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 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 ed un'altra applicazione molto frequente è la distinzione degli eventi fisici nelle categorie di segnale e background.

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). Verrà posto un focus particolare sul problema della dimensionalità degli input della sezione 5.1, che servirà da trampolino di lancio per affrontare un metodo particolare del ML, il Variational Autoencoders.

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.Come si vedrà, questo modello viene addestrato per identificare le caratteristiche peculiari degli eventi fisici di background in modo da contrapporli successivamente a quelle relative ad eventi di segnale.
  Verrà posta particolare attenzione su un particolare metodo di apprendimento non supervisionato, il Variational Autoencoder (VAEs), del quale verrà svolta una trattazione teorica approfondita ed una applicazione al campo della fisica delle particelle

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

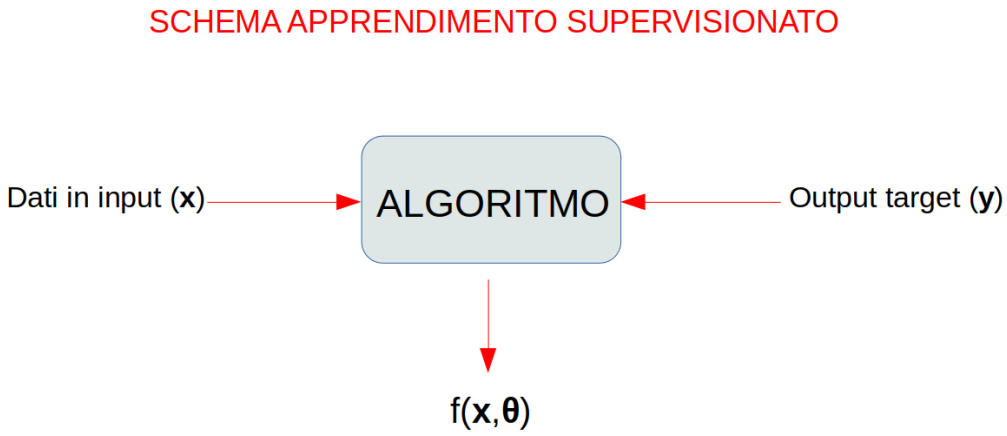

Figura 1: si riporta uno schema intuitivo del funzionamento di un algoritmo di apprendimento supervisionato

Per ogni vettore **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:

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

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 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:

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

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.
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.

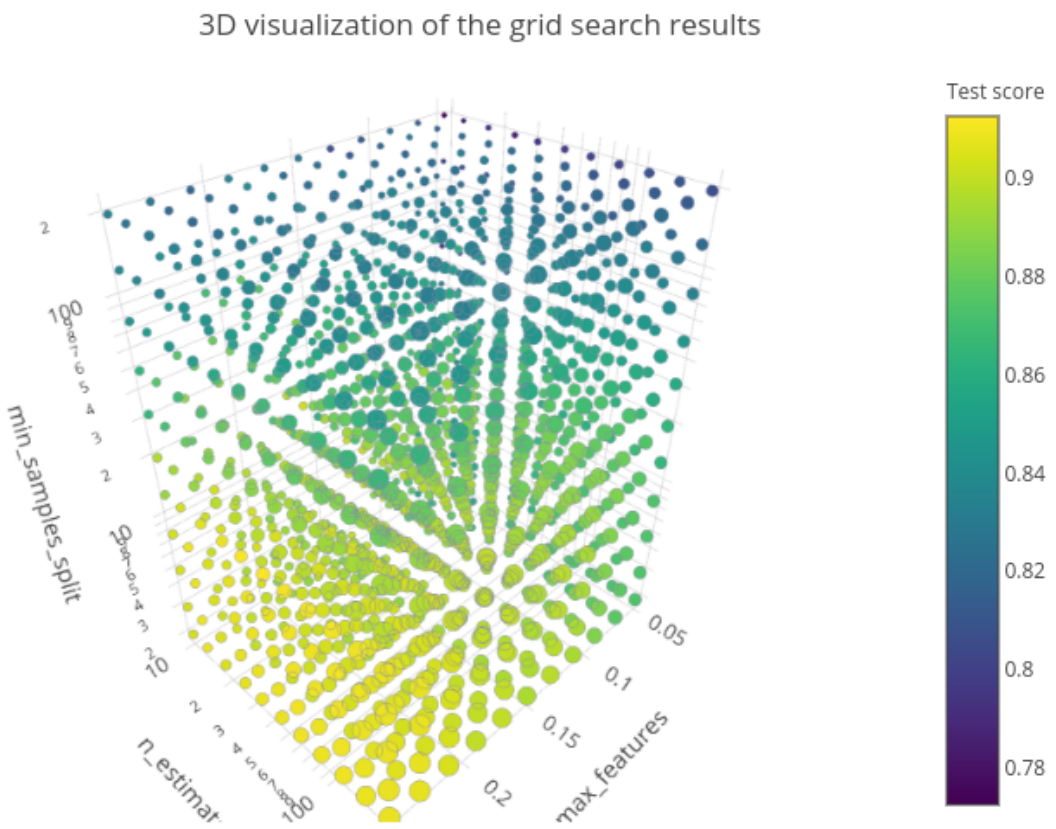

Figura 2: la figura illustra visivamente l'esito di un processo di ottimizzazione degli iperparametri attraverso il metodo Grid Search (*Knuth: Computers and Typesetting*)

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.

