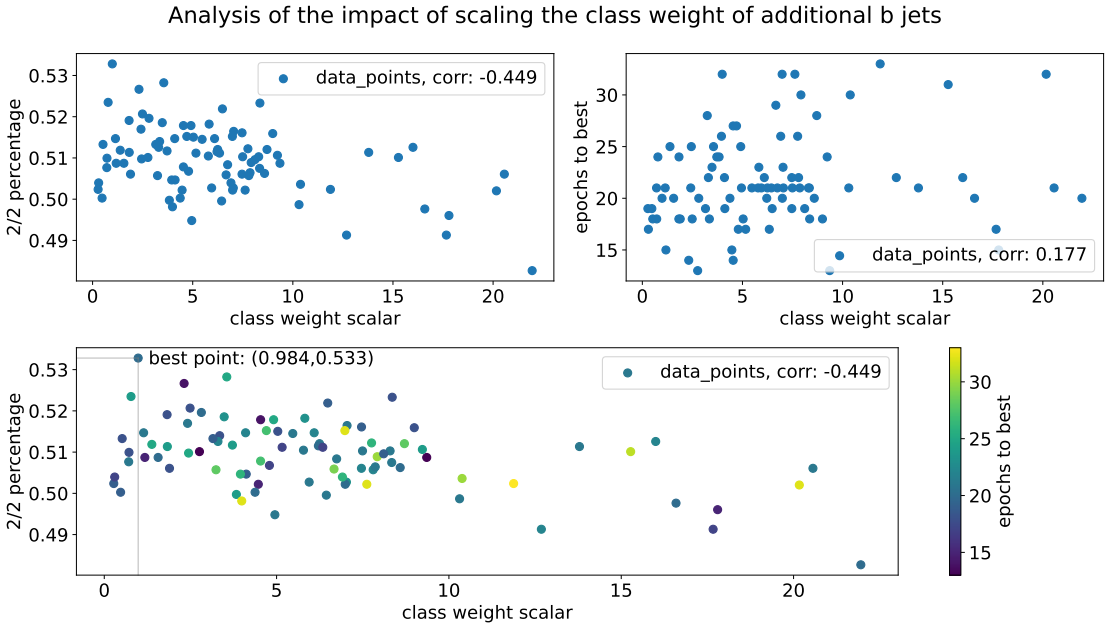

Figure 4.5: Result of the class weight analysis. As the class weight has a direct impact on the loss function, the loss cannot be used to compare network performance. Instead the 2/2 percentage is used as a performance measure for this analysis. All data points are generated using the multi-GGSNN design from [4], the plots show the result of 90 optimization iterations. The plot on the upper left compares the 2/2 percentage to the class weight scalar. A significant negative correlation is found towards very high class weights. The plot on the upper right compares the number of training epochs until the best 2/2 percentage is reached to the class weight scalar. The slight positive correlation shows that optimizations with a high class weight scalar tend to take longer until peak performance is reached. The lower plot combines both plots, this illustrates that there is very little correlation between the 2/2 percentage and the number of epochs until peak performance is reached.

section 2.4 within the node. As the network has to learn to scale down this essentially redundant information, the values are not included in the nodes for the final evaluation

$$\nu_i = (p_{T,i}, \phi_i, \eta_i, M_i, E_i, \text{btag}_i, q_{\text{J},i}). \tag{4.8}$$

5 Results

The study explored many different strategies of improving the 2/2 classification rate of additional b jets. Firstly, Bayesian optimization was implemented to optimize the GGSNN and every following network. Secondly, Hypergraphs and the networks to operate on them were introduced. Lastly, different methods of merging networks were proposed and tested.

5.1 Advanced GGSNN Studies

The previously best results of [4] on additional b jet classification were fully reproduced. For further improvements many design adjustments were considered as hyperparameters and optimized with Bayesian optimization. The parameters include, but are not limited to, the following list:

- The number of evaluations within the GGSNN.
- The number of GGSNN layers.
- The number and position of fully connected layers, as well as the number of nodes used within the fully connected layers and the option to use a bias.
- The activation function between different layers.
- The class weight scalar.

