Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Dipartimento di Fisica e Astronomia
Corso di Laurea in Fisica

APPLICAZIONI DEL MACHINE LEARNING ALLA FISICA DELLE ALTE ENERGIE

Relatore:
Prof. Alberto Cervelli

Co-relatore:
Dr. Roberto Morelli

Presentata da:
Schiazza Filippo Antonio

Anno accademico 2019/2020

Sommario

Negli ultimi anni sono stati sviluppati sistemi sempre più complessi di analisi di grandi quantità di dati: nel campo della fisica delle alte energie, il grande numero di eventi prodotti in collisionatori rendono questi sistemi molto utili per la ricerca di eventi rari. In questa tesi da prima verrà descritta l'evoluzione degli algoritmi utilizzati per l'analisi multivariata di campioni molto estesi di dati; ci si focalizzerà in particolare su sistemi di machine learning, con particolare attenzione sui Variational Autoencoders e la loro applicazione nel campo della fisica delle alte energie. Verranno quindi presentati i risultati dell'applicazione di un Variational Autoencoder per la ricerca di fisica oltre il Modello Standard (BSM). Verrà descritto il processo di addestramento di tale algoritmo, effettuato su campioni di eventi simulati secondo le predizioni del Modello Standard, e verrà valutata la sensibilità a processi di produzione elettrodebole di particelle supersimmetriche ($pp \to \tilde{\chi}_1^{\pm} + \tilde{\chi}_2^0, \tilde{\chi}_1^{\pm} \to W + \tilde{\chi}_1^0, \tilde{\chi}_2^0 \to h + \tilde{\chi}_1^0$), dove dalla collisione di due protoni si ottengono un chargino $\tilde{\chi}_1^{\pm}$ ed un neutralino pesante $\tilde{\chi}_2^0$; in seguito il chargino decade in un bosone W ed un neutralino leggero $\tilde{\chi}_1^0$, mentre il neutralino pesante decade in un Bosone di Higgs h ed in un altro neutralino leggero. La ricerca è volta all'osservazione dell'elettrone e del muone prodotti dal decadimento del W ed alla coppia di b-quarks, prodotti dal decadimento dell' h.
Con l'applicazione dell'algoritmo descritto è stato ottenuto il limite di 800 GeV per la massa del chargino per la sensibilità al segnale BSM, nell'ipotesi di massa nulla del neutralino leggero.

Indice

1 Introduzione

Il modello Standard (SM) e' una teoria che ha avuto grande successo, ed e' riuscita a spiegare e prevedere molti dei fenomeni osservati a livello subnucleare. D'altra parte, sebbene la scoperta del *Bosone di Higgs* [5] abbia confermato la rottura di simmetria elettrodebole, ha portato in primo piano lo *hierarchy problem* [15, 13, 9, 14], cioè la grande differenza tra l'accoppiamento elettrodebole e quello gravitazionale. Inoltre, nonostante l'evidenza di materia oscura nell'universo, lo SM non prevede alcuna particella che possa giustificare la presenza della materia oscura osservata, ad esempio negli aloni galattici.
Da queste considerazioni sembrerebbe essere ormai arrivati ad un punto di stallo per quanto riguarda il MS, tuttavia è evidente da ciò che è stato accennato precedentemente che rimangono aperte molte domande; una ipotesi è che il MS rappresenti il limite a basse energie di una teoria più complessa, quindi una serie di fenomeni o non avvengono alle attuali energie raggiungibili al *Large Hadron Collider* oppure sono estremamente rari.
Per esempio, la teoria della *Supersimmetria* (SUSY), che è una estensione del MS, risolve il Problema della Gerarchia introducendo un nuovo fermione/bosone per ogni fermione/bosone del MS; inoltre tali particelle sarebbero stabili e poco interagenti e quindi costituirebbero delle ottime candidate per la spiegazione della materia oscura.
Per ottenere una buona reiezione del fondo accompagnata da una buona efficienza sul segnale si possono utilizzare sistemi di analisi statistica molto complessi. Il machine learning (ML) rappresenta una serie di metodologie di natura statistica-computazionale che permettono di estrarre informazioni da enormi moli di dati senza la supervisione dell'analista. In fisica delle alte energie gli algoritmi di ML, attraverso l'apprendimento delle correlazioni tra le proprietà cinematiche delle particelle presenti in un evento, consentono di catalogare ciascun evento come affine al segnale o al fondo. In particolare i Variational Autoencoders (VAEs) [12], una cui applicazione e' descritta nel capitolo 4, si basano sulla riduzione della dimensione delle variabili che descrivono gli eventi, seguita da una fase di ricostruzione del campione. Nello specifico il VAE comprime ogni singolo evento di input che gli viene presentato non come un punto in uno spazio di dimensione minore (detto spazio latente), bensì come una distribuzione; si campiona quindi un punto nello spazio latente a partire da tale distribuzione, che viene ricostruito a seguito di un processo di decompressione.
Una volta addestrato il VAE a riconoscere e riprodurre le distribuzioni delle variabili cinematiche che descrivono gli eventi SM e' possibile, confrontando questi risultati con i dati ottenuti dall'esperimento, osservare (o valutare il limite di esclusione) anomalie che possono essere dovute alla presenza di eventi non descritti dal Modello Standard.
Questa tesi è strutturata in tre capitoli: nel primo vengono spiegate le differenze fra gli algoritmi di machine learning ed i metodi di analisi multivariata e non, nel secondo vengono approfondite le metodologie di machine learning e si presentano i Variational Autoencoders, mentre nell'ultimo viene applicato quest'ultimo metodo nel campo della fisica delle alte energie.

2 Analisi multivariata e Machine Learning

Le ricerche di eventi rari nella fisica delle alte energie hanno come scopo quello di riuscire a selezionare un campione il più puro possibile di eventi di segnale. Il parametro di merito della selezione è la sensibilità al processo a cui si è interessati, che è funzione del rapporto tra il numero di eventi di segnale attesi ed il numero di eventi di background:

$$\sigma = \frac{N_s}{\sqrt{N_s + N_b}} \tag{1}$$

In generale un evento può essere pensato come una collezione di dati e quindi lo si può rappresentare come un vettore in uno spazio n-dimensionale:

$$\vec{x} = (x_1, ..., x_n) \tag{2}$$

dove x_i sono le informazioni ottenute dalle particelle rivelate dal detector: ad esempio l'impulso di ciascuna particella, le masse invarianti ottenute sommando i quadrivettori di due o più particelle, il numero di *hit* in un detector, o l'energia trasversa mancante dovuta a particelle non rivelate.
Una volta individuate le quantità che si vogliono utilizzare per separare il campione di segnale dal fondo, si possono utilizzare varie tecniche per ottimizzare la selezione, e quindi ottenere la migliore sensibilità al processo di interesse. Si possono individuare tre classi metodologiche:

1. Sistema di tagli sulle variabili (*cut and count*). Con questo metodo si selezionano sottoinsiemi dei valori di ciascuna variabile, in modo indipendente fra loro (*cut*) per poi fare un esperimento di conteggio sulle regioni selezionate;

2. Analisi multi-variata come, ad esempio, l'analisi discriminante lineare;

3. Machine Learning, cioè sistemi che permettono l'apprendimento automatico e quindi l'algoritmo è in grado di imparare in maniera autonoma direttamente dai dati che gli vengono forniti.

E' importante sancire in maniera netta il confine fra l'analisi multivariata classica ed il *machine learning*. Si dice che un algoritmo apprende un qualche compito C dall'esperienza E, con una performance P, se la sua performance P migliora nello svolgere il compito C con l'esperienza E. Quindi, in questo caso specifico, si può affermare che la discriminante fondamentale è se l'algoritmo continui a migliorare o meno la sua prestazione all'aumentare degli eventi che gli vengono presentati.

2.1 Sistema di tagli

Con questo sistema si applicano delle selezioni sulle varie componenti x_i che definiscono un evento, in modo da ricavare un ipercubo nello spazio n-dimensionale degli eventi stessi. Per capire meglio questa metodologia si consideri un caso semplificato nel quale lo spazio in questione è bi-dimensionale e quindi i vettori di input sono del tipo $\vec{x} = (x_1, x_2)$; per raggiungere l'obiettivo di separazione bisognerà dunque applicare due tagli, uno sulla variabile x_1 ed uno su x_2, come riportato in figura 1.

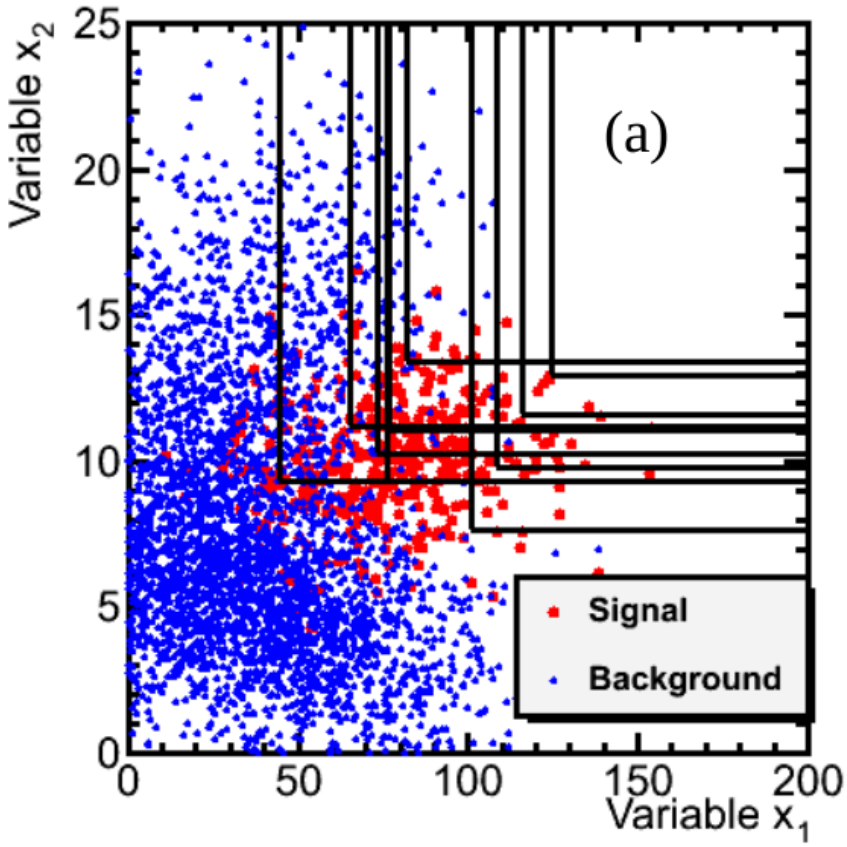

Figura 1: risultato grafico di un processo di taglio sulle due variabili x_1 e x_2 per la separazione del segnale dal background [2].

La scelta della regione in cui fare il conteggio è ottenuta grazie ad una ottimizzazione del rapporto segnale rumore al variare dei tagli. Questa ottimizzazione può essere ottenuta con molti metodi, di cui un esempio è il Random Grid Search (RGS) che verrà presentato nella sezione 3.3.
In questo caso le selezioni sulle variabili x_i sono fatte in modo indipendente, di conseguenza non si tiene conto di eventuali correlazioni tra le variabili e, se le distribuzioni di tali variabili per segnale e fondo sono molto simili, questo sistema può non essere molto efficiente.

2.2 Processi multivariati e analisi discriminante lineare

Nella sezione precedente è stato messo in evidenza il fatto che i sistemi di tagli non permettono di tener conto di eventuali correlazioni fra i dati.
Per poter tener conto delle correlazioni tra le variabili x_i in esame, e poter sfruttare al meglio le variabili poco discriminanti è possibile utilizzare delle selezioni ottenute grazie a selezioni su delle funzioni di un sottoinsieme delle variabili prese in esame.
Ogni volta che si considera una selezione fatta su una funzione di più di una variabile è possibile parlare di analisi multivariata e, per la ragione appena presentata, è necessario considerare i processi multivariati.
Il metodo di analisi multivariata più semplice è quello del discriminante lineare, o discriminante di Fisher: si immagini di avere a disposizione un determinato set di eventi di input $\vec{x}_i$, ciascuno caratterizzato da un numero n di variabili (spazio n-dimensionale) e di volerli ripartire fra segnale e background.
Si definisce la funzione discriminante lineare nel seguente modo:

$$D(x_1, x_2, ..., x_n) = c_0 + c_1 x_1 + ... + c_n x_n = c_0 + \sum_{i=0}^{n} c_i x_i \tag{3}$$

cioè 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 background), utilizzando un valore di riferimento D_0.
A questo punto l'obiettivo è quello di massimizzare la distanza fra le due classi, ossia rendere massima la differenza dei valori assunti dalla funzione $D(\vec{x})$ fra gli eventi appartenenti al background e quelli relativi al segnale.
Per ottenere una ottimizzazione del valore di selezione uno dei metodi più comuni è quello proposto da Fisher: si consideri un campione di eventi appartenenti al segnale e se ne definisca la media $\vec{\mu}_s$ e la deviazione standard σ_s ed un campione appartenente al background, definendo anche qui la media $\vec{\mu}_b$ e la deviazione standard σ_b. A questo punto la migliore configurazione dei parametri $(\vec{\theta})$ è quella che massimizza la seguente funzione:

$$F(\vec{\theta}) = \frac{(\vec{\mu}_s - \vec{\mu}_b)^2}{\sigma_s^2 + \sigma_b^2} \tag{4}$$

Il discriminante lineare e' il sistema multivariato con la forma funzionale più semplice. Si possono costruire metodi sempre più complessi nella forma funzionale, come la funzione di massima verosimiglianza. Si consideri un campione di n misure indipendenti fra loro $(\vec{x}_1, \vec{x}_2, ..., \vec{x}_n)$, allora la funzione di massima verosimiglianza L rappresenta la densità di probabilità di ottenere tali misure, una volta fissato il valore dei parametri $\vec{\theta}$ della distribuzione:

$$L(\vec{\theta}; \vec{x}_1, \vec{x}_2, ..., \vec{x}_n) = F_{\vec{x}}(\vec{x}_1, \vec{x}_2, ..., \vec{x}_n) \tag{5}$$

dove F è la PDF delle $\vec{x}$.
Inoltre vale che:

$$F_{\vec{x}}(\vec{x}_1, \vec{x}_2, ..., \vec{x}_n) = f(\vec{x}_1) f(\vec{x}_2) ... f(\vec{x}_n) \tag{6}$$

perché le n misure sono considerate indipendenti fra loro.
La stima di massima verosimiglianza è quella che si ottiene scegliendo $\vec{\theta}$ tale da massimizzare la funzione L:

$$\vec{\theta}_{MV} = \max_{\vec{\theta}} L(\vec{\theta}; \vec{x}_1, \vec{x}_2, ..., \vec{x}_n) \tag{7}$$

2.3 Machine Learning

Perché è utile il ML per l'obiettivo che è stato prefissato? Bisogna considerare il fatto che i sistemi multivariati descritti in sezione 2.2 dipendono fortemente dalla scelta dell'analista della funzione, o del metodo, da utilizzare per applicare la selezione del proprio campione. Il Machine Learning invece sfrutta algoritmi in grado di apprendere in maniera semi-autonoma la struttura dei campioni di background e segnale, ed è quindi in grado di stabilire quale sia il modo migliore per separare tali campioni.
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. A differenza dei sistemi descritti fin ora, i sistemi di selezione autonoma non necessitano di un modello fisico-matematico su cui basare la propria selezione.
Bisogna distinguere tre macro-tipologie di approccio all'analisi dati nel machine learning:

- APPRENDIMENTO SUPERVISIONATO
  In questa tipologia di apprendimento vengono presentati all'algoritmo degli input di esempio ed i relativi output desiderati, con lo scopo di apprendere una relazione generale che lega gli uni con gli altri; quindi per prima cosa si utilizza un campione di addestramento (*training data set*), in cui l'algoritmo ottimizza la selezione per legare gli input agli output forniti in fase di addestramento. Una volta addestrato, l'algoritmo viene validato utilizzando un campione di test (*test data set*) dove non vengono forniti gli output e se ne valuta l'efficienza di selezione;

- APPRENDIMENTO NON SUPERVISIONATO
  A differenza del caso supervisionato, nel *training data set* non sono presenti gli output attesi, quindi l'algoritmo deve essere in grado di apprendere autonomamente sia la struttura degli output desiderati, sia la miglior selezione per dividere i due campioni. Nel capitolo 4 verrà presentato un esempio di algoritmo non supervisionato per applicazioni nel campo della fisica delle particelle.

- APPRENDIMENTO PER RINFORZO
  Il *Reinforcement Learning* è basato sul concetto di ricompensa, cioè 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 una penalità in caso di sconfitta.

Oltre al modo in cui gli algoritmi ottimizzano la selezione a partire dai loro campioni di addestramento, si deve distinguere anche il modo in cui vengono presentati i dati in uscita. In quest'ottica si possono individuare tre differenti tipologie di algoritmi:

- CLASSIFICAZIONE
  Gli algoritmi di classificazione sono caratterizzati da un output discreto, cioè una serie di classi alle quali l'input può appartenere. Di solito questo metodo è utilizzato da sistemi con 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, proprio perché nel *training data set* gli eventi di input non sono etichettati (non è noto il relativo output) e quindi si richiede all'algoritmo di ricavare autonomamente le classi.