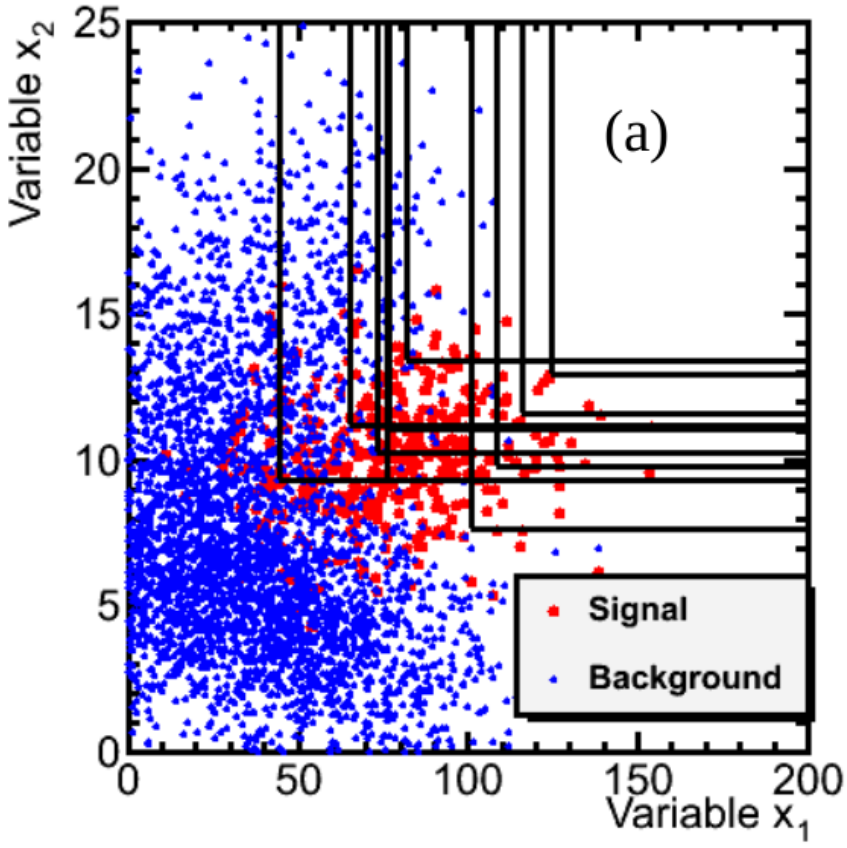

Figura 3: risultato grafico di un processo di Random Grid Search per la separazione del segnale dal fondo in un processo bi-variato. L'immagine è presa da Bhat, 2011.

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 difronte 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:

$$D(x_1, x_2, ..., x_n) = c_0 + c_1 x_1 + ... + c_n x_n = c_0 + \sum_{i=0}^{n} c_i x_i \tag{8}$$

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 $\boldsymbol{\mu}_\mathbf{s}$ e la deviazione standard σ_s ed un campione appartenente al fondo, definendo anche qui la media $\boldsymbol{\mu}_\mathbf{f}$ e la deviazione standard σ_f. A questo punto la migliore configurazione dei parametri è quella che massimizza la seguente funzione:

$$F(\mathbf{c}) = \frac{(\boldsymbol{\mu}_\mathbf{s} - \boldsymbol{\mu}_\mathbf{f})^2}{\sigma_s^2 + \sigma_f^2} \tag{9}$$

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 inspirata 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.

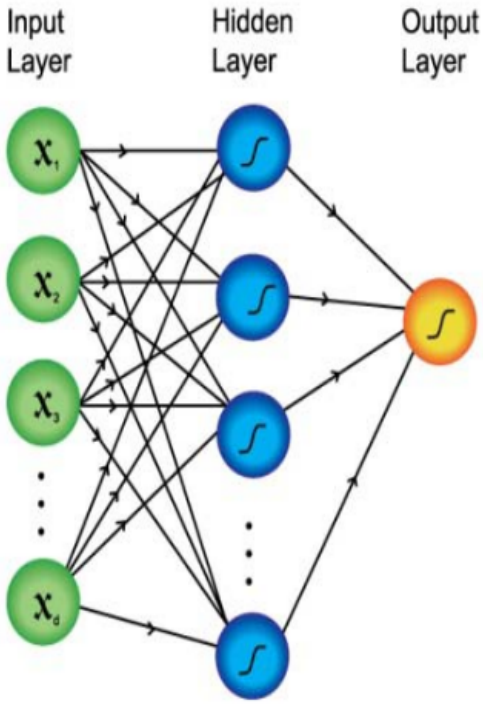

Figura 4: si riporta un esempio grafico di rete neurale formata da un unico strato nascosto. L'immagine è presa da Bhat, 2011.

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 $\mathbf{x} = (x_1, ..., x_m)$ è il vettore degli input input, $\mathbf{w}_k = (w_{k1}, ..., w_{km})$ è il vettore dei pesi, ϕ(x) è la funzione di attivazione, b_k è il valore di soglia e y_k è l'output.

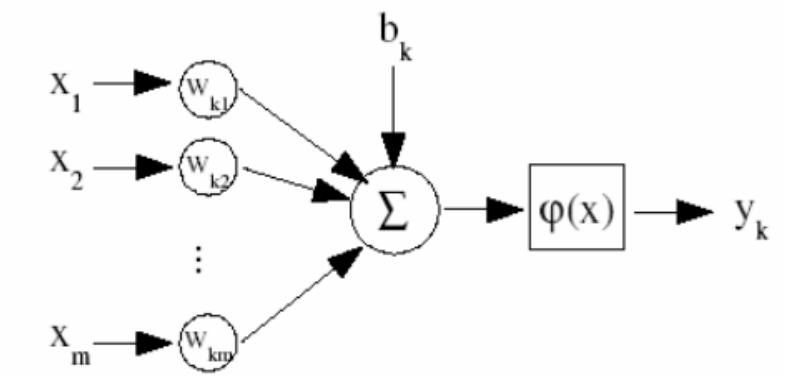

Figura 5: Illustrazione della struttura di un neurone (*Appunti di reti neurali*).

