SAPIENZA
UNIVERSITÀ DI ROMA

Classificazione di jet in fisica delle alte energie con Deep Neural Networks

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea in Fisica

Candidato

Andrea Belli Contarini
Matricola 1916927

Relatore

Prof. Stefano Giagu

Anno Accademico 2021/2022

Classificazione di jet in fisica delle alte energie con Deep Neural Networks
Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: bellicontarini.1916927@studenti.uniroma1.it

Sommario

Nel capitolo 1 di questa dissertazione si descrivono in maniera generale le particelle subatomiche, con particolare focus sui quark. Da qui, si introduce il concetto di *jet* e si trattano quindi gli algoritmi volti alla loro classificazione (in particolare il *b-tagging*). Viene descritto, in seguito, il noto acceleratore di particelle *LHC*, soffermandosi specialmente sull'esperimento *ATLAS*.

Nel capitolo 2 si introduce il *Machine Learning*, presentandone le principali caratteristiche e funzionalità. Successivamente si procede alla descrizione delle Reti Neurali Artificiali, delineandone il loro funzionamento. Il capitolo si conclude con l'introduzione a due tipologie di reti neurali: la *Feed-Forward Neural Network* e la *Long-Short Term Memory*, una delle Reti Neurali Ricorrenti più utilizzate ed efficaci.

Nel capitolo 3 viene presentato un problema di classificazione del sapore di *jet* adronici. Utilizzando la piattaforma *Google Colaboratory*, si analizza il *dataset* (frutto di una simulazione), si trovano gli iperparametri che garantiscono la corretta classificazione e si implementano ed allenano, infine, le due reti neurali discusse nel capitolo 2. I risultati ottenuti sono presentati e discussi, coadiuvati da appositi grafici, matrici di correlazione e tabelle per garantire un'efficiente analisi statistica.

La dissertazione si conclude con un sunto dei risultati finali ottenuti nel progetto di sviluppo dei due algoritmi. Inoltre, vengono presentate delle note aggiuntive finali riguardanti il terzo *run* di *LHC* ed il fondamentale ruolo che sta svolgendo (e svolgerà) il *Machine Learning* in questa attività di ricerca di nuova fisica.

Indice

Capitolo 1

Presupposti teorici

1.1 Particelle subatomiche

Una particella subatomica è una particella di massa inferiore a quella di un atomo. Essa può essere **elementare**, ossia un fermione (spin 1/2) non costituito da altre particelle (come ad esempio l'elettrone), o **composta** da altre particelle (come gli adroni). La definizione fisica esatta di particella dipende dal contesto: può essere definita come "funzione d'onda collassata", "eccitazione di un campo", o più semplicemente come un oggetto misurato da un rivelatore.

Particelle elementari

Il *Modello Standard* (*MS*), ossia la teoria fisica che descrive tre delle quattro interazioni fondamentali note (forte, elettromagnetica e debole), classifica, per composizione, le seguenti particelle elementari:

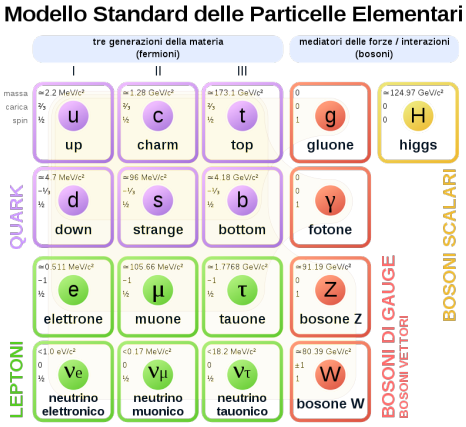

Figura 1.1. Classificazione delle particelle elementari descritte dal *Modello Standard*.

- Sei sapori[1] di **quark**: *up, down, strange, charm, bottom* e *top*.
- Sei tipi di **leptoni**: elettrone, muone, tauone e neutrino elettronico, neutrino muonico e neutrino tauonico.

[1] Il sapore è un insieme di numeri quantici, che caratterizzano tipi diversi di quark, altrimenti indistinguibili in base ad altre proprietà

- I **bosoni di gauge** (mediatori delle forze): il fotone (forza elettromagnetica), i bosoni W e Z (forza debole) ed i gluoni (forza forte).
- Il **bosone di Higgs**, teorizzato nel 1964 e rilevato per la prima volta nel 2012 negli esperimenti *ATLAS* e *CMS*.

Particelle composte

Le particelle subatomiche composte sono stati legati di particelle elementari. Esse includono gli **adroni**, particelle soggette all'interazione forte composte da quark, antiquark e gluoni. Si distinguono in due tipi:

- **Barioni**, come i nucleoni (protone e neutrone) o gli iperoni (tipo il Λ o il Σ), formati da combinazioni di tre quark o tre antiquark.
- **Mesoni**, come ad esempio il kaone o il pione, sono bosoni costituiti da una coppia quark-antiquark, aventi carica di colore[2] opposta.

1.2 Jet

Un ***jet*** è un insieme collimato di adroni e altre particelle prodotte dalla cosiddetta **adronizzazione**, ossia un fenomeno consistente nella materializzazione in adroni di uno stato costituito da quark e gluoni. La presenza dei *jet* è una prova dell'esistenza dei quark, che non possono essere osservati direttamente [12].

Come mostrato in figura 1.2, per ogni interazione si generano gruppi di particelle che raggiungono il rivelatore dopo diverse frammentazioni.

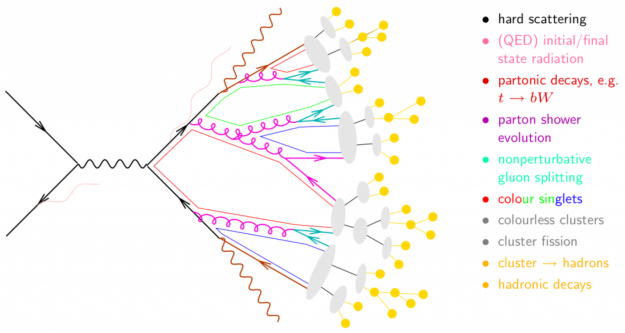

Figura 1.2. Schema del processo di produzione di *jet* adronici nell'acceleratore *LHC*: due quark iniziali interagiscono formandone altri due nello stato finale, che adronizzano per formare due *jet* [5].

La definizione di *jet* include un **algoritmo** di *jet* e uno **schema di ricombinazione** [9]. Il primo definisce come alcuni input (come ad esempio particelle o oggetti del rivelatore) vengono raggruppati in getti, mentre il secondo specifica come viene assegnata una quantità di moto ad un getto. Negli esperimenti di fisica delle particelle, i *jet* sono solitamente costruiti da gruppi di depositi di energia nel calorimetro del rivelatore. Quando si studiano processi simulati, i *jet* del calorimetro

[2] La carica di colore è un numero quantico associato ai quark e ai gluoni, che ne caratterizza le reciproche interazioni e altre proprietà, per esempio la possibilità che due quark identici si accoppino per formare una particella, nonostante, per il principio di esclusione di Pauli, non potrebbero trovarsi nello stesso stato.

possono essere ricostruiti sulla base di una risposta simulata del rivelatore.

Un buon algoritmo per la ricostruzione dei *jet* consente in genere di ottenere insiemi simili di questi, a diversi livelli nell'evoluzione dell'evento. Tipici algoritmi di ricostruzione sono, ad esempio, l'algoritmo k_T, basato sulla ricombinazione sequenziale di coppie di oggetti, nel tentativo di ricomporre le frammentazioni successive all'interazione elementare, o l'algoritmo "del cono", preordinato a trovare quelle regioni geometriche che massimizzano l'impulso degli oggetti elementari giacenti in un determinato volume.
Lo schema di ricombinazione determina come combinare i due quadrimpulsi dei due *cluster* in un singolo quadrimpulso totale. Il più utilizzato è rappresentato dal cosiddetto *schema E*, in cui i due quadrivettori vengono semplicemente sommati, al fine di ottenere quello totale; esso è preferibile anche perché invariante sotto trasformazioni di Lorentz [3].

Un esempio di tecnica di analisi di *jet* è il cosiddetto *flavor tagging*, come ad esempio il *b-tagging*, relativo ai quark *bottom* (o *beauty*).
Nel capitolo 3 verrà presentato e risolto un problema (simulato) proprio di *flavor tagging*.

1.2.1 Cenni sul *b-tagging*

Individuare il tipo di quark da cui si è originato il *jet* è importante per comprendere i processi fisici che hanno avuto luogo in seguito ad una collisione. Il metodo più importante di identificazione del tipo di quark da cui si è formato un *jet* è proprio il *b-tagging*.
Gli algoritmi di *b-tagging* si fondano sulle proprietà degli **adroni *b***, la cui fisica è fondamentalmente legata a quella dei *b*-quark. Questi ultimi hanno una vita media piuttosto lunga ($\tau \simeq 1.6\ ps$) se confrontata con il tempo medio di adronizzazione. Essi - difatti - adronizzano in barioni o mesoni prima di decadere, percorrendo una distanza misurabile, data dalla formula relativistica: $d = \beta\gamma c\tau$.
Come illustrato in figura 1.3, il punto dove si forma l'adrone b è detto "vertice primario" (PV). Viene invece detto "vertice secondario" (SV) il punto in cui esso decade.