After over 200 optimization trials no significant performance increase was achieved. Different hyperparameter optimization strategies were tested such as using a Gaussian Process Regressor instead of the Tree-structured Parzen Estimator. Most changes of hyperparameters actually lead to a decrease in performance.

This lead to the conclusion that the hyperparameters of the GGSNN were already optimal. Therefore, no improvements to the GGSNN can be made by optimizing the tested hyperparameters. However, almost everything can be considered to be a hyperparameter. Thus, improvements could be made when using a new, previously unconsidered hyperparameter.

As no new ideas for hyperparameters were left, another approach was considered which is presented in the next section.

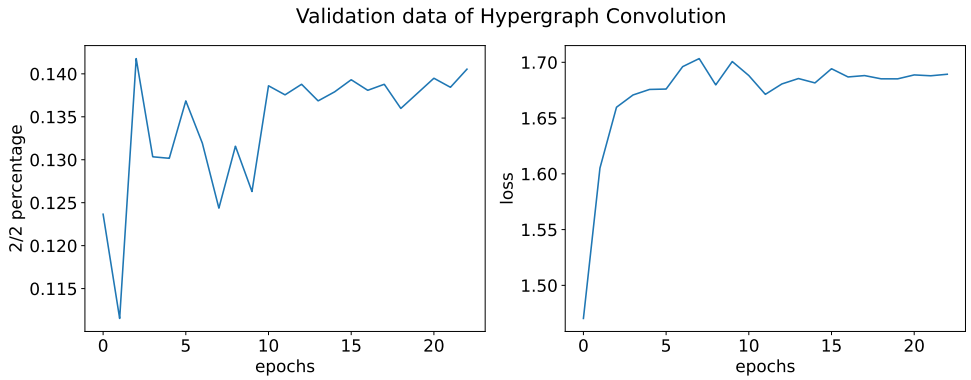

Figure 5.1: Performance of the best Hypergraph convolution layer on validation data during training. The left plot shows the evolution of the 2/2 performance over the epochs. The right plot shows the loss against the epochs. Peak performance is already reached in epoch 3 and training lacks stability.

5.2 Hypergraph Study

The essential idea of the hypergraph study was to implement the promising invariant mass of three particles using the Δm design from section 3.4.3. As the invariant mass could initially not be represented in Graphs, Hypergraphs were used. However, the only available layer for Hypergraphs in pytorch at that point was the Hypergraph Convolution.

5.2.1 Hypergraph Convolution

The first analysed Hypergraph layer is the Hypergraph Convolution from [14] which is explained in section 3.5.1. The validation data of the best performing network is shown in Figure 5.1. The network already reaches peak 2/2 performance in epoch 3 and the 2/2 performance generally does not improve in a stable manner. However, the training is still stable, which can be seen from the loss evolution. The difference in stability for the quantities stems from the chosen loss function and therefore also from the class weight scalar. The 2/2 performance training stability could possibly be improved by choosing a different loss function.

Hypergraph convolution networks with many different hyperparameter combinations were evaluated. The main considerations of the hyperparameters are:

- The Δm normalization, which just has options of using a linear normalization or none at all.
- The Δm design, section 3.4.3 presented the “lower” and “higher” option for the design.
- The threshold of the Δm design.

The network did not show any significant performance differences in the validation set for the used hyperparameters, as shown in Table 5.1. Because the expected processing of the two Δm designs is structurally different, the performance for both designs is expected to be different. However, the results are not significantly affected by this difference. This is an indication that the internal processing of the network is different from the expected behaviour. The expected processing is that the network heavily relies on the information of the hyperedges, because the number and weights of adjacent hyperedges of a node, make it, depending on the design, more or less likely for the node to stem from a top quark. As the internal calculation increases the feature values of each node in proportion to the

Hyperparameter			Validation data	
Δm normalization	Δm design	Δm threshold	1/2 percentage	2/2 percentage
Linear	lower	30.0 GeV	63.24%	14.18%
Linear	higher	56.0 GeV	62.94%	13.81%
None	lower	19.0 GeV	62.01%	13.06%

Table 5.1: Performance of the Hypergraph convolution layer with different hyperparameters.