Quindi il neurone opera la seguente somma pesata:

$$s_k = \mathbf{x} \bullet \mathbf{w_k} = \sum_{i=1}^{m} x_i w_{ki} \tag{10}$$

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

$$y_k = \phi(s_k + b_k) \tag{11}$$

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

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

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

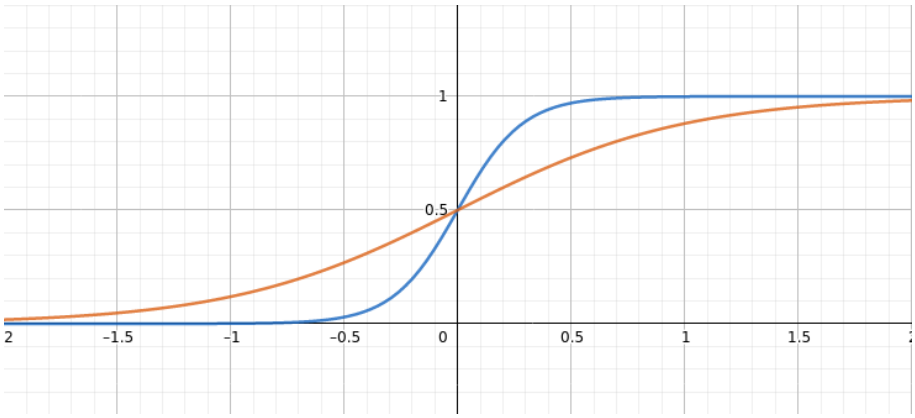

Figura 6: Si riportano due sigmoidi, dove per quella in rosso si ha α=2 e per quella in blu α=7.

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 una 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 $(x_1, ..., x_n)$ 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

$$E = \frac{1}{2} err^2 \tag{13}$$

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 corrispettivi 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:

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

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.
Addentrandosi 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:

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

   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:

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

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

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

   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

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

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

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

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

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

dove $S_k^{(j)} = s_k^{(j)} + b_k^{(j)}$ (si faccia riferimento all'equazione (10)).
Una volta calcolate le quattro derivate si ottiene:

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

e quindi:

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

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

Gli alberi decisionali rappresentano uno dei metodi più intuitivi del Machine Learning perché è possibile avere una semplice rappresentazione grafica del meccanismo del loro funzionamento.
Gli alberi decisionali rappresentano un mezzo estremamente interessante per le operazioni di classificazione (sia per output continui che discreti) ed operano attraverso una serie di test sugli attributi degli input (con il termine attributo si intende una componente del vettore di input).
Come primo passo è utile discutere come è strutturato un albero decisionale, introducendo alcune notazioni (come ausilio alla trattazione si riporta in figura 7 un esempio di albero decisionale molto semplice).
Gli elementi che caratterizzano un albero decisionale sono:

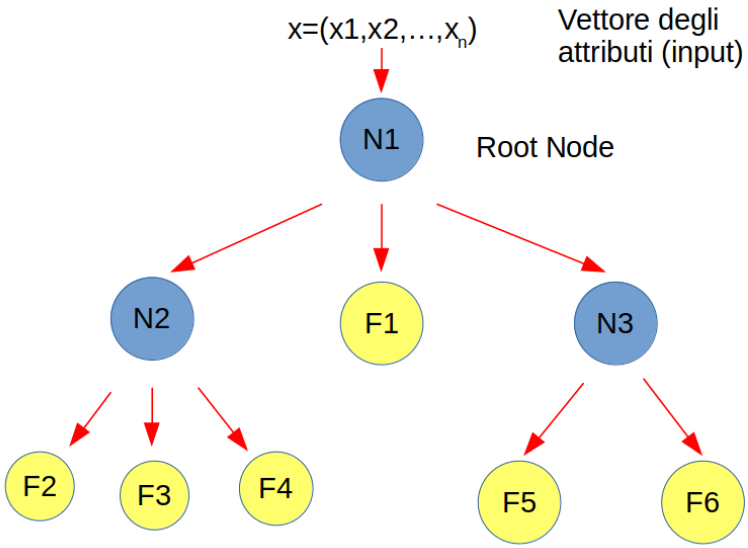

Figura 7: esempio di come è strutturato un albero decisionale

- Nodo.
  I nodi sono riportati in figura 7 come dei cerchi colorati in azzurro e contrassegnati dalla lettera N. Ogni nodo si occupa di eseguire un test su di un singolo attributo (il nodo iniziale dove avviene il primo test e quindi la prima differenziazione degli input è detto ”root node”);

- Ramo.
  I rami (detti anche archi) determinano le regole di ”splitting”, ovvero le regole attraverso le quali vengono separati i vettori in input a seconda dei loro attributi; tali rami determinano quindi i percorsi all'interno degli alberi decisionali e quindi, in ultima analisi, la classificazione finale;

- Foglie.
  Le foglie sono una classe particolare di nodi, ovvero i nodi finali che, in quanto tali, non generano nuove diramazioni ma rappresentano il risultato finale del processo di classificazione.