3 Machine Learning : metodi e caratteristiche

In questo capitolo da prima verranno descritti i due approcci principali al ML: l'apprendimento supervisionato (3.1) e non supervisionato (3.2). Successivamente verranno presentati due metodi di apprendimento supervisionato, cioè le *Reti Neurali* (3.4) e gli *Alberi Decisionali* (3.5), ed un metodo di apprendimento non supervisionato, cioè il *Variational Autoencoders* (3.8).

3.1 Apprendimento supervisionato

Come già accennato nella sezione 2.3, quando si parla di apprendimento supervisionato si hanno a disposizione sia gli input $\vec{x}$ che i corrispettivi target di output $\vec{y}$ (nella fase di addestramento); esisterà quindi una funzione $\vec{y} = f(\vec{x})$ che mette in relazione gli input con gli output. Tuttavia tale funzione non è conosciuta a priori, ed è quindi l'obiettivo dell'algoritmo ricostruirne la migliore approssimazione possibile. Nella pratica si cerca di approssimare la funzione agendo su una serie di parametri $\vec{\theta}$, in modo da ottenere una funzione del tipo: $\hat{\vec{y}} = f(\vec{x}, \vec{\theta})$.
Lo schema logico seguito durante l'addestramento di un algoritmo di apprendimento supervisionato è riportato in figura 2.

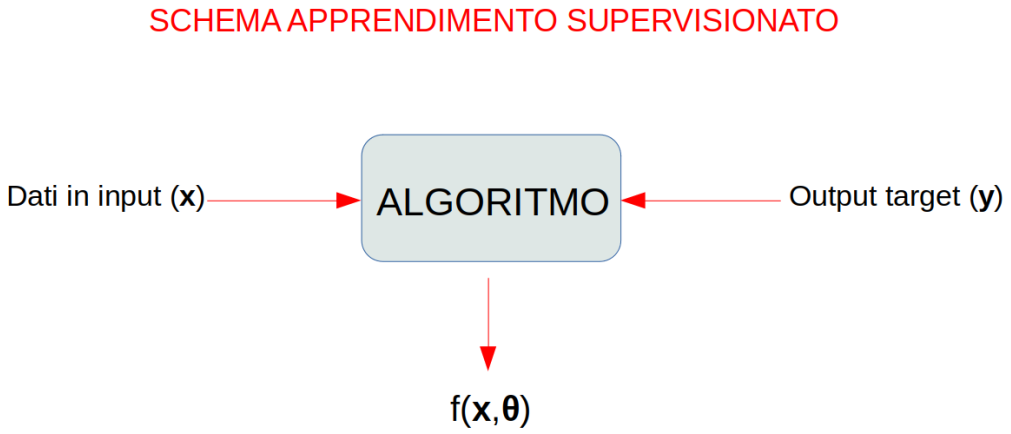

Figura 2: si riporta uno schema intuitivo del funzionamento di un algoritmo di apprendimento supervisionato

Per ogni vettore $\vec{x}$ del *training data set* è possibile definire una particolare funzione detta *Loss Function* $L(\vec{y}, f(\vec{x}, \vec{\theta}))$. Questa funzione rappresenta una misura della differenza fra l'output dell'algoritmo e quello atteso.
A questo punto è possibile fare una media di tale funzione sull'intero set di dati a disposizione, ottenendo la funzione di rischio:

$$R(\vec{\theta}) = \frac{1}{N} \sum_{k=1}^{N} L(\vec{y}, f(\vec{x}, \vec{\theta})) \tag{8}$$

dove N è il numero di eventi del *training data set*.
Quindi, mentre la *Loss Function* rappresenta una misura dell'errore compiuto dall'algoritmo nel fornire l'output per un singolo evento di input, la funzione di rischio è una media

di tale errore sull'intero campione dei dati di addestramento.
Un esempio di funzione di rischio molto diffusa è l'errore quadratico medio:

$$R(\vec{\theta}) = \frac{1}{N}\sum_{k=1}^{N}(\vec{y}_k - f(\vec{x}_k, \vec{\theta}))^2 \tag{9}$$

Quando si addestra un modello si vuole inoltre evitare il così detto *overfitting*, che consiste in un eccessivo adattamento del modello ai dati di *training* e che di conseguenza porta al non raggiungimento di una sufficiente generalità; in tal caso infatti l'algoritmo costruisce una funzione $f(\vec{x}, \vec{\theta})$, che ha una forte potenza nella classificazione del campione di addestramento, ma non ha la stessa potenza di classificazione su altri campioni statistici.
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, aggiungendo una funzione $Q(\vec{\theta})$ (aiuto); in questo modo si ottiene la funzione di costo:

$$C(\vec{\theta}) = R(\vec{\theta}) + \lambda Q(\vec{\theta}) \tag{10}$$

con λ parametro che esprime la rigidità del vincolo.
A questo punto l'obiettivo è quello di minimizzare la funzione di costo ed uno dei metodi più comuni per tale scopo è il metodo di discesa del gradiente.
Discesa del gradiente è una tecnica di ottimizzazione utilizzata per minimizzare l'errore che si introduce stimando la $\hat{\vec{y}} = f(\vec{x}, \vec{\theta})$ rispetto alla funzione "vera" $\vec{y} = f(\vec{x})$. Si può quindi definire, per quanto detto precedentemente, una *Loss Function* $L(\vec{y}, f(\vec{x}, \vec{\theta}))$ ed un vettore $\vec{\theta}$, le cui componenti sono i parametri che devono essere ottimizzati nel processo di addestramento al fine di ridurre il più possibile l'errore che viene commesso dall'algoritmo nello stimare l'output, rispetto all'output target.
Esistono tre varianti del metodo di discesa del gradiente, che differiscono per il momento nel quale avviene l'aggiornamento del vettore $\vec{\theta}$:

- *Batch Gradient Descent.*
  In questa variante l'aggiornamento del vettore dei parametri avviene una volta per ogni epoca, cioè vengono processati tutti gli eventi di input dall'algoritmo e, solo in seguito, si applica il metodo di discesa del gradiente. Quindi, come primo passo, viene calcolata la *loss* per ogni evento di input e la si somma sull'intero campione di dati di addestramento, per ottenere la *loss* totale:

  $$L_{tot} = \sum_{k=1}^{N} L(\vec{y}_k, f(\vec{x}_k, \vec{\theta})) \tag{11}$$

  Successivamente si definisce il vettore $\vec{G}$ applicando il gradiente (nello spazio dei parametri) alla *loss* totale:

  $$\vec{G} = \frac{1}{N}\vec{\nabla}_{\vec{\theta}} L_{tot} \tag{12}$$

  e, con tale risultato, viene aggiornato il vettore dei parametri nel seguente modo:

  $$\vec{\theta} - \epsilon\vec{G} \rightarrow \vec{\theta} \tag{13}$$

  con l'obiettivo di minimizzare la *loss* totale.
  Qui ϵ prende il nome di *learning rate* e regola l'aggiornamento del vettore dei parametri nella direzione opposta a quella del gradiente $\vec{G}$.

Quindi con questa tecnica si aggiornano i parametri una sola volta per ogni epoca, tuttavia si impiega molto tempo per arrivare ad una convergenza perché bisogna appunto processare tutti i dati e solo in seguito si applica il gradiente. Per questa ragione è un metodo poco adatto quando si hanno grandi moli di dati a disposizione e con alta dimensionalità.

- *Stochastic Gradient Descent.*
  Differentemente dal caso precedente, il gradiente viene valutato indipendentemente per ciascun evento di input:

$$\vec{G}_i = \vec{\nabla}_{\vec{\theta}} L(\vec{y}_i, f(\vec{x}_i, \vec{\theta})) \tag{14}$$

  e quindi il vettore dei parametri viene aggiornato per ciascun evento in modo indipendente:

$$\vec{\theta} - \epsilon \vec{G}_i \rightarrow \vec{\theta} \tag{15}$$

  Questa tecnica è, all'opposto della precedente, utile quando il numero di eventi di input è molto elevato, perché non è necessario calcolare la *loss* totale prima di procedere con l'aggiornamento dei parametri.

- *Mini Batch Gradient Descent.*
  In questo caso si calcola la variazione come nel metodo Batch ma, invece di calcolare $\vec{G}$ per il campione totale, viene calcolato per un sottocampione, in modo da ridurre i dati in esame. Così facendo si riesce a diminuire il tempo necessario per il calcolo riducendo il campione statistico.

3.2 Apprendimento non supervisionato

Gli algoritmi di apprendimento supervisionato risultano essere molto utili nel caso in cui si abbiano a disposizione sia i vettori di input che i corrispettivi output target, perché si riesce ad ottenere un'approssimazione della relazione esistente input-output; nel caso della fisica delle alte energie questo significa conoscere le proprietà degli eventi di segnale che si vogliono individuare.
Tuttavia nel caso di ricerche ad ampio spettro, che non dipendano fortemente da un modello, e che quindi non hanno una descrizione precisa degli eventi di interesse, non è possibile avere a priori un campione di segnali con il relativo output di riferimento.
In questo caso si può ricorrere all'uso di tecniche di apprendimento non supervisionato, dove l'obiettivo è quello di trovare eventuali partizioni degli input (*Clustering*), cioè vengono ricercati all'interno degli input quelli che sono diversi dalle aspettative perché hanno proprietà non affini a tutti gli alti eventi del campione.
Si consideri la figura 3 dove sono riportate tre diverse configurazioni possibili nel caso di input bidimensionali: nel caso a) è possibile la separazione in due sotto-gruppi e nel caso b) in un unico sotto-gruppo, mentre nel caso c) sembrerebbe non si possano stabilire graficamente eventuali separazioni.

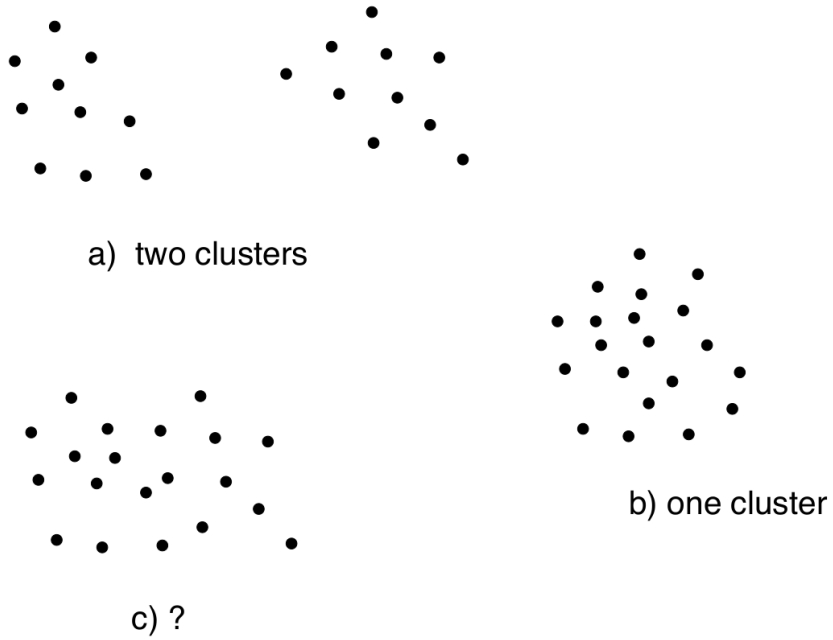

Figura 3: vettori di input in uno spazio bidimensionale in tre situazioni differenti. L'immagine è presa da [10]

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*.
Un esempio di *clustering* è quello basato sulla distanza euclidea. In questo caso bisogna

porre particolare attenzione al concetto di misura, infatti i vettori evento sono, di norma, costituiti da componenti disomogenee ma, nonostante questo, vengono rappresentati in uno spazio a più dimensioni; quindi, nel calcolare le distanze, bisogna tenere a mente questo concetto.
E' di fondamentale importanza osservare che nei metodi che si basano sul concetto di distanza è importante ri-scalare i valori delle componenti degli eventi (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; per fare ciò, ad esempio, si possono suddividere gli intervalli dei valori osservati per ogni componente in un egual numero di *bin*, e su questi calcolare le distanze.
Gli algoritmi di apprendimento non supervisionato sfruttano una misura di similarità per separare gli eventi di input nei vari *cluster*. Una possibilità è quella di utilizzare la semplice distanza euclidea per poter separare lo spazio n-dimensionale degli eventi in delle sotto-aree (i *cluster*).
Per fare ciò viene implementato un metodo iterativo, basato sulla definizione di alcuni punti particolari nello spazio degli eventi, detti *cluster seekers* (letteralmente "cercatori di cluster").
Si definiscono M punti nello spazio n-dimensionale $\vec{C}_1, ..., \vec{C}_M$ e l'obiettivo è quello di fare in modo che ogni punto si muova verso il centro di ogni singolo *cluster*, così che ogni *cluster* abbia al suo centro uno di questi *cluster seekers*. Inoltre, dato che il numero di *cluster* non e' conosciuto a priori, il numero M durante la prima iterazione e' arbitrario.
Gli eventi del *training data set* Σ vengono presentati all'algoritmo uno alla volta: per ognuno di essi ($\vec{x}_i$) si cerca il *cluster seekers* più vicino ($\vec{C}_k$) e lo si sposta verso $\vec{x}_i$ nel seguente modo:

$$\vec{C}_k + \alpha_k(\vec{x}_i - \vec{C}_k) \to \vec{C}_k \tag{16}$$

dove α_k è un parametro di apprendimento che determina di quanto $\vec{C}_k$ si muove verso il punto $\vec{x}_i$.
Dato che l'algoritmo deve spostare ciascun $\vec{C}_k$ al centro del *cluster* k-esimo, i suoi spostamenti devono essere di entità, in media, sempre minore. Per evitare che $\vec{C}_k$ abbia spostamenti sempre più ampi al crescere delle iterazioni si definisce una massa m_k e le si assegna un valore pari al numero di volte in cui $\vec{C}_k$ 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

$$\alpha_k = \frac{1}{1 + m_k} \tag{17}$$

e, dato che ad ogni iterazione che coinvolge $\vec{C}_k$ 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 gli eventi del campione di addestramento all'algoritmo, i vari *cluster seekers* andranno a convergere 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, ossia un generico punto (evento) $\vec{x}_i$ viene inserito nel *cluster* k-esimo se la distanza di tale punto dal *cluster seeker* $\vec{C}_k$ è minore rispetto alla sua distanza da tutti gli altri *cluster seekers*.
Un esempio didattico del risultato di questa partizione è riportato in figura 4.

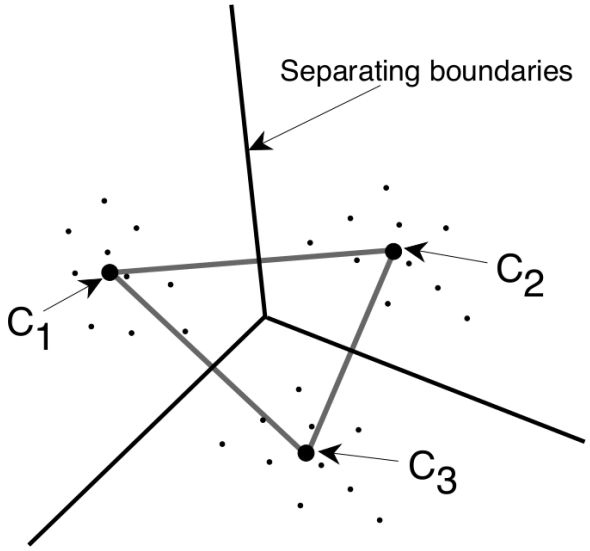

Figura 4: Si riporta un esempio di partizione dello spazio (bi-dimensionale) di Voronoi in tre sotto regioni, dove i punti C_1, C_2 e C_3 sono i cluster seeker. L'immagine è presa da [10]

Si è accennato qualche riga fa che il numero di *cluster seekers* è inizialmente scelto in maniera casuale, per poi essere ottimizzato; per il processo di ottimizzazione si utilizza la varianza degli eventi $\vec{x}_i$ per ogni *cluster*:

$$\sigma^2 = \frac{1}{L}\sum_{i=1}^{L}(\vec{x}_i - \vec{\mu})^2 \tag{18}$$

dove L è il numero di eventi nel *cluster* e $\vec{\mu}$ ne è la media:

$$\vec{\mu} = \frac{1}{L}\sum_{i=1}^{L}\vec{x}_i \tag{19}$$

A questo punto, se la distanza d_{ij} fra due *cluster seekers* $\vec{C}_i$ e $\vec{C}_j$ è minore di un determinato valore ϵ, allora si sostituiscono i due *cluster seekers* 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.
La scelta dei valori di ϵ e δ dipende dalla tipologia di dati a disposizione; nello specifico ϵ è legato alla risoluzione sperimentale dei dati, mentre δ è legato alla varianza e dipende dal fatto che si cerca di avere gli eventi compatti all'intero dei *cluster*.

