

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

Submitted by:

Schiazza Filippo Antonio

Co-supervisor: (optional)

Prof./Dr. Name Surname

Anno accademico 2019/2020

Indice

Elenco delle figure	II
Elenco delle tabelle	III
1 Introduzione	1
1.1 Tipologie di analisi dati	2
1.2 Processi multi-variati	2
2 Apprendimento supervisionato	3
2.1 Discesa del gradiente	5
3 Metodi di Machine Learning	6
3.1 Iperparametri e Grid Search	6
3.2 Analisi discriminante lineare	9
3.3 Reti Neurali	10
3.4 Alberi Decisionali	15
4 Introduction	16
4.1 Model description	16
4.2 Dataset	17
4.3 Training	19
4.3.1 Loss function	19
4.3.2 Model architecture	20
References	22
Riferimenti bibliografici	22

Elenco delle figure

1	si riporta uno schema intuitivo del funzionamento di un algoritmo di apprendimento supervisionato	3
2	la figura illustra visivamente l'esito di un processo di ottimizzazione degli iperparametri attraverso il metodo Grid Search (<i>Knuth: Computers and Typesetting</i>)	7
3	risultato grafico di un processo di Random Grid Search per la separazione del segnale dal fondo in un processo bi-variato. L'immagine è presa da Bhat, 2011.	8
4	si riporta un esempio grafico di rete neurale formata da un unico strato nascosto. L'immagine è presa da Bhat, 2011.	10
5	Illustrazione della struttura di un neurone (<i>Appunti di reti neurali</i>).	11
6	Si riportano due sigmoidi, dove per quella in rosso si ha $\alpha=2$ e per quella in blu $\alpha=7$	11
7	Distributions of the input variables for the sum of all the background processes and some signal points, as example.	18

Elenco delle tabelle

1	Preselection cuts applied both on signal and background samples.	17
2	Requirements for the three selected regions.	17

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 Hadron Collider è l'esempio principale. Il run3 del Large Hadron 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.

L'applicazione più frequente è però rappresentata dalla distinzione e quindi, dalla classificazione, degli eventi fisici nelle rispettive categorie di background/segnale

;

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:

$$\mathbf{x} = (x_1, \dots, x_n) \quad (1)$$

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 $d < n$).

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 \mathbf{x} che i corrispettivi target di output \mathbf{y} ; esisterà quindi una funzione $\mathbf{y} = f(\mathbf{x})$ 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: $\mathbf{y}' = f'(\mathbf{x}, \boldsymbol{\theta})$.

SCHEMA APPRENDIMENTO SUPERVISIONATO

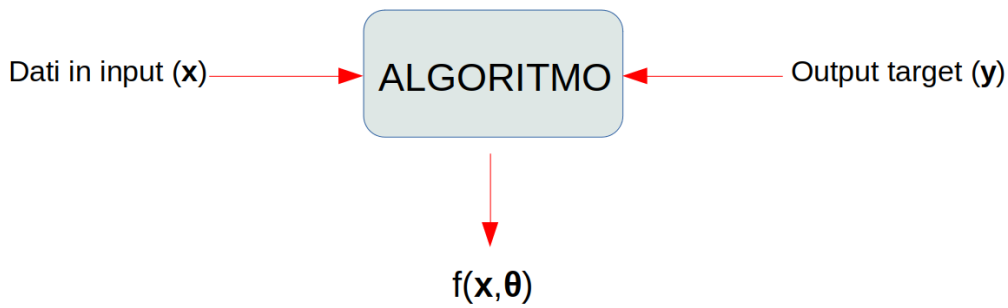


Figura 1: si riporta uno schema intuitivo del funzionamento di un algoritmo di apprendimento supervisionato

Per ogni vettore \mathbf{x} del training data set è possibile definire una particolare funzione detta "Loss function" $L(\mathbf{y}, f(\mathbf{x}, \boldsymbol{\theta}))$; a questo punto è possibile fare una media della funzione di perdita sull'intero set di dati a disposizione, ottenendo la funzione di rischio:

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

dove N è il numero di eventi del training data set.

Un esempio di funzione di rischio molto diffusa è l'errore quadratico medio:

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

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:

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

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 $\mathbf{y}' = f'(\mathbf{x}, \boldsymbol{\theta})$ 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:

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

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

$$\boldsymbol{\theta} - \epsilon \mathbf{G} \rightarrow \boldsymbol{\theta} \quad (6)$$

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 Iperparametri e 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 anch'essi 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 considerare la migliore. Per esempio si consideri il problema di dover 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:

$$\boldsymbol{\mu} = (\mu_1, \dots, \mu_k) \quad (7)$$

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 un 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.