Bisogna tenere a mente che si sta parlando di una metodologia del ML, che è quindi volta ad estrarre dai dati di input (training data set) e dai corrispettivi output target l'informazione generale, per poter poi applicare l'algoritmo a casi per i quali non si è in possesso degli output di riferimento.
Detto ciò è necessario capire in che modo possa essere costruito un albero decisionale e l'idea di base è quella di stabilire, di volta in volta, il criterio sugli attributi che più discrimina gli input.
Nella costruzione degli alberi decisionali bisogna distinguere due fasi successive:

- "Building", ovvero costruzione.
  In questo prima fase l'obiettivo è quello di far crescere l'albero in dimensione, quindi in termini di rami e nodi per avere un numero adeguato di regole di splitting così da ottenere una classificazione in classi omogenee nella fase finale. In questa prima si ottiene un albero particolarmente folto e quindi probabilmente soggetto all'overfitting (come primo argine a ciò è possibile introdurre un criterio d'arresto capace di fermare la crescita dell'albero al realizzarsi di particolari condizioni);

- "Pruning", ovvero potatura.
  Questa fase è quella che permette di evitare l'overfitting perché vengono eliminati i rami che non contribuiscono in maniera significativa al processo di classificazione.

Come già detto la prima fase è quella di "Building", durante la quale viene costruito l'albero decisionale aggiungendo nodi e rami, con il fine di ottenere una classificazione finale il più possibile omogenea; è altresì noto che ad ogni nodo corrisponde un attributo sul quale è effettuato un test, quindi è evidente che, dato un particolare numero di attributi, il numero di alberi possibili è molto elevato. Bisogna trovare un modo per disporre nella maniera più efficace i nodi all'interno dell'albero: l'idea è quella di scegliere per primo l'attributo attraverso il quale si ha una maggiore discriminazione dei dati in input. Per fare ciò si possono percorrere due strade distinte, utilizzando:

- Coefficiente di impurità di Gini.

- Guadagno informativo.

E' chiaro che entrambi questi indici vengono utilizzati con la stessa finalità, tuttavia ogni algoritmo ha una differente logica di costruzione e quindi adotterà uno solo dei due indici, con la possibilità di ottenere risultati differenti.
A questo punto l'albero decisionale è stato costruito e quindi si deve passare alla fase di "pruning", con l'obiettivo di ridurre le dimensioni dell'albero per evitare l'ormai noto overfitting. Per fare ciò le strade sono due, infatti da un lato si può seguire un approccio "top-down", partendo dalla radice e suddividendo l'intera struttura in sotto alberi e dall'altro un approccio "bottom-up", partendo dalle foglie ed analizzando l'impatto di ogni singola potatura; è altresì possibile introdurre nella fase di costruzione stessa un criterio di "early-stopping" richiedendo un valore minimo di miglioramento dell'algoritmo fra un'iterazione e l'altra.

In conclusione bisogna sottolineare che gli alberi decisionali sono particolarmente utilizzati per i seguenti motivi:

- semplicità nell'interpretazione e nella visualizzazione;
- tolleranza ad eventuali attributi mancanti per alcuni input nel training data set o nel test data set;
- insensibilità ad eventuali attributi irrilevanti nella classificazione;
- invarianza per trasformazioni monotone effettuate sugli attributi, che rende la fase di pre- processamento dei dati non necessaria.

Gli alberi decisionali hanno tuttavia una serie di limiti elencati nelle righe che seguono:

- instabilità rispetto al variare del training data set, cioè data set di allenamento di poco differenti fra loro producono risultati molto diversi;
- frequente problema dell'overfitting.

4 Apprendimento non supervisionato

Nel capitolo precedente è stata mostrata l'utilità degli algoritmi di apprendimento supervisionato, osservando che, nel caso in cui si abbiano a disposizione sia i vettori di input che i corrispettivi output target, si può ottenere un'approssimazione della relazione esistente input-output. Tuttavia non è sempre possibile avere a disposizione gli output target e bisogna capire se è comunque possibile ottenere informazioni utili dai dati.
Come già accennato nelle prime pagine di questa trattazione quando non si hanno a disposizione gli output target si possono applicare tecniche di apprendimento non supervisionato, dove l'obiettivo è quello di trovare eventuali partizioni degli input (Clustering). Si consideri la figura 8 dove sono riportate tre diverse configurazioni possibili nel caso di input bidimensionali: è evidente che nel caso a) sia possibile la separazione in due sotto gruppo e nel caso b) in un unico sotto gruppo, mentre nel caso c) sembrerebbe non si possano stabilire graficamente eventuali separazioni.

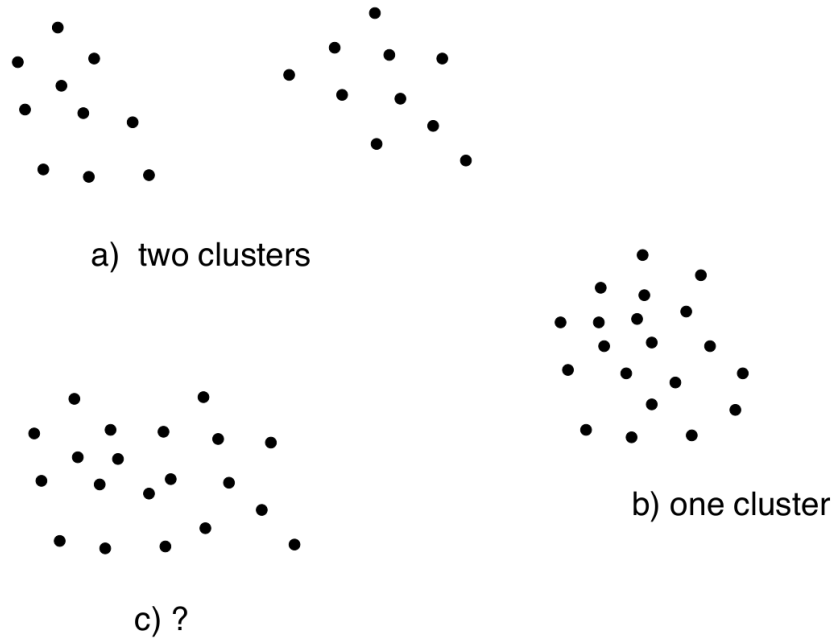

Figura 8: vettori di input in uno spazio bidimensionale in tre situazioni differenti. L'immagine è presa da Nilsson, 1998

