Domain Adversarial Learning to Reduce Training Bias in ttH(bb) Search at ATLAS

Ilyas Fatkhullin
Moscow Institute of Physics and Technology, Russia
Supervisors: Paul Glaysher, Judith Katzy

September 5, 2019

Abstract

The application of Neural Networks with a gradient reversal layer to the ttH(bb) search at ATLAS is studied. Feature distributions of the background sample slightly vary from one simulations to another. Therefore the classification model may show a training bias towards a particular simulation. Adversarial domain adaptation techniques are utilized to reduce the training bias and obtain better classification performance.

Keywords: Adversarial Training, Domain Adaptation, LHC, Higgs

Contents

1. Introduction **3**

2. Theory on domain adaptation **3**
2.1. Domain Divergence and Generalization Bound 4
2.2. Adversarial Domain Adaptation Neural Network 6

3. Data **7**

4. Details of DANN Implementation and Optimization **9**
4.1. Implementation 9
4.2. Optimization 9

5. Results **10**
5.1. Response of NN 11
5.2. Significance calculation and ROC curves 11
5.3. Looking at training curves 14

6. Conclusion **14**

7. Acknowledgements **15**

A. Variable list for NN training **16**

1. Introduction

Machine Learning (Multivariate Analysis) techniques are widely used in event selection in High Energy Physics (HEP) [1; 2]. Nowadays there is a special interest in the application of Neural Networks (NN) for this task due to the increase in dimensionality of collected data. However, when training NN classifier on simulated data one should be aware of its generalization capabilities since simulations may not describe physics precisely and there is a risk of catching wrong patterns of a particular simulation. Development of Domain Adaptation Theory [3] allows to make weaker assumptions on generated data and potentially reduce the training bias and achieve better generalization capability of the classification method. For example, one may have labelled simulated data and want to train a classifier that performs well on real data. Domain adaptation tries to achieve this goal by using additionally unlabelled real data to provide a successful transfer.
The detailed study of the Higgs boson including its rare production modes is a primary activity of the LHC experiments. We consider the search for the ttH production mode, in which the Higgs boson is produced together with a top anti-top quark pair Fig. 1 (left) at the ATLAS experiment. A measurement of this process allows to test a fundamental property of the Standard Model, namely the Yukawa coupling between the Higgs boson and the top quark. The most probable decay of the Higgs boson is to two bottom quarks (measured as b-jets) and the signal needs to be distinguished from a large background of tt+b-jets Fig. 1 (right). The expected signal and background signatures are very similar and significant separation is achieved through a neural network or Boosted Decision Tree classifier, trained on simulated, labelled Monte Carlo data. The current results for the ttH ($H \rightarrow bb$) search [1] was limited by the modelling uncertainty of the background, calculated as the discrepancy of the classifier response to different Monte Carlo generators. This uncertainty is equivalent to a training bias towards a specific Monte Carlo generator, which we show to mitigate though adversarial domain adaptation.
In this paper we study adversarial Domain Adaptation Neural Network (DANN) with Gradient Reversal Layer [4; 5] for analyzing simulated data. This paper is organized as follows: in Section 2 we introduce necessary notations and key theoretical results (mainly from the works of Ben-David et al. [3; 6]) to support the usage of DANN architecture. In Section 3 and 4 we provide a description of our dataset construction, implementation and experiments. Finally, we present the overall results in Section 5. Related work of A. Ryzhikov and A. Ustyuzhanin [7] also addresses the problem of training on simulated data for HEP.

2. Theory on domain adaptation

The optimization and evaluation of supervised learning methods generally assumes that the training and validation instances are both sampled from the same family of probability density functions. For the example presented this is not the case, as the wish is to train on the simulated data and apply the classifier to real collision data. In Fig. 4 and 5 examples of the training variables are shown, from which one can see that difference in

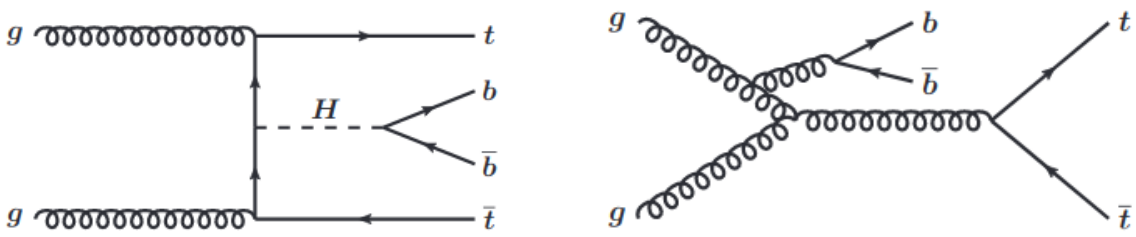

Figure 1: Feynman diagrams for ttH-signal (left) and tt+bb-background (right)

the alternative background models is of equivalent size as the difference between signal and background models. Theory of transfer learning is designed to mitigate this problem. Let us introduce the notations relevant to this study where alternative simulation models are treated as different data domains.

