POLITECNICO DI TORINO

Corso di laurea magistrale in

Ingegneria Gestionale

Tesi di Laurea Magistrale

Uso delle Reti Neurali per la previsione di serie storiche finanziarie

Relatore
Franco Varetto

Candidato
Marco Tortora

Anno Accademico 2019/2020

ABSTRACT

La previsione del movimento dei prezzi azionari è da decenni oggetto di numerosi lavori di ricerca. Da sempre vi sono due fazioni contrapposte: i sostenitori della teoria dei mercati efficienti che asseriscono che i prezzi non possono essere predetti, e un'altra parte di ricercatori che, basandosi sui fondamenti dell'analisi tecnica, secondo cui il contenuto informativo dei mercati è già insito nelle fluttuazioni, pensano che i prezzi futuri possono essere predetti con un certo grado di accuratezza utilizzando modelli appropriati.

Lo scopo di questo lavoro è l'implementazione di un framework per la previsione dei prezzi azionari basandosi su modelli di Deep Learning, una branca dell'Intelligenza Artificiale che si sta distinguendo in anni recenti grazie all'efficacia di strutture chiamate Reti Neurali Artificiali Profonde, che consentono di captare caratteristiche di funzioni altamente non lineari.

Pertanto sarà approfondita la teoria su cui si basano queste strutture, sarà spiegata la matematica che c'è dietro e come possono adattarsi alle caratteristiche del problema che si vuole affrontare.

Si passerà quindi alla raccolta e alla preparazione dei dati necessari, e all'implementazione della Rete in codice Python, sfruttando una libreria open source chiamata Keras, la più gettonata in questo momento per utilizzare i modelli Deep Learning.

La rete sarà testata su due indici azionari europei, l' Euro Stoxx 50 e il FTSE Mib, su un orizzonte di previsione di 1 anno e mezzo partendo da una base dati di 4 anni per l'addestramento del modello. Infine saranno valutate le performance del modello e, sulla base di queste, saranno tratte le conclusioni sull'effettiva efficacia del modello, sui suoi limiti e sui possibili sviluppi futuri.

Ringraziamenti

alla mia famiglia,
che sempre e incondizionatamente mi ha sostenuto per avermi dato la possibilità di raggiungere questo traguardo

a Rosy,
perché è ancora più bello poter condividere questo successo con la persona che si ama

a tutte le persone conosciute in questi anni,
perché hanno reso questo percorso divertente e indimenticabile. In particolare dedico un affettuoso saluto a Simone, Gianmarco, Vate e Marco con cui ho condiviso, tra scleri e risate, questa splendida avventura

ai miei amici,
su tutti Gennaro, Benedetto, Luca e Bruno, perché un amico è la cosa più preziosa che si possa avere e la migliore che si possa essere

al professor Varetto,
che mi ha trasmesso passione per la materia non solo con competenza e autorevolezza, ma anche con simpatia e contagiosa allegria

SOMMARIO

Capitolo I

DEFINIZIONE DELL' AMBITO DI INDAGINE.....1

Capitolo II

RETI NEURALI ARTIFICIALI.....16

Capitolo I

DEFINIZIONE DELL' AMBITO DI INDAGINE

In questo capitolo introduttivo si discuterà il problema che ci si appresta ad affrontare, le feature che lo caratterizzano, qual è lo stato dell'arte attuale e in che modo si cercherà di migliorare quanto è stato fatto fino a questo momento. Si concluderà il capitolo con una rassegna dei metodi classici di previsione, che possono essere utilizzati come benchmark per capire l'effettivo miglioramento apportato dai metodi basati sul Deep Learning.

1.1 AREE DI APPLICAZIONE FINANZIARIA

L'evoluzione dei calcolatori ha consentito lo sviluppo di numerosi argomenti di ricerca in campo finanziario. Nelle ultime decadi, infatti, sono stati pubblicati un gran numero di lavori basati su modelli Deep Learning riguardanti argomenti appartenenti a diversi campi. Infatti le aree di applicazione finanziaria sono molto diversificate. In questa sezione si passerà in rassegna le aree di applicazione[1] per capire meglio in quale campo ci si andrà a posizionare in questo lavoro di ricerca.