3.3 Iperparametri e Grid Search

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 tale processo avviene attraverso un continuo confronto con il *training data set*.
Gli iperparametri sono invece una serie di parametri che caratterizzano il modello implementato che non sono modificati nel processo di addestramento ma vengono prestabiliti dall'utente. Gli iperparametri rappresentano delle caratteristiche dell'algoritmo, che possono essere modificate prima di avviare il processo di addestramento, come ad esempio il numero di neuroni in una rete neurale (3.4), oppure il numero di epoche, cioè il numero di volte in cui viene ripresentato all'algoritmo l'intero campione di dati di addestramento.
E' molto probabile che ci siano configurazioni di iperparametri migliori di altre, cioè per le quali l'addestramento dell'algoritmo produce esiti migliori; di conseguenza anche gli iperparametri devono essere sottoposti ad un processo di ottimizzazione.
Uno dei metodi utilizzati per tale scopo è il *Grid Search*, che è piuttosto semplice 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.
Per il Grid Search si consideri un modello caratterizzato da un numero k di iperparametri, quindi è possibile creare un vettore le cui componenti siano gli iperparametri stessi:

$$\vec{\mu} = (\mu_1, ..., \mu_k) \tag{20}$$

Tale vettore appartiene 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*, e quindi la miglior configurazione sarà quella che minimizza tale funzione.
In Figura 5 nella pagina seguente è riportato per chiarezza un esempio visivo dell'esito di un processo di ottimizzazione degli iperparametri attraverso il metodo del *Grid Search*.

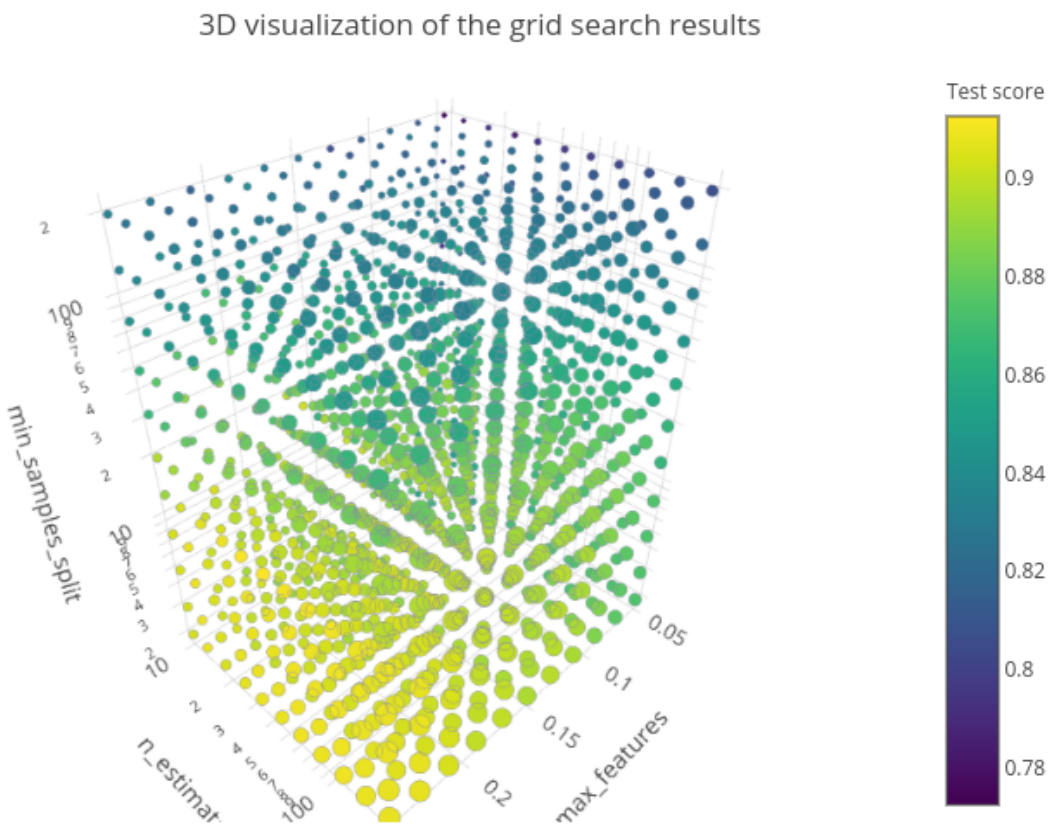

Figura 5: la figura illustra visivamente l'esito di un processo di ottimizzazione degli iperparametri attraverso il metodo del *Grid Search* [8]

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, scelta in modo arbitrario al momento dell'implementazione del RGS.
Un ulteriore accorgimento può essere quello di bloccare l'ottimizzazione di alcune configurazioni che sembrano meno promettenti, ad esempio quando superano un certo valore della *Loss function*, evitando che il processo di apprendimento venga completato su tutti i nodi. Per far ciò, è possibile implementare degli *scheduler* che tengano conto dell'andamento dei diversi training relativi alle diverse configurazioni. Un approccio diverso è quello di esplorare nuove varianti a partire dalle configurazioni iniziali. In quest'ultimo caso non è nemmeno necessario definire in modo rigido lo spazio degli iperparamentri da esplorare dato che le nuove configurazioni vengono ricercate a partire dall'andamento delle precedenti (*popultion based training*).

3.4 Reti Neurali

Le reti neurali sono probabilmente il metodo di apprendimento supervisionato più conosciuto ed utilizzato nel campo dell'analisi dati.
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.
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 6 è riportato un esempio di rete neurale con un singolo strato interno nascosto.

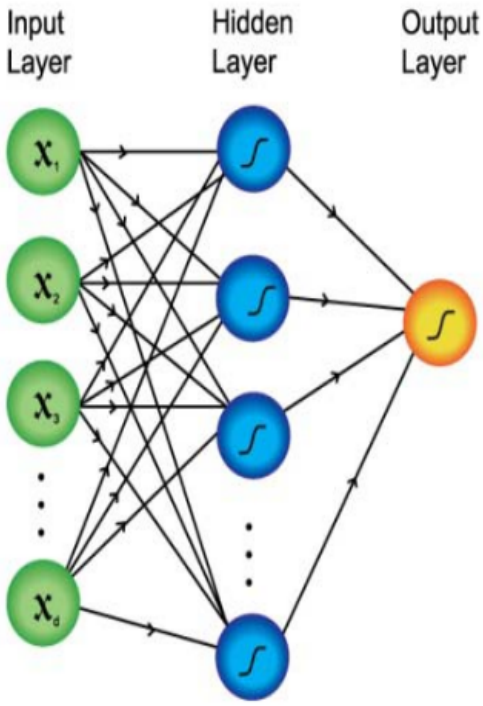

Figura 6: si riporta un esempio grafico di rete neurale formata da un unico strato nascosto. L'immagine è presa da [2].

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 7 lo schema grafico di un singolo neurone (k), dove $\vec{x} = (x_1, ..., x_m)$ è il vettore degli input, $\vec{w}_k = (w_{k1}, ..., w_{km})$ 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.

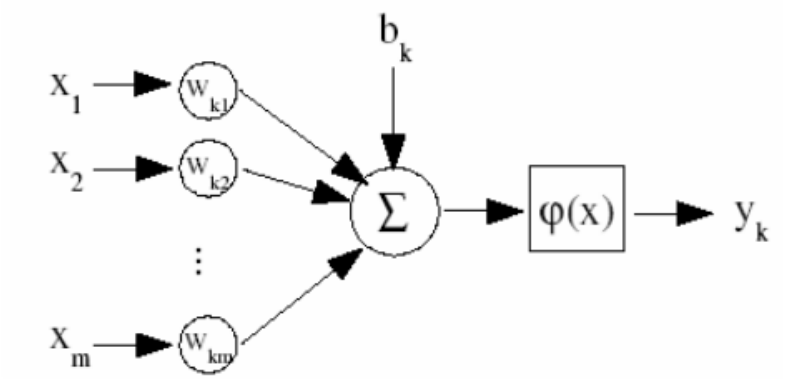

Figura 7: Illustrazione della struttura di un neurone [3].

Quindi il neurone opera la seguente somma pesata:

$$s_k = \vec{x} \bullet \vec{w}_\mathbf{k} = \sum_{i=1}^{m} x_i w_{ki} \tag{21}$$

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

$$y_k = \phi(s_k + b_k) \tag{22}$$

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

$$sig(x) = \frac{1}{1 + e^{-\alpha x}} \tag{23}$$

dove α è un parametro che permette di regolare la pendenza della curva, come si evince dalla figura 8

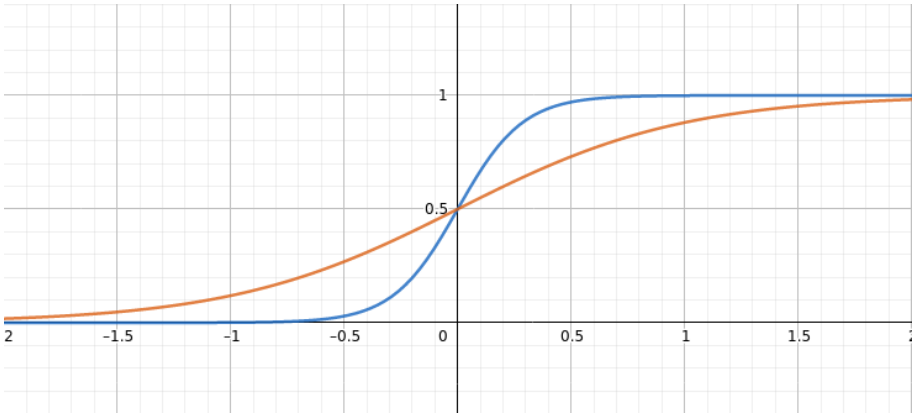

Figura 8: Si riportano due sigmoidi, dove per quella in rosso si ha α=2 e per quella in blu α=7.

Un singolo neurone è l'ingrediente fondamentale di una rete neurale e nella gran parte dei casi non è in grado di ottenere da solo nessuna classificazione degli input. 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 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.

L'apprendimento di un singolo neurone dipende dalla struttura della rete e dal tipo di input ed output fornito alla rete. Nel caso in cui l'output target sia noto si può utilizzare il metodo dell'apprendimento con correzione di errore.
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

$$E = \frac{1}{2} err^2 \tag{24}$$

sulla quale si applicherà il metodo di discesa del gradiente già discusso nel paragrafo 3.1 per ottimizzare i parametri.
Una rete neurale può svolgere diverse funzioni, tuttavia quella principale che viene utilizzata in questa tesi prende il nome di riconoscimento. 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 supervisionato). Si ipotizzi di avere a disposizione dei vettori evento con un numero n di componenti (i dati) che rappresentano dei punti in uno spazio n-dimensionale; questo spazio potrà essere allora diviso in delle regioni che corrispondono alle varie categorie ed i confini di queste regioni si ottengono a seguito del processo di addestramento.
Durante l'addestramento ogni singolo neurone deve aggiornare i suoi pesi in modo che l'output della rete sia simile a quello atteso.
Uno dei metodi migliori per addestrare la rete neurale è l'algoritmo di back-propagation. Per l'implementazione di questo algoritmo si introduce un segnale di input che deve propagarsi verso lo strato di output e, allo stesso tempo, si fa propagare un segnale di errore dallo strato di output verso quello di input.
Nel caso della back-propagation bisogna fare una distinzione tra l'addestramento dei neuroni nello strato di output e negli strati nascosti:

1. Neurone nello strato di output.
   Un neurone di output k, quando gli verrà fornito l'elemento j del *training data set*, avrà un errore:
   $$err_k^{(j)} = R_k^{(j)} - y_k^{(j)} \tag{25}$$
   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:
   $$E^{(j)} = \frac{1}{2}\sum_{k=1}^{n}(err_k^{(j)})^2 \tag{26}$$
   dove n è il numero di neuroni nello strato di output.
   Se poi N è il numero totale di elementi del *training data set*, allora la funzione di costo può essere definita nel seguente modo:
   $$E_{tot} = \frac{1}{N}\sum_{j=1}^{N} E^{(j)} \tag{27}$$
   e l'obiettivo è quello di minimizzare tale funzione di costo. Per fare ciò si procede variando 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
   $$\frac{\partial E^{(j)}}{\partial w_{ki}^{(j)}} \tag{28}$$
   e gli aggiornamenti del peso vengono applicati nel verso opposto del gradiente, ovvero
   $$\Delta w_{ki}^{(j)} = -\mu \frac{\partial E^{(j)}}{\partial w_{ki}^{(j)}} \tag{29}$$
   dove μ è il *learning rate*. Manca a questo punto il calcolo esplicito del gradiente, che può essere eseguito con la regola della catena
   $$\frac{\partial E^{(j)}}{\partial w_{ki}^{(j)}} = \frac{\partial E^{(j)}}{\partial err_k^{(j)}} \frac{\partial err_k^{(j)}}{\partial y_k^{(j)}} \frac{\partial y_k^{(j)}}{\partial S_k^{(j)}} \frac{\partial S_k^{(j)}}{\partial w_{ki}^{(j)}} \tag{30}$$

dove $S_k^{(j)} = s_k^{(j)} + b_k^{(j)}$ (si faccia riferimento all'equazione (21)).
Una volta calcolate le quattro derivate si ottiene:

$$\frac{\partial E^{(j)}}{\partial w_{ki}^{(j)}} = -err_k^{(j)} \phi'(S_k^{(j)}) y_i^{(j)} \tag{31}$$

e quindi:

$$\Delta w_{ki}^{(j)} = err_k^{(j)} \phi'(S_k^{(j)}) y_i^{(j)} \mu \tag{32}$$

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 *learning rate* 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. Per gestire meglio questo aspetto, nella pratica viene definito un *learning rate* variabile. Generalmente si fa in modo che questo fattore parta da un certo valore per decrescere mano mano che si ci avvicina ad una soluzione ottimale. In prima istanza infatti è utile avere un valore abbastanza grande da poter seguire in modo efficace la discesa del gradiente, rendendo più rapida la ricerca della soluzione ottimale. Successivamente, però, risulta utile diminuire questo stesso parametro per evitare oscillazioni attorno al punto di minimo.

3.5 Alberi Decisionali

Gli alberi decisionali sono, al pari delle reti neurali, un metodo ML di apprendimento supervisionato.
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 eventi 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 9 un esempio di albero decisionale molto semplice).

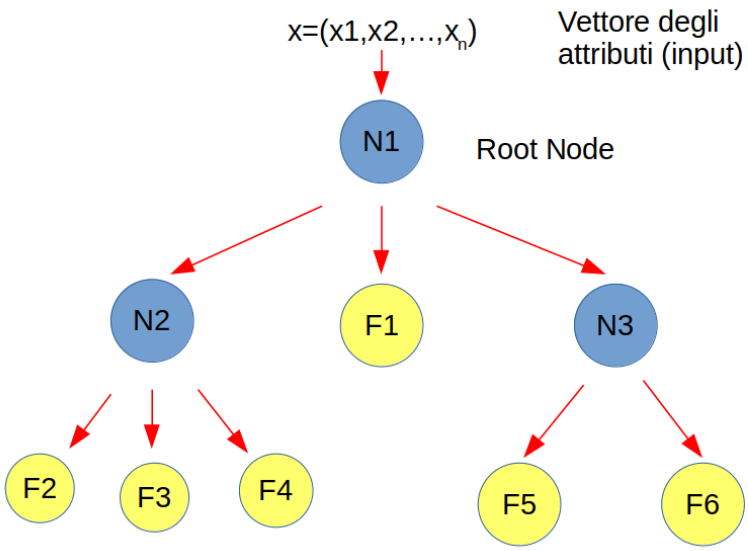

Figura 9: esempio di come è strutturato un albero decisionale

Gli elementi che caratterizzano un albero decisionale sono:

- Nodo.
  I nodi sono riportati in figura 9 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 gli esempi nelle rispettive categorie a seconda dei loro attributi; tali rami determinano quindi i percorsi all'interno degli alberi decisionali e 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.

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 fase 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 ridurre l'*overfitting* perché vengono eliminati i rami che non contribuiscono in maniera significativa al processo di classificazione.

Poiché ciascun nodo agisce su una singola variabile dei vettori di input, è possibile ottenere un gran numero di alberi differenti. Per questo motivo è necessario trovare un modo per ottimizzare la geometria dei nodi all'interno dell'albero: per farlo viene scelto 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, ovvero un indicatore della frequenza con cui un elemento casuale, appartenente al data set, sia identificato in modo non corretto.

- Guadagno informativo, ovvero un indicatore, preso in prestito dalla teoria dell'informazione e basato sul concetto di entropia, di quanto la scelta di uno specifico attributo consenta di ridurre l'entropia informativa dell'insieme (semplificando, la variabilità dei valori dell'attributo presenti all'interno di un nodo foglia).