Quindi un algoritmo di clustering si occupa della suddivisione del set di input Σ in un numero N di sottogruppi Σ_1,...,Σ_n, detti appunto cluster; si noti che lo stesso numero N non viene stabilito a priori e fornito all'algoritmo, ma viene anch'esso ricavato a partire dai dati. Una volta fatto sarà possibile implementare un classificatore per collegare nuovi vettori di input con i cluster precedentemente individuati.
Inoltre, aumentando il livello di complessità, è possibile trovare eventuali gerarchie di partizionamento, ovvero cluster di cluster.

4.1 Metodo di clustering basato sulla distanza euclidea

Gli algoritmi di apprendimento non supervisionato sfruttano una qualche misura di similarità per separare i pattern (gli input) nei vari cluster. Una possibilità è quella di utilizzare la semplice distanza euclidea per poter separare lo spazio n-dimensionale dei pattern in delle sotto-aree, che sono appunto i cluster.
Per fare ciò viene implementato un metodo iterativo, basato sulla definizione di alcuni punti particolari nello spazio dei pattern, detti "cluster seekers" (letteralmente "cercatori di cluster").
Si definiscono M punti nello spazio n-dimensionale $\mathbf{C_1}, ..., \mathbf{C_M}$ e l'obiettivo è quello di fare in modo che ogni punto si muova verso il centro di ogni singolo cluster, in modo che ogni cluster abbia al suo centro uno di questi cluster seekers.
Come è già stato spiegato precedentemente, l'algoritmo non conosce a prescindere il numero di cluster ma riesce a ricavarlo dai pattern stessi; per questa ragione il numero di cluster seekers M è inizialmente casuale ed esiste un procedura per ottimizzarlo, che verrà illustrata in seguito.
I pattern del training data set Σ vengono presentati all'algoritmo uno alla volta: per ognuno di essi ($\mathbf{x_i}$) si cerca il cluster seekers più vicino ($\mathbf{C_k}$) e lo si sposta verso $\mathbf{x_i}$ nel seguente modo:

$$\mathbf{C_k} + \alpha_k(\mathbf{x_i} - \mathbf{C_k}) \rightarrow \mathbf{C_k} \tag{23}$$

dove α_k è un parametro di apprendimento che determina di quanto il cluster seeker k-esimo si muove verso il punto $\mathbf{x_i}$.
A questo punto è utile fare in modo che più il cluster seeker è soggetto a spostamenti minore diventa l'entità dello spostamento. Per fare ciò si definisce una massa m_k e le si assegna un valore pari al numero di volte in cui $\mathbf{C_k}$ è stato soggetto a spostamenti (quindi anche il valore della massa verrà aggiornato di volta in volta); dopodiché si assegna ad α_k il seguente valore

$$\alpha_k = \frac{1}{1 + m_k} \tag{24}$$

e, dato che ad ogni iterazione che coinvolge $\mathbf{C_k}$ il valore di m_k aumenta di una unità, il parametro di apprendimento α_k diminuisce di volta in volta.
Il risultato di questo aggiustamento è che il cluster seeker si trova sempre nel punto che rappresenta la media dei punti del cluster.
Una volta che sono stati presentati tutti i pattern del training data set all'algoritmo, i vari cluster seeker saranno conversi ai "centri di massa" dei cluster e la classificazione (cioè la delimitazione dei cluster nello spazio n-dimensionale) può essere fatta con una partizione dello spazio di Voronoi, di cui si riporta la seguente definizione:

> *In ogni insieme (topologicamente) discreto S di punti in uno spazio euclideo e per quasi ogni punto x, c'è un punto in S che è il più vicino a x. Il "quasi" è una precisazione necessaria dato che alcuni punti x possono essere equidistanti da 2 o più punti di S. Se S contiene solo due punti, a e b, allora il luogo geometrico dei punti equidistanti da a e b è un iperpiano, ovvero un sottospazio affine di codimensione 1. Tale iperpiano sarà il confine tra l'insieme di tutti punti più vicini ad a che a b e l'insieme di tutti i punti più vicini a b che ad a. È l'asse del segmento ab. In generale, l'insieme dei punti più vicini a un punto c ∈ S che ad ogni altro punto di S è la parte interna di un politopo (eventualmente privo di bordi) detto dominio di Dirichlet o cella di Voronoi di c. L'insieme di tali politopi è una tassellatura dell'intero spazio e viene detta tassellatura di Voronoi corrispondente all'insieme*

> *S. Se la dimensione dello spazio è solo 2, è facile rappresentare graficamente le tassellazioni di Voronoi; è a questo caso che si riferisce solitamente l'accezione diagramma di Voronoi.*
> *(Wikipedia, Diagramma di Voronoi.)*

Un esempio didattico del risultato di questa partizione è riportato in figura 9.