In Figura 2 nella pagina seguente è riportato per chiarezza un esempio visivo dell'esito di un processo di ottimizzazione degli iperparametri attraverso il metodo Grid Search.

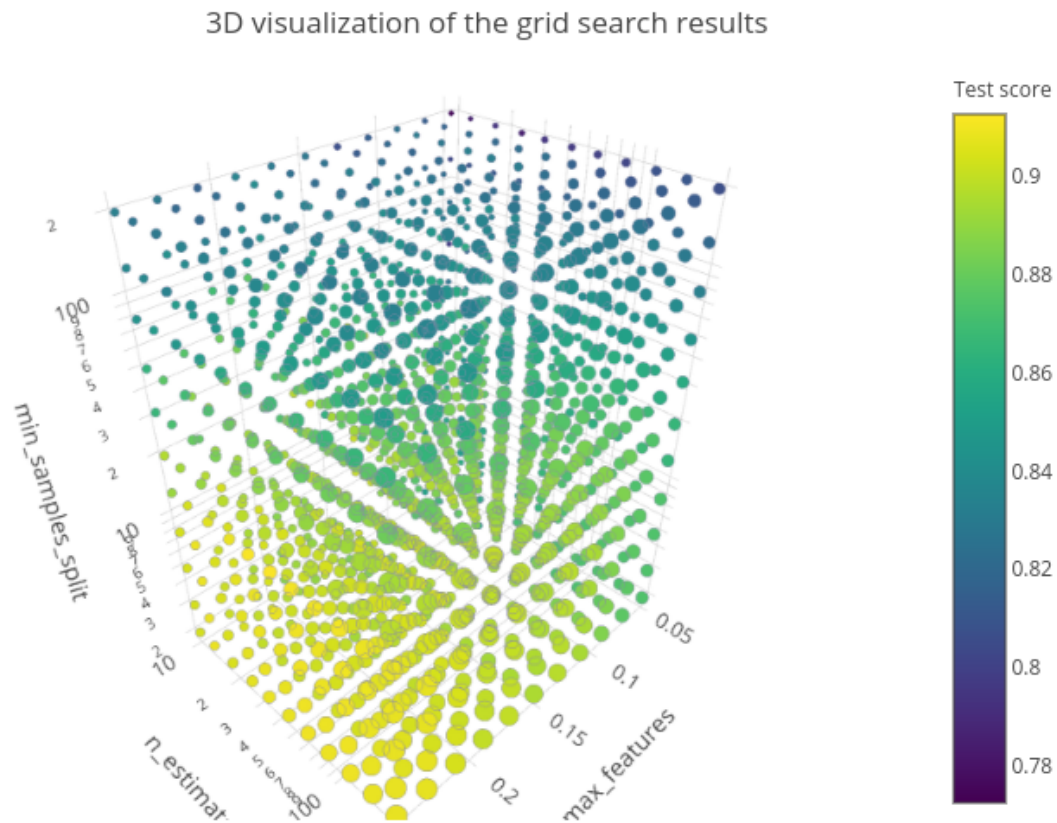


Figura 2: la figura illustra visivamente l'esito di un processo di ottimizzazione degli iperparametri attraverso il metodo Grid Search (*Knuth: Computers and Typesetting*)

Come accennato precedentemente, man mano che aumenta la complessità del modello è molto probabile che aumenti il numero degli iperparametri e quindi la dimensionalità dello spazio introdotto precedentemente; ciò implica l'aumento considerevole del numero di configurazioni degli iperparametri da esplorare attraverso il Grid Search e quindi il tempo necessario per concludere l'ottimizzazione.

E' possibile ovviare parzialmente a questo problema attraverso il Random Grid Search (RGS), dove non sono considerati tutti i nodi della griglia, ma solo una loro parte selezionata in maniera casuale secondo una particolare distribuzione (ciò permette anche di tener conto di conoscenze pregresse).

Come esempio si riporta un problema molto comune nel campo della fisica delle alte energie, ovvero la separazione del segnale dal fondo. Si consideri, per semplicità, un processo bi-variato $\mathbf{x} = (x_1, x_2)$ dove, per raggiungere l'obiettivo prefissato di separazione fra segnale e fondo, è necessario applicare un così detto taglio, ovvero stabilire un punto nello spazio bidimensionale $\mathbf{P} = (P_1, P_2)$ che permette di dividere lo stesso in due porzioni, come illustrato in Figura 3.

Chiaramente la scelta di tale punto deve essere fatta in modo da ottimizzare la separazione segnale-fondo e quindi si può utilizzare il metodo del Grid Search o meglio il RGS per le ragioni presentate precedentemente.