0/2 percentage	1/2 percentage	2/2 percentage	TNR	TPR	Test Loss
25.77%	61.25%	12.99%	83.67%	43.61%	1.66

Table 5.2: Performance of the best hypergraph convolution network on test data. Shown are 0/2, 1/2, 2/2 percentage, the true negative rate, the true positive rate and the loss.

number and weights of adjacent hyperedges, the network could decide the labels on the magnitude of the feature values. The different Δm designs are, in this case, expected to perform differently because the most important information is inherently different.

The performance of the best Hypergraph Convolution network on the test set is shown in Table 5.2. Across the board the performance is much lower compared to any of the previously used ΔR designs. In only 13 % of the events both additional b jets were found which is much worse than the 2/2 classification rate of the GGSNN of 52.5 % but still better than a random classification which would roughly yield a 2/2 classification percentage of 4.76 %.

One reason for the bad performance is the very low number of parameters within the network, the network just does not have the capacity to store the information needed for better performance.

5.2.2 Gated Hypergraph Sequence Neural Network

The moderate results of the Hypergraph Convolution led to the development of the GHGSNN as described in section 3.5.2. The GHGSNN uses the same concept as the previously very performant GGSNN, while also being capable of handling hypergraph inputs like invariant mass combinations. However, the initial performance of the network was barely any better than from the Hypergraph Convolution.

After finding the right scales and threshold values the GHGSNN performance improved by a lot, which is shown in Table 5.3. The right scales, in this case, include the scaling function of the hyperedge weights as mentioned in section 3.4.3, for which the rising part of sine wave was used. Another important scaling factor is the class weight scalar. For this the optimized value was chosen from section 4.4.2. The Δm threshold value, as defined in section 3.4.3, determines which and how many hyperedges are constructed within the dataset. The optimal value differs for the chosen Δm design. For the higher design the optimal value resulting from the Bayesian optimization is 56 GeV. This means that every hyperedge between three nodes is constructed where the combined invariant mass has at least a 56 GeV difference to the top mass. This results in a mean number of hyperedges per event of 54.4. For the lower Δm design the optimal threshold was determined to be 44 GeV with a mean number of hyperedges per event of 24.2.

Especially the choice of the Δm design had a significant impact on the performance as the higher Δm design has much better results than the lower variant. Part of the reason is

Hyperparameter			Validation data	
class weight scalar	Δm design	Δm threshold	1/2 percentage	2/2 percentage
1.0	higher	56.0	48.95%	43.68%
4.4	higher	48.0	63.04%	35.76%
4.4	lower	44.0	63.82%	19.86%

Table 5.3: Performance of the Gated Hypergraph Sequence Neural Network with different hyperparameters. The optimal Δm normalization was always linear for the GHGSNN.

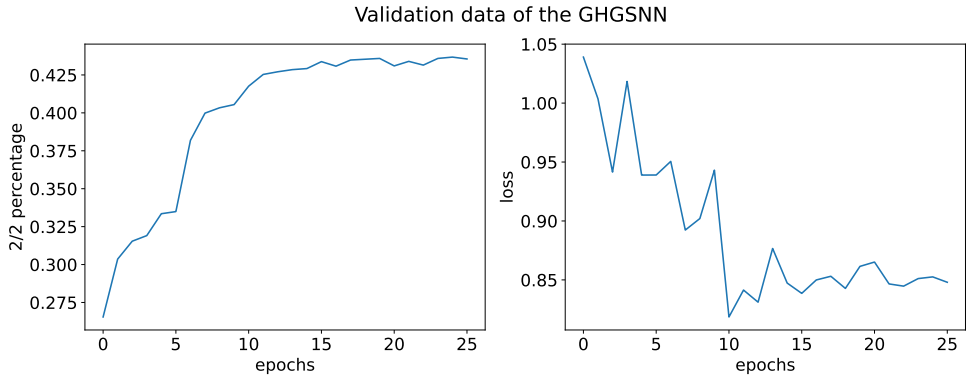

Figure 5.2: Performance validation data during training of the best Gated Hypergraph Sequence Neural Network. The 2/2 percentage (left) rises continuously and the loss (right) mostly decreases, the training is stable.