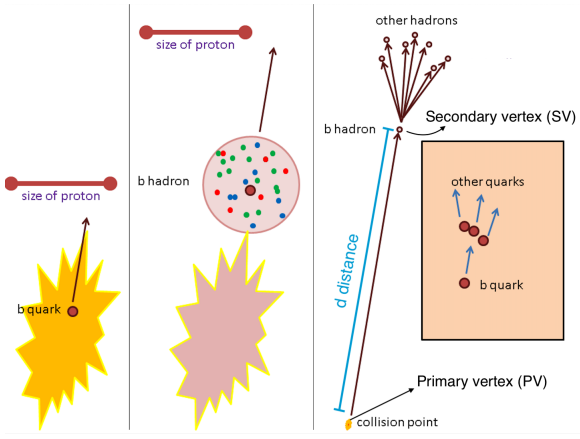

Figura 1.3. A sinistra: in una collisione pp (o $p\overline{p}$), si crea un b-quark. Al centro: attorno al b-quark si forma un adrone b. A destra: dopo aver percorso una distanza d, il b-quark decade, causando la frammentazione dell'adrone b in adroni diversi e più leggeri.

Gli algoritmi di *b-tagging* si basano sull'analisi di determinate osservabili assegnate ad ogni *jet*. Le proprietà usate per individuare *b-jet*, alla base di questi algoritmi, sono:

- Il **vertice secondario**.
- La **massa** degli adroni b. In particolare, i b-quark che li compongono sono più pesanti dei prodotti di decadimento. Così, si hanno molte particelle nel *jet* ed una grande quantità di moto trasversale, che rende i *b-jet* solitamente più ampi rispetto ai *jet* generati da quark leggeri.
- Il **parametro d'impatto** d_0, definito come la distanza minima del vertice primario dalla direzione della traccia.

Si mostra nella figura che segue (1.4) un diagramma illustrativo per l'dentificazione dei *b-jet*.

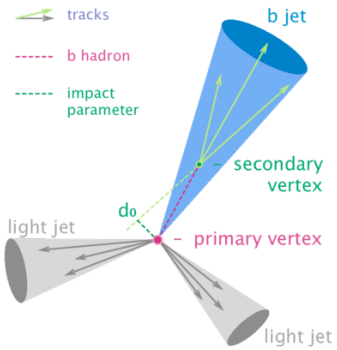

Figura 1.4. Diagramma che illustra l'identificazione di *jet* prodotti da adroni b.

Tuttavia, potendo sfruttare un numero elevato di informazioni senza penalizzazioni dal punto di vista della complessità computazionale, è utile - al fine di descrivere le proprietà del *jet* - considerare ulteriori osservabili sperimentali di alto livello, quali:

- La **significanza** del parametro d'impatto **trasversale** $S_{d_0} = d_0/\sigma_{d_0}$, dove σ_{d_0} è l'incertezza su d_0, ed il segno è definito positivo se il punto di minimo approccio è davanti al vertice primario rispetto alla direzione del *jet*, negativo altrimenti.
- La **significanza** del parametro d'impatto **longitudinale** $S_{z_0} = z_0/\sigma_{z_0}$, dove z_0 è lo spostamento longitudinale dal punto di minimo approccio al vertice primario (stessa convenzione di S_{d_0} per i segni) e σ_{z_0} è l'incertezza su z_0.
- La frazione della **quantità di moto trasversa** associata ad una traccia: $p_T^{fr} = p_T^{traccia}/p_T^{jet}$
- La **distanza angolare** tra una traccia e l'asse del *jet*: $\Delta R(traccia, jet)$, definita più avanti, nella formula (1.5).

Tutte queste grandezze sono sensibili alle caratteristiche (massa e vita media) del quark che da origine al *jet*.

1.3 LHC

Il *Large Hadron Collider* (*LHC*) è l'acceleratore di particelle più potente mai sviluppato. Esso è situato al *CERN* di Ginevra e consta di un tunnel sotterraneo circolare, di lunghezza circa 27 km, all'interno del quale vengono accelerati, a velocità prossime a quelle della luce, e fatti collidere, due fasci di particelle (protoni o ioni pesanti).

Le particelle sono guidate da campi magnetici di circa 8 T, generati da magneti superconduttori raffreddati ad una temperatura di circa 2 K con elio liquido. Essi sono dipoli magnetici (grazie ai quali viene mantenuta la traiettoria dei fasci) e quadrupoli magnetici, per mezzo dei quali si riescono a concentrare i fasci, facendoli collidere in 4 posizioni in corrispondenza dei 4 rivelatori di particelle:

- ***LHCb*** (***L**arge **H**adron **C**ollider **b**eauty*), che ha come obiettivo quello di valutare la violazione della simmetria CP (coniugazione di carica e parità) e studiare fenomeni relativi agli adroni in cui è presente il quark *beauty*, da cui il nome dell'esperimento.
- ***ALICE*** (***A** **L**arge **I**on **C**ollider **E**xperiment)*, dedicato allo studio del *quark-gluon plasma* (*QGP*), prodotto dalla collisione di nuclei pesanti (come quelli di ^{208}Pb). Il *QGP* è uno stato della cromodinamica quantistica, nel quali si ritiene che l'Universo si sia trovato, per i primi 20-30 microsecondi della sua esistenza.
- ***ATLAS*** (***A** **T**oroidal **L**HC **A**pparatu**S***), trattato più approfonditamente nella prossima sezione (1.3.1).
- ***CMS*** (***C**ompact **M**uon **S**olenoid*), che ha gli stessi obiettivi di *ATLAS*, ma utilizza tecnologie di rivelazione differenti.

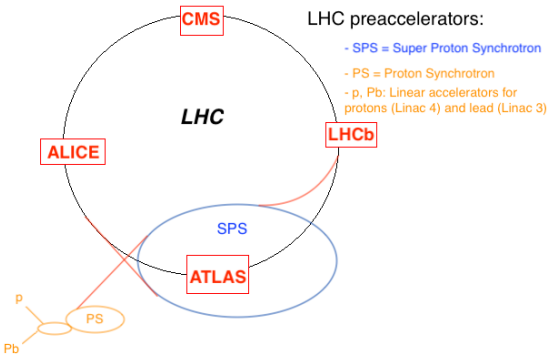

Figura 1.5. Schema illustrativo di *LHC*.

Prima di entrare nell'anello principale (in nero nella figura 1.5), le particelle vengono preparate da una serie di sistemi che ne aumentano progressivamente l'energia. Il primo sistema è l'acceleratore lineare *Linac 4*[3], che genera ioni idrogeno H^- di energia 160 MeV. Questi sono poi iniettati nel *PSB* (*Proton Synchrotron Booster*), dove i due elettroni sono "strappati" dagli ioni; rimane dunque solo il

[3]Solitamente il programma *LHC* prevede collisioni protone-protone; tuttavia, meno spesso, vengono anche studiate collisioni tra ioni pesanti, come il piombo. Tali ioni vengono accelerati in un altro acceleratore: il *Linac 3*.

nucleo contenente un protone. I protoni sono poi accelerati a 2 GeV ed entrano nel PS (*Proton Synchrotron*) ed accelerati ulteriormente fino a 26 GeV. Infine il *Super Proton Synchrotron* (SPS) incrementa tale energia a circa 450 GeV. Il fascio di protoni, dunque, entra nell'anello principale, in cui vengono ulteriormente accelerati (per circa 20 minuti) e circola per ore mentre avvengono le collisioni nelle quattro zone corrispondenti ai rivelatori.

Lo scopo di LHC è quello di creare particelle per verificare le previsioni del Modello Standard o per trovare prove di nuova fisica. L'intento principale è lo studio di collisioni fra fasci di protoni ad un'energia, nel centro di massa, di $\sqrt{s} \simeq 14\ TeV$, ad una luminosità istantanea di circa $10^{34} cm^{-2} s^{-1}$. La luminosità istantanea (L_{inst}) quantifica le capacità di produrre collisioni. Maggiore è la luminosità nell'acceleratore, maggiore è la probabilità di rivelare fenomeni rari, che hanno una sezione d'urto bassa. L_{inst} è definita dal numero di particelle collidenti (N_1 e N_2, nel caso di urto fra due fasci) per unità di tempo (f, frequenza) e per unità di superficie $S = 4\pi\sigma_x\sigma_y$ (con σ_x e σ_y che sono le dimensioni trasverse del fascio, considerato distribuito secondo una gaussiana):

$$L_{inst} = \frac{N_1 N_2}{S} f \tag{1.1}$$

Data la sezione d'urto di un processo (σ), il numero medio di interazioni al secondo, il cosiddetto *rate* (ϱ), si ottiene da:

$$\varrho = \frac{dN}{dt} = \sigma L_{inst} \tag{1.2}$$

1.3.1 Il rivelatore ATLAS

$ATLAS$ è un rivelatore di particelle lungo oltre 44 m, alto 25 m e pesa oltre 7000 tonnellate. È composto da diversi "sotto-detector", come mostrato nella figura 1.6. Il rivelatore ha una simmetria cilindrica, mostrata in figura 1.7 (b), e consiste di tre strati.

Il primo strato è l'***Inner Tracker***, progettato per ricostruire le tracce di particelle cariche create durante le interazioni. Esse sono importanti per il *b-tagging* perché permettono di ricostruirne i vertici primari e secondari.
Il secondo è composto dai **calorimetri**, principalmente elettromagnetico ed adronico, poiché l'interazione di un adrone con il materiale di un detector è differente da quella di un elettrone o di un fotone, perché esso può anche interagire attraverso la forza forte. Infine, nella regione più esterna, si trova lo **spettrometro per muoni**.