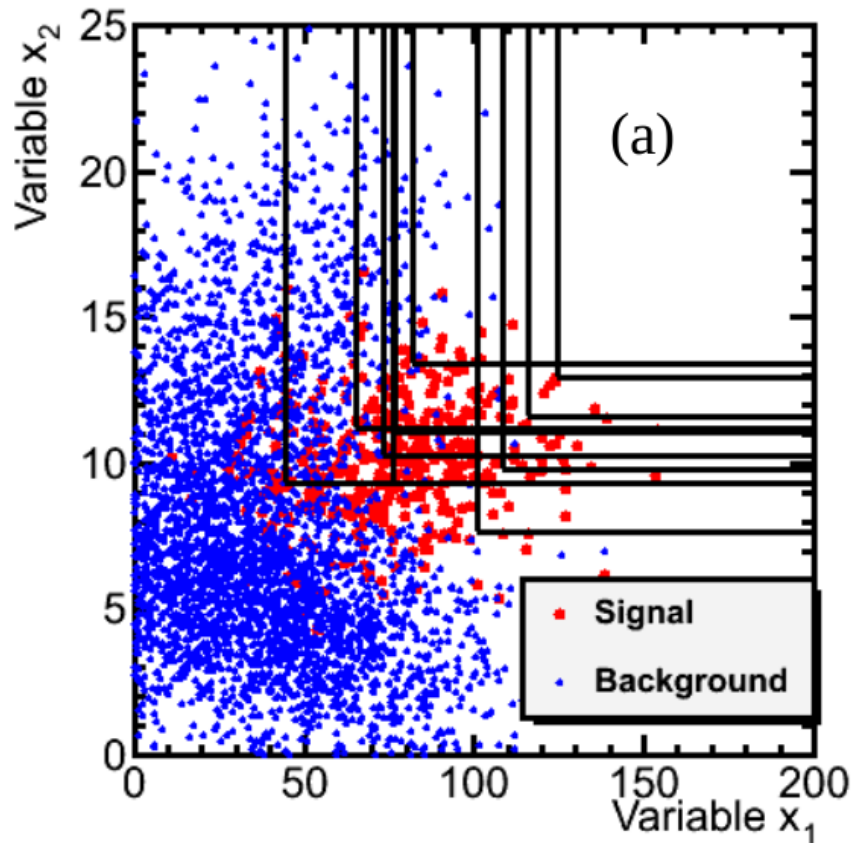


Figura 3: risultato grafico di un processo di Random Grid Search per la separazione del segnale dal fondo in un processo bi-variato. L'immagine è presa da Bhat, [2011](#).

Come si evince dalla figura, un notevole limite del Grid search in questo caso particolare (e della sua variante RDG) sta nell'essere vincolati a dei tagli paralleli agli assi e quindi ci si trova di fronte ad un risultato piuttosto limitante. Si possono introdurre a questo punto i Metodi Lineari, continuando a seguire il percorso già accennato in precedenza, ovvero partire dal metodo concettualmente e praticamente più semplice ma meno efficiente per poi salire mano mano di complessità e di efficienza.

3.2 Analisi discriminante lineare

Come detto nella sezione precedente, il GS ed il RDG possono essere utilizzati per la separazione del segnale dal fondo ma hanno dei limiti piuttosto considerevoli che sono già stati illustrati. L'analisi discriminante è un metodo che permette di raggiungere lo stesso obiettivo di separazione, ma in modo più efficiente.

L'analisi discriminante si definisce lineare quando la funzione classificatrice è, appunto, lineare.

Si immagini di avere a disposizione un determinato set di eventi in input \mathbf{x}_i , ciascuno caratterizzato da un numero n di variabili (spazio n -dimensionale) e di volerli ripartire fra segnale e fondo.

Si definisce la funzione discriminante lineare nel seguente modo:

$$D(x_1, x_2, \dots, x_n) = c_0 + c_1x_1 + \dots + c_nx_n = c_0 + \sum_{i=1}^n c_ix_i \quad (8)$$

quindi come una combinazione lineare delle componenti del vettore che rappresenta l'evento; il valore assunto dalla funzione per ogni singolo evento ne permette la separazione nelle due classi (nel presente caso segnale e fondo), utilizzando un valore di riferimento D_0 .

A questo punto l'obiettivo è quello di massimizzare la distanza fra le due classi, ovvero rendere massima la differenza dei valori assunti dalla funzione $D(\mathbf{x})$ fra gli eventi appartenenti al fondo e quelli relativi al segnale.

