Alma Mater Studiorum · University of Bologna

School of Science
Department of Physics and Astronomy
Master Degree in Physics

THESIS TITLE

Supervisor:
Prof./Dr. Name Surname

Co-supervisor: (optional)
Prof./Dr. Name Surname

Submitted by:
Name Surname

Academic Year 2020/2021

Contents

List of Figures

List of Tables

Listings

Abbreviations

1 Introduction

1.1 Model description

Like many other deep generative models, the variational auto encoders' (VAEs) aim is to learn the underlying input data distributions to generate new samples with features similar to the original ones. In order to achieve this goal, these algorithms are built basically in two steps: the first one - the *encoding stage* - where the VAE compresses the input data in a lower-dimensional space (*latent space*) and the second step where it tries to reconstruct the original input - *decoding phase* - distribution starting from this incomplete information.
Contrarily to the vanilla autoencoders, where a point wise encoding results in a less efficient and precise regeneration, the VAEs compress data in a continuous latent space. Indeed, each input is associated with a distribution within this space and the reconstruction step starts only after sampling from that distribution. In this way, the model is able to reconstruct similarly points close togheter in the latent space. This approach gives continuity and completeness to this space, making the regeneration step easier and more robust.

The variational autoencoder architecture could be thought of as the ensemble of two neural networks, one on top of the other, designed respectively with a contractive-path and an expansive one. The first of these provides a last layer/s with fewer neurons respect to the input layer, so to be able to compress the data in a lower-dimensional space. The second one, instead, starting from the latent representation, moves in the opposite direction and tries to reconstruct the compressed data in the higher-dimensional original space. Naturally, this separation is only speculative and joint optimization of all the network weights can be obtained minimizing an objective function.

The target *function loss* suitable for the final goal is constituted by two pieces. The first term is related to the model reconstruction performances, while the second one regards the Kulback-Lieber divergence between the latent space shape and its target distribution, usually a multivariate standard gaussian. So the general form of the final loss results from a weighted sum of these two terms:

$$Loss_{tot} = Loss_{reco} + \beta D_{KL} \tag{1}$$

where β is a free parameter defining the relative weight of the divergence loss term in the total final score function and D_{KL} is the Kulback-Lieber divergence, described in detail in Section 1.3.1. Under this loss definition, the model learns how to compress and successfully reconstruct the bulk of the variable distributions contained in the training samples. On the other hand, the model is prone to fail while reconstructing the more rare events. Due to this behavior, the latter kind of event is likely to end up in the right tail of the distribution loss (higher losses). This situation is even more evident when considering data samples characterized by different variable distributions with respect to the ones used for training, for example samples with non SM events which were not present in the SM-only background sample.
Given these considerations, in this study a model trained to reproduce a cocktail of Monte Carlo SM processes is presented, with the aim of testing the background modeling capa-

bilities. The signal sensitivity is tested including both background and signal events in the validation sample and expecting to find a region in the right tail of the loss distribution where maximize the purity of the signal.

1.2 Dataset

The Wh1Lbb analysis ntuples are used in this study. The same preselection of the original analysis [**?**] is applied both on the background processes and on the signal events and it is reported in Table 1. In Figure 1 the distributions of the most discriminating variables are shown for the total background (sum of all the processes) and some signal example. The definition and the data/MC agreement of each variable considered for the training are explained and described in detail in Section 6 of [**?**]. It is worth to remark that only the background events are used to train the model, while the signal events are only included in the evaluation step after the events selection.

Table 1: Preselection cuts applied both on signal and background samples.

	Preselection
Exactly 1 signal lepton	True
met trigger fired	True
$2-3$ jets with $p_T > 30GeV$	True
b-tagged jet	[1-3]
met	> 220 GeV
mt	> 50 GeV

The model is trained in three different kinematics regions described in Table 2. The aim is to test the sensitivity to the signal model in different topological spaces. The trade-off between a model-independent analysis and better signal sensitivity is one of the starting points investigated in this study. In the first case, any or only loose selection requirements would be applied to avoid cuts tailored on a specific signal hypothesis. Nevertheless, training on more selected background events could allow the model to focus more on the relevant background distinctive features finally leading to a better signal selection, albeit reducing the generality of the selection, and possibly reducing the selection sensitivity to similar models (as pMSSM, or different signal models).

Table 2: Requirements for the three selected regions.