Sistema di coordinate

$ATLAS$ utilizza un sistema di coordinate polari in cui l'asse Z è posto lungo la direzione dei fasci, l'asse X punta verso il centro dell'anello di LHC, e l'asse Y che punta "verso l'alto". L'angolo azimutale ϕ è definito attorno alla direzione di propagazione del fascio, partendo dall'asse X. L'angolo polare θ è l'angolo misurato dall'asse Z.
Poiché la variazione di angolo polare $\Delta\theta$ non è invariante sotto trasformazioni di Lorentz (a differenza di $\Delta\phi$), conviene definire una nuova variabile invariante sotto tali trasformazioni, la ***pseudorapidità***:

$$\eta = -ln\left[tan\left(\frac{\theta}{2}\right)\right] \tag{1.3}$$

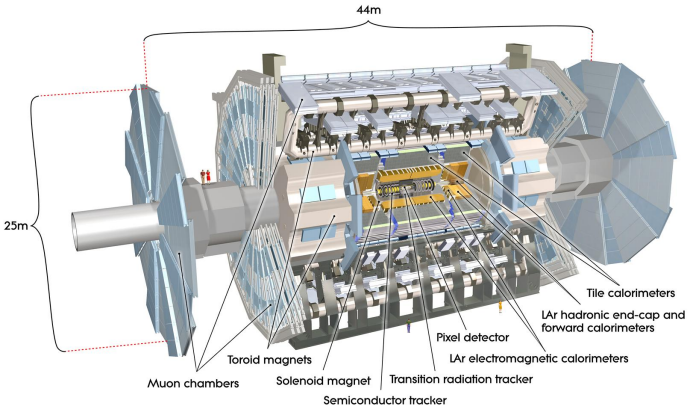

Figura 1.6. Schema illustrativo di *ATLAS* [10].

Essa, nel limite di particelle senza massa, è equivalente alla *rapidità*, definita come:

$$y = \frac{1}{2} ln\left(\frac{E + p_z}{E - p_z}\right) \tag{1.4}$$

Così facendo, $\Delta\eta$ (o Δy) sono invarianti sotto trasformazioni di Lorentz. Dunque la distanza tra due punti ΔR può essere riscritta in termini di η e ϕ:

$$\Delta R = \sqrt{\Delta\phi^2 + \Delta\eta^2} \tag{1.5}$$

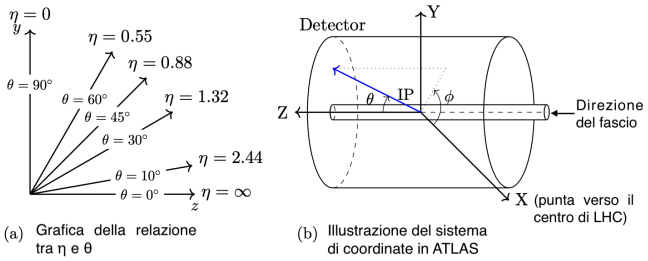

Figura 1.7. Relazione tra pseudorapidità e angolo polare (a) e simmetria cilindrica del rivelatore (b).

Mediante l'uso della pseudorapidità si individuano due particolari regioni del rivelatore:

- *Barrel region*, centrale e più vicina alla zona d'interazione ($|\eta| < 1.05$).
- *End-cap region*, riguarda le estremità del rivelatore ($1.05 < |\eta| < 2.7$).

Per $2.7 \leq |\eta| \leq 4.5$ può invece succedere che le particelle non siano rivelate correttamente perché il loro percorso risulta escluso dalla zona di sensibilità del rivelatore.

Capitolo 2

Machine Learning e Reti Neurali Artificiali

2.1 Introduzione al Machine Learning

Il *Machine Learning* (*ML*), di cui il *Deep Learning* costituisce una delle branche d'avanguardia, è un sottoinsieme dell'intelligenza artificiale (*AI*), la cui peculiarità è la capacità degli algoritmi di poter imparare da un *set* di esempi a disposizione.

Alcuni problemi tipici che possono essere affrontati da algoritmi di *ML* sono quelli di **classificazione**, come ad esempio i problemi di identificazione di particelle, la **regressione**, cioè quando è necessario associare ad ogni dato un valore continuo invece che un numero discreto legato all'appartenenza ad una classe, oppure la **generazione** di immagini o suoni a partire da alcuni esempi.

L'apprendimento di un algoritmo di *Machine Learning* può essere di tre diverse tipologie:

- *Supervised learning*: l'algoritmo viene esposto prima ad un campione di dati che presenta già una label (ossia quando è nota la classe di appartenenza o il valore target in un *task* di regressione).
- *Unsupervised learning*: durante l'apprendimento vengono forniti solo esempi non annotati, in quanto le classi non sono note a priori ma devono essere apprese automaticamente.
- *Reinforcement learning*: punta a realizzare agenti autonomi in grado di scegliere azioni da compiere per il conseguimento di determinati obiettivi tramite interazione con l'ambiente in cui sono immersi. A differenza degli altri due, questo paradigma si occupa di problemi di decisioni sequenziali, in cui l'azione da compiere dipende dallo stato attuale del sistema e ne determina quello futuro.

La **prestazione** di un algoritmo viene valutata in base alla *task* che l'algoritmo stesso deve risolvere. Per i problemi di classificazione, tipicamente si assegna un "punteggio" per ogni risposta corretta, che poi viene diviso per il numero totale di previsioni effettuate. Nei problemi di regressione, invece, vengono solitamente utilizzate delle metriche che misurano la vicinanza del risultato predetto da quello corretto.

Il vantaggio di ogni buon algoritmo di *Machine Learning* è quello di poter usare efficacemente quanto appreso da un certo campione di addestramento su un set di dati diverso, ma della stessa tipologia. Questa proprietà viene detta "generalizzazione".

Quando l'algoritmo non ci riesce, ed esso apprende anche caratteristiche specifiche del set di dati con cui era stato allenato, allora si parla di *overfitting*. Viceversa, si parla di *underfitting* se non è riuscito ad imparare l'informazione utile presente nel dataset.

Infine è importante ricordare il *No Free Lounch Theorem*, risultato teorico a cui giunse il fisico David Wolpert, secondo cui, se si media su tutte le possibili distribuzioni di probabilità, le prestazioni di ogni algoritmo di classificazione valutate su dati non osservati dall'algoritmo in precedenza sono le stesse.
Ciò significa che non può esistere un algoritmo universalmente migliore di un altro. È quindi il problema specifico a determinare la scelta di quale metodo di *ML* utilizzare.

2.2 Le Reti Neurali Artificiali

Una *Artificial Neural Network* (*ANN*) è un modello computazionale composto di "neuroni" artificiali (ispirato vagamente dalla semplificazione di una rete neurale biologica): nello specifico, ciascuno di questi neuroni (o "nodi") riceve un input e manda in output un segnale processato, adattato sulla base di alcuni parametri forniti alla rete neurale e che agiscono indipendentemente.
Le *ANN* sono spesso rappresentate tramite diagrammi (come quello in figura 2.1), che permettono di capire come i vari neuroni sono connessi tra loro[1]. È possibile notare che i nodi possono essere pensati come "unità" che agiscono indipendentemente in *layer*. Nello specifico, si distinguono:

- *Input layer*: contiene i nodi associati alle variabili di input.
- *Output layer*: contiene i nodi associati alle variabili di output.
- *Hidden layer*: sono tutti quelli compresi tra *input layer* ed *output layer*.

Matematicamente, una Rete Neurale Artificiale può essere descritta come una serie di trasformazioni funzionali: è quindi possibile ottenere M combinazioni lineari delle variabili di input $x_1, ..., x_D$ nel seguente modo:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(t)} x_i + w_{j0}^{(t)} \tag{2.1}$$

in cui $j = 1, ..., M$ e l'apice (t) indica che ci si trova nel t-esimo *layer* della rete. Inoltre, i parametri w_{ji} vengono detti *pesi* della rete, mentre i parametri w_{j0} sono i *bias*. Infine, le quantità a_j sono chiamate *attivazioni*; esse non sono ciò che verrà processato direttamente dai neuroni che si trovano nei *layer* nascosti o dall'*output*, in quanto devono essere trasformate tramite una trasformazione differenziabile, che prende il nome di ***funzione di attivazione***:

$$z_j = h(a_j) \tag{2.2}$$

I passaggi appena descritti sono quindi ripetuti, considerando gli z_j come i nuovi input da fornire all'equazione 2.1. A titolo di esempio, si riporta in figura 2.1 una schematizzazione di una rete neurale a tre *layer*.

[1] Quello in figura, più propriamente, è un *grafo computazionale*, ossia un oggetto discreto che permette di schematizzare una grande varietà di processi, consentendone delle analisi in termini quantitativi e algoritmici.

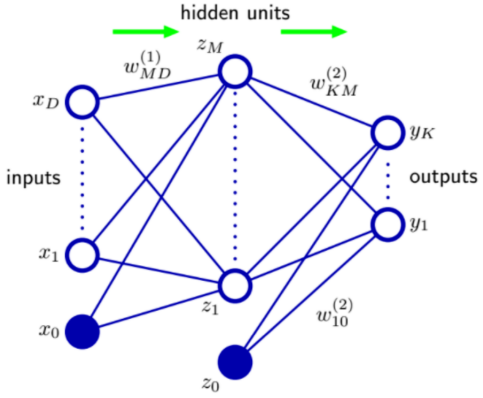

Figura 2.1. Esempio di rete neurale con tre *layer* (*input*, un *hidden* e *output*). I pesi associati a ciascun *layer* sono rappresentati dai collegamenti tra i nodi, mentre i vari *bias* sono rappresentati, rispettivamente, da input addizionali e variabili nascoste (x_0 e z_0). Le frecce (in verde) identificano la direzione di propagazione dell'informazione.