1.1.1 Algoritmi di trading

Costruire un algoritmo di trading significa in parole più semplici definire un decision-making sull'acquisto o vendita di un certo asset al fine di trarre profitto dall'aumento o dalla diminuzione del prezzo. Un algoritmo di trading può basarsi su regole semplici, come l'osservazione del trend passato o di alcuni indicatori specifici, oppure su modelli matematici altamente complessi e non lineari, che i modelli di Deep Learning tentano in qualche modo di approssimare. Gli algo-trading (abbreviazione per indicare gli algoritmi di trading) possono basarsi su modelli di previsione; in pratica si parte da modelli il cui scopo è quello di prevedere il prezzo futuro o quantomeno la sua fluttuazione, e da lì si trae la decisione sull'acquisto o vendita del titolo oggetto della previsione.

1.1.2 Risk Assessment

Il Risk Assessment è un macro-campo che a sua volta può essere visto da punti di vista molto diversi gli uni dagli altri. Si può dire che il Risk Assessment, in finanza, è una branca che identifica il rischio intrinseco di un asset, di una persona, di un'azienda, ecc. Ovviamente il rischio ha sfaccettature diverse a seconda che lo si pensi dal punto di vista di un'impresa, di una banca, di un investitore o di una persona

comune. Ma qualunque sia il lato da cui lo si guardi, il rischio è un aspetto cruciale per una moltitudine di soggetti, e la sua corretta identificazione e misura può risultare decisiva nel raggiungimento dell'obiettivo prefissato. Come si può banalmente supporre, il settore bancario è l'area dove si concentrano i maggiori lavori di ricerca legati a questo tema. Quest'interesse deriva dal fatto che il rischio di insolvenza dei debitori ha direttamente a che fare con il business core delle banche.

1.1.3 Rilevamento di frodi

Un'altra area che abbraccia diversi aspetti è il tema delle frodi finanziarie. Anche le frodi possono essere di diversa natura: frodi legate alle carte di credito, frodi assicurative, evasione fiscale ecc. Quindi anche i governi e le autorità, oltre che le aziende interessate, che anche in questo caso si rilevano essere soprattutto banche e assicurazioni, sono interessate a trovare una soluzione definitiva e permanente a riguardo. Pertanto numerosi studiosi hanno costruito modelli in grado di rilevare con una certa probabilità movimenti finanziari fraudolenti.

1.1.4 Portfolio Management

Con Portfolio Management, o gestione del portafoglio, si intende il processo di allocazione di diversi asset al fine di raggiungere un obiettivo con un certo budget su un orizzonte temporale pre-determinato. La gestione del portafoglio è un problema di ottimizzazione, dunque sviluppato per cercare la migliore soluzione per raggiungere l'obiettivo di un periodo. L'obiettivo può essere di varia natura: si può perseguire ad esempio la massimizzazione del profitto, oppure si può agire per minimizzare il rischio, per esempio coprendosi contro una variazione di prezzi inattesa.

1.1.5 Pricing di asset e derivati

Un'area fondamentale della finanza è quella che si occupa di assegnare il prezzo agli asset, in particolare ai derivati. Infatti il prezzo di un derivato viene stabilito in maniera tale da evitare possibilità di arbitraggio sfruttandone la vendita oppure l'acquisto. Se questo scopo diventa abbastanza banale con i derivati più semplici, le cose si complicano all'aumentare della complessità del derivato. Ad esempio i prezzi di Forward e Future possono essere calcolati agevolmente, sfruttando delle formule lineari, ma già per quanto riguarda le opzioni le cose si complicano. Esistono dei metodi di calcolo buoni per le opzioni europee, ma per quelle americane, e in particolare quelle esotiche diventa molto difficile stabilire il prezzo di equilibrio essendo modellato da una funzione altamente non lineare. In questo senso tornano molto utili le Reti Neurali Profonde, essendo una classe di metodi la cui peculiarità

principale è proprio quella di riuscire a mappare delle funzioni caratterizzate da elevata complessità e non linearità.