Un esempio di questo approccio è il metodo proposto da Fisher: si consideri un campione di eventi appartenenti al segnale e se ne definisca la media μ_s e la deviazione standard σ_s ed un campione appartenente al fondo, definendo anche qui la media μ_f e la deviazione standard σ_f . A questo punto la migliore configurazione dei parametri è quella che massimizza la seguente funzione:

$$F(\mathbf{c}) = \frac{(\mu_s - \mu_f)^2}{\sigma_s^2 + \sigma_f^2} \quad (9)$$

3.3 Reti Neurali

La struttura di una rete neurale prevede la presenza di unità fondamentali, dette neuroni, che sono organizzate in strati e legate fra di loro mediante delle connessioni (sinapsi), ciascuna delle quali è caratterizzata da un peso. Sono proprio questi pesi a giocare un ruolo fondamentale nel processo di apprendimento della rete perché sono loro i parametri soggetti a modifica.

Il nome rete neurale (artificiale) deriva dal fatto che la loro struttura è ispirata dalle corrispondenti strutture biologiche (seppur di molto semplificata).

In una rete neurale è sempre presente uno strato di input ed uno di output, mentre il numero di livelli nascosti può variare a seconda della complessità della rete; In figura 4 è riportato un esempio di rete neurale con un singolo strato interno nascosto.

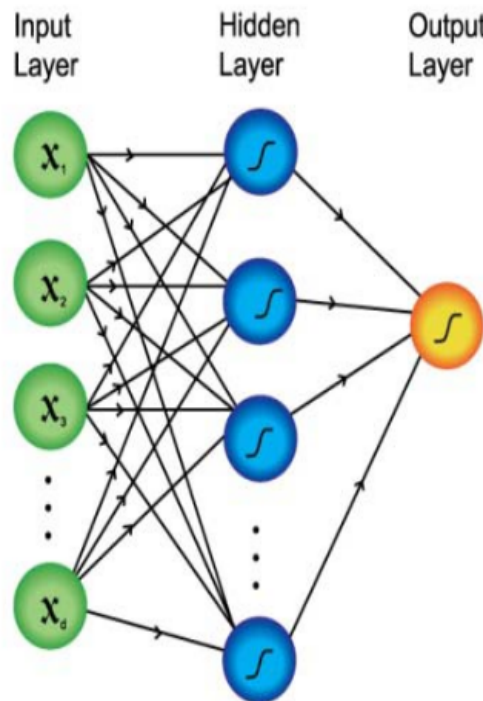


Figura 4: si riporta un esempio grafico di rete neurale formata da un unico strato nascosto. L'immagine è presa da Bhat, 2011.

Si può passare ora a presentare il modello del singolo neurone per capire com'è strutturato e quale compito svolge. Gli elementi che caratterizzano il singolo neurone sono:

1. Una serie di connessioni in ingresso (ciascuna caratterizzata da un proprio peso);
2. Un sommatore che ha il compito di svolgere la somma pesata degli input, utilizzando i pesi caratteristici delle connessioni;
3. Un output e la relativa funzione di attivazione, che viene usata per limitarne l'ampiezza (tipicamente ad intervalli $[0,1]$ o $[-1,1]$);
4. Un valore di soglia che viene usato per aumentare o diminuire il valore ottenuto dalla somma pesata.

Si riporta in figura 5 lo schema grafico di un singolo neurone (k), dove $\mathbf{x} = (x_1, \dots, x_m)$ è il vettore degli input, $\mathbf{w}_k = (w_{k1}, \dots, w_{km})$ è il vettore dei pesi, $\phi(x)$ è la funzione di attivazione, b_k è il valore di soglia e y_k è l'output.

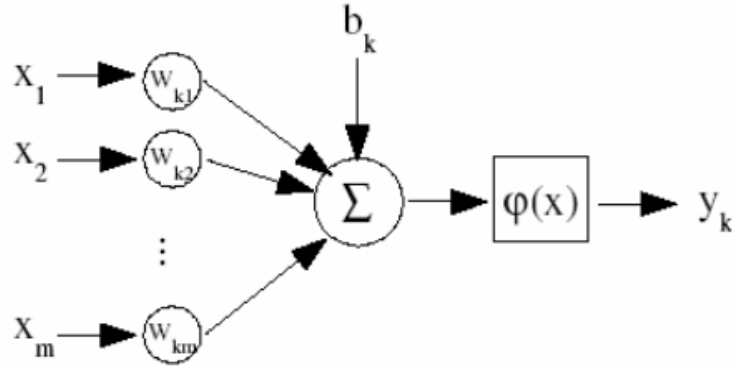


Figura 5: Illustrazione della struttura di un neurone (*Appunti di reti neurali*).