Notation. We consider a binary classification task, where $\mathcal{X}$ denotes an input space and $\mathcal{Y} = \{0, 1\}$ a set of classes. Let S and T be two datasets obtained from distributions $\mathcal{D}_S$ and $\mathcal{D}_T$ correspondingly. Both of these distributions are defined on $\mathcal{X} \times \mathcal{Y}$ and are referred to as *source* ($\mathcal{D}_S$) and *target* ($\mathcal{D}_T$) domains. Let us also denote $\mathcal{H} = \{h | \mathcal{X} \to \mathcal{Y}\}$ as a *hypothesis space*, which represents a subspace of all possible maps from $\mathcal{X}$ to $\mathcal{Y}$. We define the risk $\epsilon_{\mathcal{D}}(h)$ of a hypothesis h under distribution $\mathcal{D}$ as $\epsilon_{\mathcal{D}}(h) = \mathbf{P}_{(\mathbf{x},y)\sim\mathcal{D}}(h(\mathbf{x}) \neq y)$. We refer to the corresponding empirical risk as $\hat{\epsilon}_{\mathcal{D}}(h)$, which is calculated according to classical definition of probability. Further we also denote corresponding definitions of empirical quantities by adding a hat on top. The main goal is to build a classifier with a low risk under target distribution $\mathcal{D}_T$ while having no information about the labels on $\mathcal{D}_T$. Note that in our study we work with simulated data (see Section 3) only and none of the simulations can be given a preference to be called *target*, therefore after Section 3 we stick to different notation and denote two datasets Source 1 (S_1) and Source 2 (S_2). Most recent works on domain adaptation introduce some similarity measures between two domain distributions and try to minimize it along with usual classification loss. Successful examples of similarity measures are Maximum Mean Discrepancy (MMD) [8], High Order Moments [9], Wasserstein Distance [10] and $\mathcal{H}$-divergence [4]. Further we follow the latter approach, define the notion of $\mathcal{H}$-divergence and present theoretical results by Ben-David et al. [6] to justify the idea of DANN architecture.

2.1. Domain Divergence and Generalization Bound

Definition 1 *Given two domain distibutions $\mathcal{D}_S$, $\mathcal{D}_T$ over $\mathcal{X} \times \mathcal{Y}$ and hypothesis space $\mathcal{H}$. $\mathcal{H}$-divergence between $\mathcal{D}_S$ and $\mathcal{D}_T$ is defined as*

$$d_{\mathcal{H}}(\mathcal{D}_S, \mathcal{D}_T) = 2 \sup_{h \in \mathcal{H}} |\mathbf{P}_{(\mathbf{x},y)\sim\mathcal{D}_S}(h(\mathbf{x}) = 1) - \mathbf{P}_{(\mathbf{x},y)\sim\mathcal{D}_T}(h(\mathbf{x}) = 1)| \tag{1}$$

That is $\mathcal{H}$-divergence reflects the ability of hypothesis class $\mathcal{H}$ to distinguish between two distributions $\mathcal{D}_S$ and $\mathcal{D}_T$. Therefore minimization of this distance within fixed hypothesis space $\mathcal{H}$ results in a better generalization capability of the classifier. The

advantage of this particular definition of distance is that in the applications (given $\mathcal{H}$ with finite VC-dimension) it can be well-estimated by finite samples from $\mathcal{D}_S$ and $\mathcal{D}_T$. Ben-David et al. in [3] obtain the following results:

Statement 1 *For a symmetric hypothesis class $\mathcal{H}$ (one where for every $h \in \mathcal{H}$, the inverse hypothesis $1 - h$ is also in $\mathcal{H}$) and samples S, T of size m, the empirical $\mathcal{H}$-divergence is*

$$\hat{d}_{\mathcal{H}}(S,T) = 2\left(1 - \min_{h \in \mathcal{H}} \left[\frac{1}{m} \sum_{\mathbf{x}:h(\mathbf{x})=0} I[\mathbf{x} \in S] + \frac{1}{m} \sum_{\mathbf{x}:h(\mathbf{x})=1} I[\mathbf{x} \in T]\right]\right) \tag{2}$$

Statement 2 *(Generalization Bound) Let $\mathcal{H}$ be a hypothesis class of VC-dimension d. With probability $1-\delta$, $\delta \in (0,1)$ over the choice of samples $S \sim (\mathcal{D}_S)^m$ and $T \sim (\mathcal{D}_T)^m$, for every $h \in \mathcal{H}$*

$$\epsilon_{\mathcal{D}_T}(h) \leq \epsilon_{\mathcal{D}_S}(h) + \hat{d}_{\mathcal{H}}(S,T) + \beta + \sqrt{\frac{4}{m}(d \log \frac{2em}{d} + \log \frac{4}{\delta})} + 4\sqrt{\frac{1}{m}(d \log \frac{2m}{d} + \log \frac{4}{\delta})} \tag{3}$$

where $\beta \geq \inf_{h^* \in \mathcal{H}} (\epsilon_{\mathcal{D}_S}(h^*) + \epsilon_{\mathcal{D}_T}(h^*))$, m - the size of datasets S and T, e - the base of natural logarithm

These results provide an upper bound for hypothesis risk on target domain in terms of three important components. Firstly, given a hypothesis h the risk on source domain $\epsilon_{\mathcal{D}_S}(h)$ represents the performance of the classification on labelled training data. It implies that the fixed hypothesis h should perform well at least on samples from S. The second component is the distance measure between two datasets S and T with respect to fixed hypothesis space $\mathcal{H}$. The impact of this part is two-sided. On the one hand, ideally it is better if it is as small as possible with respect to the space of all possible hypothesis $\mathcal{H}^{all}$. On the other hand, even though that does not hold, one can make it less by choosing not rich hypothesis space $\mathcal{H}$. The third part, however, does not allow to make $\mathcal{H}$ too poor, because it should be rich enough to contain a hypothesis h^*, that performs well on both Source and Target domains. Actually, $\mathcal{H}$ should be also rich enough so that we are able to find h which serves as a good approximation of the best hypothesis h^* on this class. The last two components should be also taken into account while choosing proper $\mathcal{H}$ (i. e. number of neurons in the applications), because even though we have enough statistics ($m = 100000$) they both rise quickly with VC-dimension in the region where $d \lesssim 500000$.
One could propose an algorithm that minimizes the right part of the inequality in the Statement 2. However, the computation of $\hat{d}_{\mathcal{H}}(S,T)$ is a difficult problem, because of the "min" part in the Statement 1. For this reason following Ganin et al. [4] we approximate it by adding domain discriminator part to a conventional Feed Forward neural network.