1.1.6 Studi su Blockchain e Criptovalute

Negli ultimi anni le criptovalute hanno avuto risalto tra le notizie finanziarie per la loro elevatissima volatilità, che ha portato enormi profitti o enormi perdite ai trader che vi hanno puntato, così come la Blockchain, la tecnologia decentralizzata che consente di produrre e distribuire le criptovalute. Ovviamente Blockchain e criptovalute sono altamente correlate e in particolare gli studi sulla Blockchain, su cui aleggia un alone di mistero sulla nascita e sugli sviluppi, sono ancora in una fase embrionale.

Per quanto riguarda le criptovalute sono già stati effettuati numerosi studi sullo sviluppo di framework per la previsione dei prezzi futuri e delle fluttuazioni. Ancora tanto da fare c'è invece per quanto riguarda la Blockchain. Per esempio potrebbero essere sviluppati dei modelli basati sul Deep Learning per agevolare la tracciabilità delle transazioni.

1.1.7 Sentiment Analysis finanziaria

La Sentiment Analysis, o analisi del sentimento, è un sotto campo dello studio dei Processi del Linguaggio Naturale. Lo scopo è quello di analizzare dei segnali, che possono essere sia oggetti semplici quali testi, sia elaborazioni più complesse che coinvolgono le espressioni facciali, sfruttando tecniche di Computer Vision, al fine di trarne un'opinione.

È chiara la potenzialità che l'analisi dei sentimenti può avere in campo finanziario, dove le emozioni degli investitori giocano un ruolo chiave nelle loro decisioni di investimento. La Sentiment Analysis in campo finanziario sta acquisendo sempre più importanza, specie in appoggio ad altri campi quali la previsione dei prezzi e gli algoritmi di trading. La crescita dell'analisi dei sentimenti è stata resa possibile dallo sviluppo che hanno avuto i Social Network nell'ultimo decennio. Infatti nei Social vengono ogni giorno riversate centinaia di terabyte di informazioni che, opportunamente manipolate, possono aiutare a tracciare il profilo della persona, grazie alla disciplina dei Big Data, e nel caso degli investitori il profilo corrisponde proprio alle possibili decisioni future di acquisto o vendita di determinati asset.

1.1.8 Text Mining finanziario

Il Text Mining è una disciplina dell'intelligenza artificiale che elabora le informazioni contenute nei testi, casuali e non strutturate, trasformandole in dati in formato strutturato e normalizzato. Come è facile immaginare il Text Mining può essere strettamente collegato alla Sentiment Analysis, in quanto anche un testo può essere utilizzato come segnale d'ingresso al fine di trarre l'opinione che l'autore ha sull'argomento. In finanza vengono dati in pasto a modelli di Text Mining, oltre ai post dei social network, anche interi siti di notizie finanziarie come Bloomberg. Lo scopo è quello di riuscire a classificare le notizie sulla base dell'impatto che possono avere sui sentimenti degli investitori e, di conseguenza, come questi possono impattare sulle fluttuazioni dei prezzi azionari.