that the inner structure of the Δm design dictates that all feature values of unconnected nodes become zero after a single iteration. In the lower design the hope is that the network recognizes what these zeros mean because in this case it is unlikely for the particle to stem from a top quark. Therefore, this particles is a good candidate for an additional b jet. This however, assumes that the network transforms the zeros from the layer, to a high value in the final output. The higher design generates the low values for particles that are likely to be a top quark and high values for the other nodes. Thus, the higher design involves one calculation step less which simplifies the learning process.

The training of the GHGSNN is very stable, as illustrated in Figure 5.2. The peak 2/2 performance is reached at the final epoch and the loss decreases mostly.

The final test results of the GHGSNN in Table 5.4 show the great performance increase in comparison to the previous Hypergraph network. The 2/2 classification rate is very close to the best performance of the DNN used previously in [3]. One reason for the increase in performance is the significant increase in the number of parameters the model can use. Additionally, this model benefited greatly from the adjustment of the class weight scalar from section 4.4.2.

0/2 percentage	1/2 percentage	2/2 percentage	TNR	TPR	Test Loss
8.25%	45.95%	45.80%	90.94%	68.77%	0.82

Table 5.4: Performance of the best Gated Hypergraph Sequence Neural Network on test data. Shown are 0/2, 1/2, 2/2 percentage, the true negative rate, the true positive rate and the loss.

0/2 percentage	1/2 percentage	2/2 percentage	TNR	TPR	Test Loss
8.26%	37.37%	54.35%	92.2%	73.1%	0.72

Table 5.5: Performance of the best network using the combination design on test data. Shown are 0/2, 1/2, 2/2 percentage, the true negative rate, the true positive rate and the loss.

5.3 Merging Study

5.3.1 Combination Design

The initial combination design, introduced in section 3.6.2, actually performed worse than the pretrained GGSNN alone. The exact reason for this is speculative, as the interpretation of weights within the layer is unfeasible. The pretrained GGSNN was trained on 6 input features per node. Due to the internal mechanisms of the network it is possible to pass a higher or lower number of features to the network. Therefore, no adjustments have to be made when appending the result of the GHGSNN to the nodes. This previously empty feature still has to be learned from the GGSNN. However, changing parameters from the current local optimum to the proposed local optimum is not possible, as it requires parameter changes that temporarily worsen the training loss.

The solution used to solve this challenge is to pretrain the GGSNN on 7 features per node. The 7th feature in this case is a smudged version of the actual label. The smudged labels are constructed by randomly subtracting values between 0 and 1 from the true labels, which are represented by ones, and randomly adding values between 0 and 1 to the false labels, which are represented by zeros. The true positive and true negative rate of the smudged labels can be adjusting by choosing specific PDFs for the random samplings. By using this training method the network already uses the 7th feature and the interpretation when actually using the GHGSNN output as the 7th feature does not change. Another, possibly simpler, solution is to append all Hypergraph classifications to the data set and use an untrained GGSNN for the training. However, from a programming perspective it is much simpler to evaluate the Hypergraph during runtime. This also has the additional benefit of allowing for further training of the GGSNN.

Looking at the evolution of validation data in Figure 5.3 reveals that there are major sources of instability during training. One source definitely is the large number of parameters, as the model uses two Sequence Neural Networks which each contain many parameters.

The test results of the combination design in Table 5.5 show great improvements compared to the previous networks with a 2/2 classification rate of 54.35 % on the test set. Compared to the final 2/2 classification rate from [4] of 53.49 % this is a performance increase of 0.86 %.

5.3.2 Addition Design

The final addition design, as defined in section 3.6.1, used the GGSNN and the GHGSNN as the merged networks. The training of the network started with a very high 2/2 classification rate, as both networks were already pretrained on the same training data. The training of the full design has major instabilities as the loss only rises after epoch 10 until the early stopping algorithm stops the training in epoch 34. As in the combination design a probable reason for the instability is the huge number of learnable parameters and a slight mismatch between the loss function and the actual optimization goal, the 2/2 classification rate.