Alla fine del processo, le unità di attivazione dell'output sono quindi trasformate tramite la funzione di attivazione per dare un set di variabili di output: $y_1, ..., y_K$.

La funzione di attivazione non può essere lineare per tutte le unità nascoste; in questo modo la rete sarebbe equivalente ad un'altra priva di *hidden layer*, visto che la composizione di successive trasformazioni lineari è anch'essa una trasformazione lineare.
Solitamente, dunque, si usano funzioni come la sigmoide o la $ReLU$, definite come:

$$\sigma(x) = \frac{1}{1+e^{-x}}; \quad ReLU(x) = MAX(0, x) \tag{2.3}$$

Si possono chiaramente trovare variazioni di queste funzioni o altre diverse, come ad esempio la tangente iperbolica, spesso giustificate su base euristica.

Le ***Feed-Forward Neural Network*** ($FFNN$) sono una diretta applicazione di quanto analizzato sinora: l'informazione in questo caso procede dall'input all'output, senza che vi siano delle connessioni di *feedback* che permettano di fornire le informazioni sull'output ai *layer* precedenti. In altre parole, le informazioni procedono dagli input, passando per ciascun *hidden layer*, producendo alla fine degli output. Uno schema di una rete neurale di questo tipo è presentato in figura 2.2.

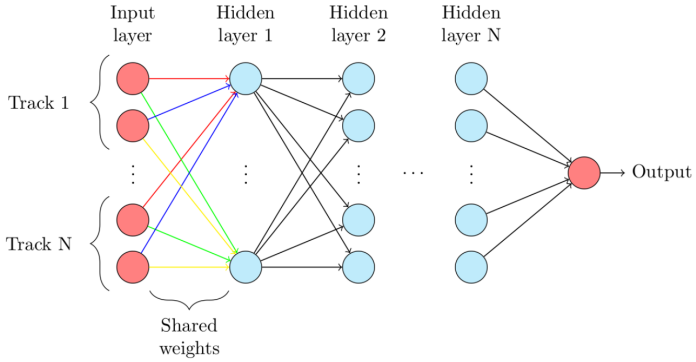

Figura 2.2. Schema della struttura di una rete neurale di tipo *Feed-Forward* [2].

Nel caso più semplice si ha solo un *hidden layer*; reti con questo tipo di architettura vengono dette *shallow*, ma è possibile inserire molti altri *hidden layers* e in questo caso si definiscono *Deep Neural Networks* (*DNN*). Nella pratica, è stato notato che per problemi complessi e ad alta dimensionalità, una rete *deep* con più *layer* nascosti riesce ad essere addestrata più efficacemente.

Una volta introdotta la rete neurale è necessario spiegare come essa apprenda (processo di *training*). Nei casi di *supervised learning*, il *training* viene fatto minimizzando una funzione detta ***loss function***, che misura la distanza tra la funzione predetta $\vec{N}(\vec{\theta}, \vec{x})$ ed il vero valore $\vec{f}(\vec{x})$. La migliore approssimazione possibile si ottiene quindi con il set di parametri interni alla rete $\vec{\theta}$ che rendono minima la *loss function*, cioè quelli che ne azzerano il gradiente rispetto ai parametri:

$$\vec{\nabla}_{\vec{\theta}} \mathcal{L}[\vec{N}(\vec{\theta}, \vec{x}), \vec{f}(\vec{x})] \stackrel{!}{=} 0$$

Poiché la vera funzione $\vec{f}(\vec{x})$ non è nota, ma si conoscono solo i valori presi in corrispondenza dei punti del *test set*, si applica un algoritmo detto di ***backpropagation***, che permette di calcolare i gradienti locali.

Una tipica *loss function* utilizzata nei problemi di classificazione, come quello presentato nel capitolo 3, è la cosiddetta *Cross Entropy*, che, date due distribuzioni discrete f e N (che fissata f misura quanto N differisce da f), è definita da:

$$H(f, N) = -\sum_{\vec{x}} f(\vec{x}) log[N(\vec{x})] \tag{2.4}$$

Riassumendo, l'algoritmo di apprendimento procede allora in tre fasi:

1. Calcolo dell'*output* della rete: $\vec{Y}_{\vec{N}} = \vec{N}(\vec{\theta}, \vec{x})$
2. Calcolo della *loss*: $\mathcal{L} = |\vec{Y}_{\vec{N}} - \vec{Y}_{true}|$
3. Propagazione, sui parametri, dell'andamento della *loss function* attraverso la *backpropagation*; dopo aver fatto ciò, sarà però necessario riaggiornare i parametri, al fine di apprestarsi ad uno dei minimi della *loss*. Questo processo è noto come *Gradient Descent* ("discesa lungo il gradiente").

2.2.1 Recurrent Neural Network

Le *Recurrent Neural Network* (*RNN*) sono una classe di reti neurali artificiali che include neuroni collegati tra loro in un ciclo.

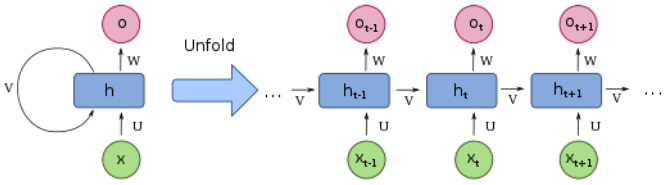

Figura 2.3. Questo grafico mostra i diversi modi d'illustrare i grafi computazionali per una *RNN*, passando da una rappresentazione più sintetica ad una che si basa sull'*unfolding*, che permette di evidenziare la struttura ripetitiva della rete.

L'idea alla base è piuttosto diversa da quella delle *FFNN* e una rappresentazione della loro struttura è riportata in figura 2.3. Nello specifico, si consideri una sequenza di vettori x_t, con $t = 1, ..., \tau$, dove t assume la connotazione di indice di posizione all'interno di una sequenza. Nelle *RNN*, ciascun membro dell'output (con indice t) è una funzione dei precedenti membri dell'output (indicati con l'indice $t-1$).

Long Short-Term Memory

Le ***Long Short-Term Memory*** (*LSTM*) sono nate per ovviare ad uno dei più grandi problemi delle *RNN*: l'aggiornamento dei pesi tramite la *backpropagation*. Durante questa fase, in una generica *RNN*, gli errori calcolati nel processo di minimizzazione della *loss function* tendono ad "esplodere" o ad annullarsi, portando al malfunzionamento dell'algoritmo.
Il modello *LSTM* consente di risolvere questo problema introducendo delle *celle*, che presentano una ricorrenza interna (detta *self loop*). Così, l'informazione è gestita da altri *hidden layer*, che prendono il nome di *gate*. Le componenti principali di una cella sono dunque le seguenti (schematizzate in fig. 2.4):

- *State unit*: funziona da unità di memorizzazione della cella; essa viene aggiornata e gestita tramite i vari *gate*.
- *Input gate*: controlla l'informazione in ingresso, preservandola da perturbazioni in *input*.
- *Forget gate*: permette di gestire i pesi associati al *self loop*, decidendo quale informazione debba essere rimossa e permettendo, così, di aggiornare la *state unit*.
- *Output gate*: accede all'informazione connessa alle celle in uscita e preserva l'informazione da eventuali perturbazioni contenute in esse.

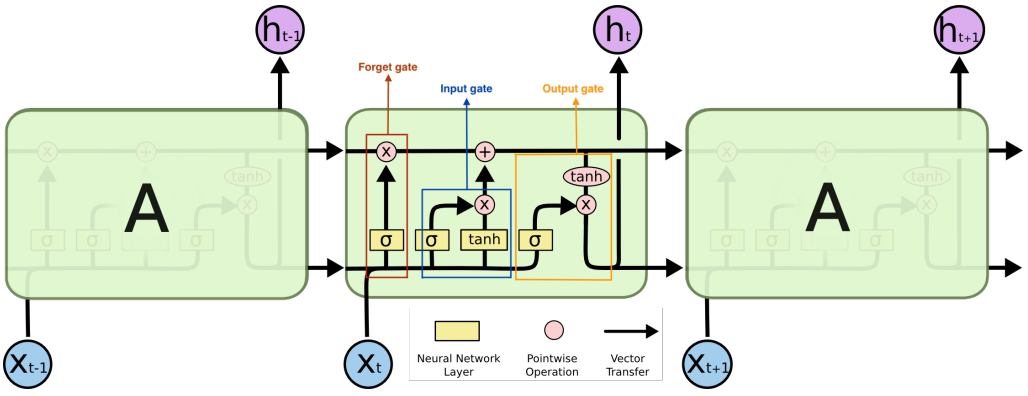

Figura 2.4. Schema di una *LSTM* [6].

Il *forget gate* agisce sui vettori d'input $\vec{x}_t$ e $\vec{h}_{t-1}$ calcolando

$$\boldsymbol{f_t} = \sigma(\vec{x}_t U^f + \vec{h}_{t-1} W^f) \tag{2.5}$$

dove U è la matrice dei pesi che connette gli *input* al *layer* nascosto e W la connessione ricorrente tra il precedente *layer* di *input* e quello attuale. La sigmoide, definita

nella formula 2.3, permette che il peso $\boldsymbol{f_t} \in [0, 1]$. L'apice identifica il *gate* a cui ci si riferisce: $f \longrightarrow$ *forget gate*; $i \longrightarrow$ *input gate*; $o \longrightarrow$ *output gate*.

