Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Dipartimento di Fisica e Astronomia
Tesi triennale in Fisica

APPLICAZIONI DEL MACHINE LEARNING ALLA FISICA DELLE ALTE ENERGIE

Supervisor:
Prof./Dr. Name Surname

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

Submitted by:
Schiazza Filippo Antonio

Anno accademico 2019/2020

Indice

Elenco delle figure

Elenco delle tabelle

Listings

Abbreviations

1 Introduzione

Il Modello Standard è, senza ombra di dubbio, il fiore all'occhiello della fisica del Novecento. Tuttavia sono sempre più le evidenze che suggeriscono come esso si limiti a spiegare solo una parte della struttura profonda della natura: si è fatta strada sempre con più forza l'idea che esista una così detta fisica oltre il Modello Standard e, per indagarla, si costruiscono acceleratori di particelle sempre più potenti, di cui il Large Hidron Collider è l'esempio principale. Il run3 del Large Hidron Collider è previsto per Maggio 2021, tuttavia non vi è stato un miglioramento notevole da un punto di vista energetico. Allo stesso tempo si stima che la produzione di dati sarà fino a dieci volte maggiore rispetto al run precedente, quindi ci si chiede se sia possibile trattare in maniera innovativa questa enorme mole di dati per provare a trovare segnale di nuova fisica. Nello specifico la domanda è se le metodologie di machine learning possano giocare un ruolo centrale per analizzare i dati prodotti nel prossimo ciclo di funzionamento di LHC.
Con il termine machine learning si intende una serie di metodologie di natura statistico-computazionale che permettono di estrarre informazione utile da enormi moli di dati, altrimenti difficilmente processabili dall'uomo. I dati, per la loro stessa natura, sono disomogenei e caotici, quindi risulta particolarmente complesso analizzarli per ottenerne dei risultati. Qui entra in gioco il machine learning, ovvero l'apprendimento automatico della "macchina", perché permette di trovare relazioni nascoste fra i dati autonomamente, ovvero senza la continua supervisione dell'essere umano. Uno dei concetti fondamentali del machine learning è quello di apprendimento, che consiste nella possibilità di addestrare il modello in maniera iterativa.

1.1 Tipologie di analisi dati

Quando si parla di analisi dati ci si può essenzialmente ricondurre a tre macro-categorie di operazioni:

1. CLASSIFICAZIONE
   Questa tipologia è probabilmente la principale quando si ha a che fare con la fisica delle alte energie e consiste nell'associare un evento/oggetto ad una categoria. Per esempio, una volta rilevata una particolare particella bisogna stabilire se questa è un elettrone, un protone, etc;

2. STIMA DI PARAMETRI
   In questa tipologia ricadono tutti quei processi attraverso i quali si estraggono dei parametri (ad esempio la massa di una tipologia di particelle) attraverso un fitting del modello teorico con i dati sperimentali;

3. STIMA DI FUNZIONI
   Si ricava una funzione continua di una o più variabili a partire dai dati sperimentali.

1.2 Processi multi-variati

Nelle prime righe del paragrafo precedente si è introdotto il termine evento o oggetto, senza meglio specificare come questo fosse collegato ai dati. Un evento può essere pensato come una collezione di dati e quindi lo si può rappresentare come un vettore in uno spazio n-dimensionale:

$$\mathbf{x} = (x_1, ..., x_n) \tag{1}$$

In realtà l'utilizzo del termine vettore è improprio ogni qual volta si abbia a che fare con componenti (i dati) disomogenee tra loro, tuttavia lo si continuerà ad utilizzare per una questione di comodità tenendo a mente questa specifica. A questo punto risulta evidente la necessità di trattare questi eventi attraverso processi multi-variati.
Bisogna aggiungere che è possibile che i dati (e quindi le componenti del vettore) siano tra loro correlati: in questa situazione è possibile ridurre la dimensionalità dello spazio di cui si è parlato precedentemente da n a d (con d<n).