Quindi il neurone opera la seguente somma pesata:

$$s_k = \mathbf{x} \bullet \mathbf{w}_k = \sum_{i=1}^m x_i w_{ki} \quad (10)$$

e si ottiene l'output attraverso la funzione di attivazione:

$$y_k = \phi(s_k + b_k) \quad (11)$$

Risulta utile spendere qualche parola in più sul tipo di funzione di attivazione più utilizzata, ovvero la funzione sigmoide:

$$\text{sig}(x) = \frac{1}{1 + e^{-\alpha x}} \quad (12)$$

dove α è un parametro che permette di regolare la pendenza della curva, come si evince dalla figura 6

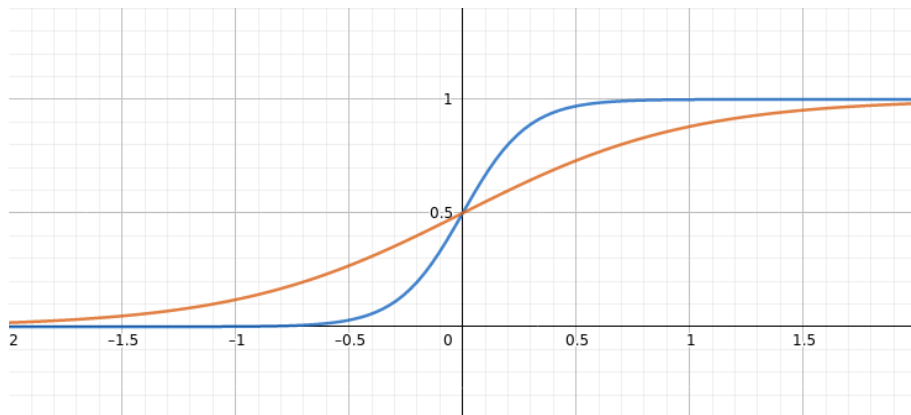


Figura 6: Si riportano due sigmoidi, dove per quella in rosso si ha $\alpha=2$ e per quella in blu $\alpha=7$.

Un singolo neurone è l'ingrediente fondamentale di una rete neurale e nella gran parte dei casi non è in grado di svolgere da solo nessun compito interessante ma deve sempre essere inserito in una rete. Tuttavia esiste un caso particolare nel quale un singolo neurone può portare a termine un compito di classificazione; affinché ciò sia possibile è necessario che i vettori evento siano riconducibili a due sole categorie e che il loro spazio possa essere separato (in relazione alle due categorie) da un singolo iper-piano. In questo caso esiste un teorema di convergenza che garantisce appunto la convergenza dei pesi nel processo di addestramento.

A questo punto è possibile passare ad un livello di complessità superiore, osservando in che modo possono essere organizzati i neuroni per formare la rete neurale; si distinguono due tipologie di reti:

1. Reti feedforward con uno o più strati: in questo caso il segnale si propaga dai nodi di input verso quelli di output, senza connessioni fra i neuroni di uno stesso strato;
2. Reti feedback: sono reti cicliche dove il segnale si propaga anche fra i neuroni di uno stesso strato.

Bisogna chiedersi ora in che modo apprende il singolo neurone. Una delle possibilità (nel caso in cui sia noto l'output target) è l'apprendimento con correzione di errore, che viene presentato velocemente nel seguito. Si consideri un singolo neurone che ha in ingresso una serie di input (x_1, \dots, x_n) e quindi produce un valore di output y attraverso la somma pesata già introdotta precedentemente; tale valore può quindi essere confrontata con il risultato atteso R , ottenendo così un errore $err = R - y$; si può quindi definire la funzione di costo

$$E = \frac{1}{2}err^2 \quad (13)$$

sulla quale si applicherà il metodo di discesa del gradiente già discusso nel paragrafo 2.1 per ottimizzare i parametri.

Una volta capito in che modo avviene l'addestramento di un singolo neurone, è possibile trattare le diverse funzioni che può svolgere una rete neurale:

1. L'associazione, a sua volta divisibile in autoassociazione ed eteroassociazione. Nel primo caso vengono presentati alla rete neurale una serie di vettori evento nella fase di training per poi verificare se uno di questi vettori, ripresentato parzialmente, viene nuovamente riconosciuto dalla rete e completato (tale funzione ben si presta ad essere ottenuta a seguito di un processo di apprendimento non supervisionato); nel secondo caso si utilizza un vettore non ancora noto alla rete neurale come richiamo di una già processato;
2. Il riconoscimento consiste nell'associazione da parte della rete di un vettore evento ad una delle varie categorie possibili. Tale obiettivo può essere ottenuto a seguito di una fase di addestramento dove vengono forniti alla rete sia i vettori in input che le categorie alle quali questi appartengono (si tratta chiaramente di un processo di apprendimento non supervisionato). Si ipotizzi di avere a disposizione dei vettori evento con un numero n di componenti (i dati) e, chiaramente, possono essere pensati come dei punti in uno spazio n -dimensionale; questo spazio potrà essere allora diviso in delle regioni che corrispondono alle varie categorie di cui si è parlato

precedentemente ed i confini di queste zone si ottengono a seguito del processo di addestramento;