L'*input gate* esegue due operazioni:

$$\tilde{\boldsymbol{C}}_{\boldsymbol{t}} = tanh(\vec{x}_t U^c + \vec{h}_{t-1} W^c); \quad \boldsymbol{i_t} = \sigma(\vec{x}_t U^i + \vec{h}_{t-1} W^i) \tag{2.6}$$

in cui, con la sigmoide, si "decide" quale nuova informazione allocare nella cella; poi, per mezzo della tangente iperbolica, viene a crearsi un nuovo possibile vettore di candidati, $\tilde{\boldsymbol{C}}_{\boldsymbol{t}}$, da poter aggiungere alla *state unit*.
Questi due vettori vengono quindi combinati per aggiornare la cella di stato da $\boldsymbol{C_{t-1}}$ a $\boldsymbol{C_t}$:

$$\boldsymbol{C_t} = \sigma(\boldsymbol{f_t} \cdot \boldsymbol{C_{t-1}} + \boldsymbol{i_t} \cdot \tilde{\boldsymbol{C}}_{\boldsymbol{t}}) \tag{2.7}$$

Infine, l'*output gate* gestisce l'informazione che viene passata alle celle successive, calcolando:

$$\vec{h}_t = tanh(\boldsymbol{C_t}) \cdot \boldsymbol{o_t}; \quad \boldsymbol{o_t} = \sigma(\vec{x}_t U^o + \vec{h}_{t-1} W^o) \tag{2.8}$$

Capitolo 3

Jet flavor tagging con deep neural networks

Il progetto ha come obiettivo quello di classificare *jet* adronici in funzione del sapore di quark che li ha prodotti. Sono state valutate le prestazioni di due diverse architetture di reti neurali nel compito di classificazione binaria: una *Feed-Forward Neural Network* ed una *Long Short-Term Memory Neural Network*. Si sono poi confrontati i risultati ottenuti con quelli presentati in [2].

3.1 Descrizione del problema e strategia di risoluzione

Nell'ambito del problema generale dell'identificazione delle particelle prodotte in eventi collisionali a *LHC*, il *jet flavor tagging* ha un ruolo cruciale. Soprattutto, è importante riconoscere segnali di quark pesanti in mezzo ad un ampio fondo di segnali di quark *charm* e leggeri.
Nel nostro caso, vogliamo distinguere, analizzando il *jet* adronico in esame, i quark *bottom* (*label 0*) dai quark *charm* o *light* (*label 1*). Rientrano in questa categoria i quark *up*, *down* e *strange*; i quark *top*, com'è noto, non adronizzano. Dunque il problema si riduce ad una **classificazione binaria**.

Come già accennato, si implementano due modelli di reti neurali profonde, la *FFNN* e la *LSTM*, ampliamente discusse nella sezione 2.2.1. Entrambe prendono in input un vettore a 16 componenti (*features*) e producono lo stesso numero di *output* (2, il numero di classi).

La ***FFNN*** è costituita da 11 *layer* densi, di cui 9 nascosti (ciascuno con numero di neuroni compreso tra 64 e 4096) con attivazione *Rectified Linear Unit* (*ReLU*, in formula 2.3). L'ultimo *hidden layer* è connesso al *layer* di uscita tramite una *LogSoftmax*:

$$LogSoftmax(x_i) = ln\Big[Softmax(x_i)\Big] = ln\left[\frac{exp(x_i)}{\sum_j exp(x_j)}\right] \tag{3.1}$$

al fine di non compromettere il modello in termini di ottimizzazione e stabilità numerica, cosa che potrebbe accadere utilizzando semplicemente la *Softmax*[1] [7].

[1] La *Softmax* è una generalizzazione di una funzione logistica che comprime un vettore k-dimensionale di valori reali arbitrari in un vettore k-dimensionale di valori compresi tra 0 e 1, la cui somma è 1.

Nella figura A.3 dell'appendice è riportato un sommario di quanto appena descritto.

La rete ricorrente impiegata si compone di un'unità ***LSTM*** con `hidden_dim` ottimizzabile e di un susseguente *MLP* da quattro strati, tra i quali sono stati interposti dei *layer* di *dropout*[2] con stessa probabilità di diluizione `pdrop` (cfr figura A.1). Anche qui, la funzione di attivazione utilizzata è la *ReLU*.

3.2 Analisi del dataset

Il *dataset*[3] a disposizione è frutto di simulazioni i cui dettagli si trovano nell'articolo di riferimento del progetto [2].

Per il *training* delle reti neurali (sviluppate principalmente utilizzando il *framework PyTorch*, in *Google Colaboratory*) sono state impiegate in tutto 16 *features*: l'impulso trasverso p_T, la pseudorapidità η (1.3) ed altre 14 *features* di alto livello suddivisibili in:

- **Variabili di traccia**: la significanza trasversale (S_{d_0}) e longitudinale (S_{z_0}) di seconda e terza traccia, la probabilità di avere un *light jet*, la larghezza del *jet* in η e in ϕ ed il numero di tracce con $S_{d_0} > 1.8\ \sigma_{d_0}$.
- **Variabili di vertice**: il numero di vertici secondari ed il numero di tracce associate ad essi, la *significance* combinata del vertice, la distanza angolare ΔR tra *jet* e vertice, la massa della catena di decadimento (somma delle masse invarianti di tutti i vertici ricostruiti) e la frazione di energia associata ai vertici secondari.

Mostriamo la distribuzione di tutte e 16 le *features* e l'esposizione delle matrice di correlazione tra esse in figura A.4. Dalla figura emerge subito la difficoltà del problema: le *features* sembrano avere, prese singolarmente, scarso potere di separazione, come si può valutare dalla forte sovrapposizione tra gli istogrammi di gran parte di esse; inoltre dalle matrici di correlazione possiamo evincere come le variabili risultino scarsamente correlate tra di loro, a sola eccezione delle variabili di alto livello relative ai vertici del jet.

Il *dataset* è stato fornito in formato `json`, leggibile con libreria `pandas`. Al fine di "alleggerire" il set di dati, esso è stato preliminarmente convertito da `pandas dataframe` a `numpy array`. Quest'ultimo è stato poi salvato in un *file* `.npz` per facilitarne la presa in *input*.

Inoltre, alcuni eventi del campione, per alcune *features*, esibivano valori irrappresentabili, come $\pm$`NaN` e $\pm\infty$; per altre, come ad esempio `n_secondary_vertices`, valore -1. Durante la fase di conversione del formato del *dataset*, per poter facilitare le *DNN* a riconoscere ciò, si è uniformata la scelta di porre tali valori porprio a -1.

3.3 Training delle reti

Per entrambi i modelli è stata impiegata un'ottimizzazione *Stochastic Gradient Descent* con *momentum* 0.9 (metodo che permette di accelerare la convergenza) e *learning rate* decrescente da $2{\cdot}10^{-3}$ fino a $1{\cdot}10^{-4}$ per la *FFNN* e fino a $2{\cdot}10^{-5}$ per la *LSTM*. Ciò favorisce la stabilizzazione della rete nelle fasi finali dell'allenamento.

[2] Il *dropout* è una tecnica di regolarizzazione per ridurre l'*overfitting* nelle *ANN*, in cui vengono selezionati randomicamente alcuni neuroni da ignorare durante il *training*.

[3] *Dataset* scaricabile al link: http://mlphysics.ics.uci.edu/data/hb_jet_flavor_2016/

Tale decrescita è stata ottenuta tramite l'utilizzo di uno `scheduler` [8].
Poiché il problema è di classificazione binaria, si sceglie la *Cross Entropy* (definita in formula 2.4) come funzione di *loss* e come metrica la *Area Under the Curve of the Receiver Operating Characteristic* (definita con l'acronimo *AUROC*, o più semplicemente *AUC*).
Inoltre, poiché le reti sono risultate dispendiose per il tempo impiegato per singola epoca, sono stati aggiunti degli appositi comandi per salvare l'ultimo ed il miglior modello dopo ogni epoca di *training* su *Google Drive* (`last_model` e `best_model`). Analogamente, si sono salvati sul *Drive* anche i valori delle *loss* e delle metriche (per *train* e *validation*), per poter poi graficare i loro andamenti ed analizzare i risultati ottenuti.

Ottimizzazione degli iperparametri

Prima di iniziare la vera e propria fase di *training*, si sono andati a studiare ed ottimizzare gli iperparametri al fine di massimizzare le prestazioni della rete. Fissando il `batch_size`, il numero di *layer* e la costanza della probabilità di *droput*, sono state allenate le reti con una piccola porzione del dataset (solo 25000) per 10 epoche per ogni combinazione. Al termine del processo di ottimizzazione, è stato scelto l'insieme di iperparametri che ha permesso alle reti di avere la metrica di validazione più alta. Dunque si scelgono:

Feed-Forward	
# *neuroni layer* 1	4096
# *neuroni layer* 2	2048
# *neuroni layer* 3	1024
# *neuroni layer* 4	512
# *neuroni layer* 5	64
`pdrop`	0.0
Best Vali Loss	0.187
Best Vali Accuracy	0.907

LSTM	
# *neuroni layer* 1	256
# *neuroni layer* 2	128
# *neuroni layer* 3	64
`hidden_dim`	14
`pdrop`	0.0
Best Vali Loss	0.196
Best Vali Accuracy	0.910

Tabella 3.1. Iperparametri ottimizzati per *FF*, a sinistra, per *LSTM* a destra.