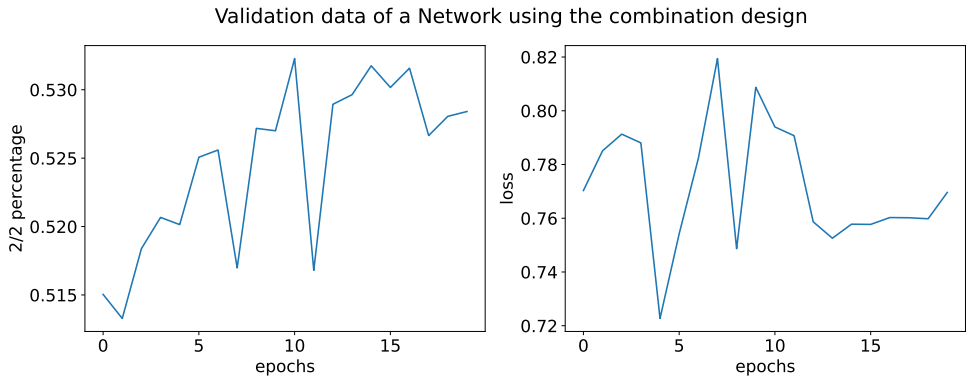

Figure 5.3: Performance of the best network using the combination design on validation data during training. The peak performance of is already reached in epoch 9 and the training has major instabilities.

0/2 percentage	1/2 percentage	2/2 percentage	TNR	TPR	Test Loss
6.24%	35.69%	58.07%	93.03%	75.91%	0.68

Table 5.6: Performance of the best network using the addition design. Shown are 0/2, 1/2, 2/2 percentage, the true negative rate, the true positive rate and the loss.

The test result of the best performing addition design in Table 5.6 is considered to be an outlier, as the 2/2 classification rate on the validation set is only 54.16%. Other than that the results show a great improvement compared to any other tested network.

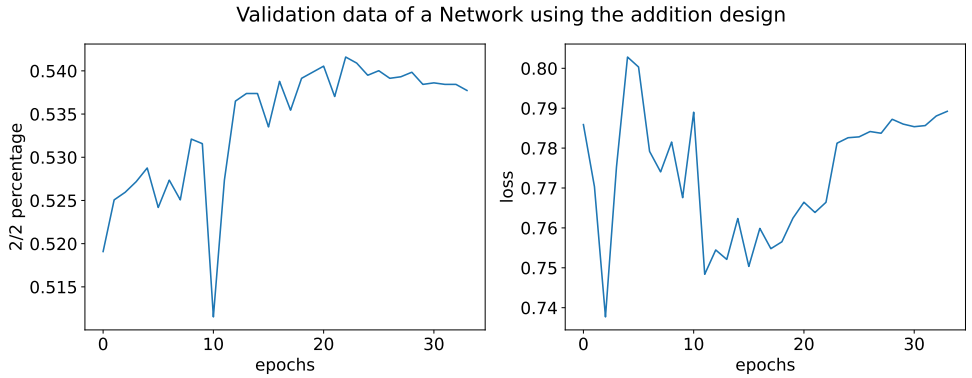

Figure 5.4: Performance validation data during training of the best network using the addition design. The 2/2 classification rate (left) rises as expected except for the large drop at epoch number 10. The training of the loss function (right) is very unstable, as the loss actually only becomes larger after about epoch 10. As the loss only rises, the early stopper stops the training at epoch 34.

6 Summary and Outlook

The primary objective of the study has always been the $t\bar{t} + b\bar{b}$ process with the underlying goal of making a contribution to further the understanding of the Higgs Boson and QCD in proton-proton collisions at the LHC. The challenge of comprehending the $t\bar{t}+b\bar{b}$ process lies within the identification of additional b jets which are not a result of top decays. Previous studies on this subject have already been done. The most recent study [4] implemented a GNN design and found the best performing layer, the GGSNN.