3. L'approssimazione di funzioni, dove si hanno a disposizione gli input \mathbf{x}_i ed i corrispondenti output \mathbf{y}_i . Quello che si cerca di fare è approssimare al meglio la funzione $\mathbf{y} = f(\mathbf{x})$ vera con una $g(\mathbf{x})$, tale per cui la distanza euclidea è inferiore ad un valore prefissato positivo (piccolo) ϵ :

$$\|g(\mathbf{x}) - f(\mathbf{x})\| < \epsilon \quad (14)$$

Arrivati a questo punto è possibile esporre la trattazione su come una rete neurale viene addestrata. Precedentemente si è introdotta la struttura di una rete neurale, specificando le differenze fra lo strato di input, quello di output e gli strati nascosti. Ogni singolo neurone nel suo processo di addestramento deve aggiornare i suoi pesi, in modo che l'output della rete neurale sia simile a quello atteso.

Uno dei metodi migliori per addestrare la rete neurale è l'algoritmo di back-propagation. Una rete neurale è caratterizzata da due tipologie di segnale: da un lato vi è un segnale di funzione che si propaga dallo strato di input verso quello di output e, dall'altro, vi è un segnale di errore che ha origine nello strato di output e si propaga verso quello di input. E' il segnale di errore a giocare un ruolo fondamentale nel processo di apprendimento tramite ottimizzazione dei pesi che caratterizzano la rete neurale.

Addentrando nell'algoritmo di back-propagation bisogna fare una distinzione fra il modo in cui esso viene applicato allo strato di output ed il modo in cui viene applicato agli strati nascosti:

1. Neurone nello strato di output.

Si consideri uno strato di output con un numero n di neuroni e ci si focalizzi sul k -esimo. In un certo momento del processo di apprendimento, alla rete neurale si starà presentando il j -esimo elemento del training data set, quindi per il neurone k si otterrà il seguente segnale di errore:

$$err_k^{(j)} = R_k^{(j)} - y_k^{(j)} \quad (15)$$

dove con la lettera y si intende il valore ottenuto in output dal neurone e con R il valore atteso.

L'errore totale dello strato di output per il vettore evento j -esimo viene definito nel seguente modo:

$$E^{(j)} = \frac{1}{2} \sum_{k=1}^n (err_k^{(j)})^2 \quad (16)$$

Se poi N è il numero totale di elementi del training data set, allora la funzione di costo può essere definita nel seguente modo:

$$E_{tot} = \frac{1}{N} \sum_{j=1}^N E^{(j)} \quad (17)$$

e l'obiettivo è quello di minimizzare tale funzione di costo. Per fare ciò si procede aggiustando i pesi a seguito della presentazione di ogni singolo vettore evento. Si utilizza il metodo di discesa del gradiente, procedendo nel seguente modo:

il gradiente è dato da

$$\frac{\partial E^{(j)}}{\partial w_{ki}^{(j)}} \quad (18)$$

e gli aggiornamenti del peso vengono applicati nel verso opposto del gradiente, ovvero

$$\Delta w_{ki}^{(j)} = -\mu \frac{\partial E^{(j)}}{\partial w_{ki}^{(j)}} \quad (19)$$

con μ fattore di apprendimento. Manca a questo punto il calcolo esplicito del gradiente, che può essere eseguito con la regola della catena

$$\frac{\partial E^{(j)}}{\partial w_{ki}^{(j)}} = \frac{\partial E^{(j)}}{\partial err_k^{(j)}} \frac{\partial err_k^{(j)}}{\partial y_k^{(j)}} \frac{\partial y_k^{(j)}}{\partial S_k^{(j)}} \frac{\partial S_k^{(j)}}{\partial w_{ki}^{(j)}} \quad (20)$$

dove $S_k^{(j)} = s_k^{(j)} + b_k^{(j)}$ (si faccia riferimento all'equazione (10)).

Una volta calcolate le quattro derivate si ottiene:

$$\frac{\partial E^{(j)}}{\partial w_{ki}^{(j)}} = -err_k^{(j)} \phi'(S_k^{(j)}) y_i^{(j)} \quad (21)$$

e quindi:

$$\Delta w_{ki}^{(j)} = err_k^{(j)} \phi'(S_k^{(j)}) y_i^{(j)} \mu \quad (22)$$

2. Neurone in uno strato nascosto

In questo caso l'output del neurone non ha un diretto valore con il quale può essere confrontato, quindi il segnale di errore deve essere determinato a partire dai segnali di errore di tutti i neuroni dello strato successivo, da cui il nome di back-propagation proprio perché il segnale di errore prosegue all'indietro dall'output verso l'input.