1.3 Machine Learning

L'approccio classico all'analisi dei dati prevede la disponibilità di un modello matematico, che dipende da una serie di parametri incogniti. Questi parametri vengono ricavati a partire dai dati sperimentali attraverso processi che possono essere sia analitici che numerici. Quando si parla di machine learning la prospettiva viene ribaltata, perché il modello matematico non è noto a priori.
Bisogna distinguere tre macro-tipologie di approccio all'analisi dati nel machine learning:

- APPRENDIMENTO SUPERVISIONATO
  In questa tipologia di apprendimento vengono presentati al computer degli input di esempio ed i relativi output desiderati, con lo scopo di apprendere una relazione generale che lega gli input con gli output; in questo caso si utilizza il così detto "training data set", mentre per testare il modello ottenuto si considera il "test data set" dove non vengono forniti al computer gli output. L'apprendimento supervisionato verrà trattato in maniera più approfondita nel prossimo paragrafo.

- APPRENDIMENTO NON SUPERVISIONATO
  In questo caso non vengono forniti al computer gli output attesi fin dalle prime fasi di apprendimento del modello e quindi lo scopo è quello di scoprire una qualche struttura fra i dati di input.

- APPRENDIMENTO PER RINFORZO
  Il Reinforcement Learning è basato sul concetto di ricompensa, ovvero si permette all'algoritmo di esplorare un così detto ambiente e, in base all'azione compiuta, gli si fornisce un feedback positivo, negativo o indifferente. Un esempio classico prevede di voler addestrare un algoritmo per un particolare gioco: si farà in modo di fargli compiere una serie di partite in maniera iterativa e gli si assegnerà una ricompensa in caso di vittoria o un malus in caso di sconfitta.

Una ulteriore distinzione che è necessario fare è fra algoritmi di classificazione, regressione e clustering:

- CLASSIFICAZIONE
  Gli algoritmi di classificazione sono caratterizzati da un output discreto, cioè una serie di classi alle quali l'input può appartenere.Questa tipologia di meccanismo viene in genere portata avanti tramite metodi di apprendimento supervisionato. Un esempio di algoritmo di classificazione è quello che permette di distinguere se un particolare oggetto è presente o meno in un'immagine.

- REGRESSIONE
  La regressione è simile alla classificazione con la differenza che, in questo caso, l'output è continuo. Anche gli algoritmi di regressione sono adatti ad essere trattati con metodologie di apprendimento supervisionato.

- CLUSTERING
  Nel clustering l'obiettivo è sempre quello di dividere gli input in delle classi, tuttavia in questo caso tali classi non sono stabilite a priori. La natura di algoritmi di

questo tipo li rende adatti ad essere trattati tramite metodi di apprendimento non supervisionato.

2 Apprendimento supervisionato

In questa sezione viene portata avanti una descrizione più approfondita e formale dell'apprendimento supervisionato.
Come già accennato precedentemente, quando si parla di apprendimento supervisionato si hanno a disposizione sia gli input **x** che i corrispettivi target di output **y**; esisterà quindi una funzione **y** = f(**x**) che mette in relazione gli input con gli output. Tuttavia, come detto, tale funzione è incognita ed è quindi ciò che viene ricercato con l'algoritmo di apprendimento. Nella pratica si cerca di approssimare la funzione agendo su una serie di parametri $\boldsymbol{\theta}$, quindi si avrà un qualcosa del tipo: **y'** = f'(**x**,$\boldsymbol{\theta}$).
Per ogni vettore **x** del training data set è possibile definire una particolare funzione detta "Loss function" L(**y**,f(**x**,$\boldsymbol{\theta}$)); a questo punto è possibile fare una media della funzione di costo sull'intero set di dati a disposizione, ottenendo la funzione di rischio:

$$R(\boldsymbol{\theta}) = \frac{1}{N}\sum_{k=1}^{N} L(\mathbf{y}, f(\mathbf{x}, \boldsymbol{\theta})) \tag{2}$$

dove N è il numero di eventi del training data set.
Un esempio di funzione di rischio molto diffusa è l'errore quadratico medio:

$$R(\boldsymbol{\theta}) = \frac{1}{N}\sum_{k=1}^{N} (\mathbf{y}_k - f(\mathbf{x}_k, \boldsymbol{\theta}))^2 \tag{3}$$