The key work of the study was the practical implementation of the improvement measures for additional b jet classification. Beginning with the implementation of Bayesian optimization that set the foundation for later improvements. The initial goal of implementing Bayesian optimization was to optimize the GGSNN performance. This led to expanding the idea of hyperparameters and creating new ones. However this did not have a direct impact on the performance of the GGSNN. As a consequence, a deeper look into the base data was taken.

With the goal of optimizing the GGSNN, the Graph data structure was adjusted. To reduce the effect of oversmoothing, Graph edges were disconnected based in the ΔR. This also did not lead to an improvement of the 2/2 classification rate, the rate with which both additional b jets are correctly assigned.

Instead of further trying to optimize the GGSNN, a new idea was necessary: the previously very performant invariant mass difference to the top quark had no way of being implemented within a Graph structure. Thus, a new data structure was proposed: the hypergraph. The hypergraph by itself did not lead to the intended improvement of the 2/2 classification. Actually, it performed much worse than expected. The first idea was to extend the functionality of the previously best performing GGSNN to allow for hypergraph inputs. However, this did not only lead to the construction of the GHGSNN layer but revealed that any Graph layer can be used to evaluate Hypergraph input data by applying the transformation developed in this thesis. The GHGSNN in combination with the Bayesian optimization led to a much better 2/2 classification rate, which has previously not been achieved without directly using ΔR values.

Nonetheless, even the best performing GHGSNN could not outperform the GGSNN and the GGSNN itself could not be improved. Taking a new perspective allowed to see both designs as part of a new design combining any number of models. The introduction of the combination and addition design allowed to finally increase the previously best 2/2

classification rate by 0.86 %. This is a big improvement considering the many previous studies that already went into exploration of this topic.

The next possible steps for further improvements include the correction of the loss function. Because even with an adapted class weight scalar, the loss function in this study is not as strongly correlated to the 2/2 classification rate as it should be. As the network performance is only optimized with regard to the loss function, improvements to the loss function directly benefit the training and the stability of the training. Other than that the performance can be improved by finding more different methods of classification and merging them with the previously obtained networks.

Bibliography

[1] CMS Collaboration. "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC". In: *Physics Letters B* 716.1 (2012), pp. 30–61. DOI: https://doi.org/10.1016/j.physletb.2012.08.021. URL: https://www.sciencedirect.com/science/article/pii/S0370269312008581.

[2] ATLAS Collaboration. "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC". In: *Physics Letters B* 716.1 (2012), pp. 1–29. ISSN: 0370-2693. DOI: https://doi.org/10.1016/j.physletb.2012.08.020. URL: https://www.sciencedirect.com/science/article/pii/S037026931200857X.

[3] E. Pfeffer. "Studies on tt+bb production at the CMS experiment". ETP-KA/2021-01. MA thesis. Karlsruhe Institute of Technology (KIT), 2021.

[4] T. Halenke. "Studien zu Graph Neural Networks in tt + bb- Prozessen am CMS-Experiment". BA thesis. ETP-BACHELOR-KA/2021-13. Karlsruhe Institute of Technology (KIT), 2021.

[5] B. Povh. *Teilchen und Kerne : Eine Einführung in die physikalischen Konzepte*. Ed. by K. Rith et al. Berlin, Heidelberg, 2014. URL: http://swbplus.bsz-bw.de/bsz39819646xcov.jpghttps://doi.org/10.1007/978-3-642-37822-5.

[6] *Representation of the Standard Model of particle physics*. https://en.wikipedia.org/wiki/Standard_Model. Accessed: 2022-13-02.

[7] *Image of the CERN complex*. https://cds.cern.ch/record/2684277/files/CCC-v2019-final-white.png?subformat=icon-1440. Accessed: 2022-13-02.

[8] *Transverse slice of the CMS experiment*. https://cds.cern.ch/record/2205172l. Accessed: 2022-13-02.

[9] "Performance of the DeepJet b tagging algorithm using 41.9/fb of data from proton-proton collisions at 13TeV with Phase 1 CMS detector". In: (Nov. 2018). URL: https://cds.cern.ch/record/2646773.