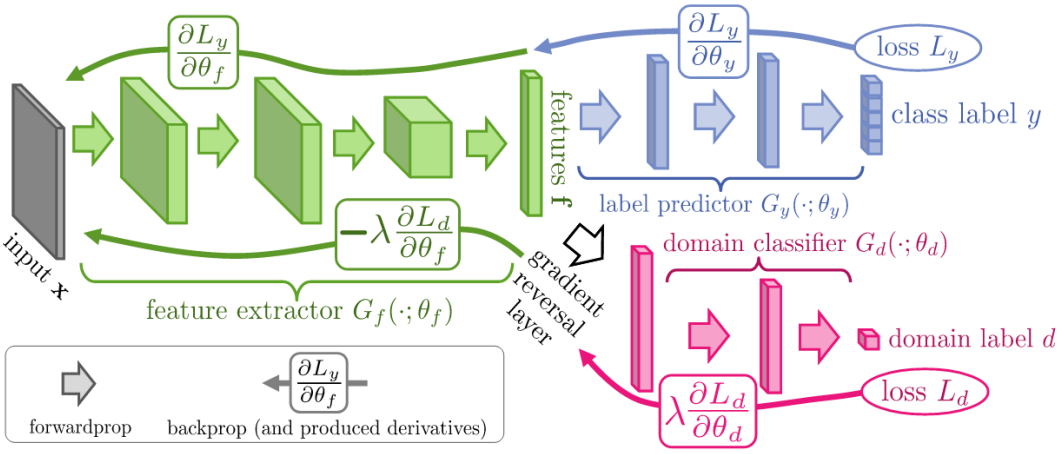

Figure 2: Neural network architecture with domain classifier (discriminator) and gradient reversal layer [4], for detailed description of DANN architecture refer to the text.

2.2. Adversarial Domain Adaptation Neural Network

Thus, DANN architecture consists of three main parts Fig. 2. The first one is a *feature extractor* $G_f(\cdot, \theta_f) : \mathcal{X} \to \mathcal{R}^D$ which maps the input variables into latent representation space (or features). Here and later θ parameters, which define a corresponding maps, are called *weights* of the classifier. The second part $G_y(\cdot, \theta_y) : \mathcal{R}^D \to [0, 1]$ is a *label predictor* that maps features to the class scores between 0 and 1 that serve as an estimate of class labels (if the score of an event is close to 1, then it is signal, otherwise it is predicted to be background). The third is *domain classifier* $G_d(\cdot, \theta_d) : \mathcal{R}^D \to [0, 1]$. This part predicts if an event comes from Source (0) or Target (1) i. e. the domain classifier is designed to distinguish between the source and target instances and approximate the "min" part of the $\mathcal{H}$-divergence in the Statement (1). Both label predictor and domain classifier have corresponding *cross entropy* loss functions $\mathcal{L}_y(\theta_f, \theta_y)$ and $\mathcal{L}_d(\theta_f, \theta_d)$ that are computed as follows:

$$\mathcal{L}_y(\theta_f, \theta_y) = \frac{1}{m}\sum_{i=1}^{m}\left[y_i \log \frac{1}{G_y(G_f(\mathbf{x}_i, \theta_f), \theta_y)} + (1 - y_i)\log\frac{1}{1 - G_y(G_f(\mathbf{x}, \theta_f), \theta_y)}\right] \quad (4)$$

$$\mathcal{L}_d(\theta_f, \theta_y) = \frac{1}{2m}\sum_{i=1}^{2m}\left[d_i \log \frac{1}{G_d(G_f(\mathbf{x}_i, \theta_d), \theta_d)} + (1 - d_i)\log\frac{1}{1 - G_d(G_f(\mathbf{x}, \theta_f), \theta_d)}\right] \quad (5)$$

where d_i is a *domain label* of an event $(\mathbf{x}_i, y_i)$. $d_i = 0$, if $\mathbf{x}_i \in S$ and $d_i = 1$ if $\mathbf{x}_i \in T$. Note that $\mathcal{L}_y(\theta_f, \theta_y)$ is computed from Source instances only, while $\mathcal{L}_d(\theta_f, \theta_d)$ uses both Source and Target data. A total loss function is introduced as

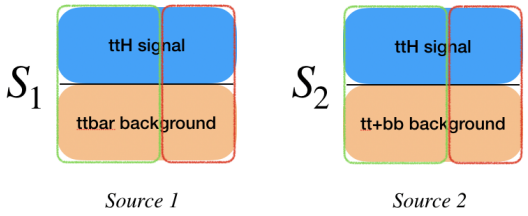

Figure 3: Construction of two datasets from different simulations

$$\mathcal{L}(\theta_f, \theta_y, \theta_d) = \mathcal{L}_y(\theta_f, \theta_y) + \lambda \mathcal{L}_d(\theta_f, \theta_d) \tag{6}$$

However, instead of minimizing this total loss with respect to three groups of weights $\theta_f, \theta_y, \theta_d$ the *gradient reversal layer* is utilized. The idea of the *gradient reversal layer* is to reverse the sign of the derivative of discriminator loss $\mathcal{L}_d$ with respect to the weights of the feature extractor. So the optimizer acts as the gradients are calculated as follows:

$$\frac{\partial \mathcal{L}(\theta_f, \theta_y, \theta_d)}{\partial \theta_y} = \frac{\partial \mathcal{L}_y(\theta_f, \theta_y)}{\partial \theta_y} \tag{7}$$

$$\frac{\partial \mathcal{L}(\theta_f, \theta_y, \theta_d)}{\partial \theta_d} = \lambda \frac{\partial \mathcal{L}_d(\theta_f, \theta_d)}{\partial \theta_d} \tag{8}$$

$$\frac{\partial \mathcal{L}(\theta_f, \theta_y, \theta_d)}{\partial \theta_f} = \frac{\partial \mathcal{L}_y(\theta_f, \theta_d)}{\partial \theta_f} - \lambda \frac{\partial \mathcal{L}_d(\theta_f, \theta_d)}{\partial \theta_f} \tag{9}$$