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.
Una volta costruito l'albero decisionale, i metodi per il *pruning* sono sostanzialmente due: 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. E' altresì possibile introdurre nella fase di costruzione stessa un criterio di *early-stopping* o *pre-pruning* richiedendo un valore minimo di miglioramento dell'algoritmo fra un'iterazione e l'altra: ad ogni passaggio di separazione dell'albero decisionale viene fatto un controllo sulla *Loss function* e, se tale errore non diminuisce significativamente tra un passaggio e l'altro, si interrompe il processo di costruzione dell'albero. Il problema dell'*early stopping* è che non tiene conto dell'impatto della selezione sui nodi successivi; per questa ragione in genere si utilizzano entrambi i metodi di *pruning* e di *early stopping* parallelamente, per poi confrontare i risultati.
Gli alberi decisionali hanno vari pregi, fra i quali quello di essere caratterizzati da una particolare semplicità nell'interpretazione e nella visualizzazione, ed essere insensibili ad eventuali componenti degli eventi irrilevanti nella classificazione; inoltre sono caratterizzati da una tolleranza ad eventuali attributi mancanti per alcuni input nel *training data set* o nel *test data set* e da una invarianza per trasformazioni monotone effettuate sulle variabili degli input, che rende la fase di pre-processamento dei dati non necessaria. Dall'altro lato ci sono una serie di limiti, come l'instabilità rispetto al variare del *training data set*, cioè data set di allenamento di poco differenti fra loro producono risultati molto diversi,

oppure il frequente problema dell'*overfitting*.
E' possibile anche combinare più alberi decisionali insieme per ottenere migliori prestazioni predittive. Una delle tecniche per tale combinazione è il *Bagging*, dove vengono generati in maniera casuale dei sottogruppi del training data set ed ognuno di questi viene utilizzato per l'addestramento di un albero decisionale. Il risultato sarà una collezione di alberi decisionali e, come decisione finale, si utilizzerà la media delle decisioni dei singoli alberi. Esiste un'estensione di questo metodo conosciuta con il nome di *Random Forest*; in questo caso viene aggiunto un passaggio ulteriore al processo appena illustrato perché viene scelto casualmente anche un sottogruppo delle componenti dei vettori di evento, ottenendo così un metodo capace di agire anche su data set ad alta dimensionalità e che permette di ridurre sensibilmente il problema dell'*overfitting*.

3.6 Curse of dimensionality e riduzione della dimensionalità

I metodi di apprendimento supervisionato, come descritto, necessitano anche di output target per essere addestrati. Nel caso questi non fossero disponibili l'unica soluzione è utilizzare un metodo non supervisionato di ML. Un moderno esempio e' il Variational Autoencoder, di cui nel capitolo 4 verrà mostrata un'applicazione nel campo della fisica delle alte energie.
Quando si parla di eventi di input 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).
Spesso, nel campo della fisica delle particelle, il numero di variabili che descrivono ciascun evento è molto grande e questo fa si che lo spazio di riferimento sia ad altissima dimensionalità. Come visto, in molti metodi l'aumento della dimensionalità crea delle problematiche riguardo la complessità dell'algoritmo e del tempo di computazione necessario per ottenere dei metodi ottimizzati. Questa problematica prende il nome di *Curse of Dimensionality*.
All'aumentare della dimensionalità la densità degli eventi nello spazio diminuisce, e questo fa si che il numero di dati necessari per poter avere dei risultati statisticamente significativi aumenta enormemente. Tutto ciò ha un grande impatto soprattutto quando si vuole fare una classificazione di eventi molto rari.
Per ovviare a questo problema si possono utilizzare tecniche attraverso le quali viene ridotto il numero di variabili che caratterizzano gli eventi.; 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, limitando il più possibile la perdita di informazione.
Degli input con più bassa dimensionalità permettono di avere anche meno parametri (gradi di libertà) e quindi una struttura più semplice del modello. Inoltre sistemi meno complessi sono meno soggetti all'*overfitting*, oltre a garantire una migliore efficacia statistica.
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 se tale processo 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*

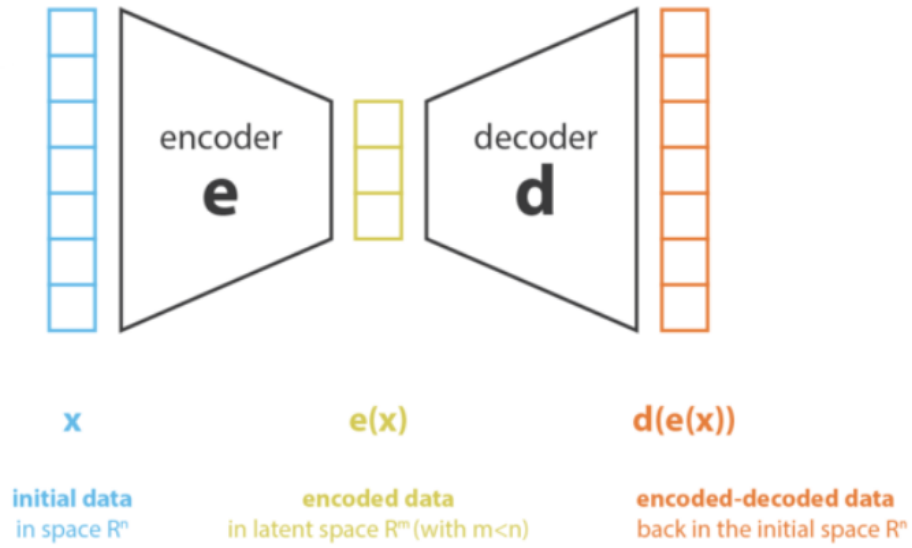

Figura 10: Strutture generale di un processo di riduzione della dimensionalità . L'immagine è presa da [12]

Il vettore di input $\vec{x}$ (n-dimensionale) viene compresso dall'encoder **e** in un vettore $\mathbf{e}(\vec{x})$ di uno spazio m-dimensionale (con m<n), detto *spazio latente*.
Il decoder **d** svolge la funzione opposta, ovvero decomprime il vettore $\mathbf{e}(\vec{x})$ in $\mathbf{d}(\mathbf{e}(\vec{x}))$ per tornare allo spazio originario n-dimensionale.
Nel caso in cui $\vec{x} = \mathbf{d}(\mathbf{e}(\vec{x}))$ (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 $\vec{x} \neq \mathbf{d}(\mathbf{e}(\vec{x}))$, 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 decodifica.
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:

$$(e, d) = \min_{E \times D} \epsilon(\vec{x}, \mathbf{d}(\mathbf{e}(\vec{x}))) \tag{33}$$

dove $\epsilon(\vec{x}, \mathbf{d}(\mathbf{e}(\vec{x})))$ è la grandezza attraverso la quale viene quantificata la quantità di informazione persa nel processo di riduzione.
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. I FSM si includo i *wrapper methods* ed i *filter methods*: i primi valutano il modello con varie combinazioni di subset delle variabili originali e selezionano quella per la quale si ha la maggiore accuratezza del modello, mentre i secondi si basano maggiormente sulle caratteristiche intrinseche dei dati (correlazioni, contenuto informativo, etc...).
I metodi di estrazione, invece, si basano fortemente sull'algebra lineare; in particolare vengono utilizzati spesso per la riduzione della dimensionalità i metodi di fattorizzazione

delle matrici per cogliere la parte più importante dei dati, cioè quella per la quale la perdita di informazione nel processo di ricostruzione è minore.
Il più comune di questi metodi prende il nome di *Principal Component Analysis* (PCA). L'idea del PCA è quella di costruire un numero n_e di nuove variabili indipendenti che siano combinazione lineare delle n variabili di partenza; tale costruzione viene fatta in modo tale che la proiezione delle vecchie variabili sul nuovo sottospazio generato da quelle nuove sia il più possibile vicina ai dati iniziali, dove la vicinanza è da intendere in termini della distanza euclidea. In altre parole con il PCA si ricerca il sottospazio dello spazio degli eventi di partenza per il quale l'errore che viene compiuto nell'approssimazione dei dati tramite proiezioni sia il più piccolo possibile. Un esempio grafico è riportato in figura 11.

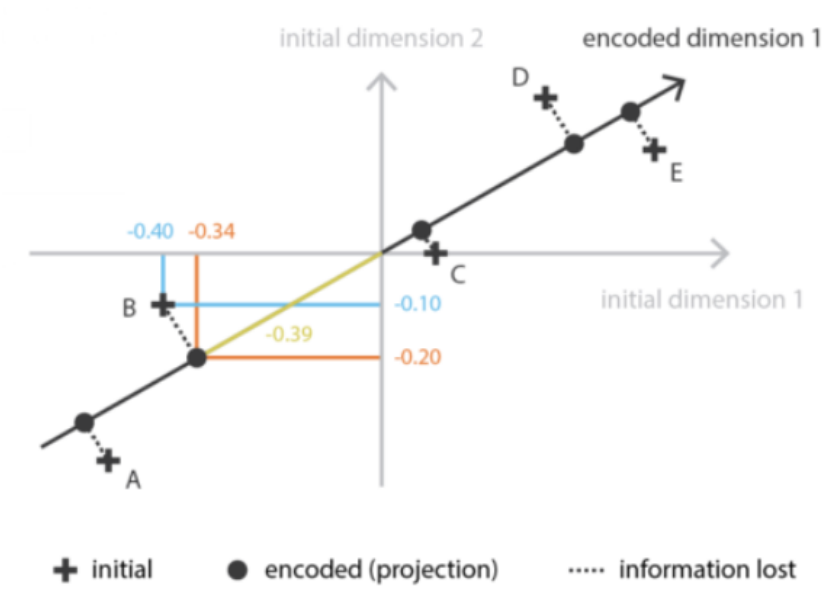

Figura 11: Illustrazione del processo di PCA nel caso di uno spazio degli eventi iniziali bi-dimensionale [12].

3.7 Autoencoders

Gli *autoencoders*, come ogni altro metodo di riduzione della dimensionalità, sono costituiti da un *encoder* e da un *decoder*; tuttavia in questo caso la peculiarità è che sia l'*encoder* che il *decoder* sono delle reti neurali, come è possibile vedere in figura 12.

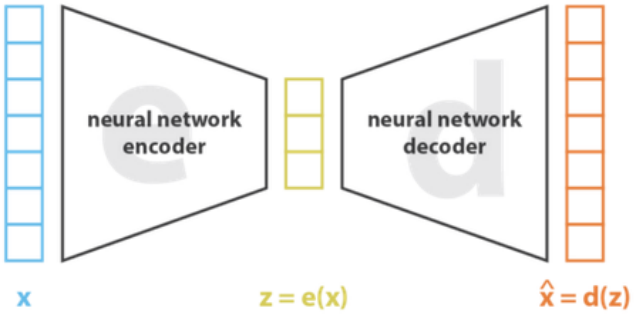

Figura 12: Struttura di un generico *autoencoder* [12].

L'obiettivo è anche in questo caso quello di individuare la coppia *encoder-decoder* che ottimizza il processo di ricostruzione degli input e ciò viene fatto attraverso il seguente processo iterativo: si presentano all'*encoder* gli eventi di input uno alla volta, subiscono un processo di riduzione della dimensionalità e poi vengono ricostruiti (tornando alla dimensionalità di partenza), viene calcolato l'errore dal confronto fra l'input iniziale e quello ricostruito ed avviene l'aggiornamento dei pesi della rete neurale mediante il meccanismo di *backpropagation*, già incontrato nella sezione 3.4.
Intuitivamente l'*autoencoder* può essere pensato come un collo di bottiglia, attraverso il quale solo una parte dell'informazione riesce a passare oltre e a formare i vettori dello spazio latente. Facendo riferimento alla figura 12 si osserva che, a partire dagli eventi di input $\vec{x}$, come primo passo si costruisce lo spazio latente degli $\vec{z} = e(\vec{x})$ per poi procedere alla fase di decodifica nella quale si ottengono gli eventi ricostruiti $\hat{\vec{x}} = d(\vec{z})$; si procede successivamente al calcolo degli errori nel seguente modo:

$$L = \|\vec{x} - \hat{\vec{x}}\| \tag{34}$$

dove L è l'errore di ricostruzione.
Una considerazione necessaria circa l'errore, che potrebbe sembrare in contraddizione con quanto detto fino ad ora sul concetto di ottimizzazione, è che si vuole di norma evitare che $\vec{x} = \hat{\vec{x}}$, perché questo vuol dire che l'*autoencoder* ha imparato la funzione identità e, come conseguenza, la struttura dello spazio latente, che è quella interessante per il processo di riduzione della dimensionalità, non porta alcuna informazione interessante; ciò è dovuto al fatto che l'*encoder* non impara se vi siano variabili più o meno importanti di altre o se esse possano essere compattate in nuove variabili di dimensionalità minore.
Per fornire un esempio pratico si consideri un insieme di vettori di input N dimensionali; una possibilità è quella di prendere una per una le componenti degli eventi e disporle lungo una retta (spazio latente 1-dimensionale) nella fase di codifica, per poi procedere in maniera inversa nella fase di decodifica. L'errore con questo procedimento sarà nullo ma non si può essere soddisfati essenzialmente per due motivi: da un lato lo spazio latente non è interpretabile e sfruttabile e, dall'altro, in un processo di riduzione della dimensionalità

si vuole fare in modo che i dati continuino a conservare una qualche struttura.
Una possibilità per evitare il risultato appena illustrato è di aggiungere alla funzione L un fattore di regolarizzazione che penalizza i risultati per i quali $\vec{x} = \hat{\vec{x}}$.
Quindi bisogna sempre porre particolare attenzione alla scelta della profondità dell'*encoder*, ovvero alla sua capacità di riduzione della dimensionalità.
Per completezza nella trattazione si osserva che gli *autoencoder* possono essere sia lineari che non; il primo caso si ottiene quando non si inserisce una funzione di attivazione non lineare e si utilizzano solo due strati, quindi le trasformazioni possono essere rappresentate come matrici e si ottiene un risultato simile a quello del PCA (3.6).
Il caso di *autoencoder* non lineari (*deep autoencoder*) può essere pensato come un passo successivo per quanto riguarda la riduzione della dimensionalità. Infatti, come è stato già detto, il PCA ricerca il miglior iperpiano nello spazio degli eventi originali sul quale questi possano essere proiettati in modo da ridurre la perdita di informazione; dall'altro lato gli *autoencoders* non lineari non si limitano alla ricerca di iperpiani, ma possono esplorare anche superfici più complesse, come illustrato in figura 13.

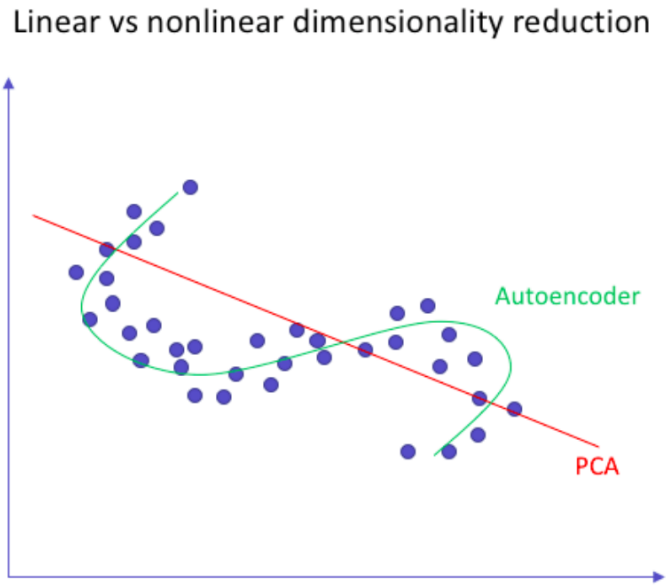

Figura 13: Differenza fra i metodi lineari (PCA) e gli autoencoder non lineari [7].

3.8 Variational Autoencoders (VAEs)

Nella sezione precedente è stato presentato l'autoencoder come un metodo di riduzione della dimensionalità; tuttavia, apportando una sostanziale modifica, tale metodo può essere utilizzato per la generazione di eventi ed in questo caso si parla di *Variational Autoencoders* (VAEs).
L'autoencoder agisce su di un evento originale $\vec{x}$ trasformandolo in un $\vec{z} = e(\vec{x})$ nello spazio latente (fase di *encoding*) e poi trasforma nuovamente $\vec{z}$ per tornare allo spazio originale. Per la generazione, basterebbe prendere un punto a caso nello spazio latente, e quindi decodificarlo, ottenendo un nuovo evento non legato direttamente a quello di input, ma solo alla distribuzione nello spazio latente. Questo non tiene pero' conto della regolarità dello spazio latente.
L'*autoencoder* codifica in modo puntuale ogni evento associando ad ognuno di essi un punto specifico e diverso dagli altri all'interno dello spazio latente. In questo modo, se viene dato in input un evento mai incontrato durante la fase di addestramento, si presentano delle difficoltà nella sua successiva rigenerazione. In questo caso infatti il modello comincia la fase di rigenerazione a partire da un punto nello spazio latente a cui non è associato nessun input, come se in quel punto lo spazio latente fosse discontinuo (un esempio è riportato in figura 14). Per ovviare a questo problema, la fase di codifica deve assumere un significato probabilistico, associando ai vari vettori evento non più un solo punto ma un'intera distribuzione. In questo modo lo spazio latente sarà meno soggetto ad avere discontinuità al suo interno, facilitando così il processo di generazione.