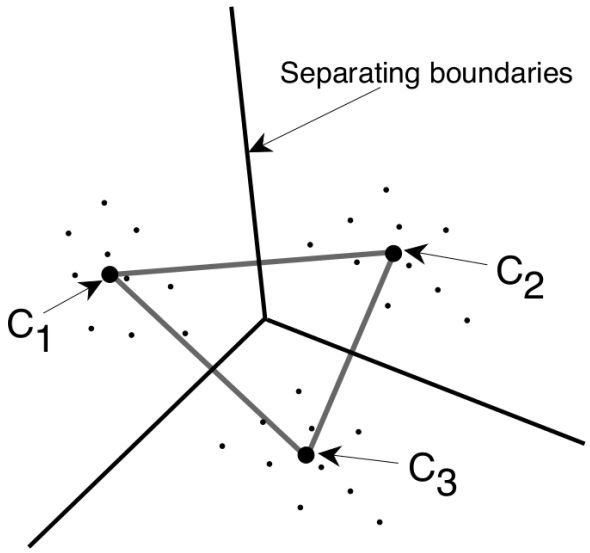

Figura 9: Si riporta un esempio di partizione dello spazio (bi-dimensionale) di Voronoi in tre sotto regioni . L'immagine è presa da Nilsson, 1998

Si è accennato qualche riga fa che il numero di cluster seeker è inizialmente scelto in maniera casuale, per poi essere ottimizzato; per il processo di ottimizzazione si utilizza la varianza dei pattern $\{\mathbf{x_i}\}$ per ogni cluster:

$$\sigma^2 = \frac{1}{L}\sum_{i=1}^{L}(\mathbf{x_i} - \boldsymbol{\mu})^2 \tag{25}$$

dove L è il numero di pattern nel cluster e $\boldsymbol{\mu}$ ne è la media:

$$\boldsymbol{\mu} = \frac{1}{L}\sum_{i=1}^{L}\mathbf{x_i} \tag{26}$$

A questo punto, se la distanza d_{ij} fra due cluster seeker $\mathbf{C_i}$ e $\mathbf{C_j}$ è minore di un determinato valore ϵ, allora si sostituiscono i due cluster seeker con uno nuovo posto nel loro centro di massa (tenendo conto delle due masse m_i e m_j); dall'altro lato, se vi è un cluster per il quale la varianza σ^2 è più grande di un valore δ, si aggiunge un nuovo cluster seeker vicino a quello già esistente e si eguagliano entrambe le loro masse a zero.
Come osservazione finale bisogna dire che nei metodi che si basano sul concetto di distanza è importante ri-scalare i valori delle componenti dei pattern (in linea di principio si possono avere componenti diverse con ordini di grandezza di molto differenti) in modo da evitare che alcune componenti pesino più di altre.

5 Variational Autoencoders (VAEs)

A questo punto si è giunti finalmente al cuore di questa trattazione, dove vengono illustrate le basi teoriche di un metodo di apprendimento non supervisionato, il Variational Autoencoders (VAEs), del quale si studierà nella prossima sezione un'applicazione al campo della fisica delle alte energie.
Il percorso seguito in questa sezione è il seguente:

- Problema della dimensionalità;
- Autoencoders;
- Variational Autoencoders.

5.1 Curse of dimensionality e riduzione della dimensionalità

Come è stato più volte detto in questa trattazione, quando si parla di input (o pattern) ci si riferisce a dei vettori, le cui componenti sono i dati veri e propri; questi vettori, in quanto tali, possono essere pensati all'interno di un opportuno spazio n-dimensionale (con n=numero di componenti del vettore).
Quando si parla di "Curse of dimensionality" (letteralmente "la maledizione della dimensionalità") ci si riferisce ad una serie di problemi che ci si trova ad affrontare quando bisogna trattare spazi con un'alta dimensionalità, che altrimenti non comparirebbero in spazi a bassa dimensionalità.
Dato che all'aumentare della dimensionalità i volumi nello spazio aumentano in maniera significativa, ci si troverà nella situazione per cui i pattern risultano sparsi nello spazio e questo è chiaramente un problema per ogni analisi che ne si vuole fare basata sulla statistica; infatti, per ottenere dei risultati significativi a livello statistico, la quantità di dati necessari aumenta in maniera esponenziale e questo risulta essere un problema a livello pratico.
Quando si parla di riduzione della dimensionalità ci si riferisce ad una serie di tecniche, attraverso le quali viene ridotto il numero delle variabili che caratterizzano i vettori di input; l'obiettivo di base è quello di proiettare gli elementi dello spazio n-dimensionale (i vettori di input) su di uno spazio a dimensione inferiore, cogliendo l'essenza stessa dei dati.
Ciò che è necessario notare è che avere degli input con più bassa dimensionalità permette di avere anche meno parametri (gradi di libertà) e quindi una struttura più semplice del modello. Il prediligere la semplicità alla complessità, oltre che per gli ovvi motivi, deriva dal fatto che la seconda è molto soggetta al fenomeno dell'overfitting.
Il processo di riduzione della dimensionalità è una metodologia di preparazione dei dati, per poi essere presentati all'algoritmo di apprendimento, che si troverà di fronte delle informazioni più compatte e quindi più facilmente processabili.
Inoltre bisogna notare che, se il processo di riduzione della dimensionalità viene svolto sul training data set, allora deve essere attuato anche sul test data set, per garantire un processo di verifica valido.