Intuitively this means that the feature extractor tries to maximize $\mathcal{L}_d$ making instances from two domains indistinguishable in the feature space. Therefore domain classifier acts like a regularization term and promotes the generalization capability in the target domain. The discriminator loss $\mathcal{L}_d$ is also multiplied by a positive *discriminator importance parameter* λ that sets the relative importance of label classification and domain adaptation. Zhao et. al. in [5] argue that λ parameter can be data dependent so it is important to find an appropriate value for our application. Technically we set λ to zero to measure the performance of the corresponding baseline Feed Forward architecture without domain classifier.

3. Data

Two datasets S_1 and S_2 Fig. 3 are constructed from simulated data as follows. Each dataset consist of 100000 ttH(bb) signal events from MadGraph/Herwig6 simulation. In addition, both of them has background events. But S_1 is filled with background events from MadGraph/Pythia6, while S_2 from Powheg Pythia8. Each event is represented by 41 variables that simulate ATLAS detector response on processes shown in Fig. 1. A description of each variable is provided in the Table 2. Distributions of some input

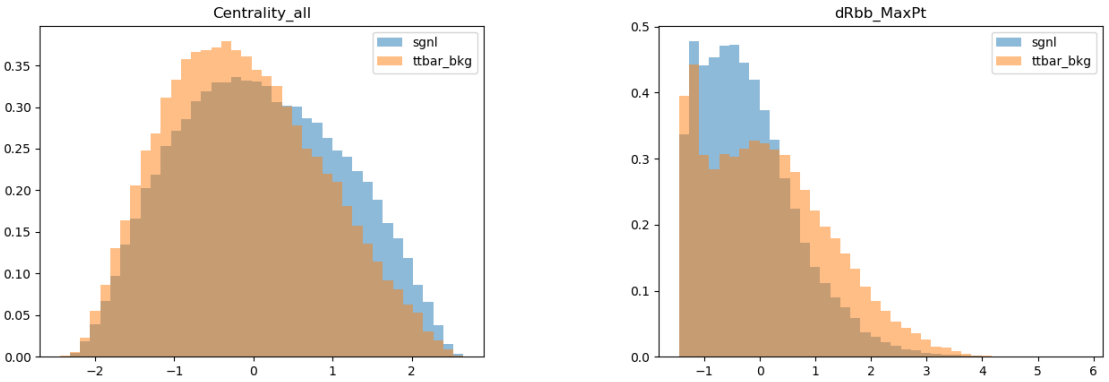

Figure 4: Variable distributions for Centrality_all (left) and dRbb_MaxPt (right)

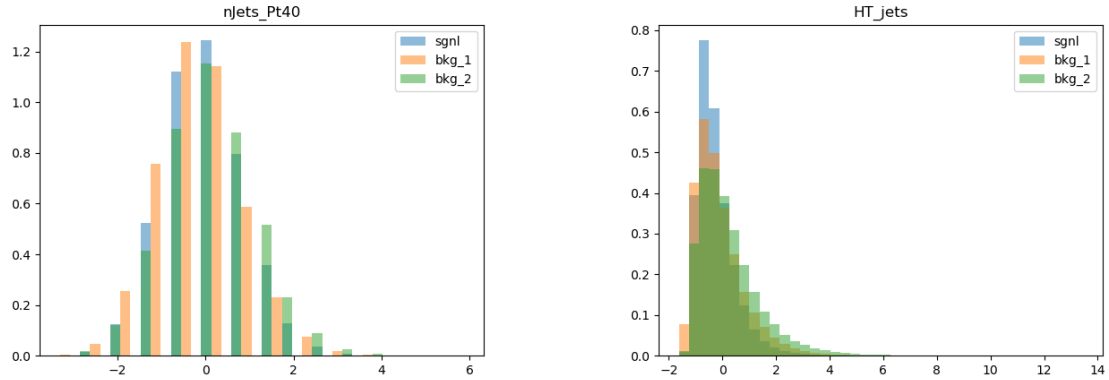

Figure 5: Variable distributions for nJets_Pt40 (left) and dRlb3 (right)

Jets are selected if they have at least 25 GeV transverse momentum. We select events that have at least 5 jets, of with at least 3 are tagged as b-jets. We require events to have exactly one charged lepton, either electron or muon. We are therefore looking at events in which one top decays to a lepton, thereby suppression background. bkg_1 is ttbar background, bkg_2 is ttbb background.

variables are shown in Fig. 4. The difference in the background simulations may be seen from distributions in Fig. 5, for example, $nJets\_Pt40$ and HT_jets (distributions are centered to zero and normalized for better performance of the NN). 66% of each dataset (green part in Fig. 3) are used for training and classification performance is evaluated on statistically independent *validation* (red in Fig. 3).