1.1.9 Altre applicazioni finanziarie

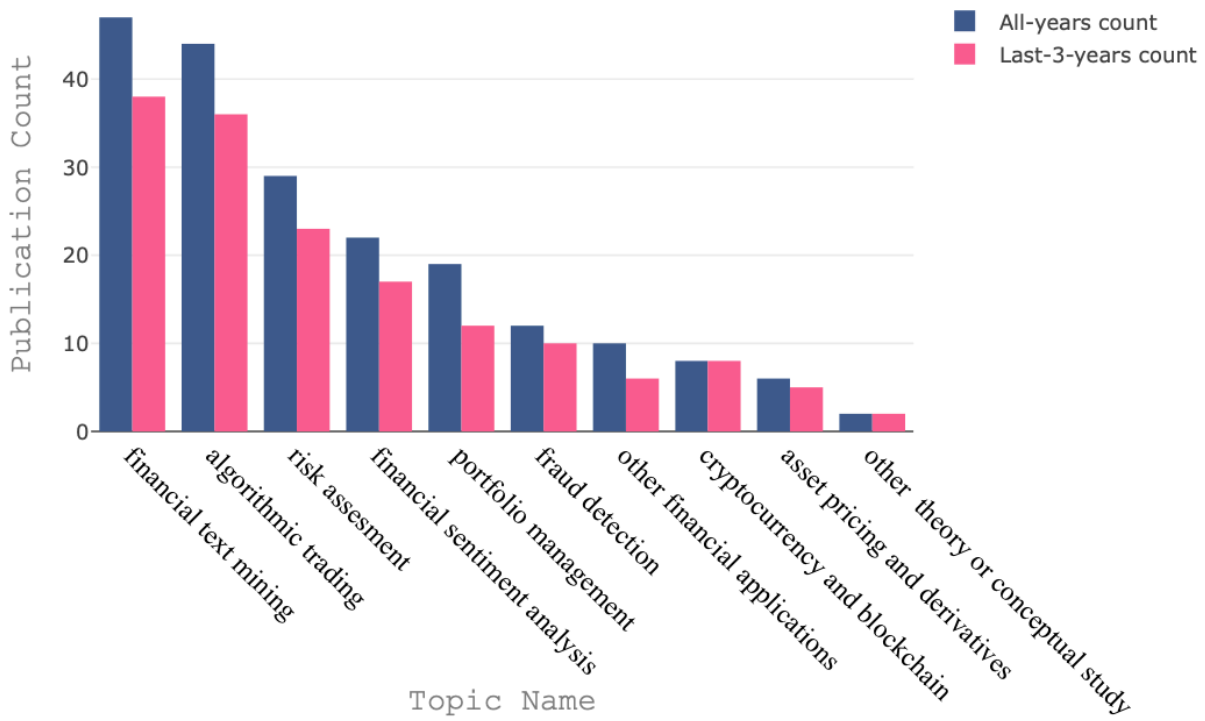
Vi sono infine altri ambiti di ricerca che non rientrano in nessuna delle precedenti aree di applicazione ma che comunque hanno a che fare con l'industria dei servizi finanziari. Alcuni esempi di questi studi hanno a che fare con l'instant payment e con lo studio dell'hardware e del software che implementano transazioni finanziarie, e riguardano in particolare sicurezza e velocità. Per quanto riguarda i metodi Deep Learning, questi lavori sono altamente specifici e molto inferiori in percentuale rispetto alle tesi prodotte per le altre macro aree, che sono di interesse maggiore per gli esperti del settore finanziario.

1.2 STATO DELL'ARTE

Prendendo spunto da dati tratti da "Deep Learning for Financial Applications: a Survey"[2], è stato fatto il punto sui progressi fatti dai modelli di Deep Learning in campo finanziario fino a questo momento storico (il paper è stato pubblicato in febbraio 2020).