	Preselection	mid. region	$2-3$b region
Exactly 1 signal lepton	True	True	True
met trigger fired	True	True	True
$2-3$ jets with $p_T > 30GeV$	True	True	True
b-tagged jet	[1-3]	[1-3]	[2-3]
met	> 220 GeV	> 220 GeV	> 220 GeV
mt	> 50 GeV	> 50 GeV	> 50 GeV
mbb		$[100-140]$ GeV	$[100-140]$ GeV
mct		> 100 GeV	> 100 GeV

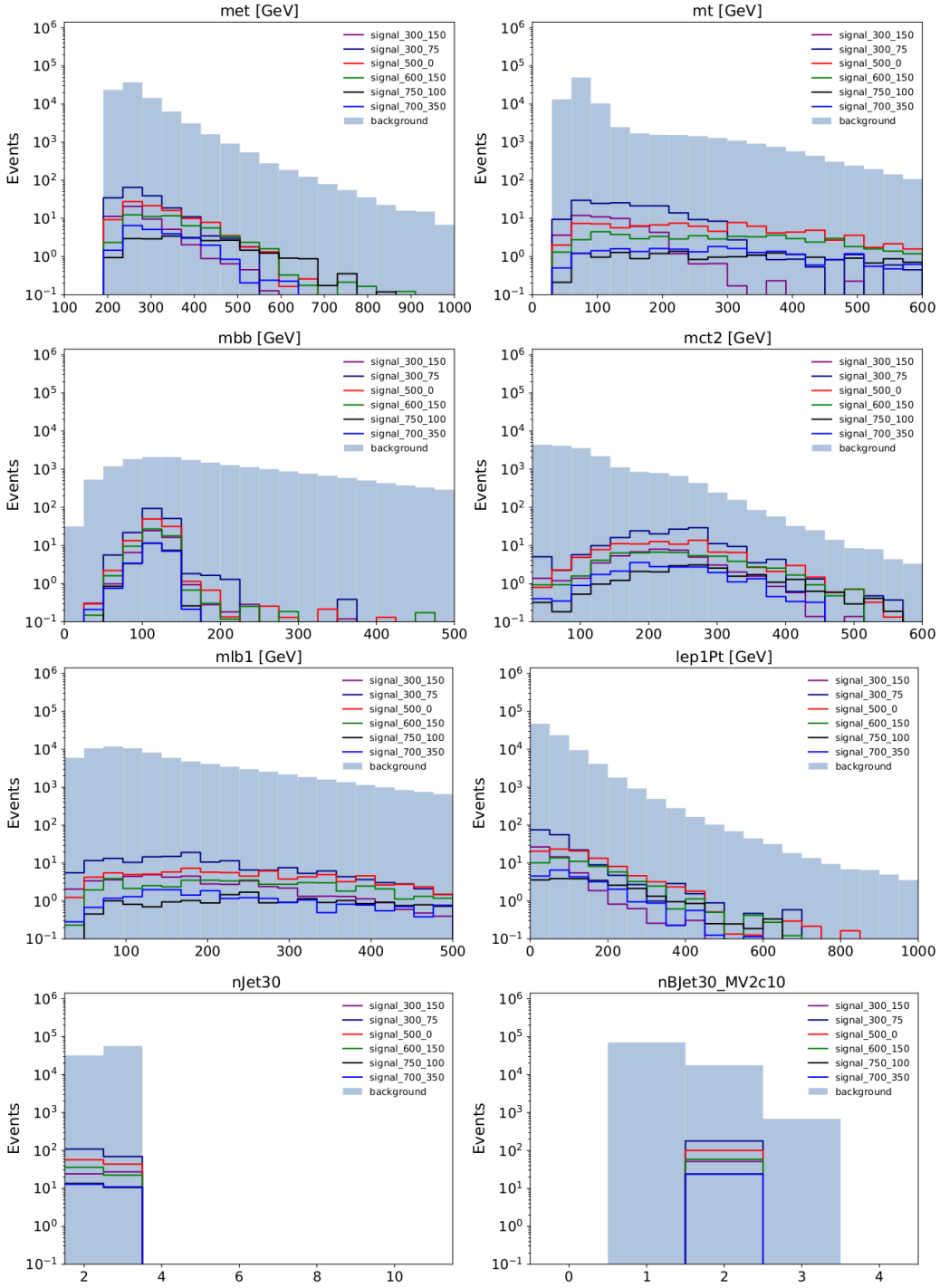

Figure 1: Distributions of the input variables for the sum of all the background processes and some signal points, as example.

1.3 Training

The model is trained in the three regions separately after a $60\% - 40\%$ training-validation split. Due to the different selection depth, the total training-validation event number varies region by region. In order not to reduce this number drastically, a threshold of 10^6 events for the training-validation sample is fixed as a lower limit and the 2 - 3*b region* is taken as the region with the smallest selected number of events.

1.3.1 Loss function

The training loss function described above (eq. 1) consists of two terms. The goal of the training is to find the best trade-off between good reconstruction performances and a regular latent space. A closer description of these two terms is given in the following.