Come ultima considerazione sulle reti neurali bisogna sottolineare che il coefficiente di apprendimento deve essere scelto in maniera accurata, infatti se fosse troppo piccolo si avrebbe una convergenza estremamente lenta e, viceversa, un valore troppo grande porterebbe ad una instabilità con comportamento oscillatorio.

3.4 Alberi Decisionali

Un problema di classificazione può essere sempre trattato in forma di albero decisionale; applicazioni in sequenza di una serie di test, fatti sugli attributi l'albero decisionale non è altro che una funzione che a partire dal vettore di input, le cui componenti in questo caso le chiamo attributi, a seguito di una sequenza di test restituisce un singolo valore di output

gli attributi in input possono essere sia continui che discreti (caso più semplice)

output a due valori -> classificatore booleano (o sì o no, o segnale o fondo)

differenza tra nodo interno, dove avviene il test sul valore di uno degli attributi e gli archi che ne dipartono corrispondono ai possibili valori e nodo foglia (terminale) che mi dà il possibile risultato finale del processo decisionale

come posso capire a partire dall'abbinamento fra vettore degli attributi e output l'ordine con il quale avvengono i test? Si vanno a cercare di volta in volta i criteri più discriminanti -> servirà un algoritmo ricorsivo

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 together 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:

$$Loss_{tot} = Loss_{reco} + \beta D_{KL} \quad (23)$$

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 7 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 $p_T > 30\text{GeV}$	True
b -tagged jet	[1-3]
met	$> 220\text{ GeV}$
mt	$> 50\text{ GeV}$

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 – 3b region
Exactly 1 signal lepton	True	True	True
met trigger fired	True	True	True
2 – 3 jets with $p_T > 30\text{GeV}$	True	True	True
b -tagged jet	[1-3]	[1-3]	[2-3]
met	$> 220\text{ GeV}$	$> 220\text{ GeV}$	$> 220\text{ GeV}$
mt	$> 50\text{ GeV}$	$> 50\text{ GeV}$	$> 50\text{ GeV}$
mbb		[100 – 140] GeV	[100 – 140] GeV
mct		$> 100\text{ GeV}$	$> 100\text{ GeV}$