Quando si addestra un modello si vuole inoltre evitare il così detto overfitting, ovvero il fatto che il modello si è adattato troppo bene ai dati del training data set, non raggiungendo la generalità richiesta. Un modo per verificare un eventuale overfitting è quello di verificare se il modello è nettamente migliore per il data set di allenamento rispetto al data set di test.
Per arginare questo problema è possibile modificare la funzione di rischio, definendo la funzione di costo:

$$C(\boldsymbol{\theta}) = R(\boldsymbol{\theta}) + \lambda Q(\boldsymbol{\theta}) \tag{4}$$

con λ parametro. A questo punto l'obiettivo è quello di minimizzare la funzione di rischio (o di costo in caso di overfitting) e per fare ciò esistono diversi metodi, fra i quali il più comune è il metodo di discesa del gradiente.

2.1 Discesa del gradiente

La discesa del gradiente è una tecnica di ottimizzazione utilizzata per minimizzare l'errore che si introduce stimando la **y'** = f'($\mathbf{x}$,$\boldsymbol{\theta}$) rispetto alla funzione "vera" $\mathbf{y}$ = f($\mathbf{x}$).
Si consideri quindi una Loss function L($\mathbf{y}$,f($\mathbf{x}$,$\boldsymbol{\theta}$)) ed un vettore dei parametri $\boldsymbol{\theta}$ con i quali è possibile calcolare:

$$\mathbf{G} = \frac{1}{N}\sum_{k=1}^{N} \boldsymbol{\nabla}_{\theta} L(\mathbf{y}, f(\mathbf{x}, \boldsymbol{\theta})) \tag{5}$$

Una volta calcolato $\mathbf{G}$ è possibile aggiornare il vettore dei parametri $\boldsymbol{\theta}$ nel modo seguente:

$$\boldsymbol{\theta} - \epsilon\mathbf{G} \rightarrow \boldsymbol{\theta} \tag{6}$$

Qui ϵ prende il nome di passo ed ha il ruolo di calibrare di quanto debba essere modificato il vettore $\boldsymbol{\theta}$ nella direzione opposta a quella del gradiente $\mathbf{G}$.
Per completezza si riporta il fatto che, quando si ha un elevato campione nel training data set, si utilizza il metodo di discesa del gradiente stocastico, che è strutturato allo stesso modo di quello appena descritto ma viene limitato ad un sottoinsieme del data set di allenamento per una questione di lunghezza di calcolo.

3 Metodi di Machine Learning

In questa sezione verranno presentate, in maniera descrittiva, le più popolari metodologie di machine learning.

3.1 Grid Search

Prima di parlare del Grid Search è necessario introdurre il concetto di **iperparametro**. Come detto nelle sezioni precedenti, un modello di apprendimento è caratterizzato da una serie di parametri che vengono modificati in maniera iterativa in modo da minimizzare la Loss function e, come noto, tale processo avviene attraverso un continuo confronto con il training data set. Quando si parla di iperparametri si intende invece una serie di parametri che caratterizzano il modello implementato che non sono modificati nel processo di addestramento con il training data set ma vengono prestabiliti dall'utente.
Chiaramente al variare degli iperparametri cambia anche la qualità del processo di apprendimento del modello e quindi anche gli iperparametri devono essere sottoposti ad un processo di ottimizzazione. A questo punto entra in gioco il metodo del Grid Search che è appunto un metodo di ottimizzazione degli iperparametri.
Il Grid Search è piuttosto semplice sia da comprendere concettualmente sia da implementare nella pratica; fa parte dei così detti "Brute-Force Search", cioè di quei metodi che si basano sulla sistematica verifica di tutte le possibili soluzioni ad un problema per poi andare a considerare la migliore. Per esempio si consideri il problema di dover andare a cercare i divisori di un numero n: un approccio "Brute-Force" prevedrebbe di considerare tutti i numeri minori di n e verificare quelli per i quali la divisione non dà resto. Questo esempio permette anche di mettere in evidenza il limite principale di tale tipologia di approccio: il numero di possibilità da esplorare può aumentare molto velocemente, soprattutto se si considera un processo multivariato.
Tornando ora nello specifico al Grid Search, si consideri un modello caratterizzato da un numero k di iperparametri. Si può definire, in analogia a ciò che è stato fatto con i parametri, un vettore le cui componenti sono appunto gli iperparametri:

