CHAPTER 6

Neural Network Setup

Before the neural networks introduced in the previous chapter can be utilized to analyze the datasets described in chapter 4, the data needs to be prepared. The preprocessing involves the definition of a dataset splitting strategy to properly treat of overfitting and the selection of representative input features for both FNNs and RNNs. Moreover, the transformation of the input features and renormalization of the Monte Carlo event weights.

For the practical implementation of the neural networks, the API Keras with Tensorflow as a backend is used throughout this thesis.

6.1 Splitting Strategy

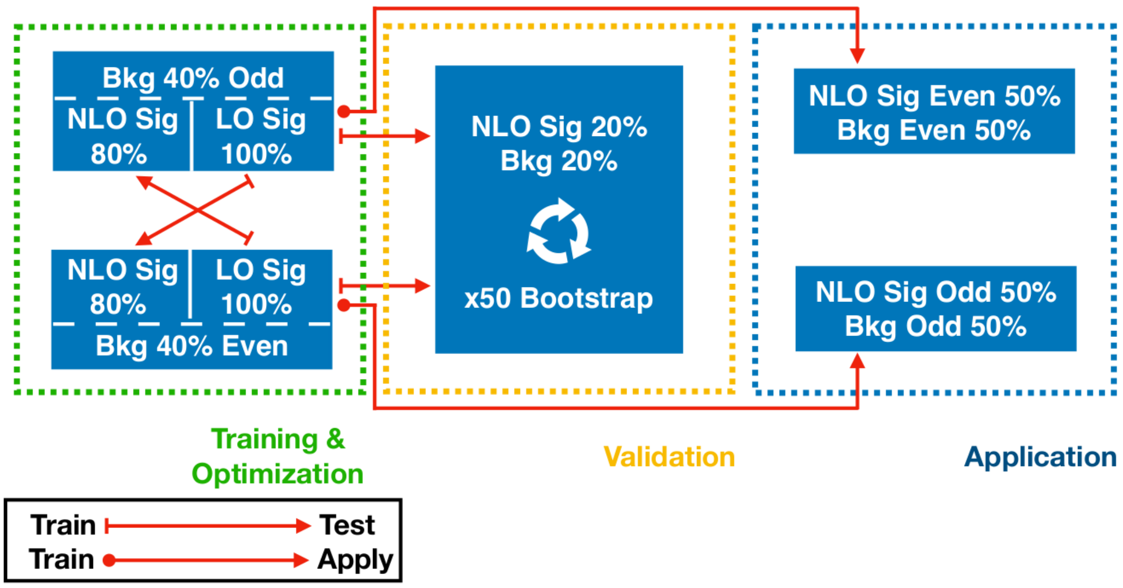

Figure 6.1: Splitting Strategy that is applied to ensure an unbiased estimation of both overtraining and performance [].

The two essential quantities of every machine learning model are the performance measure and its overtraining. As discussed in Section **??** the performance measure of choice is the AUC Γ. It can be evaluated on a single dataset. Overtraining, however, measures how well a model trained on one dataset generalizes to a second dataset. Thus, at least two datasets are needed to evaluate the overtraining of a model. An important side note is that the two datasets need to be orthogonal i.e. they don't have an event in common. The original dataset, therefore, is split into two independent datasets called the training and the testing dataset. As the name suggests, the training of the neural network is performed on the training set. After all trainable parameters are fixed the model is *tested* on the testing set. The overtraining O_v as a function of the two obtained AUCs Γ_{test} and Γ_{train} can be expressed as

$$O_v = 1 - \frac{\Gamma_{\text{test}}}{\Gamma_{\text{train}}} \tag{6.1}$$

The neural network with the highest AUC which has a reasonable O_v typically under 5% is considered the best model. The detailed strategy to discover the best model is described in chapter **??**. The optimization relies on training numerous models with different hyperparameters and architectures. However, since the same testing dataset is exploited multiple times, this approach introduces a bias into the measurement of the overfitting. To compensate for this, a third orthogonal dataset called the *validation set* is introduced. Similar to the testing set, it is speared during training and additionally also remains unused during the optimization. Only after all hyperparameters are fixed the $\Gamma_{\text{validation}}$ is calculated. Thus, allowing for unbiased estimation of O_v. Equation 6.1 its to be adjusted for the final evaluation of the overtraining by replacing Γ_{test} with Γvalidation. The error of the obtained overfitting is then evaluated on the validation set utilizing the method of *Bootstrapping*[???].
The background datasets available for the four top analyses are split into 20% validation set and 80% for testing and training. The second dataset is moreover subdivided into two equally large datasets. The orthogonality is ensured by splitting on the event index (even or odd). This procedure is applied for each background dataset individually such that the three final datasets have a proper number of events for each process. The $t\bar{t}t\bar{t}$ signal set, since it holds events calculated both at leading order and next-to-leading order, is treated differently. All LO events are used for training to avoid the negative event weights problem which will be discussed in section 6.3. The NLO events are split into 80% testing dataset and 20% for validation dataset. The complete splitting strategy is summarized in Figure 6.1. The application step indicated corresponds to further studies of the $t\bar{t}t\bar{t}$ process which are performed on the NLO events. As the red arrows imply, two neural networks with the same hyperparameters are trained. The first one having the even dataset as the training dataset and the odd dataset as the testing dataset (yielding Γ_{even}). The definition of the testing and training dataset is then interchanged for the second neural network (yielding Γ_{odd}). The combined AUC Γ_{total}, therefore, can be determined as