Il processo di riduzione della dimensionalità può essere portato avanti attraverso due metodologie differenti:

- Selezione, dove solo alcune componenti dei vettori di input vengono conservate;
- Estrazione, dove viene creato un numero ridotto di nuove componenti a partire da quelle originali.

A prescindere da questa distinzione, bisogna sottolineare che tutti i processi di riduzione della dimensionalità hanno una struttura comune, ovvero sono caratterizzati da una fase di *encoding* (che rappresenta il vero e proprio processo di riduzione della dimensionalità) e da una fase di *decoding*, nella quale si verifica quanta informazione è stata persa nel processo.
Si riporta in figura 10 l'illustrazione grafica del processo *encoding-decoding*

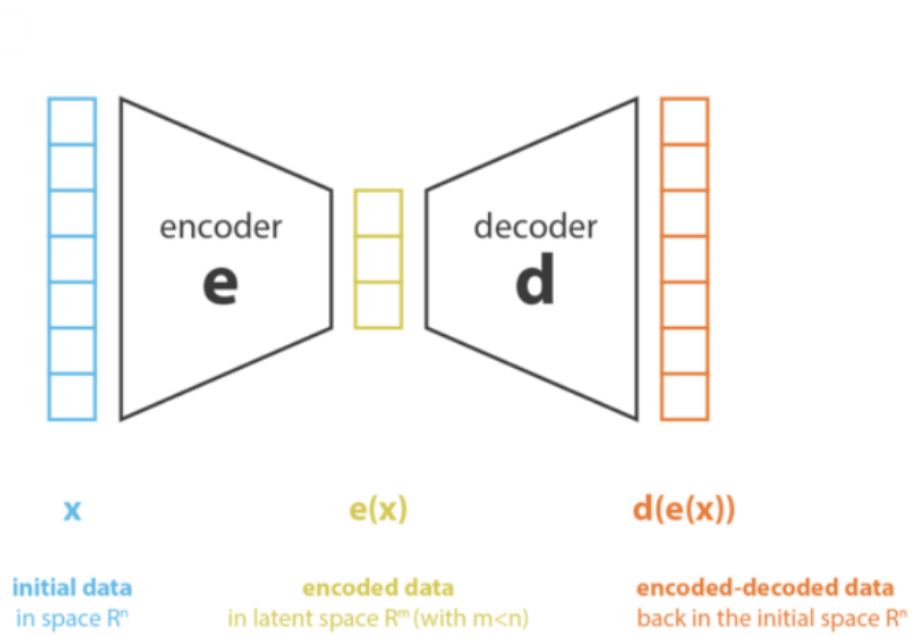

Figura 10: Strutture generale di un processo di riduzione della dimensionalità . L'immagine è presa da *Understanding Variational Autoencoders (VAEs)*

Il vettore di input $\mathbf{x}$ (n-dimensionale) viene compresso dall'encoder $\mathbf{e}$ in un vettore $\mathbf{e}(\mathbf{x})$ di uno spazio m-dimensionale (con m<n), detto *spazio latente*; l'encoder, come detto, può agire per selezione o per estrazione.
Il decoder $\mathbf{d}$ svolge la funzione opposta, ovvero decomprime il vettore $\mathbf{e}(\mathbf{x})$ in $\mathbf{d}(\mathbf{e}(\mathbf{x}))$ per tornare allo spazio originario n-dimensionale.
Nel caso in cui $\mathbf{x} = \mathbf{d}(\mathbf{e}(\mathbf{x}))$ (caso ideale) si dice che il processo è un *lossless encoding*, ovvero non c'è stata perdita di informazioni nella riduzione della dimensionalità; viceversa, se $\mathbf{x} \neq \mathbf{d}(\mathbf{e}(\mathbf{x}))$, si parla di un *lossy encoding*, cioè un processo nel quale parte dell'informazione viene persa e non può essere recuperata con la fase di decoding.

Come conseguenza di ciò che è stato appena illustrato, l'obiettivo di un processo di riduzione della dimensionalità è quello di trovare la coppia encoder-decoder (e,d) fra una famiglia di encoder E e di decoder D, che minimizzi l'informazione persa:

$$(e, d) = \min_{E \times D} \epsilon(\mathbf{x}, \mathbf{d}(\mathbf{e}(\mathbf{x}))) \tag{27}$$

dove $\epsilon(\mathbf{x}, \mathbf{d}(\mathbf{e}(\mathbf{x})))$ è la grandezza attraverso la quale viene quantificata la quantità di informazione persa nel processo di riduzione.

A questo punto è possibile illustrare le varie metodologie di riduzione della dimensionalità, secondo la distinzione già incontrata fra selezione ed estrazione.
I metodi di selezione ("Feature Selection Methods (FSM)") sono metodi attraverso i quali vengono selezionate le componenti dei vettori di input da tenere e quelle da eliminare perché irrilevanti per le analisi successive; a proposito dei

6 Introduction

6.1 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:

$$Loss_{tot} = Loss_{reco} + \beta D_{KL} \tag{28}$$

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 6.3.1. 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.