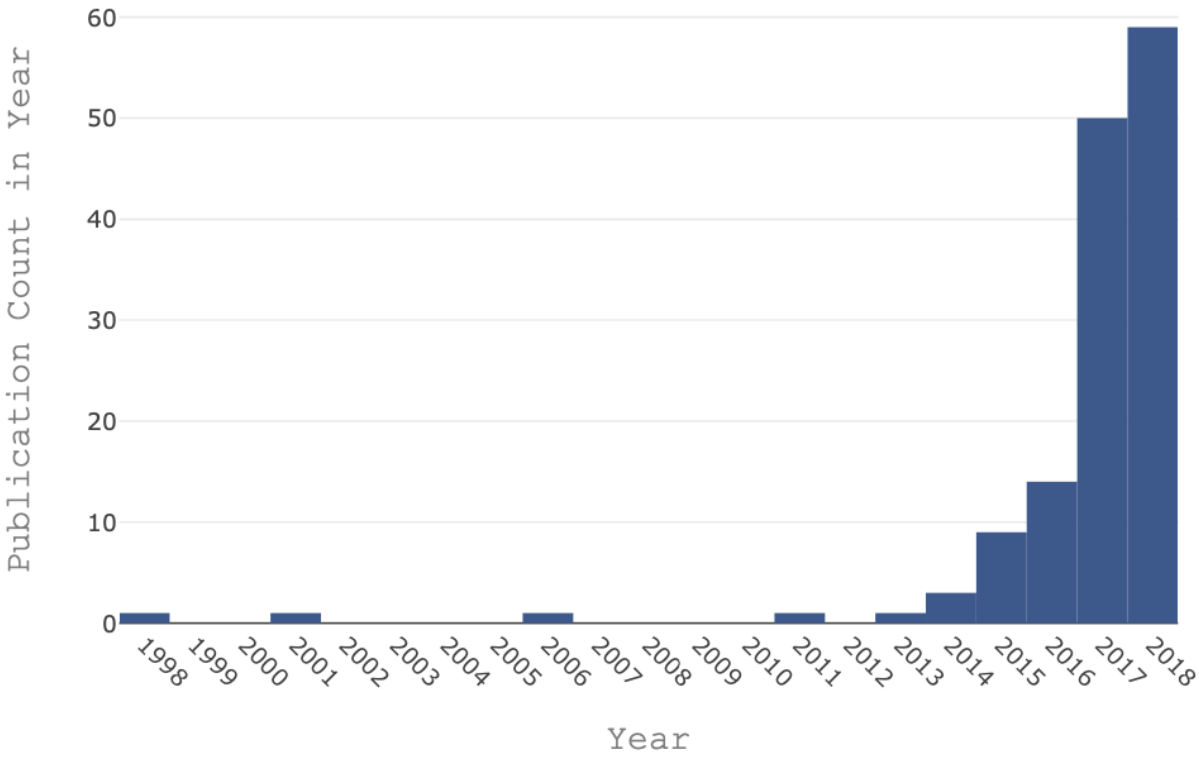
La classificazione non è fatta solo per area di applicazione, ma anche rispetto ad altri parametri quali il dataset di partenza utilizzato, la struttura di Deep Learning impiegata, il tipo di problema analizzato, e anche il linguaggio di programmazione utilizzato per sviluppare il modello e per effettuare i test.

Partendo dalla suddivisione in aree fatta nel paragrafo precedente è possibile tracciare un primo grafico che metta in luce gli argomenti più gettonati nei lavori di ricerca riguardanti l'uso del Deep Learning in campo finanziario.