LSTM training

L'allenamento della rete ricorsiva è stato effettuato utilizzando tutti gli 11 milioni di vettori di *features* presenti nel *dataset*. Vista la gran mole di dati utilizzati, il ciclo di training è risultato essere una procedura piuttosto lenta: circa 7 ore per 55 epoche ($\sim$ 470 secondi ad epoca).

Google Colab - nella versione non a pagamento - impone delle limitazioni sul *runtime* (e non solo). Difatti, dopo 45 epoche, ossia dopo poco più di 5 ore e mezza, *Colab* interrompe il *runtime*, cancellando quindi i progressi fatti fino a quel momento. Dunque, per aggirare questo problema, sono stati inseriti appositi comandi per poter salvare il modello allenato e tutti i valori di *loss* e metrica (in appositi *file* `.txt`) su *Google Drive*. Una volta che si interrompe il *runtime* così, basta ricaricare il modello finora allenato, rieseguire tutte le procedure preliminari (tipo ricaricare le librerie, il *dataset* ecc.) e continuare con una seconda *tranche* di *training*, che riparte dall'epoca 45 e "gira" - nel nostro caso - per altre 10 epoche (circa 1 ora e mezza).

Si nota, infine, che, a causa del cambio di *learning rate* che si verifica nella seconda *tranche* di allenamento, si osserva un "salto" negli andamenti di *loss* e *AUC*, che vengono riportati di seguito.

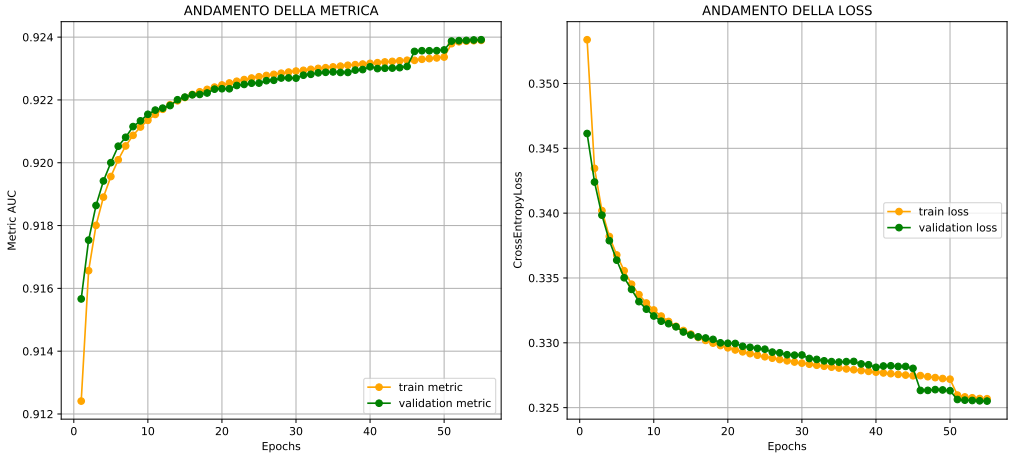

Figura 3.1. Andamento della metrica *AUC* a sinistra, andamento della *loss* a destra, per la rete *LSTM*.

Si nota - inoltre - che si ha un'ottima convergenza tra le due *loss*. La rete non sembra dunque soffrire di problemi di *underfitting* o *overfitting*.

FFNN training

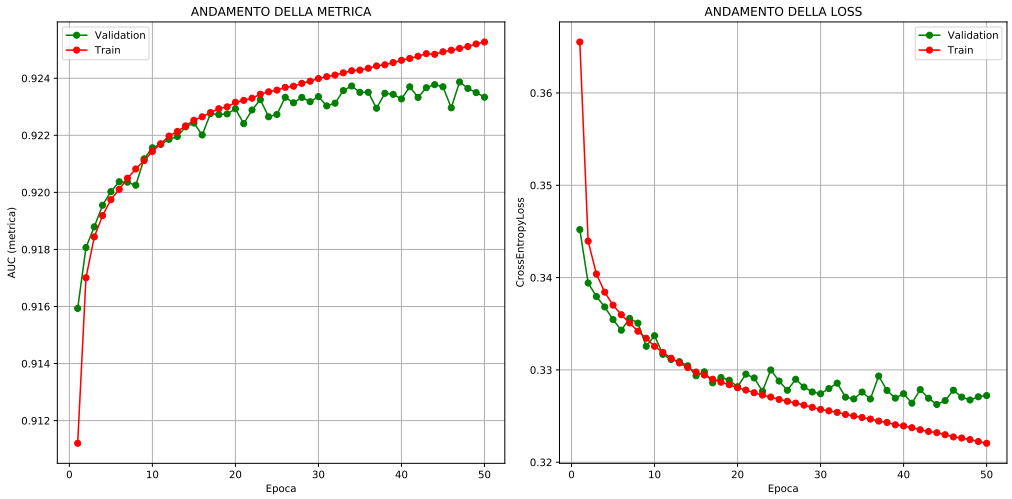

Figura 3.2. Andamento della metrica *AUC* a sinistra, andamento della *loss* a destra, per la rete *FFNN*.

La rete ottimizzata, che presenta circa 33 milioni di parametri (cfr. figura A.3), risulta essere molto pesante. Pertanto si è allenato il modello solamente su 4 milioni di dati, utilizzando dei batch di dimensione 300 per 50 epoche totali. Il tempo impiegato per il *training* è stato di circa 3 ore e mezza ($\sim$ 245 secondi ad epoca).

Anche qui, si nota che si ha una convergenza abbastanza buona tra le due *loss*. La rete non sembra dunque soffrire di *underfitting* o *overfitting*.

3.4 Analisi dei risultati

Si riporta nella figura che segue la *ROC* calcolata sul *test set*, per entrambe le reti.

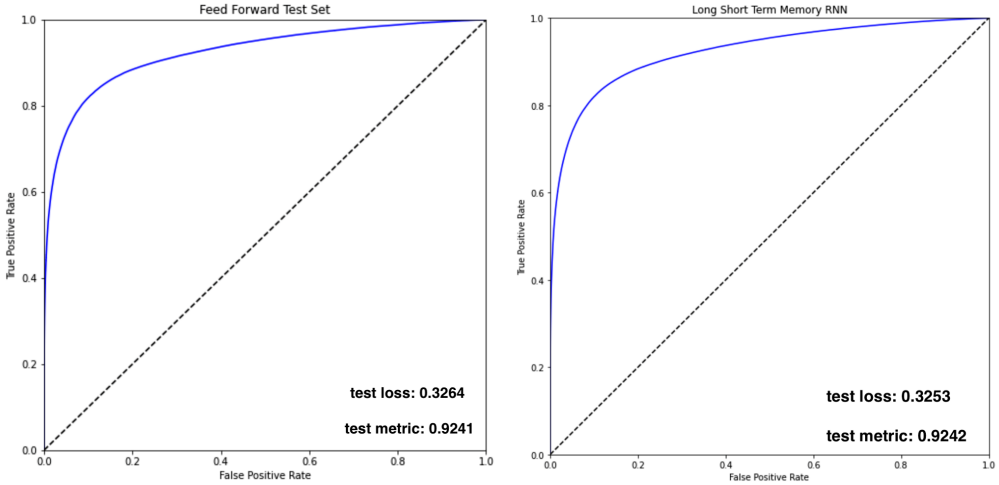

Figura 3.3. Curva *ROC* calcolata sul *test set*, a sinistra per la *FFNN*, a destra per la *LSTM*.

Si riporta, inoltre, di seguito, una tabella riassuntiva delle prestazioni per le due reti , per un confronto con i risultati dell'articolo di riferimento [2].

	***AUC* ottenuta**	***AUC* articolo**
FFNN	0.924	0.924
LSTM	0.924	0.925

Tabella 3.2. Tabella riassuntiva delle prestazioni: *AUC* sul *test set*.

Si è dunque eguagliata la prestazione di riferimento [2], per quanto riguarda la rete *Feed-Forward*. La *LSTM* è invece risultata meno efficace, ma di poco: 0.001 di differenza, un valore consistente con le fluttuazioni statistiche legate alla inizializzazione random dei pesi della rete.

Matrici di confusione

È necessario, infine, fare un'ultima importante osservazione.
Il *dataset* presenta una distribuzione non omogenea tra *label 0* e *label 1*, come si può notare dall'istogramma riportato in figura A.2.
Su un campione sbilanciato, c'è il rischio che l'*accuracy* media su di esso, possa non fornire l'informazione sufficiente per capire le prestazioni del problema. Avendo una classe predominante sull'altra, la rete potrebbe dare un'*accuracy* media molto buona semplicemente rispondendo bene agli esempi di quella classe e con *accuracy* zero su quelli dell'altra.

Vengono dunque calcolate, e riportate in tabella 3.3, le **matrici di confusione**, per avere una rappresentazione dell'accuratezza di classificazione statistica per ciascuna classe.

Feed-Forward

True Positive	False Negative
0.9219	0.0781
False Positive	**True Negative**
0.2087	0.7913

LSTM

True Positive	False Negative
0.9210	0.0790
False Positive	**True Negative**
0.2062	0.7938

Tabella 3.3. Matrici di confusione: a sinistra per la *FF*, a destra per *LSTM*.