[10] E. Fiesler. "Neural Network Classification and Formalization". In: *Computer Standards Interfaces* 16 (1994), pp. 231–239.

[11] K. He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[12] R. J. Trudeau. *Introduction to graph theory*. Dover ed. Dover books on advanced mathematics. New York, NY: Dover Publ., 1993, pp. 19–21. ISBN: 0486678709.

[13] T. N. Kipf and M. Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG].

[14] S. Bai, F. Zhang, and P. H. S. Torr. *Hypergraph Convolution and Hypergraph Attention*. 2020. arXiv: 1901.08150 [cs.LG].

[15] J. Bergstra, D. Yamins, and D. Cox. "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by S. Dasgupta and D. McAllester. Vol. 28. Proceedings of Machine Learning Research 1. Atlanta, Georgia, USA: PMLR, 2013, pp. 115–123.

[16] J. Bergstra et al. "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor et al. Vol. 24. Curran Associates, Inc., 2011. URL: https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

Acronyms

CDF cumulative density function. 29

CMS Compact Muon Solenoid. 4, 6

DNN Deep Neural Network. 1, 12, 25, 38

EI expected improvement. 28, 29

GGSNN Gated Graph Sequence Neural Network. 17, 22, 23, 31–33, 35, 37, 39, 43

GHGSNN Gated Hypergraph Sequence Neural Network. 22, 23, 31, 37–39, 43

GNN Graph Neural Network. 1, 15–17, 25, 32, 43

GP Gaussian Process. 28, 29, 35

LHC Large Hadron Collider. 4, 43

MET missing transverse energy. 6

MPNN Message Passing Neural Network. 16, 17, 20, 21

MSE Mean Squared Error. 10

NN Neural Network. 9, 10, 12, 13, 15–17, 19–22, 25, 29, 30, 39, 49

PDF probability density function. 29, 39

SM Standard Model. 1, 3, 4

SMBO Sequential Model-based Global Optimization. 25, 26, 28

TNR True negative rate. 37–40

TPE Tree-structured Parzen Estimator. 26, 28, 29, 31, 35

TPR True positive rate. 37–40

Appendix

A Proofs and Algorithms

A.1 Reason for the need of an activation function

Proof. Assume a Neural Network with a single hidden layer, the activation function σ and no bias:

$$\mathbf{x} \in \mathbb{R}^{l^0}, \quad \mathbf{h} \in \mathbb{R}^{l^1}, \quad \mathbf{y} \in \mathbb{R}^{l^2}, \quad \theta^{\mathbf{1}} \in \mathbb{R}^{l^0 \times l^1}, \quad \theta^{\mathbf{2}} \in \mathbb{R}^{l^1 \times l^2} \tag{6.1}$$

Thus the feedforward calculation becomes,

$$\mathbf{h} = \sigma\left(\mathbf{x} \cdot \theta^{\mathbf{1}}\right) \quad \text{and} \quad \mathbf{y} = \sigma\left(\mathbf{h} \cdot \theta^{\mathbf{2}}\right) \tag{6.2}$$

hence,

$$\mathbf{y} = \sigma\left(\mathbf{x} \cdot \theta^{\mathbf{1}}\right) \cdot \theta^{\mathbf{2}} \tag{6.3}$$

If the activation function is linear, the output $\mathbf{y}$ simplifies to

$$\mathbf{y} = \mathbf{x} \cdot \theta^{\mathbf{1}} \cdot \theta^{\mathbf{2}} = \mathbf{x} \cdot \theta^{*} \tag{6.4}$$

which is the same as a linear model i.e. a network without hidden layers. □

A.2 The gradient is the direction of steepest ascent

Proof. Let $\mathbf{u}$ be a unit vector and $f(\mathbf{x})$ a multivariate, differentiable function. The directional derivative is:

$$\begin{aligned} \text{grad}(f(\mathbf{x})) \cdot \mathbf{u} &= |\text{grad}(f(\mathbf{x}))|\,|\mathbf{u}| \cos(\theta) && (6.5) \\ &= |\text{grad}(f(\mathbf{x}))| \cos(\theta) && (6.6) \end{aligned}$$

This expression is maximized if $\cos(\theta) = 1$, meaning that $\mathbf{u}$ has to point in the same direction as $\text{grad}(f(\mathbf{x}))$ for the steepest ascend, thus $\text{grad}(f(\mathbf{x}))$ is the direction of steepest ascend. □

A.3 Graph layers can be used to evaluate Hypergraphs

Assertion: A weighted adjacency matrix $\mathbf{A}$ can be constructed from the incidence matrix $\mathbf{H}$ and the weights $\mathbf{W}$ of a hypergraph.