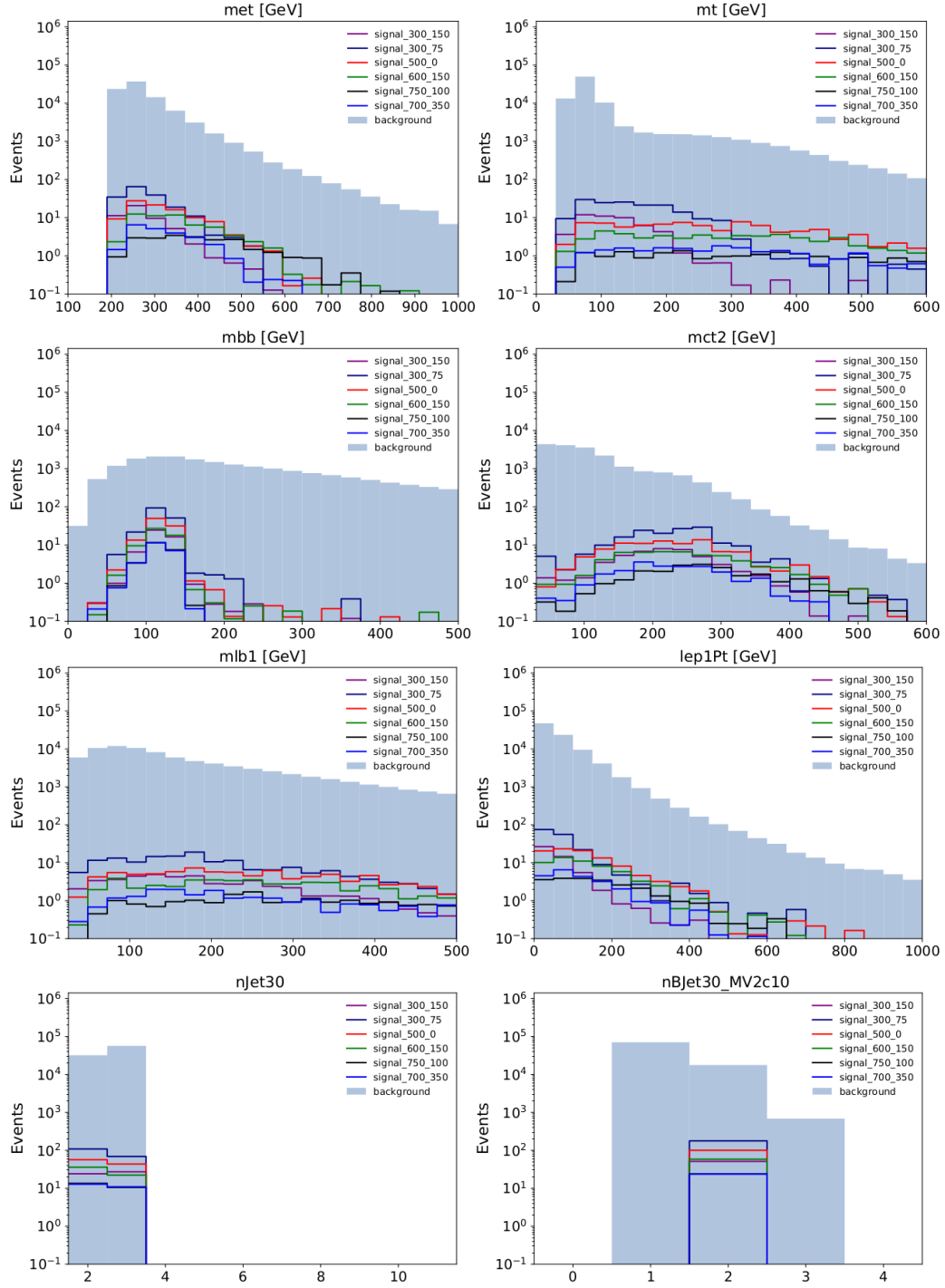


Figure 7: Distributions of the input variables for the sum of all the background processes and some signal points, as example.

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 - 3b 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. 23) 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-Liebr 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:

$$D_{KL} = \frac{1}{2k} \sum_{i,j=1} (\sigma_p^j \sigma_z^{i,j})^2 + \left(\frac{\mu_p^j \mu_z^{i,j}}{\sigma_p^j} \right)^2 + \ln \frac{\sigma_p^j}{\sigma_z^{i,j}} - 1 \quad (24)$$

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:

$$\begin{aligned} Loss_{reco} &= -\frac{1}{k} \sum_{i,j=1} \ln [P(x|\alpha_1, \alpha_2, \dots \alpha_n)] \\ &= -\frac{1}{k} \sum_{i,j=1} \ln [f_{i,j}(x_{i,j}|\alpha_1^{i,j}, \alpha_2^{i,j}, \dots \alpha_n^{i,j})] \end{aligned} \quad (25)$$

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*:

$$P(x|\alpha_1, \alpha_2, \alpha_3) = \begin{cases} \alpha_3 \delta(x) \frac{1-\alpha_3}{x\alpha_2 + \sqrt{2}\pi} e^{-\frac{(\ln x - \alpha)^2}{2\alpha_2^2}} & \text{for } x \geq 10^{-4} \\ 0 & \leq 10^{-4} \end{cases} \quad (26)$$

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

$$\Theta(x) \left[\text{erf} \left(\frac{n + 0.5 - \alpha_1}{\alpha_2 + \sqrt{2}} \right) - \text{erf} \left(\frac{n - 0.5 - \alpha_1}{\alpha_2 + \sqrt{2}} \right) \right] \quad (27)$$

where the normalization factor N is set to:

$$N = \frac{1}{2} \left(\frac{-0.5 - \alpha_1}{\alpha_2 + \sqrt{2}} \right) \quad (28)$$

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: 26, 27): 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 **vae: Cerri_2019**.

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 training, 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.

Riferimenti bibliografici

- Bhat, Pushpa (2011). “Advanced Analysis Methods in Particle Physics”. In: *Annual Review of Nuclear and Particle Science*.
- Bulò, Samuel Rota. *Appunti di reti neurali*. URL: <https://www.dsi.unive.it/~srotabul/files/AppuntiRetiNeurali.pdf>.
- Einstein, Albert (1905). “Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]”. In: *Annalen der Physik* 322.10, pp. 891–921. DOI: <http://dx.doi.org/10.1002/andp.19053221004>.
- Knuth, Donald. *Knuth: Computers and Typesetting*. URL: <https://towardsdatascience.com/using-3d-visualizations-to-tune-hyperparameters-of-ml-models-with-python-ba2885eab2e9>.
- Nilsson, Nils J. (1998). *Introduction to Machine Learning*. Department of Computer Science, Stanford University.