Dai risultati in tabella, si nota che la differenza non è trascurabile: 92% per la *label 0*, contro il 79% della *label 1* di corretta classificazione. In ambo i casi, questa è una differenza non enorme, ma comunque visibile e presumibilmente dovuta proprio allo sbilanciamento tra gli eventi nelle due classi nel campione di *training*. Sarebbe possibile - in principio - equalizzare maggiormente le prestazioni in *accuracy* per le due classi, bilanciando il dataset di *training* (via *undersampling* o *oversampling*), oppure usando una funzione di *loss* pesata con le frazioni relative degli eventi nelle due classi.

Conclusioni

Si può concludere che l'obiettivo di eguagliare le prestazioni presentate nell'articolo di riferimento [2] è stato praticamente raggiunto, nonostante una differenza di circa il 10% di corretta identificazione delle *label* (come denotato dalle matrici di correlazione in tab. 3.3). Tale differenza è tuttavia migliorabile, ad esempio con l'introduzione di una *weighted loss*, come accennato nella sez. 3.4.
I due modelli addestrati sono riusciti a raggiungere, entro le incertezze, i valori di *AUC* riportati nell'articolo (cfr. tabella riassuntiva 3.2). Probabilmente, aumentando il numero delle epoche di addestramento si sarebbero potute eguagliare, se non addirittura migliorare, le *performance* dell'articolo di riferimento.

Uno sguardo al futuro: il terzo *run* di LHC

A dieci anni dalla scoperta del bosone di Higgs e dopo oltre tre anni di *shutdown*, è iniziato l'atteso *run 3* di *LHC*. Il 5 luglio scorso, il *Large Hadron Collider* ha realizzato le prime collisioni nel centro di massa ad una energia record di $\sqrt{s} \simeq 13.6\ TeV$. Come spiega Rende Steerenberg - responsabile delle operazioni degli acceleratori al CERN - queste collisioni più energiche dovrebbero aumentare le probabilità di produrre particelle nelle regioni ad alta energia in cui potrebbe trovarsi la "nuova fisica" [4].

Inoltre, i fasci dell'acceleratore genereranno pacchetti (*bunch*) di particelle più compatti, aumentando la probabilità di collisioni. Così facendo, *LHC* riuscirà a mantenere il suo picco di luminosità più a lungo, permettendo di registrare una quantità di dati pari a quella dei primi due *run* messi insieme. Per ottenere ciò, è stato necessario un aggiornamento dei rivelatori *ATLAS* e *CMS*, per fare in modo che proprietà come energia, quantità di moto o pseudorapidità, vengano misurate in maniera più efficace e precisa. L'introduzione delle *GPU* (*Graphics Processing Unit*) è un esempio di tali aggiornamenti effettuati, nell'ambito dei sistemi di *trigger*. Le *GPU* sono vantaggiose, poiché in grado di ricostruire la traiettoria delle particelle più rapidamente dei processori convenzionali [4].

Con questi aggiornamenti, si avrà una maggiore precisione nelle misurazioni relative alle particelle note, come il bosone di Higgs. Si potrà verificare, inoltre, se una serie di risultati anomali ottenuti con i dati del *run 2*, siano delle vere anomalie oppure il risultato di fluttuazioni statistiche. Tra questi, si può citare la massa del **bosone W**, che è risultata, dalla recente misura eseguita dall'esperimento *CDF-II* al Tevatron del Fermilab, significativamente più alta di quanto previsto dal Modello Standard.
Inoltre, si studierà con particolare attenzione il decadimento dei **mesoni B**. Le analisi di *LHCb*, difatti, suggeriscono che essi tendono a produrre elettroni più spesso (circa il 15% in più) di quanto non producano i muoni, nonostante il Modello

Standard preveda che la natura non debba preferire un processo rispetto all'altro.

Molti esperimenti di *LHC* usano già il *machine learning* per distinguere dai processi di fondo particolari collisioni ricercate. Per fare ciò si utilizzano apprendimenti di tipo "supervisionato": all'algoritmo vengono rese note a priori le classi da cercare. Tuttavia, per il terzo *run* di *LHC*, si stanno sviluppando algoritmi cosiddetti di *anomaly detection*, che presentano un approccio differente rispettto a quello dei supervisionati.

L'obiettivo è quello di costruire dei modelli non supervisionati (come ad esempio un *AutoEncoder*), che vengono allenati a ricostruire correttamente gli eventi "normali", cioè del Modello Standard. Tale scopo viene perseguito applicando una forte compressione dei dati tale da poter permettere di ricostruire gli eventi sulla base di un *set* minimo di informazioni.

Quando un evento diverso da quelli utilizzati per allenare il modello viene analizzato dal modello stesso, ci si aspetta che quanto più l'evento è differente dagli eventi normali tanto più la ricostruzione dell'*AutoEncoder* risulta sbagliata. Si può quindi costruire una "misura di anomalia" come distanza tra evento di *input* ed evento ricostruito dall'*AE* (come ad esempio una *MAE*[4]). Per gli eventi normali usati per allenare l'*AE*, la *MAE* sarà piccola, mentre per gli eventi anomali si avrà una *MAE* maggiore.
In questo modo non si sono mai usati eventi anomali per allenare l'algoritmo e quindi esso prescinde dal modello di nuova fisica caratterizzato dai medesimi eventi anomali.

Il *machine learning* potrebbe risultare quindi uno strumento chiave per la scoperta di nuova fisica. È auspicabile dunque che questa branca dell'intelligenza artificiale sia sempre più presente in tutte le attività di ricerca scientifica in cui essa è utilizzabile.

[4] *Mean Absolute Error*: $MAE = \frac{1}{N}\sum_i |x_i^{obs} - x_i^{pred}|$

Appendice A

Appendice immagini

#DEFINIZIONE DELLA CLASSE PYTHON

```python
class myLSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim=14,
                 l1=256, l2=128, l3=64, pdrop=0.0):
        super().__init__()
        self.input_dim = input_dim  # num. di features
        self.hidden_dim = hidden_dim
        self.num_layers = 1 #uso una LSTM con un solo layer

        self.lstm = nn.LSTM(
            input_size=input_dim,
            hidden_size=hidden_dim,
            batch_first=True, # necessario, input shape: batch, seq_len, feat
            num_layers=self.num_layers
        )

        self.layer1 = nn.Linear(in_features=self.hidden_dim, out_features=l1)
        nn.init.xavier_uniform(self.layer1.weight)
        self.drop1 = nn.Dropout(p=pdrop)
        self.layer2 = nn.Linear(l1, l2)
        nn.init.xavier_uniform(self.layer2.weight)
        self.drop2 = nn.Dropout(p=pdrop)
        self.layer3 = nn.Linear(l2, l3)
        nn.init.xavier_uniform(self.layer3.weight)
        self.drop3 = nn.Dropout(p=pdrop)
        self.layer4 = nn.Linear(l3, 2)
        nn.init.uniform_(self.layer4.weight, a=-0.05, b=0.05)
```

```python
def forward(self, x):
    batch_size = x.shape[0]
    # valore iniziale per hidden state e cell state (inzializzo a 0)

    h0 = torch.zeros(self.num_layers, batch_size,
                     self.hidden_dim).requires_grad_().to(device)
    c0 = torch.zeros(self.num_layers, batch_size,
                     self.hidden_dim).requires_grad_().to(device)


    on, (hn, cn) = self.lstm(x, (h0, c0))

    # la prima dimensione di h è il numero di layers, pari a 1

    x = self.layer1(hn[0])
    x = F.relu(x)
    x = self.drop1(x)
    x = self.layer2(x)
    x = F.relu(x)
    x = self.drop2(x)
    x = self.layer3(x)
    x = F.relu(x)
    x = self.layer4(x)
    out = F.log_softmax(x, dim=1)
    return out
```

Figura A.1. ***LSTM***: definizione della *classe Python*.

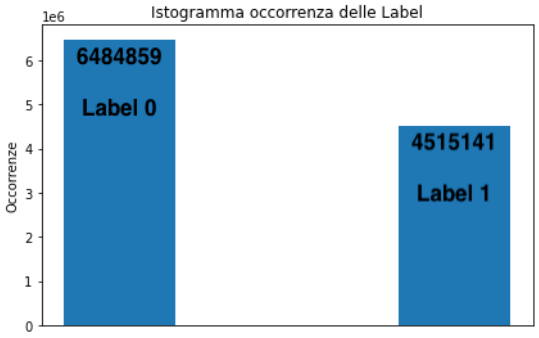

Figura A.2. Istogramma che mostra l'effettivo sbilanciamento tra le due *label*.