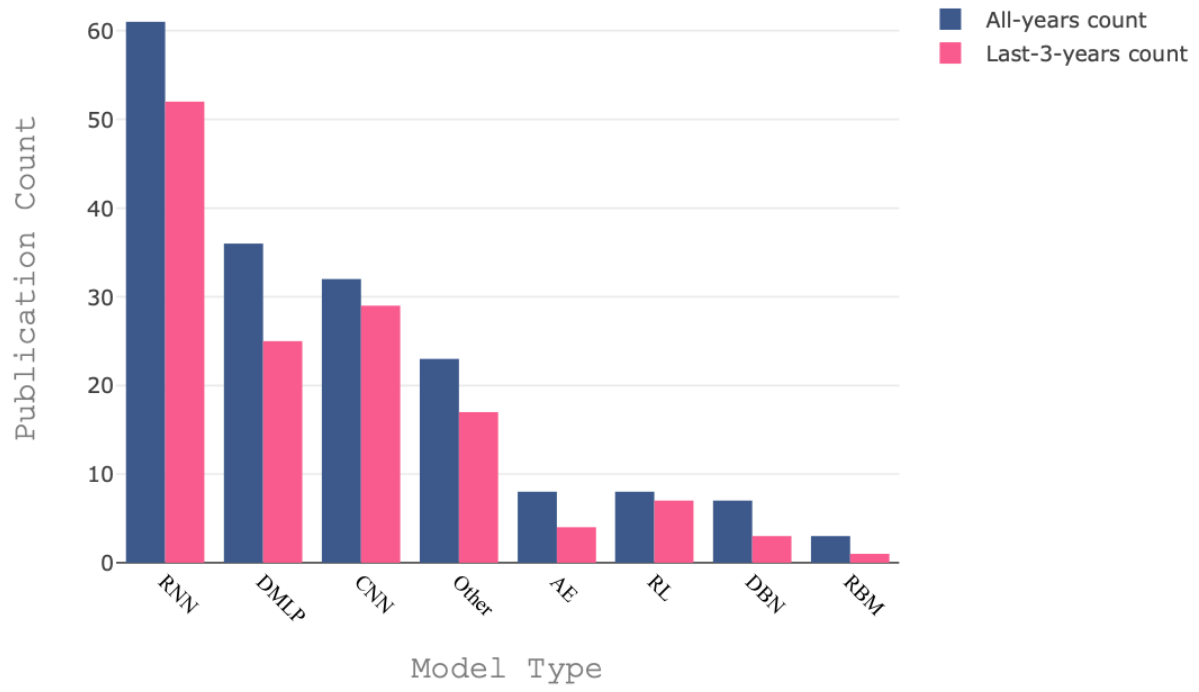


Dall' istogramma si evince come le aree decisamente più ricercate sono gli algoritmi di trading e il text mining finanziario, che come detto nella sezione precedente sono altamente connessi tra loro. Questo a prova del fatto che la possibilità di previsione dei prezzi azionari, al fine di generare profitti, eccita di fatto i sogni degli investitori e delle aziende connesse, che quindi investono molti fondi in ricerche di questo tipo. Un problema legato a questo tipo di ricerca però è che l'interesse nello sviluppo appartiene soprattutto ad aziende private, che perseguono il proprio profitto investendo enormi fondi in ricerche private, pertanto la conoscenza pubblica a riguardo è probabilmente minore di quella raggiunta nella realtà. Ad esempio se un broker riuscisse ad implementare un algoritmo di previsione dei prezzi, il risultato rimarrebbe oscuro al pubblico in quanto al broker non gioverebbe la divulgazione alla concorrenza.

Un altro aspetto molto interessante è il fatto che la ricerca nel Deep Learning è cresciuta in maniera decisa in anni recentissimi, cosa che rappresenta una prova di quanto sia un argomento di forte innovazione. Infatti, per tutte le aree applicative, vediamo come la barra in rosso (che rappresenta i lavori di ricerca degli ultimi 3 anni) sia una grande percentuale dei lavori totali, rappresentati dalla barra in blu. Il grafico seguente, che rappresenta il numero di pubblicazioni per ogni anno dal 1998 al 2018, dà un'indicazione ancora più chiara di quanto recente sia l'exploit dei modelli Deep Learning.