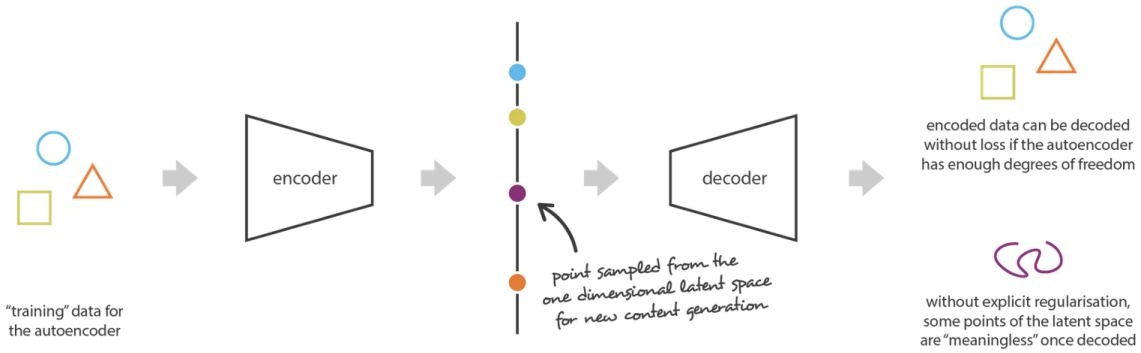

Figura 14: Illustrazione grafica e semplificata dei limiti nell'applicazione degli *autoencoder* per fini di generazione di nuovi eventi [12].

Si può quindi pensare al VAEs come ad un *autoencoder* per il quale si attua una regolarizzazione dello spazio latente durante il processo di addestramento ed è quindi adatto alla generazione di nuovi eventi.
Lo spazio latente si dice regolare se:

- è continuo, ovvero due punti vicini portano ad un risultato simile una volta decodificati;
- è completo, nel senso che un qualunque punto porta ad un risultato sensato una volta decodificato.

I VAEs, al pari degli *autoencoders*, sono costituiti da una coppia di *encoder* e *decoder* con la differenza che, per gli *autoencoders* l'evento di partenza viene codificato come un punto nello spazio latente ($\vec{z}$), mentre per i VAEs la codifica avviene tramite una distribuzione nello spazio latente ($p(\vec{z}|\vec{x})$). Il processo seguito nella fase di addestramento è il seguente:

1. L'evento iniziale viene codificato nello spazio latente come una distribuzione di probabilità;
2. Si campiona un punto dello spazio latente a partire dalla distribuzione del punto precedente;
3. Tale punto viene decodificato dal *decoder*, ottenendo un evento ricostruito;
4. Avviene il confronto fra l'evento iniziale e quello ricostruito da cui si calcola l'errore, che viene poi propagato mediante il meccanismo della *backpropagation*.

Nella pratica si cerca di fare in modo che la distribuzione ottenuta alla fine del processo di codifica sia il più possibile vicina ad una distribuzione Normale. In particolare si farà in modo che l'*encoder* restituisca la media e la matrice di covarianza della distribuzione Normale.
Seguendo questa linea, si ottiene che il processo di addestramento è regolato da una funzione di perdita, composta da un termine relativo alla ricostruzione ed uno relativo alla regolarizzazione dello spazio latente; tale termine di regolarizzazione viene espresso mediante la *Divergenza di Kullback-Leibler*, che rappresenta una misura della differenza ta due distribuzioni di probabilità.
Il solo fatto di codificare gli eventi di input come delle distribuzioni non garantisce la continuità e la completezza, perché se non si inserisce il termine di regolarizzazione nella funzione di perdita il VAE continuerà a comportarsi come un semplice *autoencoder*, tendendo semplicemente a minimizzare l'errore di ricostruzione in due possibili modi, cioè codificando gli eventi come distribuzioni da un lato con varianze molto piccole (quasi come singoli punti) e dall'altro con medie molto diverse fra loro (punti molto lontani nello spazio latente); nel primo caso non viene garantita la continuità e nel secondo la completezza.
Per ottenere la regolarità dello spazio latente si richiede allora che le distribuzioni con cui vengono codificati gli eventi siano il più possibile vicine a distribuzioni Normali con media zero e matrice di covarianza uguale all'identità; le medie saranno allora vicine con conseguente sovrapposizione delle distribuzioni, anche perché la matrice di covarianza così fatta impedisce la codifica come punti nello spazio latente. Il prezzo da pagare sarà un più alto errore nella fase di ricostruzione. Questo fatto deriva direttamente dal differente modo in cui gli *autoencoders* ed i VAEs codificano gli eventi di input, infatti per i primi vi è una relazione biunivoca fra input ed output e si ha perdita di informazione solo nella fase di compressione, mentre per i secondi si ha una doppia perdita di informazione, cioè sia nella fase di codifica sia in quella di ricostruzione a seguito del campionamento dalla distribuzione nello spazio latente.
Prima di passare alla formulazione matematica dei VAEs, bisogna notare che la regolarità dello spazio latente implica la presenza di un gradiente, il quale permette di mischiare le caratteristiche degli eventi in input e quindi di dare un significato al campionamento nello spazio latente.

3.8.1 Formulazione matematica dei VAEs

Per formulare in maniera rigorosa i VAEs è necessario utilizzare l' inferenza variazionale e verranno dati alcuni concetti fondamentali della teoria dell'informazione.
La quantità di informazione in una preposizione (nel nostro caso l'evento) è definita come:

$$I = \log p(x) \tag{35}$$

dove x è l'evento.
Quindi ad eventi certi o molto probabili corrisponde una quantità di informazione nulla o molto bassa, mentre ad eventi poco probabili corrisponde una quantità di informazione più alta.
Un'altra quantità fondamentale nella teoria dell'informazione è l'*entropia*, ovvero l'informazione media, ed è definita nel seguente modo:

$$H = \sum p(x) \log p(x) \tag{36}$$

La divergenza di KL è la misura di dissimilarità tra due distribuzioni (p(x) e q(x)):

$$KL(p(x)||q(x)) = -\sum p(x) \log q(x) + \sum p(x) \log p(x) = -\sum p(x) \log \frac{q(x)}{p(x)} \tag{37}$$

e si nota come essa sia molto simile alla differenza delle entropie delle due distribuzioni perché due distribuzioni che hanno un'informazione media uguale saranno pressoché uguali. Le due proprietà fondamentale della *KL divergency* sono le seguenti:

1. $$KL(p(x)||q(x)) \geq 0 \tag{38}$$

2. $$KL(p(x)||q(x)) \neq KL(q(x)||p(x)) \tag{39}$$

cioè la divergenza di KL è semi-definita positiva (perché vi è un aumento dell'entropia) e non è simmetrica (come conseguenza del modo in cui è stata definita nell'equazione 37).
Ora è possibile ricollegarsi al discorso sui VAEs e si definisca con $\vec{x}$ la grandezza osservabile e con $\vec{z}$ quella nascosta dello spazio latente. Il teorema di Bayes [6] permette di scrivere:

$$p(\vec{z}|\vec{x}) = \frac{p(\vec{x}|\vec{z})p(\vec{z})}{p(\vec{x})} \tag{40}$$

dove

$$p(\vec{x}) = \int p(\vec{x}|\vec{z})p(\vec{z})dz \tag{41}$$

tuttavia tale integrale è difficile da calcolare e quindi è possibile approssimare $p(\vec{z}|\vec{x})$ con $q(\vec{z}|\vec{x})$ utilizzando l'inferenza variazionale. Si assume inoltre che quest'ultima debba essere scelta dalla famiglia delle distribuzioni Normali, andando a variare i parametri in modo che risulti il più possibile simile a $p(\vec{z}|\vec{x})$, che viene detta *prior* e rappresenta la scelta a priori per la modellizzazione dello spazio latente; generalmente viene scelta una distribuzione normale multivariata standard e la $q(\vec{z}|\vec{x})$ deve approssimare tale distribuzione durante la

fase di addestramento. Per vincolare la forma di $q(\vec{z}|\vec{x})$ a quella di $p(\vec{z}|\vec{x})$ viene utilizzata la *KL divergency*:

$$KL(q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})) = -\sum q(\vec{z}|\vec{x}) \log \frac{p(\vec{z}|\vec{x})}{q(\vec{z}|\vec{x})} \tag{42}$$

e sostituendo $p(\vec{z}|\vec{x})$ con la 40 si ottiene:

$$\begin{aligned} KL(q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})) &= -\sum q(\vec{z}|\vec{x}) \log \frac{p(\vec{x}|\vec{z})p(\vec{z})}{p(\vec{x})q(\mathbf{z}|\vec{x})} \\ &= -\sum q(\vec{z}|\vec{x}) \log \frac{p(\vec{x}|\vec{z})p(\vec{z})}{q(\vec{z}|\vec{x})} + \sum q(\vec{z}|\vec{x}) \log p(\vec{x}) \end{aligned}$$

ma in entrambi i termini della somma la sommatoria è estesa sulle z, quindi:

$$\sum q(\vec{z}|\vec{x}) \log p(\vec{x}) = \log(\vec{x}) \sum q(\vec{z}|\vec{x}) = \log p(\vec{x})$$

perché $\sum q(\vec{z}|\vec{x}) = 1$.
Si arriva quindi al seguente risultato:

$$\log p(\vec{x}) = KL(q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x})) + \sum q(\vec{z}|\vec{x}) \log \frac{p(\vec{x}|\vec{z})p(\vec{z})}{q(\vec{z}|\vec{x})} \tag{43}$$

La sommatoria nell'equazione 43 prende il nome di *Variational lower bound* e viene indicata con la lettera $\mathcal{L}$.
A questo punto si deve osservare che $\vec{x}$ è fissato e quindi il termine sinistro dell'equazione 43 è una costante; di conseguenza minimizzare la *KL divergency* equivale a massimizzare la $\mathcal{L}$, che può essere riscritta in maniera semplificata:

$$\begin{aligned} \mathcal{L} &= \sum q(\vec{z}|\vec{x}) \log \frac{p(\vec{x}|\vec{z})p(\vec{z})}{q(\vec{z}|\vec{x})} \\ &= \sum q(\vec{z}|\vec{x}) \log p(\vec{x}|\vec{z}) + \sum q(\vec{z}|\vec{x}) \log \frac{p(\vec{z})}{q(\vec{z}|\vec{x})} \\ &= E_q \log p(\vec{x}|\vec{z}) - KL(q(\vec{z}|\vec{x})||p(\vec{z})) \end{aligned} \tag{44}$$

Quindi l'obiettivo è quello di trovare la distribuzione $p(\vec{z}|\vec{x})$ che è troppo complessa per essere calcolata in modo analitico: si cerca di approssimarla con una $q(\vec{z}|\vec{x})$ scelta tra un'opportuna famiglia e, per scegliere quella più vicina, si deve minimizzare $KL(q(\vec{z}|\vec{x})||p(\vec{z}|\vec{x}))$, che come visto equivale a massimizzare la $\mathcal{L}$.
In figura 15 è possibile osservare in che modo si passa da $\vec{x}$ a $\vec{z}$ e viceversa.

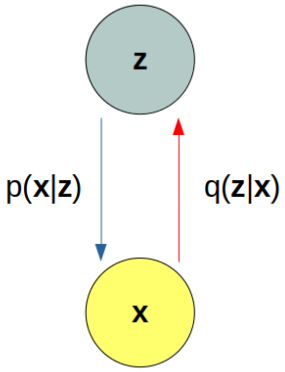

Figura 15: Illustrazione grafica della modalità attraverso la quale si ottiene il passaggio dalla variabile $\vec{x}$ osservabile alla $\vec{z}$ nello spazio latente e viceversa.

Il significato del termine di *KL divergency* che compare in $\mathcal{L}$ suggerisce che la distribuzione $q(\vec{z}|\vec{x})$ debba essere il più possibile simile ad una distribuzione $p(\vec{z})$ che può essere scelta e quindi si assume una distribuzione Normale multivariata standard, per la quale la *KL divergency* è più facile da calcolare.
L'altro termine in $\mathcal{L}$ è riconducibile ad un errore di ricostruzione, infatti il processo di decodifica, una volta campionato $\vec{z}$ è deterministico e quindi si ottiene che:

$$p(\vec{x}|\vec{z}) = p(\vec{x}|\hat{\vec{x}}) \tag{45}$$

dove $\hat{\vec{x}}$ è l'evento ricostruito. Inoltre se si considerano distribuzioni gaussiane si troverà che:

$$p(\vec{x}|\hat{\vec{x}}) \propto e^{-|\vec{x}-\hat{\vec{x}}|^2} \tag{46}$$

e quindi

$$\log p(\vec{x}|\hat{\vec{x}}) \propto -|\vec{x}-\hat{\vec{x}}|^2 \tag{47}$$

Quindi si osserva che l'*autoencoder* tende a minimizzare semplicemente $|\vec{x}-\hat{\vec{x}}|^2$, mentre il VAE tende a minimizzare la seguente quantità:

$$|\vec{x}-\hat{\vec{x}}|^2 + KL(q(\vec{z}|\vec{x})||N(\vec{\mu},\vec{\Sigma})) \tag{48}$$

Nella pratica si costruisce la rete neurale che si occupa della fase di codifica in modo che restituisca i parametri della distribuzione Normale e quindi la media e la matrice di covarianza, che si impone essere diagonale per semplicità; da qui viene campionato un punto dello spazio latente a partire da tale distribuzione e avviato verso il *decoder* per ottenere l'evento ricostruito da confrontare con quello iniziale (nella fase di addestramento). Lo schema di questo processo è riportato in figura 16.

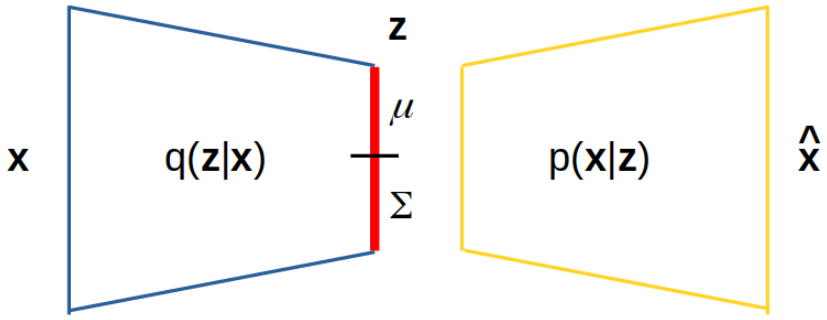

Figura 16: Schema di funzionamento del VAE, dove viene messo in evidenza che l'*encoder* è forzato a restituire come output i parametri della distribuzione Normale, ovvero la media e la matrice di covarianza.

Infine, per la fase di generazione di nuovi eventi, sarà sufficiente eliminare l'*encoder* e campionare dalla distribuzione che si è ottenuta alla fine dell'addestramento e fornire tale punto al *decoder* per costruire un nuovo evento.

4 Ricerca di fisica Beyond Standard Model con i VAEs

In quest'ultimo capitolo verrà presentata una possibile applicazione dei *Variational Autoencoders* nel campo della fisica delle alte energie, con lo scopo di ricercare segnali di nuova fisica BSM, cioè oltre il Modello Standard.
Gli esperimenti portati avanti al *Large Hadron Collider* hanno l'obiettivo di esplorare la fisica spingendosi sempre a più alte energie; attualmente, nonostante la scoperta del bosone di Higgs abbia confermato la validità dello SM, questo modello non riesce a dare alcune risposte a domande aperte nel campo della fisica delle alte energie, quali lo *Hierarchy problem* e l'origine della materia oscura.
Nella ricerca di nuova fisica possono essere portati avanti due approcci, detti *model dependent* e *model independent*. Nel primo caso la ricerca di nuova fisica avviene con un particolare modello in mente, così che è possibile creare un algoritmo di ricerca ottimizzato per il modello target. D'altra parte questo approccio fa si che si focalizzi solo su un particolare canale di ricerca, e quindi manca di generalità. Dall'altro lato una ricerca *model independent* ha il pregio di non essere legata ad una particolare teoria fisica e quindi è capace di ricercare eventuali segnali di nuova fisica senza fare alcuna ipotesi sulla loro natura, o la loro origine. Come descritto nel capitolo 3 quest'ultimo approccio ha molte analogie con il modo in cui i metodi di ML non supervisionati classificano i dati.
Nel caso descritto il VAE verrà addestrato su dati SM simulati, in modo da fare un modello del fondo. L'approccio seguito sarà model dependent: infatti la ricerca sarà ottimizzata per un processo supersimmetrico di produzione elettrodebole di una coppia chargino/-neutralino (partner fermionici dei bosoni elettrodeboli). In questo canale il chargino $\tilde{\chi}_1^{\pm}$ decade in un Bosone W ed in un neutralino $\tilde{\chi}_1^0$, mentre il neutralino pesante decade in un Bosone di Higgs h ed in un altro neutralino leggero.
Se l'addestramento si dovesse rivelare efficace, e la sensibilità al processo a cui si è interessati fosse buona, il processo verrà ripetuto utilizzando i dati dell'esperimento, per cercare di osservare l'anomalia direttamente nel campione di dati raccolti.
Questo ulteriore passaggio è possibile solo in virtù dell'approccio *Unsupervised* per cui non è necessario dare all'algoritmo le etichette fondo/segnale durante la fase di addestramento. Per lo stesso motivo si capisce perché un algoritmo *Supervised* non possa essere in generale direttamente applicato ai dati sperimentali; infatti questa seconda categoria di modelli richiede nella fase di addestramento le etichette fondo/segnale per ciascun evento fisico al fine di impararne la distinzione. Si deve perciò ricorrere necessariamente alle simulazioni MC con tutte le incertezze modellistiche annesse.