$$\Gamma_{\text{total}} = \frac{\Gamma_{\text{even}} + \Gamma_{\text{odd}}}{2} \tag{6.2}$$

This scheme ensures that the concluding performance measure Γ_{total} is independent of the concrete choice of the training and testing datasets.

6.2 Feature Selection

The ingredient of the neural network that impacts the achievable AUC the most is the selection of the input features. To pick representative features that discriminate the signal events from the background events is pivotal. Selecting too many or irrelevant features can slow down the training and even decrease the performance. The usage of prior information about the problem such as the different physical characteristics of the processes at hand can be used as a guideline. In the case of $t\bar{t}t\bar{t}$ process, theoretically the number of quarks and b quarks should be higher for signal events (cf. Section **??**). As can be seen in Figures 6.3 and 6.2 this difference is reflected in the number of jets (n_j) and the sum of the continuous score derived by the MV2C10 b-tagging algorithm ($\sum_j$MV2C10). In practice, resolution and detector effects can smear out expected distinctions. Therefore, it's necessary to validate which features increase Γ_{total}. As the first figure of merit, the ranking of features provided by a boosted decision tree[???] is adopted. The ranking is derived by counting how often a feature is used to split decision tree nodes, and by weighting each split occurrence by the separation gain squared it has achieved [tmva note] [???? 32]. To ensure that the highest-ranking features are also well suited for neural networks a study using an iterative removal algorithm[1] is performed. The features selected in this manner are summarized in table 6.1 together with the respective BDT ranking scores. As previously mentioned N_j and $\sum_j$MV2C10 are two of the highest-scoring features alongside the angular distance features $\Delta R(x, y)$ which are calculated according to

$$\Delta R(x, y) = \sqrt{(\eta_x - \eta_y)^2 + (\phi_x - \phi_y)^2} \tag{6.3}$$

where x and y are two particle types. An example of an angular distance distribution is shown in Figure 6.4. $\min(\Delta R(b, b))$ refers to the minimum angular distance between any b-jet pair.

The approach chosen to select features for the RNN is different and loosely inspired by [???]. The four momenta and the decaying angles of the reconstructed objects in the final state (cf. Chapter 3) are used directly as input features. In the case where multiple particles of the same type are present in the final state of the event, the particles are sorted according to their p_T. Thus, an input feature such as the pseudorapidity of any jet ϕ_j is a p_T sorted sequence of length n_j where n_j is the number of jets observed in the event. In practice, all sequences in the event have to have the same length. This is accomplished by padding sequences that do not have the desired length with zeros. In addition to the features described above some high-level features such as $\sum_j$MV2C10, are selected. The combination of the two types of features was shown to increase performance in several applications [??? 13-25]. The input features selected are ϕ, η, p_T of electron, muons and jets as well as $E_\mathrm{T}^\mathrm{miss}$, the N_j and $\sum_j$MV2C10. All features for FNNs and RNNs are shown in detail in the Appendix ???.
To have some kind of feature selection a small l1 regularization term is applied to the RNN. As briefly mentioned in Section 5.4 this regularization tends to eliminate connections to the least important features. Thus an automatic feature selection is provided which suppresses redundant information.

[1] The algorithm aims to find the best k features out of a set of m features. This is done by training m neural networks with the same hyperparameters in the first step where each has a different feature removed from the feature set. After the evaluation, the feature with the lowest AUC is removed from the initial set and the process is repeated for $m - 1$ features.