An algorithm is suggested for the conversion of Graphs to Hypergraphs. There is not just one way to implement such a conversion. This specific conversion tries to mimic the functionality of the Hypergraph as close as possible. Let $\mathcal{H}$ be a Hypergraph with the vertex set $V = \{1, 2, 3, 4, 5, 6\}$, hyperedge set $\mathbf{E} = \{\{1, 2, 3\}, \{1, 2, 4\}, \{3, 4, 5\}\}$ and the hyperedge weights $\mathbf{W} = \operatorname{diag}(0.2,\, 0.8,\, 0.4)$, as represented in Figure 3.8.

1. The hyperedges connect every vertex within themselves, therefore, when converting this to a Graph, every connection of pairs of the nodes from the hyperedges must be made. These connections have the same weights as the hyperedge they stem from.

   When applying this step, the first hyperedge of the Hypergraph $\{1, 2, 3\}$, turns into $\{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$ with the edge weights $(0.2, 0.2, 0.2)$.

2. As the hyperedges do not have directions, the inverse connections must also be made. They have the same edge weights as the original hyperedge.

   The edges $\{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$ are inverted to $\{\{2, 1\}, \{3, 1\}, \{3, 2\}\}$ and retain the same edge weights $(0.2, 0.2, 0.2)$.

3. Every hyperedge also connects the vertices with themselves, therefore self loops for the respective vertices have to be added.

   The first hyperedge contains the vertices 1, 2 and 3. Therefore the self loops $[\{1, 1\}, \{2, 2\}, \{3, 3\}]$ are constructed with the edge weights $(0.2, 0.2, 0.2)$.

4. The results of step (1) to (3) are appended into an ordered edge vector $\mathbf{c}$ and a vector with the corresponding weights $\mathbf{w_c}$.

5. The vector $\mathbf{c}$ can contain duplicate entries. The weights of duplicate entries are summed to still retain all weights. Then duplicate entries are removed in $\mathbf{c}$ until all only one of the duplicate entries is left. As order matters, it is crucial that when removing the i-th component in $\mathbf{c}$, the i-th component of $\mathbf{c}_w$ is also removed. The resulting vectors are called $\mathbf{e}$ and $\mathbf{e}_w$.

   Part of $\mathbf{c}$ for the Hypergraph $\mathcal{H}$ is $(\{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2\})$ with the edge weights $(0.2, 0.2, 0.2, 0.8)$, applying the transformation yields $(\{1, 2\}, \{1, 3\}, \{2, 3\})$ with the edge weights $(1, 0.2, 0.2)$.

6. The adjacency matrix $\mathbf{A}$ is initialized as a matrix of zeros of the dimension $v \times v$. Now, the i-th entry of $\mathbf{e}$ contains the row and column of the matrix $\mathbf{A}$ where the value is set to the i-th entry of $\mathbf{e}_w$.

   Assuming $\mathbf{e} = (\{1, 2\}, \{2, 3\}, \{2, 1\}, \{3, 2\}, \{1, 1\}, \{2, 2\}, \{3, 3\})$, $v = 4$ and $\mathbf{e}_w = (0.4, 0.7, 0.4, 0.7, 0.4, 1.3, 0.7)$ yields the weighted adjacency matrix $\mathbf{A}$, which is exactly the same as the multiplication of $\mathbf{HWH}^\intercal$:

$$\mathbf{A} = \begin{pmatrix} 0.4 & 0.4 & 0 & 0 \\ 0.4 & 1.3 & 0.7 & 0 \\ 0 & 0.7 & 0.7 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{6.7}$$