4. Details of DANN Implementation and Optimization

4.1. Implementation

The DANN architecture described in Section 2.2 is implemented in Python 3.5 using Pytorch framework based on the code of [5]. The feature extractor $G_f(\cdot, \theta_f)$ is modeled by 3 hidden layers and $ReLu$ activation function in each layer. Both label the predictor $G_y(\cdot, \theta_y)$ and the domain classifier $G_d(\cdot, \theta_d)$ represent shallow one layer perceptrons with Cross entropy loss. Batch size of 5000 events is used in most of the experiments as it appears to be a good trade-off between epoch computation costs and stability of optimizer steps. The feature extractor is tested in two different modes: the first is conventional - all data instances from both domains (S_1, S_2) are passed to the discriminator, the second one is data specific and uses the knowledge of S_1 data labels for discrimination task - only background events are passed to the discriminator part. Though for our experiments the second approach does not show clear improvements, it has been found that it affects the choice of λ. Hyper-parameters (number of neurons in each layer and λ) have been selected from cross-validation and the optimal value is found to be $[45, 30, 25]$, $\lambda : 1.2$ (when only the background events are shown to the discriminator). Scan on the λ parameter is performed at 20 values between 0 and 2. For values of $\lambda < 1$ the optimization converges, but does not show satisfactory results on S_2. For values between 1.5 and 2 the optimization does not converge. Overall, the training behaviour of the network is found to be sensitive to λ.

4.2. Optimization

Pytorch implementation of Adadelta optimizer [11] is used in most of the experiments. Weights are updated in all three parts of the network simultaneously according to the derivatives in Eq. (7, 8, 9). Note that partial derivatives with respect to θ_f weights are deliberately calculated putting " $-$ " for the second component to implement the idea of gradient reversal layer. This, however, creates an additional challenge to the optimizer in preserving the balance between classification $\mathcal{L}_y(\theta_f, \theta_y)$ and discrimination $\mathcal{L}_d(\theta_f, \theta_d)$ losses. The Adadelta method updates the weights $\theta_t = (\theta_f^t, \theta_y^t, \theta_d^t)$ of NN at time t as follows:

$$\theta_{t+1} = \theta_t + \Delta\theta_t \tag{10}$$

$$\Delta\theta^t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} \cdot g_t \tag{11}$$

$$RMS[g]_t = \sqrt{E[g^2]_t + \varepsilon} \tag{12}$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta x^2]_t + \varepsilon} \tag{13}$$

$$E[\Delta\theta^2]_t = \rho E[\Delta\theta^2]_{t-1} + (1-\rho)\Delta\theta_t^2 \tag{14}$$

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho)g_t^2 \tag{15}$$

where ε ($\simeq 10^{-6}$) is a small constant to avoid division by zero and zero update at the beginning, $\rho \in (0,1)$ is a decay constant which is set to 0.9.
Here θ_{t+1}, $\Delta\theta_t$, $E[g^2]_t$, $E[\Delta\theta^2]_t$, $RMS[g]_t$ and $RMS[\Delta\theta]_t$ are vectors of the same dimension as $\theta_t = (\theta_f^t, \theta_y^t, \theta_d^t)$, and vector operations in Eq. 11 should be considered as component-wise division and multiplication. Accumulation variables $E[g^2]_t$, $E[\Delta\theta^2]_t$ play a role of running averages for squared gradients and weight updates. An idea of the Adadelta method is to accumulate sums of squared components of the gradient g over a time-window, which is defined by ρ parameter. The weight updates $\Delta\theta_t$ are modified Eq. 11 (in comparison with standard gradient descent, where $\Delta\theta_t = -const \cdot g_t$) so that the gradient g_t is divided by accumulated sums for each component. It is also multiplied by accumulated sums of previous updates to match the units of updates to the units of θ. Intuitively, this approach helps to treat all NN weights more or less equally and allows to update those weights that remain not updated for a long time. This strategy turns out to be especially fruitful in application to DANN architecture. As the gradient reversal layer is applied the NN the total loss function $\mathcal{L}$ is no longer directly optimized during the training. Instead different components of the total gradients in Eq. 7, 8, 9 are responsible either for improving classification power or generalization capability. Thus ideally to provide a good trade-off between these two goals all weight groups should be treated equally even though it is natural that in the beginning they might influence the total loss differently. In the experiments related to optimization a default setting of the NN is fixed (number of neurons: $[45, 30, 25]$, $\lambda : 1.2$). We roughly compare the performance of three optimization methods: Nesterov's accelerated gradient [12], Adagrad and Adadelta [11] in pytorch implementations to justify the above reasoning. Adagrad and Adadelta are similar in that both methods tend to give preference to rarely updated weights implementing the component-wise division of g_t by accumulated squares of gradient.

5. Results

Response, ROC and training curves are used to illustrate the classification performance and training bias. Further we calculate significance, ROC AUC and accuracy to quantify the results Table 1. Note that both datasets are split into training and validation parts with $2:1$ ratio as it is shown with red and green lines in Fig. 3 and all above-mentioned curves and performance measures have been evaluated on "red" validation data instances that are statistically independent of "green" training data.

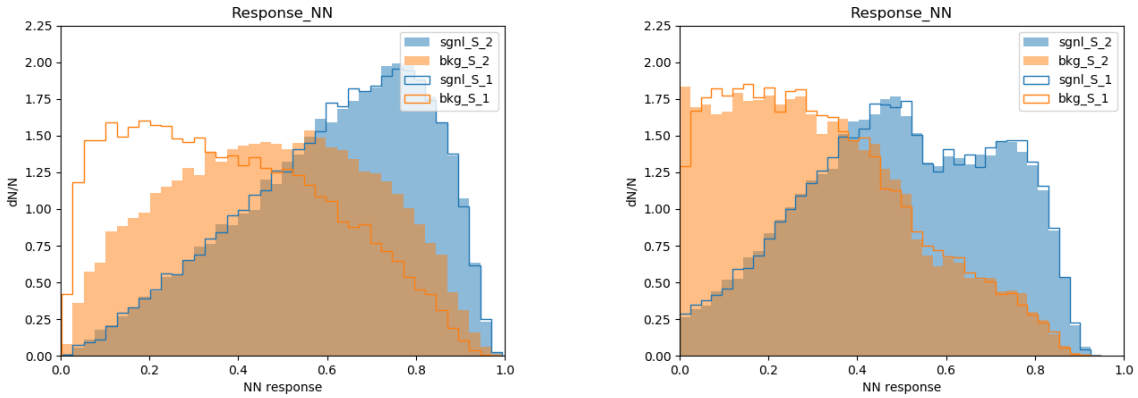

Figure 6: # neurons: 45, 30, 25; λ: 0 (left), 1.2 (right); Only background events from S_1 are passed through discriminator