4.1 Dataset

Per l'addestramento del modello e per la successiva fase di verifica sono stati utilizzati i dati prodotti attraverso simulazioni Montecarlo (MC), in base alla teoria di riferimento (SUSY). Le variabili fisiche che verranno utilizzate per descrivere ogni evento sono otto (*met*=, *mt*, *mbb* = massa invariante dei due *b-jets*, *mct2*, *mlb1*, *lep1Pt*, *njet30*, *nBjet30-MV2c10*) e sono le stesse utilizzate nell'analisi fisica relativa allo studio [4]. Di conseguenza lo spazio iniziale, che dovrà essere compresso dal VAE, sarà 8-dimensionale.
Attraverso la simulazione MC vengono prodotti eventi sia di background che di segnale. Prima di passare alla fase di codifica, gli eventi (sia di segnale che di background) sono stati sottoposti ad una serie di tagli di preselezione sulle variabili, come riportato nella tabella 1. Questo viene fatto per selezionare, fra tutti gli eventi di fondo, quelli più simili al segnale. Se non si applicassero questi tagli di preselezione, l'algoritmo verrebbe addestrato su tutto il fondo e quindi sia sugli eventi di fondo molto diversi dal segnale sia su quelli simili; come risultato si otterrebbe un algoritmo non sensibile al segnale, perché non riuscirebbe a rilevare le piccole variazioni esistenti fra gli eventi di segnale e quelli di fondo simili al segnale stesso. Si può affermare che, adoperando i tagli di preselezione, è come se si facesse uno zoom per aumentare la sensibilità dell'algoritmo.

	Preselezione
Esattamente un leptone di segnale	Vero
met trigger	Vero
$2-3$ jets con $p_T > 30GeV$	Vero
b-tagged jet	[1-3]
met	$>$ 220 GeV
mt	$>$ 50 GeV
mbb	$[100-140]$GeV
mct	$>$ 100GeV

Tabella 1: Sono riportati i tagli di preselezione applicati sia agli eventi di segnale che di background, prodotti attraverso una simulazione MC.

Gli eventi di background prodotti con il metodo MC sono stati divisi, come si richiede in un processo di apprendimento automatico, in una *training data set* (30 %), un *validation data set* (30 %) ed un *test data set* (40 %). Successivamente, per rendere coerente la selezione del segnale simulando una raccolta dati con le attuali capacità e specifiche del LHC, gli eventi di *validation* e test sono stati ripesati per ottenere la giusta luminosità.
Il numero di eventi prodotti per ciascun processo dello SM e SUSY é di solito molto più grande di quello che è stato prodotto ad LHC. Quindi, per ottenere un modello di fondo e segnale affine alla realtà, a ciascun evento simulato viene associato un peso (di solito minore di 1) in modo che la somma di tali pesi sia uguale al numero di eventi attesi per la luminosità dei dati in esame.

QUESTA PARTE DOVE FORNISCO LA FORMULA LA TOGLIEREI COMPLETAMENTE

Per ottenere i nuovi pesi w_j' a partire da quelli validi prima dello split w_j, è stata usata

la seguente formula [1]:

$$w_j^{'} = w_j \frac{\sum_i w_i \mathbf{1}\{y_i = fondo\}}{\sum_{i \in S'} w_i \mathbf{1}\{y_i = fondo\}} \tag{49}$$

Dove y_i è l'etichetta (segnale o fondo) dello i-esimo evento e $\mathbf{1}$ è la funzione indicatrice ($\mathbf{1}$ y_i = il fondo è zero per un evento di segnale e pari ad uno per un evento di fondo).

4.2 Architettura del modello

Per quanto riguarda la struttura del VAE si specifica che il numero di neuroni negli strati nascosti è pari a cinquanta e che la dimensione dello spazio latente è pari a tre; inoltre la funzione di attivazione dei neuroni è la ReLU, cioè una funzione definita nel seguente modo:

$$f(x) = \max(0, x) \tag{50}$$

Parlando invece del processo di apprendimento, sono state impostate 2000 epoche, cioè tutti i dati vengono riproposti all'algoritmo per duemila volte; inoltre la *loss function* utilizzata è quella standard, composta da un termine legato all'errore di ricostruzione ed uno relativo alla *KL divergency* (moltiplicato per un fattore $\beta = 0.6$). Per il processo di discesa del gradiente è stato impostato inizialmente un *learning rate* pari a 0.003, con una diminuzione del 20% ogni qualvolta il modello non migliora per venti epoche consecutive; inoltre è stata impostata una *batch size* pari a 200, quindi l'aggiornamento dei pesi della rete avviene dopo aver cumulato l'errore su 200 eventi di *training*. Infine si sottolinea che il processo di addestramento termina o perché si è arrivati alle 2000 epoche o perché la *Validation Loss* non migliora per 50 epoche consecutive.
Il modello è stato sviluppato in Python usando le librerie Keras (tensorflow backend).

4.3 Addestramento del VAE

Il VAE deve essere addestrato in modo tale da rilevare eventuali indizi di fisica BSM come delle anomalie, cercando di dimostrarsi sensibile ad una ampia gamma di possibili segnali. Un classificatore binario, ovvero un modello capace di discriminare fra due sole categorie (segnale e background), viene addestrato su un *training data set* i cui eventi di segnale sono generati facendo riferimento ad un determinato modello; tuttavia quando ne verranno presentati altri prodotti con diversi modelli, allora la classificazione risulterà totalmente arbitraria ed è qui che si evince il limite principale di una ricerca model *dependent*. Un ulteriore vantaggio del VAE ed in generale degli approcci model *independent*, rispetto a quelli model *dependent*, è quello di poter essere applicato direttamente sui dati come anticipato nella sezione introduttiva di questo capito (4). In questo modo si evitano quei problemi di incertezze modellistiche legate alle simulazioni montecarlo.
In linea con ciò che è già stato illustrato nel capitolo 3.8, gli eventi di segnale e di background, che sono stati prodotti attraverso simulazioni MC, sono rappresentabili in uno spazio 8-dimensionale. Durante il processo di addestramento del VAE gli eventi di background vengono compressi nello spazio latente (tridimensionale), decompressi per essere ricostruiti e poi confrontati con quelli iniziali per il calcolo dell'errore e quindi per dare il via al processo di *backpropagation* (3.4).
Dopo la fase di addestramento si verifica che il VAE abbia imparato come ricostruire gli eventi fisici di background usati per l'addestramento, dopo averli compressi nello spazio latente. A tal proposito vengono confrontate le distribuzioni originali date in input con quelle rigenerate dalla rete. Allo stesso tempo, si ci aspetta che il modello commetta un errore di ricostruzione maggiore quando, invece che eventi di background, vengono dati in input eventi di segnale. La limitata capacità di generalizzare su questa nuova categoria di eventi mai visti durante la procedura di addestramento dovrebbe indurre il VAE ad una ricostruzione meno accurata. Come si vedrà è possibile allora usare la distribuzione dell'errore di ricostruzione per eventi di background e segnale per discriminare tra queste

due tipologie di eventi.
Tra i contributi originali di questa tesi vi è anche la possibilità di pesare in maniera diversa il contributo che le diverse variabili fisiche che definiscono un evento possono apportare al processo di discriminazione e, per questo motivo, verranno presentati due casi: nel primo le otto variabili avranno tutte lo stesso peso, mentre nel secondo si proverà a capire se, dando maggiore importanza ad alcune di esse, si otterrà un processo di discriminazione più efficiente. E' infatti possibile che il VAE, concentrandosi su di una variabile più significativa per la ricostruzione del background ma non altrettanto per il segnale, possa sfruttare questa differenza per aumentare la forbice di errore di ricostruzione tra background e segnale. Rimane a questo punto da capire quali delle variabili possano aiutare in questo compito. Le possibilità sono almeno due: da un lato si possono sfruttare conoscenze o intuizioni teoriche riguardo il potere discriminante di alcune delle variabili, mentre dall'altro si può procedere in maniera *brute-force* sulle possibili configurazioni dei pesi da dare alle diverse variabili.

4.4 Risultati

4.4.1 Processo di rigenerazione degli eventi

Come primo passo bisogna capire se, a seguito del processo di addestramento, il VAE è in grado di ricostruire gli eventi di background in maniera ottimale. Il risultato del processo di ricostruzione è riportato in figura 17, dove vengono confrontate le distribuzioni dei dati in input (punti blu) con quelle ricostruite dal VAE (punti rossi).

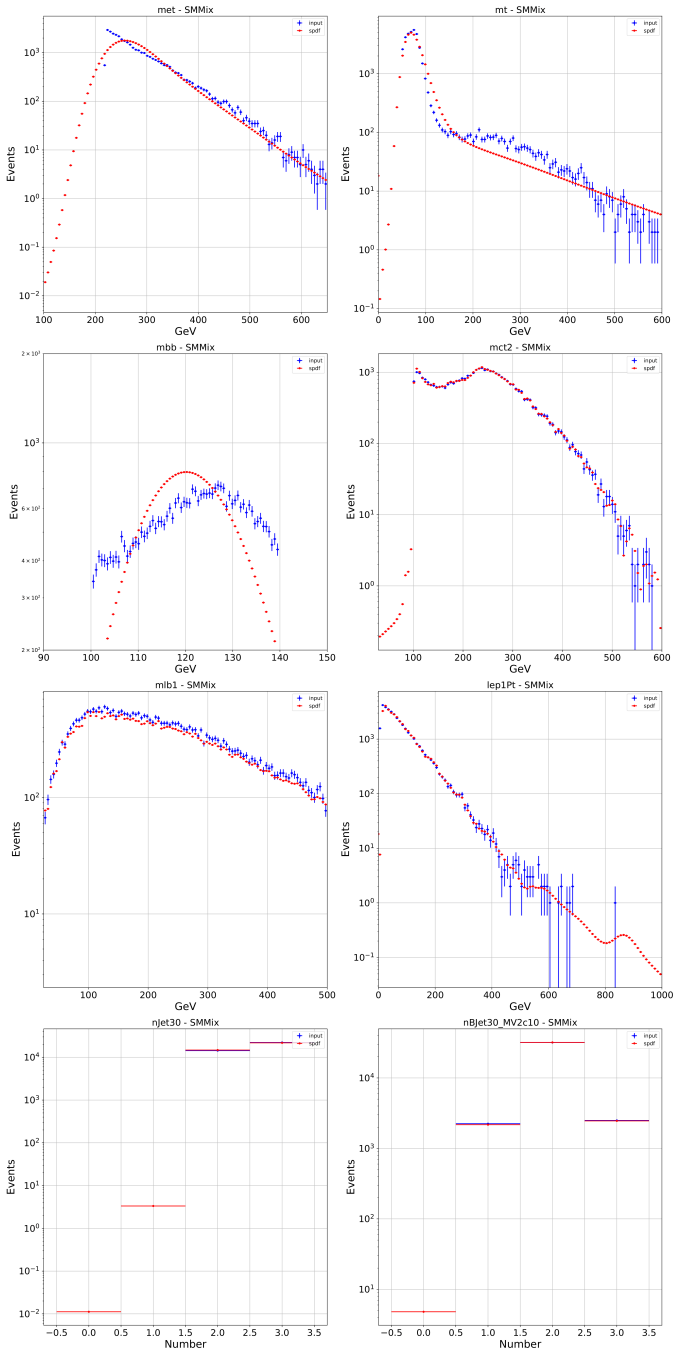

Figura 17: Confronto tra gli input in ingresso del VAE (in blu) e quelli ricostruiti (in rosso) per le otto componenti degli eventi di input relativi al background.

Dall'osservazione qualitativa della figura 17 emerge che il processo di ricostruzione del VAE risulta piuttosto accurato per tutte le variabili ad eccezione della *mbb*.
Tuttavia, come si vedrà più avanti, è possibile correggere questo risultato impostando un peso maggiore per tale variabile (ad esempio un peso pari a due o tre rispetto agli altri tutti pari ad uno). Per ogni singola variabile il peso viene moltiplicato per il suo errore di ricostruzione ed il risultato viene sommato per tutte le variabili nel seguente modo:

$$L_{tot} = w_0 L_{met} + w_1 L_{mt} + w_2 L_{mbb} + w_3 L_{mct2} + w_4 L_{mlb1} + w_5 L_{lep1Pt} + w_6 L_{njet30} + w_7 L_{nBjet30-MV2c10} \tag{51}$$

Inoltre, dopo aver constatato che è possibile indirizzare particolare attenzione sulla ricostruzione di specifiche variabili, si capirà come sfruttare questa situazione per ottenere un modello più sensibile ai vari tipi di segnale.

4.4.2 Distribuzione della loss di ricostruzione

Per rendere un segnale riconoscibile è opportuno che il suo indice di anomalia lo contraddistingua rispetto a quello degli eventi di fondo. In questa tesi la misura adottata per discriminare tra fondo e segnale è l'errore di ricostruzione che il VAE commette durante la ricostruzione degli eventi. Altre scelte possibili sarebbero state quelle di utilizzare la *loss* totale data dalla somma della *Kullback-Lieber divergency* e della *loss* di ricostruzione oppure, al contrario, solo la *Kullback-Lieber divergency*.
Ad ogni modo, per ottenere l'errore di ricostruzione per ogni evento è sufficiente sommare quello sulle otto variabili. Da questi valori si ottiene quindi la distribuzione della *loss*, come quella riportata in figura 18.

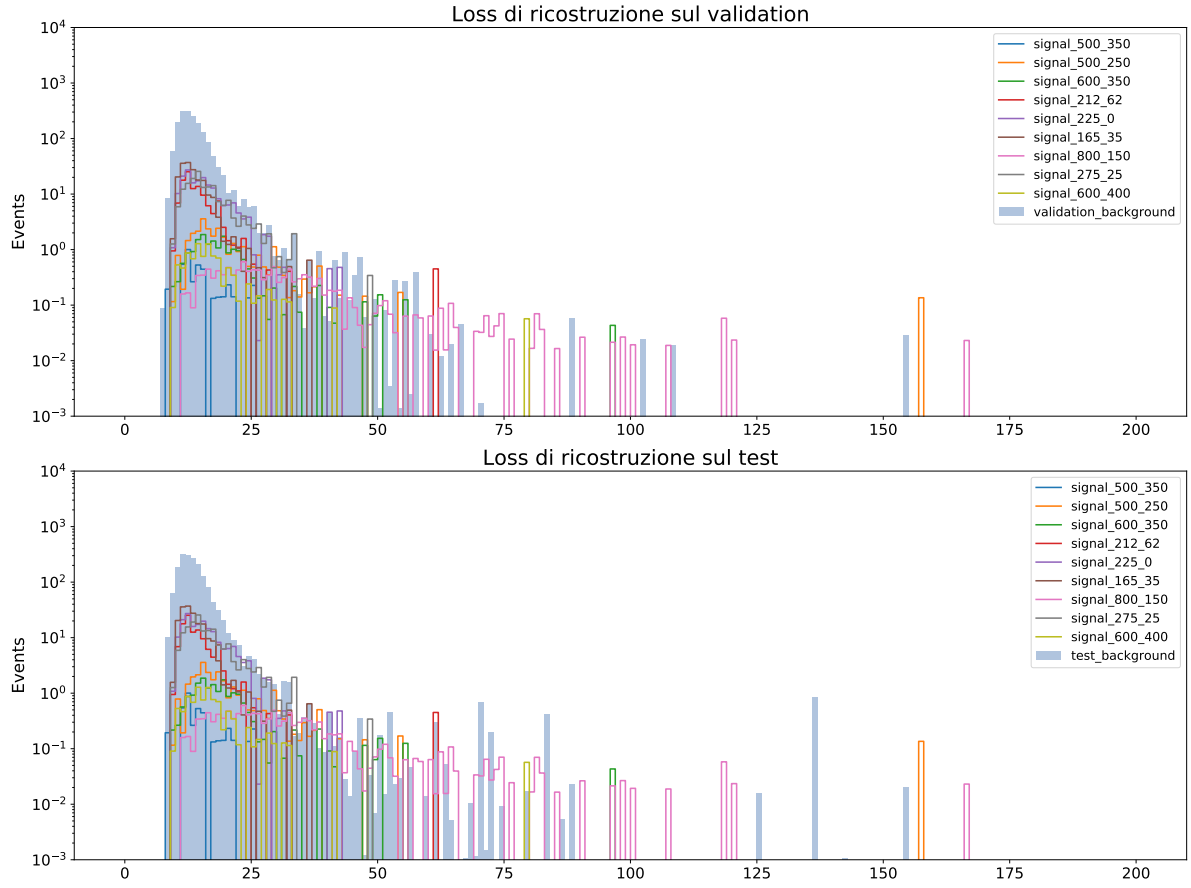

Figura 18: Distribuzione della *Loss* per gli eventi di background e per quelli di segnale relativi ad alcune combinazioni delle masse di chargino-neutralino. La prima immagine è relativa al *Vaidation data set*, mentre la seconda al *test data set*.