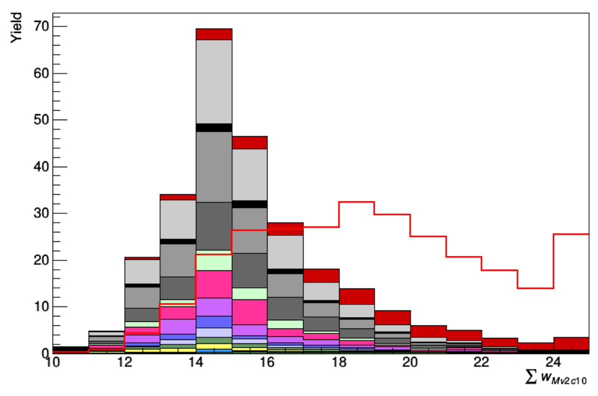

Figure 6.2: Sum of the continuous MV2c10 b-tagging score

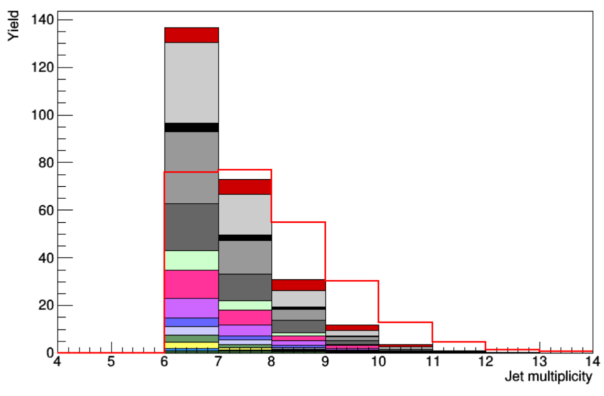

Figure 6.3: Jet multiplicity

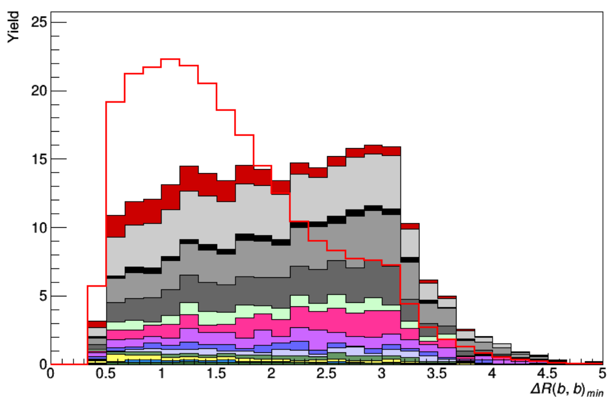

Figure 6.4: minimum angular distance between any b-jet pair

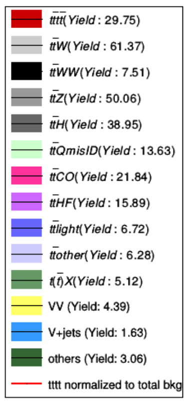

Figure 6.5: Legend

Feature	Definition	BDT ranking score
$\sum_j$MV2C10	Sum of continuous mv2c10 score over all jets	0.325
N_j	Jet multiplicity	0.088
H_T^0	Scalar sum of all lepton and jet p_T's excluding the leading jet	0.062
E_T^{miss}	missing transverse energy	0.026
$p_T^{j,0}$	p_T of the leading jet	0.004
$p_T^{j,1}$	p_T of the sub-leading jet	0.012
$p_T^{j,5}$	p_T of the fifth highest jet	0.084
$p_T^{b,0}$	p_T of the leading b-jet	0.033
$p_T^{\ell,0}$	p_T of the leading lepton	0.040
$p_T^{\ell,1}$	p_T of the sub-leading lepton	0.030
$\max(\Delta R(l,b))$	Maximum angular distance between any lepton and b-jet	0.018
$\max(\Delta R(l,l))$	Maximum angular distance between any lepton pair	0.024
$\min(\Delta R(l,l))$	Minimum angular distance between any lepton pair	0.017
$\min(\Delta R(b,b))$	Minimum angular distance between any b-jet pair	0.084
$\min(\Delta R(l,b))$	Minimum angular distance between any lepton and b-jet	0.030
$\min(\Delta R(l,j))$	Minimum angular distance between any lepton and jet	0.016
$\sum_l \Delta R(l,l)$	Sum of all angular distance between any lepton pair	0.030

Table 6.1: The 19 selected features for the FNNs with there respective BDT ranking score

6.3 Feature Transformation and Event Weight Treatment

As discussed in Section **??** the backpropagation algorithm calculates the gradients of the loss function for each weight starting from the output layer and working backward to the input layer. Each gradient of the current step, therefore, depends on the gradients derived in the previous step. Thus, when the selected input features have small numerical values the gradients start to vanish[???], leaving the weights virtually unchanged. On the other hand, if the features have big numerical values the gradients can get very large, leading to the *exploding gradient problem*[???]. Both cases can hinder the convergence of the neural network. One way of dealing with this problem is to transform all input features so that they will be normally distributed i.e. 0 mean and a variance of 1, before feeding them to the neural network. Additionally, the SELU activation function can be utilized to normalize the neuron outputs at each layer. A similar problem comes up due to the Monte Carlo event weights, discussed in Chapter **??**, which can turn out to be very small or very large once again slowing down convergence. To avoid this, a renormalization is applied such that the sum of all weighted signal training events equals the sum of all weights of the background training events, following the approach described in [tvma guide]. Furthermore, all negative events weights[2] are set to 0 during the training since ordinary interpretation as probabilities can not be applied.

[2] Negative weights can arise from the calculation of loop diagrams in next-to-leading order processes for more details [Computational challenges for MC event generation]