5.1. Response of NN

Response curve shows a distribution of NN output scores (on validation) which vary from 0 to 1. Here 0 corresponds to the prediction of background event and 1 to signal event. Fig. 6, 7 demonstrate NN response curves after 300 epochs for different NN architectures. The novelty of this approach addresses the specific difficulty of the physics analysis that relies on a uniform response of the classifier over its full spectrum. In these plots orange lines correspond to the response on *ttbar* background events, while orange shaded region to *ttbb* background events. The difference between two is the measure of a training bias. Blue color shows the response on signal events. All histogram bins are set to 40 to distinguish histogram shapes obtained after training. Fig. 6, 7 compare the training bias of NN for baseline method without adversarial part (left) with DANN (right) described in Sections 2.2, 4. DANN architecture clearly reduces the training bias of NN in the signal region. The improvement can be also seen in case when both signal and background events are passed to the discriminator Fig. 8.

5.2. Significance calculation and ROC curves

To quantify our results we calculate *significance* Z_A taking into account class imbalance (5% of signal) and integrated cross section (139 fb^{-1}) according to Eq. 16, 17, 18, 19. We fix an arbitrary cut of the classification score at 0.6 to roughly estimate the results.

$$Z_A = \frac{s}{\sqrt{b_2 + \sigma_b^2}} \tag{16}$$

$$s = 5 \cdot 10^4 \frac{s^{cut}}{s^{all}} \cdot 5\% \tag{17}$$

$$b_{2(1)} = 5 \cdot 10^4 \frac{b_{2(1)}^{cut}}{b_{2(1)}^{all}} \cdot 95\% \tag{18}$$

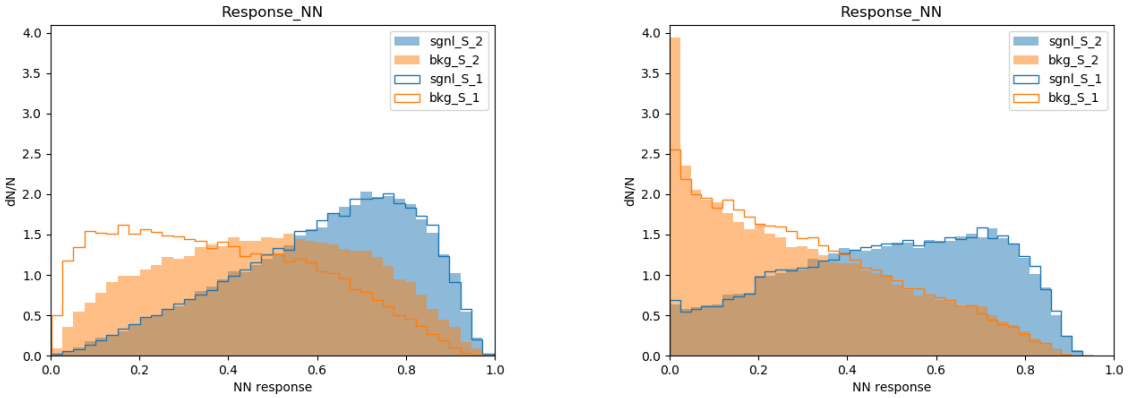

Figure 7: # neurons: 35, 30, 15; λ: 0 (left), 1.3 (right); Only background events from S_1 are passed through discriminator

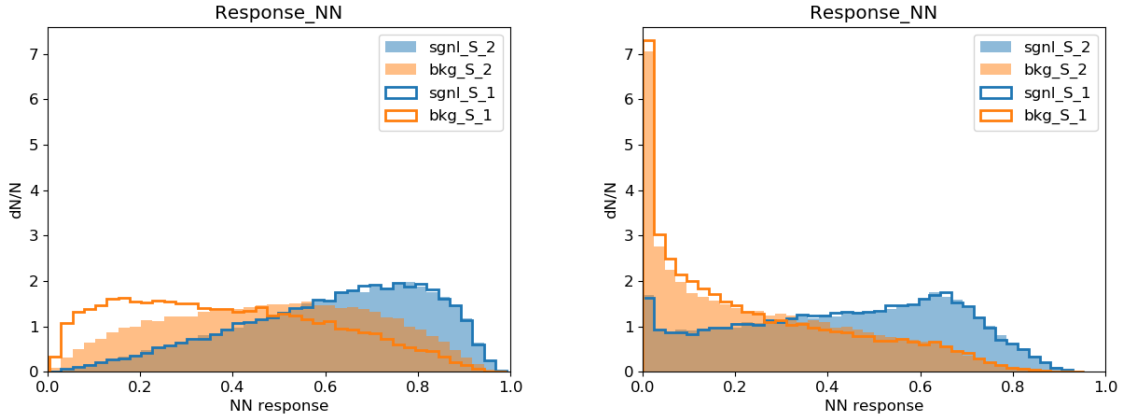

Figure 8: # neurons: 45, 30, 25; λ: 0 (left), 2 (right); Signal and Background events from S_1 are passed through discriminator

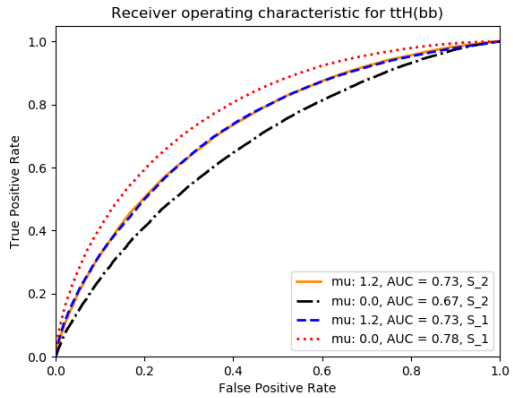

Figure 9: ROC curves
The data instances from S_1 are used for training and the evaluation is performed on the statistically independent validation parts of S_1 and S_2.