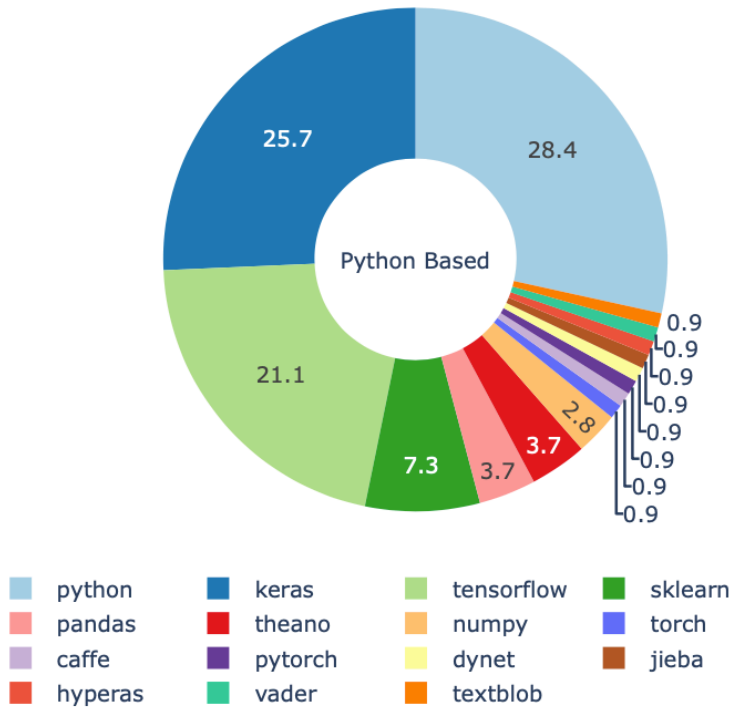


Un'altra statistica interessante, i cui dettagli tecnici saranno oggetto dei capitoli successivi, è quella che riguarda le strutture impiegate per affrontare i problemi. Infatti il Deep Learning è una classe di numerosi modelli, diverse strutture, ognuna con le proprie peculiarità e caratteristiche che si adattano ad affrontare alcune classi di problemi meglio di altre.

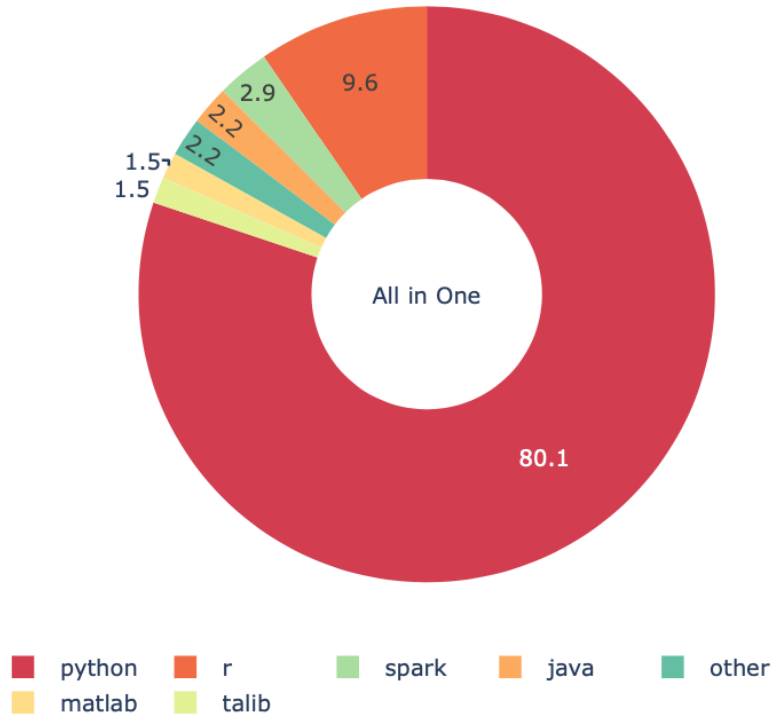


Le RNN (Reti Neurali Ricorrenti), le DMLP (Deep Multilayer Perceptron) e le CNN (Convolutional Neural Network) sono le strutture più diffuse. Le RNN in particolare rappresentano la maggioranza dei lavori effettuati fino a questo momento, per le loro caratteristiche che le rendono perfette per essere utilizzate su sequenze come le serie storiche, comprese quelle finanziarie. Le CNN sono molto innovative. Solo in anni recentissimi si sta diffondendo il loro utilizzo nelle analisi finanziarie, perché nate strutturalmente per lavorare su segnali bi-dimensionali come le immagini, mentre solo negli ultimi tempi questa classe di modelli è stata ristrutturata per essere adattata a lavorare su segnali mono-dimensionali come le serie storiche. Comunque si tratteranno nel dettaglio le caratteristiche di questi modelli nei capitoli successivi.

Un ultimo aspetto da considerare è quello relativo ai software, o per meglio dire ai linguaggi di programmazione, utilizzati per costruire questi modelli e testarli. Bisogna fare una distinzione tra quelli che si chiamano linguaggi di programmazione ad alto livello e linguaggi di programmazione a basso livello. Un linguaggio di programmazione è più ad alto livello quanto più si utilizzano delle funzioni built-in precostruite basate su altre funzioni che sono già implementate e testate all' interno di file chiamati librerie, che possono essere incorporate nel codice principale per poi essere richiamate ed utilizzate. Un linguaggio di basso livello costituisce invece le funzioni presenti nelle librerie, su cui si appoggiano i linguaggi di alto livello. Di solito si parte quindi dai linguaggi di basso livello. Il grafico seguente ci mostra i linguaggi di basso livello (o di back-end) che implementano funzioni per costruire e manipolare le Reti Neurali Profonde.