$$\boldsymbol{\mu} = (\mu_1, ..., \mu_k) \tag{7}$$

Tale vettore apparterrà ovviamente ad uno spazio k-dimensionale, sul quale può essere costruita una griglia i cui nodi corrispondono a particolari combinazioni degli iperparametri.
A questo punto si può avviare l'apprendimento del modello per ogni particolare configurazione degli iperparametri ed ottenere un valore per la Loss function. Si arriva allora ad avere una valore della Loss per ogni nodo della griglia e quindi basta considerare quello per il quale la Loss è minore, ottenendo la miglior configurazione degli iperparametri.

4 Introduction

4.1 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:

$$Loss_{tot} = Loss_{reco} + \beta D_{KL} \tag{8}$$

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 4.3.1. 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 capabi-

lities. 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.

4.2 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.

Tabella 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).

Tabella 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

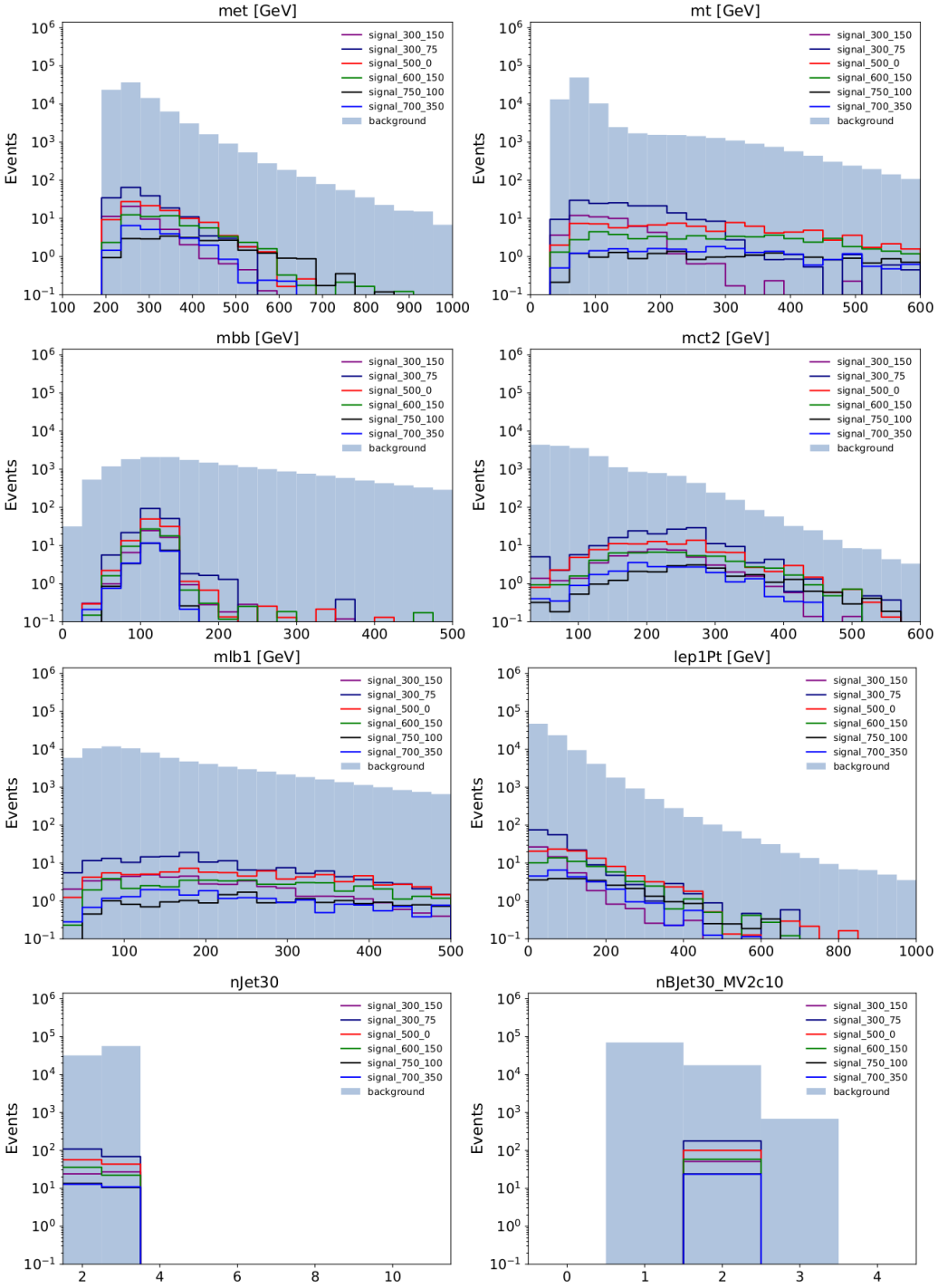

Figura 1: Distributions of the input variables for the sum of all the background processes and some signal points, as example.

4.3 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.

4.3.1 Loss function

The training loss function described above (eq. 8) 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:

$$D_{KL} = \frac{1}{2k} \sum_{i,j=1} \left(\sigma_p^j \sigma_z^{i,j}\right)^2 + \left(\frac{\mu_p^j \mu_z^{i,j}}{\sigma_p^j}\right)^2 + ln\frac{\sigma_p^j}{\sigma_z^{i,j}} - 1 \tag{9}$$

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:

$$\begin{aligned} Loss_{reco} &= -\frac{1}{k} \sum_{i,j=1} \ln\left[P\left(x|\alpha_1, \alpha_2, ...\alpha_n\right)\right] \\ &= -\frac{1}{k} \sum_{i,j=1} \ln\left[f_{i,j}\left(x_{i,j}|\alpha_1^{i,j}, \alpha_2^{i,j}, ...\alpha_n^{i,j}\right)\right] \end{aligned} \tag{10}$$

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*:

$$P\left(x|\alpha_1,\alpha_2,\alpha_3\right) = \begin{cases} \alpha_3\delta\left(x\right)\frac{1-\alpha_3}{x\alpha_2+\sqrt{2\pi}}e^{-\frac{(\ln x-\alpha)^2}{2\alpha_2^2}} & for x \geq 10^{-4} \\ 0 & \leq 10^{-4} \end{cases} \tag{11}$$

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

$$\Theta\left(x\right)\left[erf\left(\frac{n+0.5-\alpha_1}{\alpha_2+\sqrt{2}}\right)-erf\left(\frac{n-0.5-\alpha_1}{\alpha_2+\sqrt{2}}\right)\right] \tag{12}$$

  where the normalization factor N is set to:

$$N = \frac{1}{2}\left(\frac{-0.5-\alpha_1}{\alpha_2+\sqrt{2}}\right) \tag{13}$$

4.3.2 Model architecture

The network architecture is briefly described in the 4.1. 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: 11, 12): 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.

[Tur38]

Riferimenti bibliografici

[Tur38] Alan Mathison Turing. *Systems of Logic Based on Ordinals: a Dissertation.* Ph.D. dissertation, Cambridge University, Cambridge, UK, 1938.