6.2 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 11 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

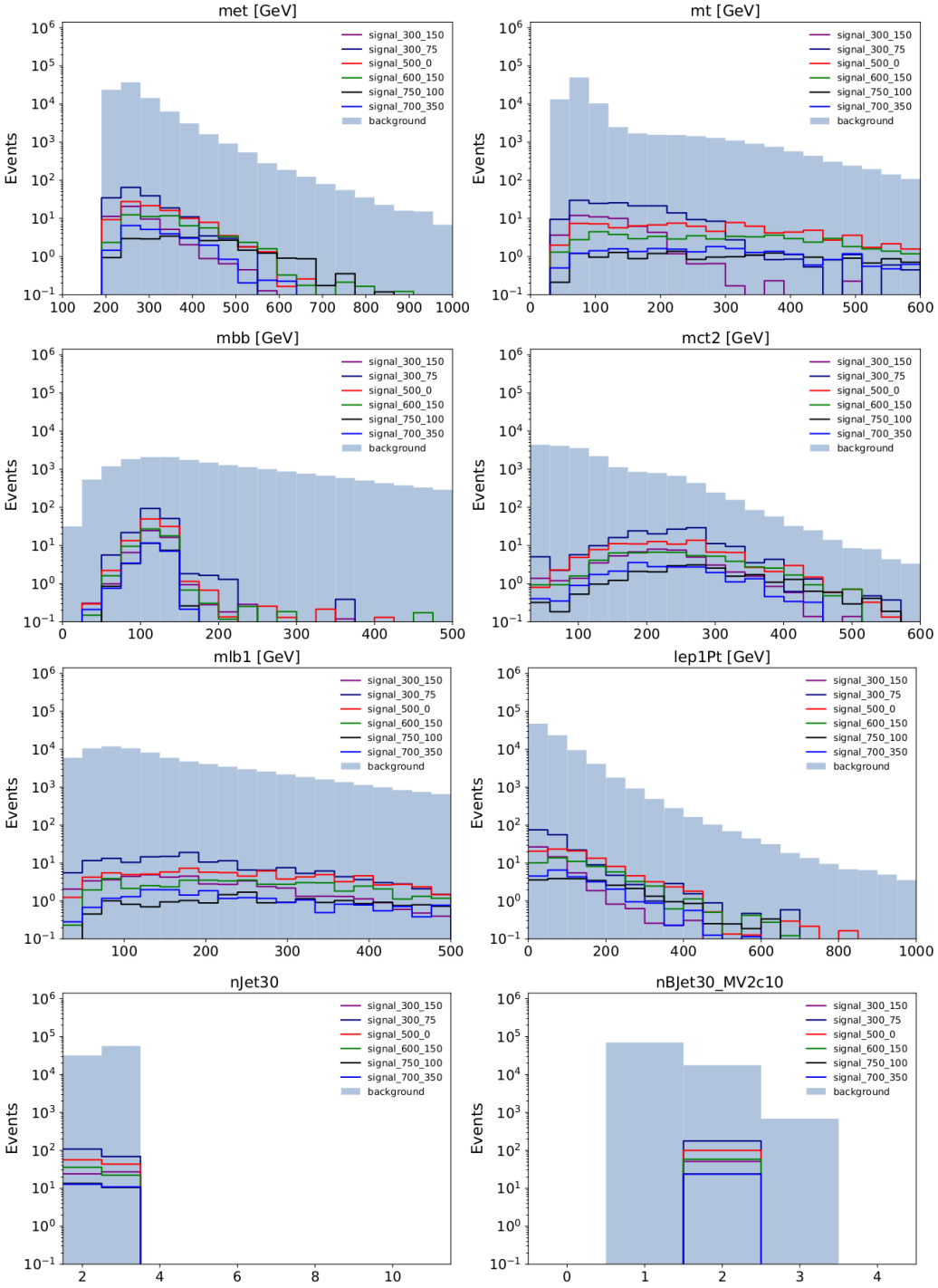

Figura 11: Distributions of the input variables for the sum of all the background processes and some signal points, as example.

6.3 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.

6.3.1 Loss function

The training loss function described above (eq. 28) 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:

$$D_{KL} = \frac{1}{2k} \sum_{i,j=1} \left(\sigma_p^j \sigma_z^{i,j}\right)^2 + \left(\frac{\mu_p^j \mu_z^{i,j}}{\sigma_p^j}\right)^2 + ln \frac{\sigma_p^j}{\sigma_z^{i,j}} - 1 \tag{29}$$

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:

$$\begin{aligned} Loss_{reco} &= -\frac{1}{k} \sum_{i,j=1} \ln \left[P\left(x|\alpha_1, \alpha_2, ...\alpha_n\right)\right] \\ &= -\frac{1}{k} \sum_{i,j=1} \ln \left[f_{i,j}\left(x_{i,j}|\alpha_1^{i,j}, \alpha_2^{i,j}, ...\alpha_n^{i,j}\right)\right] \end{aligned} \tag{30}$$

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*:

$$P\left(x|\alpha_1,\alpha_2,\alpha_3\right) = \begin{cases} \alpha_3\delta\left(x\right)\frac{1-\alpha_3}{x\alpha_2+\sqrt{2\pi}}e^{-\frac{(\ln x-\alpha)^2}{2\alpha_2^2}} & for x \geq 10^{-4} \\ 0 & \leq 10^{-4} \end{cases} \tag{31}$$

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

$$\Theta\left(x\right)\left[erf\left(\frac{n+0.5-\alpha_1}{\alpha_2+\sqrt{2}}\right)-erf\left(\frac{n-0.5-\alpha_1}{\alpha_2+\sqrt{2}}\right)\right] \tag{32}$$

  where the normalization factor N is set to:

$$N = \frac{1}{2}\left(\frac{-0.5-\alpha_1}{\alpha_2+\sqrt{2}}\right) \tag{33}$$

6.3.2 Model architecture

The network architecture is briefly described in the 6.1. 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: 31, 32): 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 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.

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.

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.

Rocca, Joseph. *Understanding Variational Autoencoders (VAEs)*. URL: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73.