The Kullback-Lieber divergence term regards the encoding phase outputs and forces all the predicted distributions to stay as close as possible to the prior choice. For the sake of simplicity, usually a multivariate gaussian is selected as target distribution. In this study, the Kulback-Lieber divergence has the following closed analytical form:

$$D_{KL} = \frac{1}{2k} \sum_{i,j=1} \left(\sigma_p^j \sigma_z^{i,j}\right)^2 + \left(\frac{\mu_p^j \mu_z^{i,j}}{\sigma_p^j}\right)^2 + ln\frac{\sigma_p^j}{\sigma_z^{i,j}} - 1 \tag{2}$$

where,k is the batch size selected for the training, i runs over the samples and j over the latent space dimensions. The parameters predicted from the model for each event are: (1) μ_z and σ_z that define the distribution shape in the latent space; (2) μ_p, and σ_p that represents the prior shape parameters. It is important to observe that, following the work described in [**?**], also this latter couple of parameters are optimized during the training letting the model to select the optimal latent space target distribution.

The reconstruction loss term is instead represented by the average negative-log-likelihood of the inputs given the shape parameter values predicted by the model during the decoding phase:

$$\begin{aligned} Loss_{reco} &= -\frac{1}{k} \sum_{i,j=1} \ln\left[P\left(x|\alpha_1, \alpha_2, ...\alpha_n\right)\right] \\ &= -\frac{1}{k} \sum_{i,j=1} \ln\left[f_{i,j}\left(x_{i,j}|\alpha_1^{i,j}, \alpha_2^{i,j}, ...\alpha_n^{i,j}\right)\right] \end{aligned} \tag{3}$$

In the equation, j runs over the input space dimensions, $f_{i,j}$ is the functional form chosen to describe the pdf of the *i-th* input variable and α_n are the parameters of this function and represent also the final output of the network. Two different functional forms are selected to describe the distribution of the variables defining each physical events inside the training dataset. Specifically:

- the clipped log-normal function is used for all the continuous variables: *met, mt,*

mbb, mct2, mlb1, lep1Pt:

$$P\left(x|\alpha_1,\alpha_2,\alpha_3\right) = \begin{cases} \alpha_3\delta\left(x\right)\frac{1-\alpha_3}{x\alpha_2+\sqrt{2\pi}}e^{-\frac{(\ln x-\alpha)^2}{2\alpha_2^2}} & for x \geq 10^{-4} \\ 0 & \leq 10^{-4} \end{cases} \tag{4}$$

- A truncated discrete gaussian for for the discrete variables is: *njet30, nBjet30_MVc10*:

$$\Theta\left(x\right)\left[erf\left(\frac{n+0.5-\alpha_1}{\alpha_2+\sqrt{2}}\right)-erf\left(\frac{n-0.5-\alpha_1}{\alpha_2+\sqrt{2}}\right)\right] \tag{5}$$

where the normalization factor N is set to:

$$N = \frac{1}{2}\left(\frac{-0.5-\alpha_1}{\alpha_2+\sqrt{2}}\right) \tag{6}$$

1.3.2 Model architecture

The network architecture is briefly described in the 1.1. It consists mainly of a contractive path followed by an expansive one. So, the feed-forward step goes first through a stack of fully-connected layers that progressively builds a representation of the inputs in the latent space and, then, towards a second fully-connected set of layers whose goal is to output the parameters shape (eq: 4, 5): the latter are used to describe the variables probability distributions for each event and represent the final output of the variational autoencoder.

The configuration of the network here trained is strongly inspired by the work [**?**].

Nevertheless, the model configuration is also affected by the choice of a set of hyperparameters: that is, all the parameters fixed a priori, whose value is not learned automatically during the training. The hyperparameters are opposed to the network weights, linking the neurons in the model architecture for which an optimization occurs through the back-propagation of the loss. The weights update happens batch by batch during the traininig, following the gradient descents related to the loss minimization. The training success also depends on the hyperparameters that, in some way, fix the boundary condition of the learning procedure. Usually, their choice proceeds by trial and error, but some hints to address the initial guesses come from the problem context. For example, increasing the number of hidden layers goes along the complexity of the model to be related to the dataset size. A more detailed list of hyperparameter examples is reported here:

- learning rate;
- number of neurons per layer;
- latent space dimension;
- batch size;
- beta weight in the total loss sum;
- kind of activation layer;

- penalization weights on one/more variable loss.

The list is quite long, and it is helpful to figure out how to handle the fine-tuning process in an effective way. For that reason, the *tune* python library has been exploited and modified accordingly to work for this study. One of the strengths of this library is the possibility to run a hyperparameters optimization at any scale. Deploying on a cluster of many GPUs, as in this case, allows for an extensive search among all the possible parameters configuration, reducing the time and the human effort. A training summary is stored during the training and all the model performances can be compared. A final rank based on the evaluation metric helps to focus on the more promising architectures and, finally, to select the best one.

2 Conclusion

This is the last chapter of this thesis [**?**].