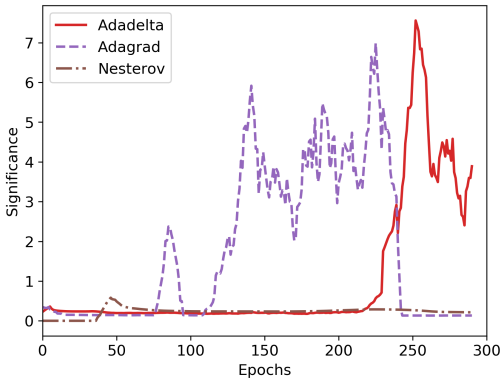

Figure 10: Significance curves
The significance is measured after each epoch and the curves are smoothed with a 10 epoch window. The performance of the three optimization methods is compared.

Neurons	λ	Accuracy	AUCROC	Significance
[35, 30, 15]	0.00	0.62	0.67	0.21
[35, 30, 15]	1.30	0.65	0.73	4.41
[40, 25, 25]	0.00	0.62	0.67	0.22
[40, 25, 25]	1.24	0.64	0.73	3.95
[45, 30, 25]	0.00	0.62	0.67	0.21
[45, 30, 25]	2.00	0.64	0.72	11.99

Table 1: Classification performance for different DANN settings
Significance values are improved by a huge amount by using domain adaptation.

$$\sigma_b = b_2 - b_1 \tag{19}$$

where s^{all} - number of signal events from source 2, s^{cut} - number of signal events from source 2 with classification score above 0.6, $b^{all}_{2(1)}$ - number of background events, $b^{cut}_{2(1)}$ - number of background events with classification score above 0.6 (here indexes 1 and 2 relate to source 1 and 2 to correspondingly).
Receiver operating characteristic (ROC) curves Fig. 9 also indicate the trade-off achieved by the application of domain classifier. The difference between red dotted and black dash-dotted lines illustrate the difference in the performance of baseline method on different domains. Notably the performance on the S_2 is worse because NN is trained on S_1 only. However, this gap shrinks to almost identical orange solid and blue dashed lines when domain classifier is applied.

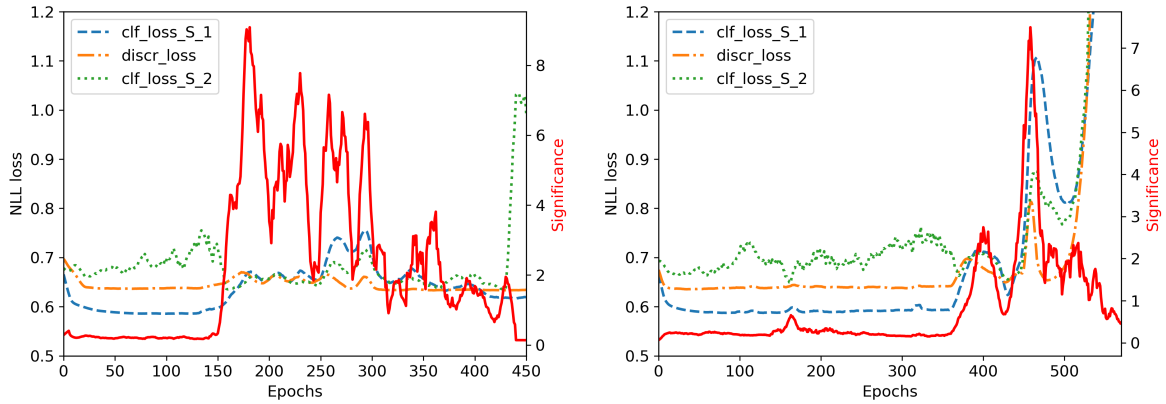

Figure 11: Training curves for the NN setting of [45, 30, 25] neurons
The two plots illustrate the training process for the different strategies of passing data through the discriminator. For the left plot only background events from S_1 are shown to the discriminator. The right plot is the result of the training when all data instances from S_1 are used for the discriminating task.

5.3. Looking at training curves

The competing behaviour between the two parts of DANN is observed in Fig. 11 when Adagrad or Adadelta optimizer is applied. Blue dashed and green dotted curves correspond to the classification cross entropy loss on the first and the second sources during the training. Orange dash-dotted and red solid lines show the discriminator cross entropy loss and the significance correspondingly. The curves are smoothed with a 10 epoch window to get rid of local fluctuations. Interestingly, all three losses synchronize at some point and later oscillate together with nearly the same frequency and amplitude. At the same time significance increases rapidly as a result of this competition and also oscillates regularly reaching its local maximum values together with the three losses. The significance curve usually has a distinct global maximum and this analysis may indicate that the DANN architecture shows the best performance on one of the first local maximum of the blue or orange curves when the competing behaviour has only started. Three different optimization methods have been tested Fig. 10. Both Adadelta and Adagrad optimizers remarkably improve the significance. Notably, the application of Nesterov's accelerated method (with different learning rate and momentum factor) monotonically slightly decreases both classification and discrimination losses on S_1, which however does not result in the significance improvement.

6. Conclusion

Adversarial Domain Adaptation approach has been tested in application to ttH(bb) search in simulated data. We have demonstrated that it can reduce the training bias towards a given simulation. This concept can be applied to minimize an impact of any

systematic uncertainty. A natural extension to this work could be to use real collision data as the target, thereby correcting the labelled MC to data. The code and more plots can be found here.