In questa immagine, oltre alla distribuzione relativa agli eventi di background (istogramma blu) sia per il dataset di validation (sopra) che per quello di test (sotto), sono presenti anche le distribuzione relative ad alcune delle ipotesi di segnale (curve colorate) contraddistinte dalla diverse ipotesi sulle masse delle due particelle (chargino/neutralino). E' importante ricordare che le simulazioni MC degli eventi di segnale non sono mai state usate fino ad ora e solo in questa fase di test vengono date in input al modello.
Una situazione ideale prevede una distribuzione degli errori per gli eventi di fondo concentrata su valori della Loss inferiori rispetto alle analoghe distribuzioni relative agli eventi di segnale. In questo modo, infatti, la selezione del segnale risulta facilitata e il rapporto

segnale/background aumenta.
Osservando le figure, una prima considerazione riguarda proprio questo aspetto. Infatti, la distribuzione della *loss* per gli eventi di background presenta un picco spostato verso sinistra rispetto alle distribuzione relative agli eventi di segnale. Allo stesso tempo questa separazione non risulta molto netta, in particolare per alcuni tipi di segnale, caratterizzati da una struttura cinematica più simile a quella degli eventi di fondo. Per questo motivo si cercherà di trovare un modello che risulti più efficace in questo compito, sfruttando, ad esempio, una diversa pesatura delle variabili che caratterizzano un evento fisico.
Un altro aspetto non trascurabile e verificabile confrontando i due grafici della stessa figura 18, riguarda l'*overfitting*. Infatti, giudicando la distribuzione degli eventi di background nei due grafici, si può vedere come la forma sia pressoché la stessa; questo permette di concludere che il modello è in grado di generalizzare, non adattandosi troppo al *validation data set* per poi essere in grado di generalizzare anche ad un set di dati mai visto prima (*test data set*).
Un'ultima osservazione può essere fatta pensando ad una possibile applicazione di questo algoritmo sui dati sperimentali, laddove la distinzione fra background e segnale non è nota a priori; infatti, osservando come le distribuzioni dei segnali si pongono rispetto a quella degli eventi di fondo e selezionando eventi con *loss* molto alta è possibile selezionare quelli più affini a processi di nuova fisica, e quindi fare un test di ipotesi su questi campioni selezionati.

4.4.3 Esperimento di conteggio e regione di esclusione

A questa prima analisi qualitativa ne viene fatta seguire una quantitativa, con la quale si vogliono definire per quali combinazioni delle masse delle due particelle il VAE riesce a discriminare, con una certa confidenza statistica, gli eventi di segnale da quelli di background. In questo modo sarà possibile delineare una regione di esclusione per i diversi segnali caratterizzati da diverse ipotesi circa le masse delle due particelle candidate.
Per portare a termine questo obiettivo si designa un esperimento di conteggio per confrontare l'ipotesi nulla, cioè di presenza di solo fondo nei dati, con quella alternativa che prevede invece anche la presenza del segnale. Nello specifico si procede in questo modo: dopo aver determinato il numero di eventi di fondo N_b da selezionare, si ricava il valore di soglia della Loss relativa a questo numero. Successivamente, utilizzando lo stesso valore di soglia, si selezionano anche tutti gli eventi di segnale N_s alla destra di tale valore per poi procedere all'esperimento di conteggio. In figura 19 si riporta il caso in cui si vogliano selezionare 10 eventi SM. La soglia di selezione è quindi indicata dalla linea verticale della stessa immagine.

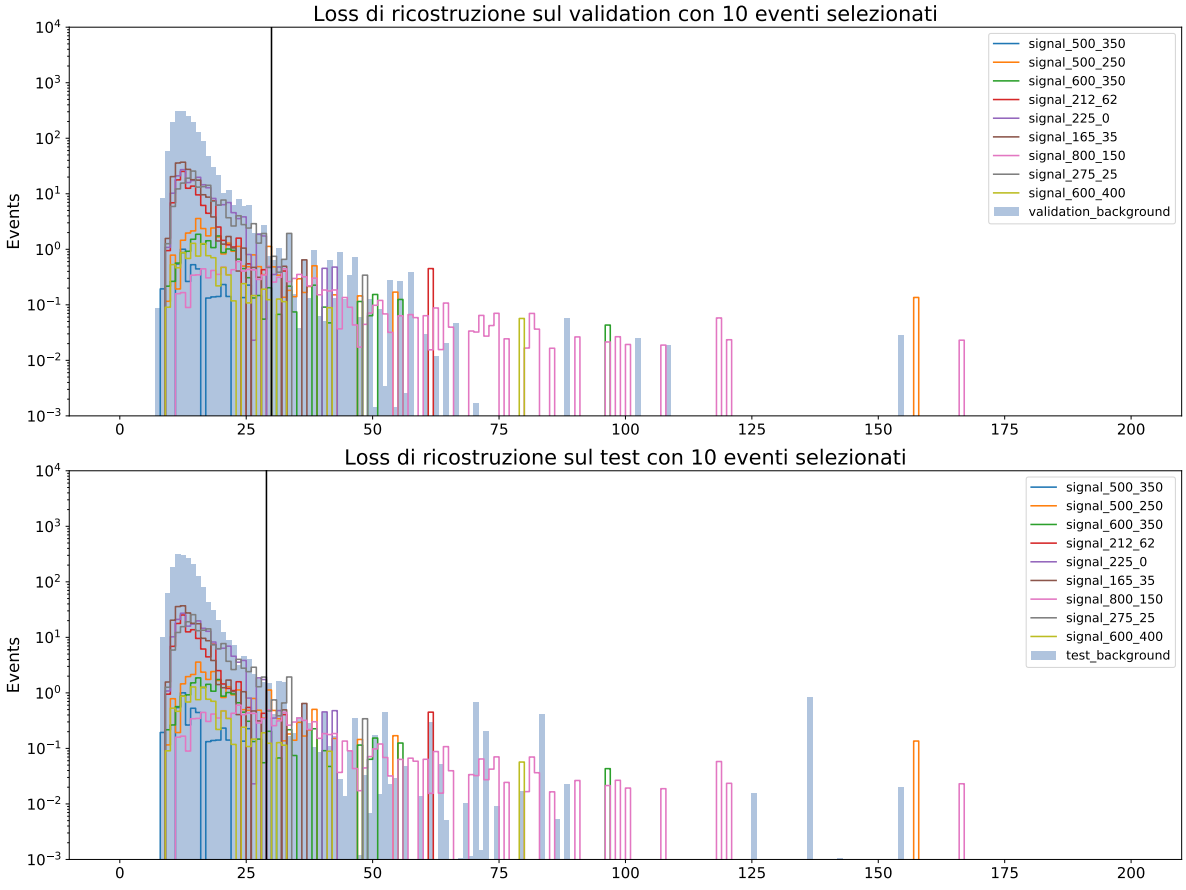

Figura 19: Figura analoga all 18, dove però la linea in nero mette in evidenza il processo di selezione degli eventi di background nella parte destra della distribuzione.

Quindi, dopo aver ricavato N_b ed N_s, si procede alla verifica dell'ipotesi nulla, ovvero la presenza di solo fondo nei dati; nello specifico si verifica che la probabilità di avere la somma $N_s + N_b$ di eventi sia compatibile con una distribuzione di Poisson centrata su di un valore pari a N_b (in altre parole si calcola $p(N_b + N_s|N_b)$). Nel caso in cui tale probabilità sia inferiore al 5%, si procede all'esclusione dell'ipotesi nulla in favore di quella

alternativa di presenza di segnale nei dati.
Nelle figure 20 e 21 sono riportati i risultati dell'esperimento di conteggio e rappresentano la cosidetta regione di esclusione dove, lungo l'asse delle ascisse sono riportate le masse, considerate degeneri, delle due particelle prodotte nella collisione pp ($\chi_1^{\pm}$ e χ_2^0), mentre lungo quello delle ordinate la massa della particella SUSY piu' leggera (χ_1^0). Questi risultati sono riportati al variare del numero N_b di eventi di fondo da selezionare per vedere se è in qualche modo possibile ottimizzare questo parametro in funzione della selezione del segnale. In altre parole si opera uno scan sui possibili valori N_b che tornerà utile nella prossima sezione.
Tornando alla descrizione di queste immagini, si osservano sia punti rossi che verdi. I primi rappresentano le particolari combinazioni delle masse delle due particelle per le quali il VAE è in grado di discriminare fra segnale e background con sufficiente confidenza statistica, mentre i punti verdi indicano quei segnali per cui tale discriminazione non può essere compiuta.
E' importante osservare come, aumentando il numero di eventi da selezionare nella zona di ipotesi con basse masse ($m < 300GeV$), si evidenzia un incremento della sensibilità al segnale; l'opposto avviene invece per le zone con $m > 600GeV$, mentre per la zona intermedia un buon risultato sembra emergere nel range $50GeV - 80GeV$ eventi di fondo da selezionare. Questo è dovuto al fatto che la sezione d'urto attesa per la produzione delle particelle SUSY diminuisce lungo l'asse delle ascisse. Questo fa si che il campione statistico del segnale sia più grande a basse masse e più piccolo a masse più alte. In questo modo la selezione deve essere maggiormente stringente al crescere della massa così da ottenere un campione adeguato per il test di ipotesi.
In ogni caso scegliendo tre selezioni diverse, una per regione, è possibile designare un'unica regione di esclusione avendo ottimizzato la procedura nelle diverse regioni di massa. E' giusto rimarcare anche come questo scan sia stato condotto su dati di validation in quanto relativo ad un processo di ottimizzazione degli iperparametri e quindi riconducibile ad un fine-tuning del modello. Come tale non può essere condotto sugli stessi dati di test su cui i risultati finali verrano estratti. Infatti, solo i risultati finali della prossima sezione prendono in considerazione il test set per evitare possibili bias sugli esiti finali.

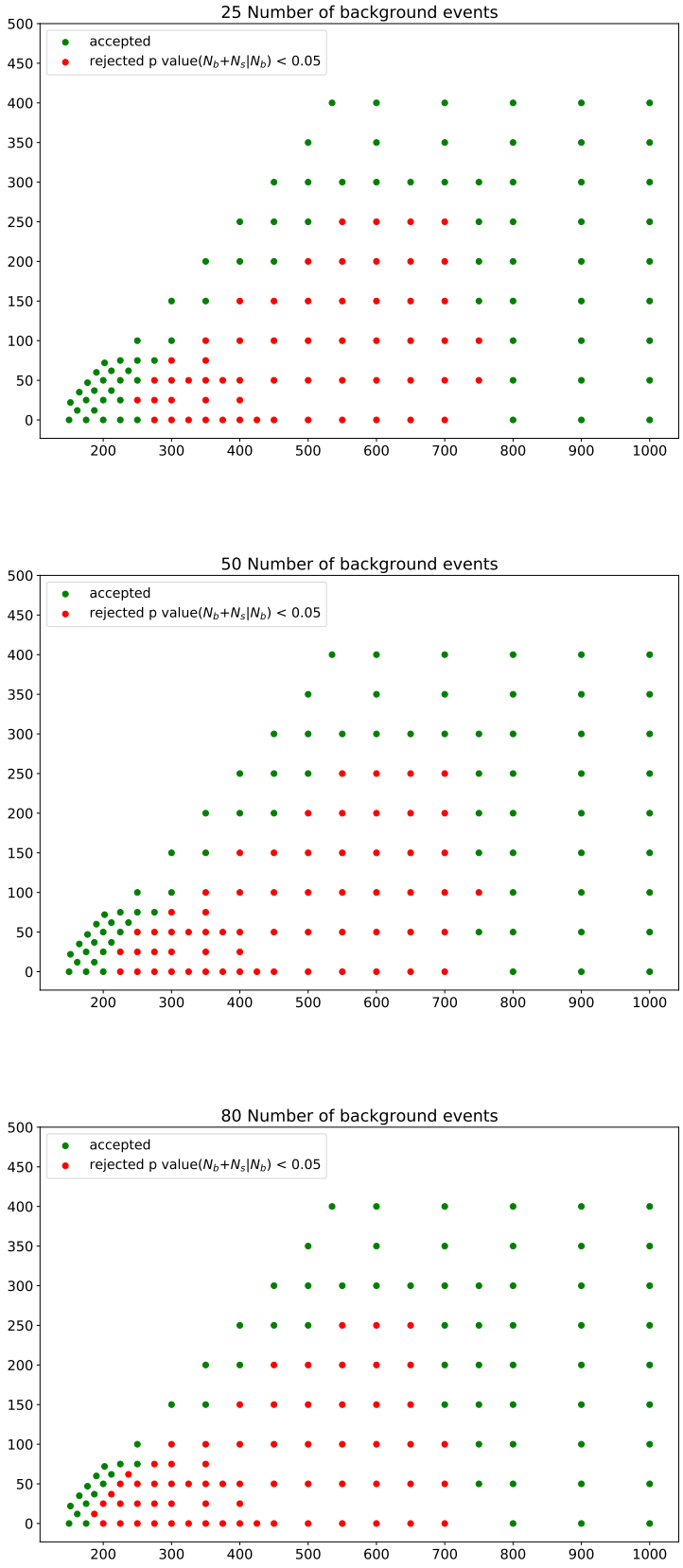

Figura 20: Risultati degli esperimenti di conteggio per, rispettivamente, 25, 50 e 80 eventi di background selezionati nella parte destra della distribuzione della *loss*.

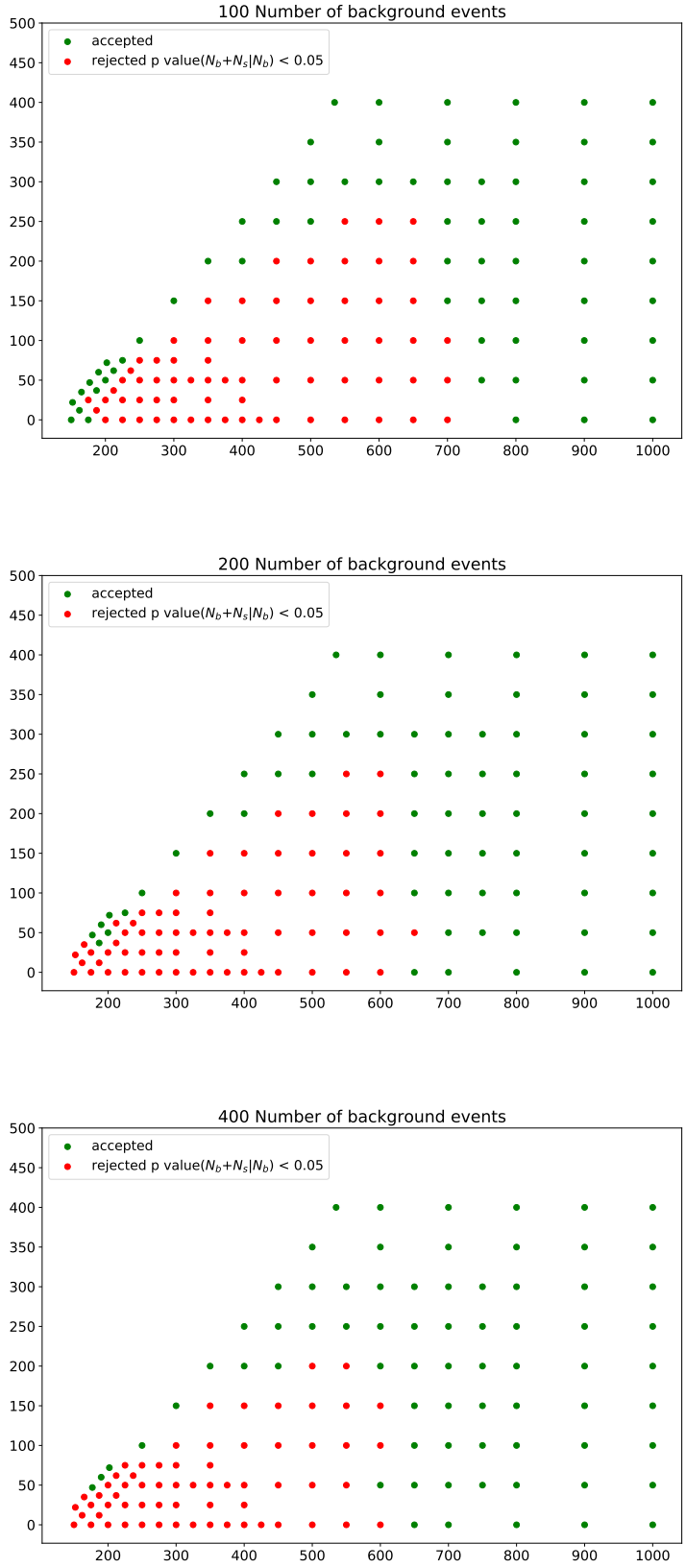

Figura 21: Risultati degli esperimenti di conteggio per, rispettivamente, 100, 200 e 400 eventi di background selezionati nella parte destra della distribuzione della *loss*.