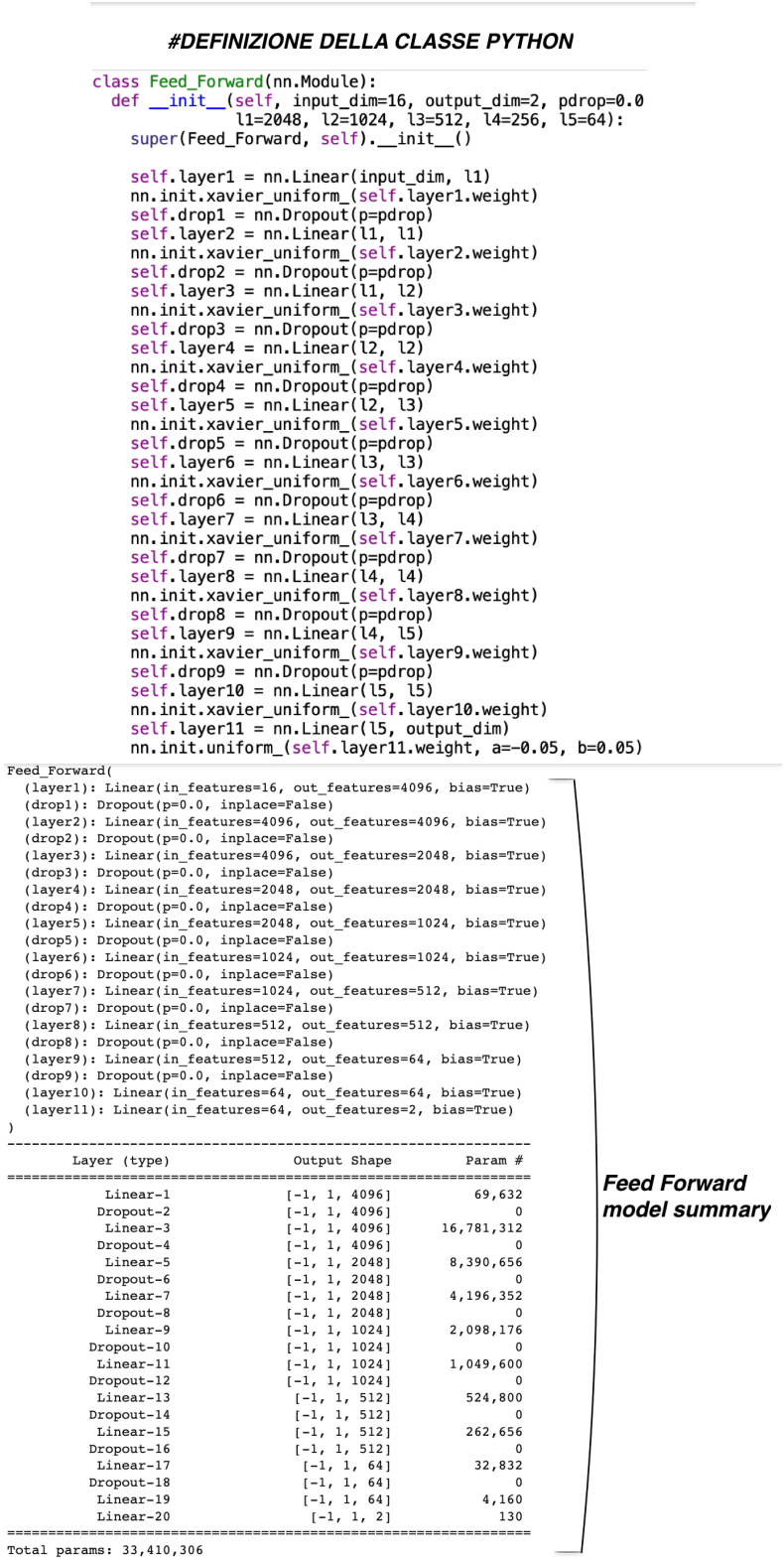

Figura A.3. ***FFNN***: definizione della *classe* in alto, *printout* del modello in basso.

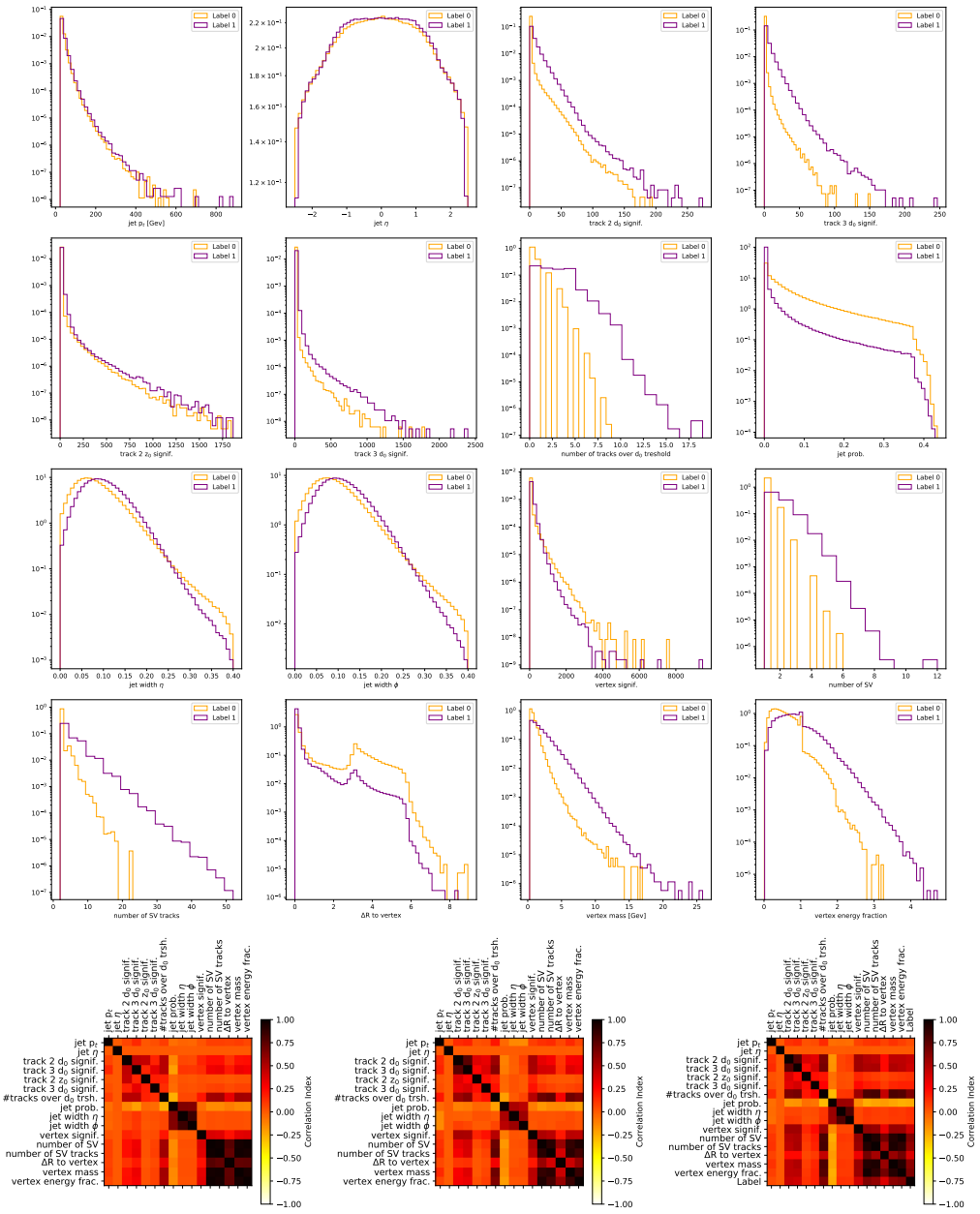

Figura A.4. Distribuzione delle 16 *features* in alto. Esposizione delle matrici di correlazione tra esse (*heatmaps*) in basso, da sinistra a destra: *label 0*, *label 1* e intero *dataset*.

Bibliografia

[1] Baldi Pierre, Bauer Kevin, Eng Clara, Sadowski Peter e Whiteson Daniel, *Jet substructure classification in high-energy physics with deep neural networks*, 27 Maggio 2016. https://arxiv.org/abs/1607.08633

[2] Baldi Pierre, Collado Julian, Guest Daniel, Hsu Shih-Chieh, Urban Gregor e Whiteson Daniel, *Jet Flavor Classification in High-Energy Physics with Deep Neural Networks*, 9 Settembre 2016. https://arxiv.org/pdf/1607.08633.pdf

[3] Couchman Jonathan, *Recombination schemes*, UCL Department of Physics and Astronomy - High Energy Physics, 2 Ottobre 2002. https://www.hep.ucl.ac.uk/~jpc/all/ktjet/ktjet/node10.html

[4] Gibney Elizabeth, *LHC: Third Time Lucky?*, «Nature», Vol 605, 26 maggio 2022. https://media.nature.com/original/magazine-assets/d41586-022-01388-6/d41586-022-01388-6.pdf

[5] Malito Davide, *Study of b-tagging effciencies with the ATLAS detector*, tesi di laurea in fisica 2019, Università Della Calabria.

[6] Olah Christopher, *Understanding LSTM Networks*, «colah's blog», 27 Agosto 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[7] *PyTorch 1.12 Documentation - LogSoftmax.* https://pytorch.org/docs/stable/generated/torch.nn.LogSoftmax.html

[8] *PyTorch 1.12 Documentation - torch.optim - How to adjust learning rate.* https://pytorch.org/docs/stable/optim.html

[9] Salam Gavin Philip, *Towards jetography*, Eur. Phys. J. C 67, 637–686 (2010). https://doi.org/10.1140/epjc/s10052-010-1314-6

[10] The ATLAS Collaboration *et al 2008 JINST* 3 S08003, 14 Agosto 2008. https://iopscience.iop.org/article/10.1088/1748-0221/3/08/S08003

[11] The ATLAS Collaboration, *Optimisation and performance studies of the ATLAS b-tagging algorithms for the 2017-18 LHC run*, 5 Luglio 2017. http://cds.cern.ch/record/2273281/files/ATL-PHYS-PUB-2017-013.pdf?version=1

[12] Wikipedia, *Jet (particle physics).* https://en.wikipedia.org/wiki/Jet_(particle_physics)