7. Acknowledgements

The author expresses gratitude to Paul Glaysher and Judith Katzy for providing so challenging and interesting project. Numerous discussions, constant assistance and guidance have been invaluable. Special thanks go to the ATLAS group for creating such friendly and enjoyable atmosphere. Lastly, I would like to express my appreciation to the DESY Summer Student Program organizers for arranging this highly educational and enjoyable experience.

References

1. *Collaboration A.* Search for the Standard Model Higgs boson produced in association with top quarks and decaying into a $b\bar{b}$ - pair in pp collisions at $\sqrt{s} = 13$ TeV with the ATLAS detector // arXiv:1712.08895. — 2018.
2. *Guest D.*, *Cranmer K.*, *Whiteson D.* Deep Learning and Its Application to LHC Physics // Annual Review of Nuclear and Particle Science. — 2018. — Vol. 68, no. 1. — P. 161–181. — DOI: 10.1146/annurev-nucl-101917-021019. — eprint: https://doi.org/10.1146/annurev-nucl-101917-021019.
3. A theory of learning from different domains / S. Ben-David [et al.] // Machine Learning. — 2010. — May. — Vol. 79. — P. 151–175. — DOI: 10.1007/s10994-009-5152-4.
4. Domain-Adversarial Training of Neural Networks / Y. Ganin [et al.] // arXiv: 1505.07818. — 2016.
5. Adversarial Multiple Source Domain Adaptation / H. Zhao [et al.] // Advances in Neural Information Processing Systems 31 / ed. by S. Bengio [et al.]. — Curran Associates, Inc., 2018. — P. 8559–8570.
6. Analysis of Representations for Domain Adaptation. / S. Ben-David [et al.] //. Vol. 19. — 2006. — P. 137–144.
7. *Ryzhikov A.*, *Ustyuzhanin A.* Domain adaptation with gradient reversal for MC/real data calibration // Journal of Physics: Conference Series. — 2018. — Sept. — Vol. 1085. — P. 042018. — DOI: 10.1088/1742-6596/1085/4/042018.
8. *Long M.*, *Wang J.*, *Jordan M.* Deep Transfer Learning with Joint Adaptation Networks. — 2016. — May.
9. Moment Matching for Multi-Source Domain Adaptation / X. Peng [et al.] // ArXiv. — 2018. — Vol. abs/1812.01754.
10. *Drossos K.*, *Magron P.*, *Virtanen T.* Unsupervised Adversarial Domain Adaptation Based On The Wasserstein Distance For Acoustic Scene Classification // ArXiv. — 2019. — Vol. arXiv:1904.10678.
11. *Zeiler M. D.* ADADELTA: An Adaptive Learning Rate Method // arXiv: 1212.5701. — 2012.
12. On the importance of initialization and momentum in deep learning / I. Sutskever [et al.] // Proceedings of the 30th International Conference on Machine Learning. Vol. 28 / ed. by S. Dasgupta, D. McAllester. — Atlanta, Georgia, USA : PMLR, 2013. — P. 1139–1147. — (Proceedings of Machine Learning Research ; 3).

A. Variable list for NN training

Num	Name	Description
1	nJets	number of jets
2	MET	missing transverse energy
3	Mjj_MaxPt	mass of jet pair with largest transverse momentum
4	nJets_Pt40	number of jets above 40 GeV p_T
5	nbTag	number of b-tagged jets
6	nHiggsbb30	number of Higgs boson candidates from b-jets, in 30 GeV window
7	nHiggsjj30	number of Higgs boson candidates from all jets, in 30 GeV window
8	pT_jet5	transverse momentum of 5th leading jet
9	pT_lep	transverse momentum of charged lepton
10	HT_jets	sum of transverse momentum of jets
11-14	Hi_all	ith Fox-Wolfram moment, i=0,1,2,3
15	H2_jets	2nd Fox-Wolfram moment (jets only)
16	dEtajj_MaxdEta	largest difference in pseudo rapidity
17	Centrality_all	Scalar sum of the p_T divided by the sum of E for all jets and the lepton
18	dRbb_avg	average opening angle between b-jets
19	Mbb_MindR	mass of closest b-jets
20	Mbj_MaxPt	mass of two jets (one of them a b-jet), with largest p_T
21	dRbb_MaxPt	opening angle of two b-jets with largest P_T
22	Aplanarity_jets	1.5 times second eigenvalue of the momentum tensor
23	Mjj_MindR	mass of two closest jets
24	dRbj_Wmass	opening angle of two jets (one of them a b-jet), closest two W-mass
25	Mbj_Wmass	mass of two jets (one of them a b-jet), closest two W-mass
26	Mbj_MindR	mass of two closest jets (one of them a b-jet)
27	dRlj_MindR	smallest opening angle between lepton and a jet
28	dRlj_MaxdR	largest opening angle between lepton and a jet
29	pT_jet3	transverse momentum of 3rd leading jet
30	dRbb_MaxM	opening angle of two b-jets with largest invariant mass
31	dRjj_MindR	smallest opening angle of any two jets
32	Aplanarity_bjets	1.5 times second eigenvalue of the momentum tensor of b-jets
33	Mjjj_MaxPt	mass of three jets with largest p_T
34	Mbb_MaxM	largest mass of two b-jets
35	Mjj_MinM	smallest mass of two jets
36	dRlbb_MindR	smallest opening angle between lepton and bb-system
37	dRluu_MindR	smallest opening angle between lepton and two untagged jets
38	dRlbb_MaxdR	largest opening angle between lepton and bb-system
39-41	dRlbi	opening angle between lepton and ith b-jet, i=1,2,3

Table 2: Description of the input variables used for training neural network