4.4.4 Regione di esclusione ottimizzata

A questo punto dopo aver individuato le tre zone lungo l'asse delle ascisse, rispettivamente per valori $m(\chi_1^{\pm}/\chi_2^0) \leq 300GeV$, $300GeV < m \leq 600GeV$ e $m > 600GeV$, è necessario ricavare il numero ottimale di eventi di background da selezionare in modo da ottimizzare la sensibilità agli eventi di segnale. Da un semplice conteggio dei punti evidenziati in rosso si ottiene che la combinazione ottimale prevede di selezionare 400 eventi di background per la prima zona, 100 per la seconda e 25 per la terza. In figura 22 viene riportato l'esito finale dell'esperimento, selezionando per ognuna delle tre zone il valore ottimale di eventi di background da selezionare.

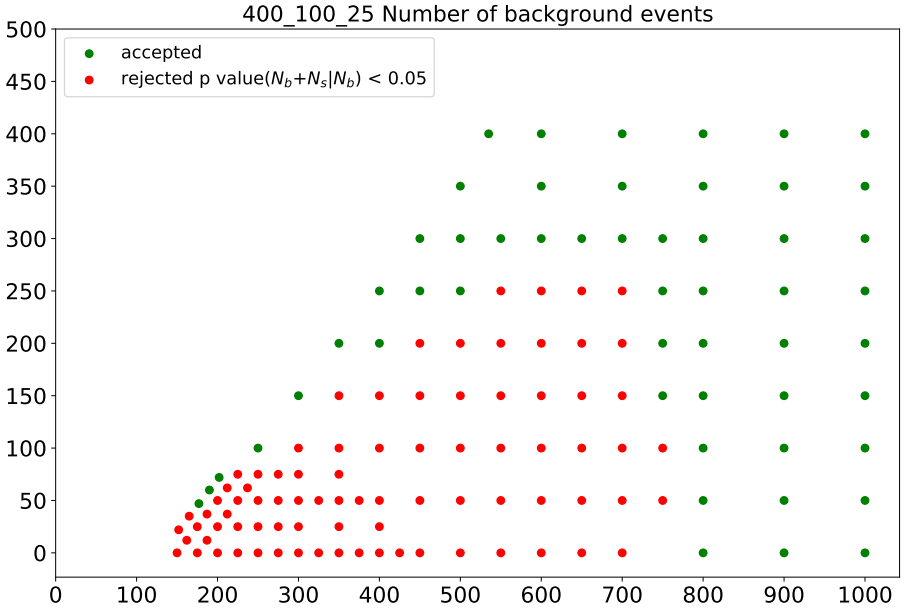

Figura 22: Risultato del processo di ottimizzazione nella distinzione fra background e segnale. Sono stati utilizzati i risultati ottimali in ciascuna delle tre zone individuate.

Questa ottimizzazione consente di raggiungere delle sensibilità al segnale SUSY analoghe a quella dei metodi *cut and count*, come si evince dalla figura 23.

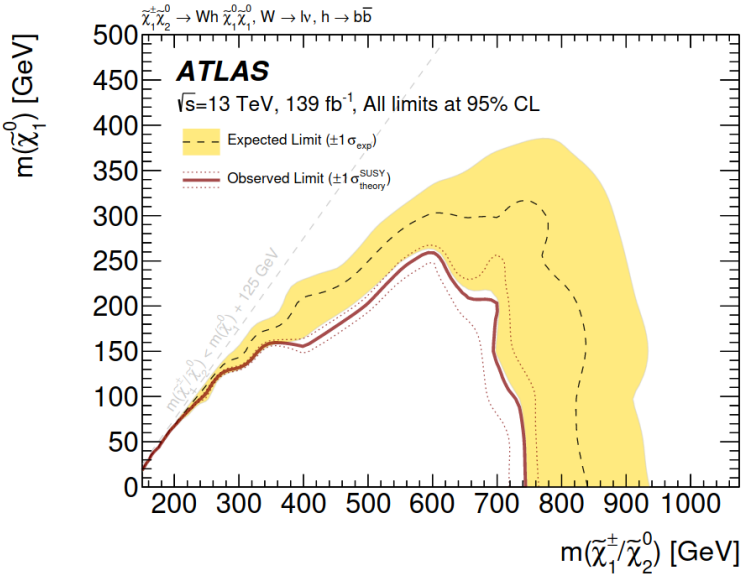

Figura 23: E' riportato l'esito di un processo *cut and count* per capirne la sensibilità al segnale per le varie combinazioni delle masse del chargino e del neutralino. L'immagine è presa da [4].

Rimangono pero' alcune questioni aperte, legate alla cattiva ricostruzione della variabile *mbb*, e se ci siano variabili più sensibili delle altre tra quelle utilizzate nella presente analisi.

4.4.5 Effetti della variazione dei pesi sulle variabili fisiche nel processo di apprendimento

Per ottimizzare ulteriormente la selezione è possibile operare sugli iperparametri del VAE. In particolare si vuole vedere in che modo cambia la sensibilità del modello agli eventi di segnale cambiando i pesi relativi alle diverse variabili che costituiscono un evento fisico. Nella precedente ottimizzazione la rigenerazione di una delle variabili (mbb) non è stata soddisfacente. Per questo motivo si è provato ad impostare un peso maggiore per questa variabile ed il risultato è riportato in figura 24 (nello specifico è stato scelto un peso pari a tre per la variabile mbb e ad uno per le altre).

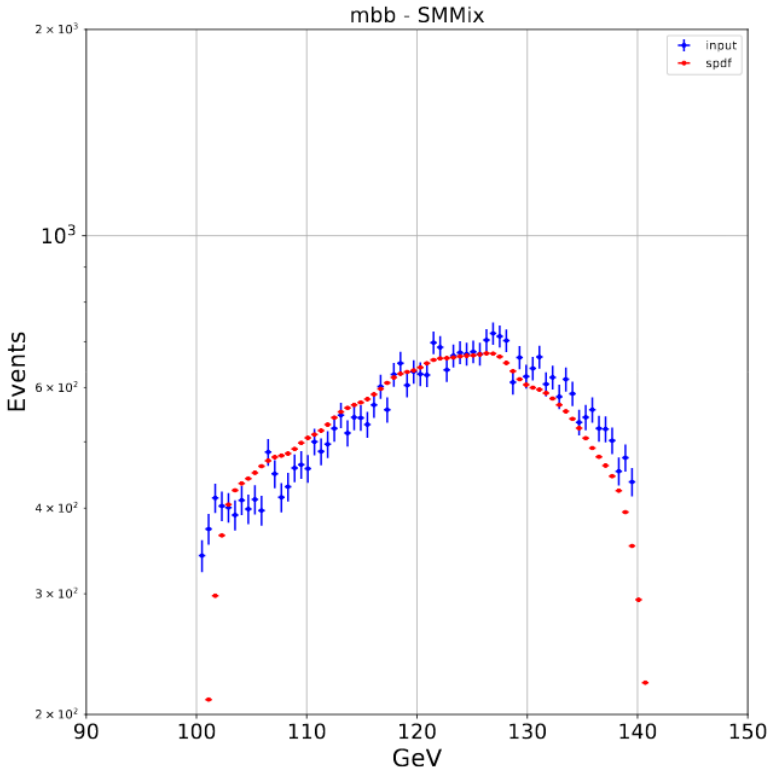

Figura 24: Esito del processo di ricostruzione della variabile mbb dopo aver impostato un peso pari a tre per tale variabile e mantenendo quelli delle altre variabili pari ad uno. In blu sono riportati i dati originali ed in rosso quelli ricostruiti.

Risulta evidente che questo piccolo accorgimento in fase di simulazione ha permesso di ottenere un'ottima ricostruzione della variabile mbb, mantenendo inalterata la qualità delle altre. Ciò è avvenuto perché, aumentando il peso su tale variabile, aumenta anche il relativo errore nella ricostruzione (l'errore viene moltiplicato per il peso); di conseguenza l'algoritmo, operando la minimizzazione della *loss* totale, viene forzato ad occuparsi maggiormente della ricostruzione della variabile mbb.
Questo spunto suggerisce che è effettivamente possibile condizionare il modello sulla ricostruzione delle diverse variabili, di conseguenza ci si chiede se vincolando l'algoritmo su alcune grandezze fisiche particolari sia possibile ottenere un processo di discriminazione migliore.

Si passa quindi a verificare se nel processo di classificazione in segnale e background vi siano alcune variabili più discriminanti di altre; per far emergere ciò bisogna assegnare pesi diversi alle diverse variabili e osservare il conseguente risultato: emerge che ci sono effettivamente tre variabili più discriminanti delle altre, cioè *met*, *mt* e *mct2*). Nello specifico sono stati assegnati i pesi rispettivamente pari a 5,10,10 a queste tre variabili ed il risultato finale ottenuto è stato riportato in figura 25, per poter essere confrontato con il risultato in figura 22 dove i pesi erano tutti pari ad uno.

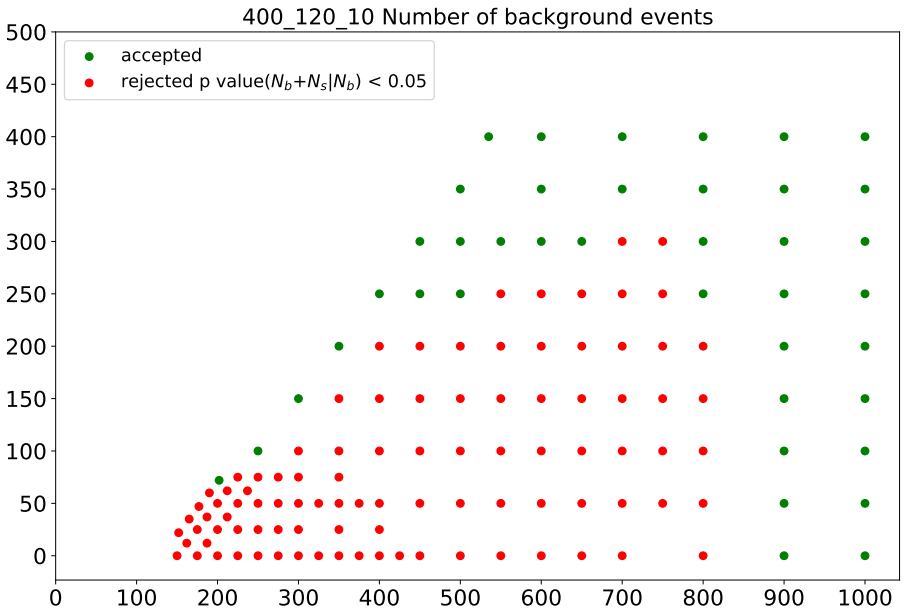

Figura 25: Risultato analogo a quello riportato in figura 22, ma utilizzando pesi differenti per le variabili più discriminanti.

Emerge dal confronto fra le due figure che quest'ultima configurazione di iperparametri permette la distinzione del segnale di background per un numero maggiore di possibili combinazioni delle masse delle due particelle. Nel primo caso con pesi tutti pari ad uno il numero totale di combinazioni delle masse delle particelle e quindi di modelli è 82, mentre in questo secondo caso si arriva a quota 96.
Quindi si giunge alla conclusione che, in questo caso specifico, pesare in maniere differente le variabili che costituiscono gli eventi permette di ottenere una sensibilità maggiore, ovvero il VAE riesce ad essere discriminante per un maggior numero di possibili eventi di segnale.

5 Conclusioni

In questa tesi è stato presentato un possibile approccio alla ricerca BSM, ovvero alla ricerca di nuova fisica oltre il Modello Standard. Sono state presentate le varie possibilità che si hanno a disposizione per la discriminazione degli eventi di segnale, ovvero quelli riconducibili a nuova fisica, e gli eventi di background, ovvero quelli riconducibili al Modello Standard già noto. Il più avanzato di questi approcci prevede l'utilizzo di algoritmi di apprendimento automatico (*machine learning*), dei quali è stata fatta un'ampia panoramica delle caratteristiche e delle metodologie più note ed utilizzate. In particolare ci si è focalizzati su un metodo di apprendimento non supervisionato, il *Variational Autoencoder* (VAE), per verificare se possa essere utilizzato nel processo di discriminazione fra segnale e background; per fare ciò il VAE è stato addestrato sugli eventi generati attraverso una simulazione Montecarlo ed i risultati finali sono stati assolutamente positivi. Nello specifico si è osservato che, a seguito del processo di addestramento sui dati di background, il VAE può essere utilizzato per la ricerca di eventi di segnale grazie al fatto che tali eventi, una volta ricostruiti, hanno un errore di ricostruzione tendenzialmente maggiore rispetto agli eventi di background. Di quest'ultimo aspetto è stata compiuta una dimostrazione qualitativa utilizzando la distribuzione della *loss* ed una quantitativa, andando a verificare per quali combinazioni delle masse delle due particelle ricercate (chargino e neutralino) il VAE fosse in grado di attuare la discriminazione.
In ultima analisi è stato effettuato un tentativo di ottimizzazione del processo tramite una variazione degli iperparametri, ovvero pesando in maniera diversa le variabili che compongono i pattern ed il risultato è stato incoraggiante perché è emerso che tale variazione degli iperparametri permette di rendere il VAE discriminante per delle combinazioni di masse per le quali precedentemente non lo era.
Quindi, per ciò che è stato appena detto, è stata dimostrata l'utilità del VAE per una ricerca di processi supersimmetrici sulla SUSY. In conclusione, come possibili sviluppi futuri, si potrebbe verificare se il VAE così addestrato risulti discriminante anche per eventi di segnale riconducibili a teorie diverse dalla SUSY.

Riferimenti bibliografici

[1] Claire Adam-Bourdarios et al. "The Higgs boson machine learning challenge". In: *HEPML* (2014).

[2] Pushpa Bhat. "Advanced Analysis Methods in Particle Physics". In: *Annual Review of Nuclear and Particle Science* (2011).

[3] Samuel Rota Bulò. *Appunti di reti neurali*. URL: https://www.dsi.unive.it/~srotabul/files/AppuntiRetiNeurali.pdf.

[4] Alberto Cervelli et al. "Search for squarks, gluinos and electroweakinos in events with an isolated lepton, jets and missing transverse momentum at $\sqrt{s} = 13$ TeV with the ATLAS detector". In: (2018). URL: https://cds.cern.ch/record/2648719.

[5] ATLAS Collaboration. "Observation of a new particle in the search for the Standard Model Higgs bosonwith the ATLAS detector at the LHC". In: *Physics Letters B 716 (2012) 1–29* (2012).

[6] T. Del-Prete. *Methods of Statistical Data Analysis in High Energy Physics*. Istituto Nazionale di Fisica Nucleare, Sezione di Pisa, Italia, 2010.

[7] Jeremy Jordan. *Introduction to autoencoders*. 2018. URL: https://www.jeremyjordan.me/autoencoders/.

[8] Donald Knuth. *Knuth: Computers and Typesetting*. URL: https://towardsdatascience.com/using-3d-visualizations-to-tune-hyperparameters-of-ml-models-with-python-ba2885eab2e9.

[9] G. Ross L. Ibanez. "Low-energy predictions in supersymmetric grand unified theories". In: *Physics Letters B* (1981). URL: https://www.sciencedirect.com/science/article/abs/pii/0370269381912004?via%3Dihub.

[10] Nils J. Nilsson. *Introduction to Machine Learning*. Department of Computer Science, Stanford University, 1998.

[11] Filippo Schiazza Roberto Morelli. *Vae code git repository*. 2020. URL: https://github.com/robomorelli/vae_bachelor_thesis_code.

[12] Joseph Rocca. *Understanding Variational Autoencoders (VAEs)*. 2019. URL: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73.

[13] F. Wilczek S. Dimopoulos S. Raby. "Supersymmetry and the scale of unification". In: *Physical Review D* (1981). URL: https://journals.aps.org/prd/abstract/10.1103/PhysRevD.24.1681.

[14] H. Georgi S. Dimopoulos. "Softly broken supersymmetry and SU(5)". In: *Nuclear Physics B* (1981). URL: https://www.sciencedirect.com/science/article/pii/0550321381905228?via%3Dihub.

[15] N. Sakai. "Naturalness in supersymmetric GUTS". In: *Zeitschrift für Physik C Particles and Fields* (1981). URL: https://link.springer.com/article/10.1007%2FBF01573998.

[16] Emily Smith. *The Hierarchy Problem*. 2019. URL: http://theory.uchicago.edu/~sethi/Teaching/P445-S2019/Emily_Smith_QFT_III_Final_Paper.pdf.