Dal grafico si evince che le librerie più utilizzate sono keras, tensorflow e alcune librerie di Python stesso. Altre degne di nota sono theano, una libreria implementata da Google e numpy, nata come grande libreria contenente funzioni di calcolo e statistiche che ha implementato una routine di funzioni dedicate al Machine Learning, tra cui dei metodi per implementare e gestire le Reti Neurali.
Il grafico seguente mostra invece i linguaggi di alto livello che si possono utilizzare per l'implementazione.



In questo caso Python la fa nettamente da padrone, seguito da R. Altri linguaggi degni di nota sono Java e Matlab. Si anticipa che in questo lavoro è stato utilizzato Python per l' implementazione come linguaggio di alto livello, essendo l'ambiente più diffuso e pertanto anche quello dove è possibile trovare più materiale e informazioni, sfruttando in back-end le funzioni della libreria Keras, che a sua volta si appoggia al linguaggio di ancor più basso livello Tensorflow.

1.3 OBIETTIVO DELL'ANALISI

Lo sviluppo dell'Intelligenza Artificiale nella forma delle Reti Neurali Profonde ha consentito l'evolversi dei tentativi di risolvere una problematica che accomuna diversi campi scientifici: lo studio delle serie storiche, ed in particolare la previsione dei valori futuri della serie. Ma che cos' è una serie storica?

Una serie storica è semplicemente definita come un insieme di dati, raccolti istante per istante, osservati su un determinato fenomeno. I dati raccolti possono essere equi-spaziati, ovvero scanditi da istanti di tempo regolari (ogni secondo, ogni minuto, ogni ora, ogni giorno, una volta al mese, una volta l'anno…) oppure non equi-spaziati, il che significa che la cadenza temporale di raccolta non è regolare. Le serie storiche possono presentare delle oscillazioni di varia natura intorno ad un andamento di lungo periodo. Queste oscillazioni sono classificate a seconda della tipologia:

- Trend: definito come un movimento tendenziale di lungo periodo che causa un'evoluzione strutturale del fenomeno;
- Ciclo: movimento originato dal presentarsi di condizioni più o meno favorevoli, prima di espansione e poi di contrazione;
- Stagionalità: oscillazione scandita da particolari periodi (ad esempio le stagioni);
- Accidentalità: movimenti originati da componenti di disturbo provocati da circostanze che non modificano il fenomeno in maniera strutturale, ma soltanto in un lasso di tempo limitato.

Nell'ambito della previsione delle serie storiche vi sono due approcci differenti: uno classico in cui si suppone che il movimento della serie sia composto da una parte deterministica e da una parte probabilistica; uno moderno in cui si suppone che la serie storica sia generata da diverse componenti stocastiche correlate o meno tra loro. Qualunque sia l'impostazione, le previsioni altro non sono che un'informazione sul probabile valore futuro del fenomeno. La previsione delle serie storiche è molto difficile perché in aggiunta ai normali problemi di regressione e classificazione c'è la problematica dell'interdipendenza temporale dei dati. Infatti quello che diversifica i dati della serie storica da quelli di un normale dataset è proprio il fatto che il dato di una serie esiste in quanto successivo al precedente e precedente al successivo. Il modello deve dunque tenere conto di questi legami.

Come accennato esistono diverse applicazioni scientifiche in cui si sfruttano le serie storiche. Ad esempio un'azienda potrebbe studiare la serie storica delle vendite mensili degli ultimi anni al fine di tentare di prevedere quali saranno le vendite dei mesi successivi. Oppure in un processo chimico si può studiare la variazione nel tempo di una variabile come la temperatura.

Quelle di interesse per la tesi che si sta sviluppando sono le serie storiche finanziarie. Una serie storica finanziaria è composta da dati ottenuti da rilevazioni effettuate su attività finanziarie. È possibile prendere in considerazione diversi dati e diverse cadenze temporali. Ad esempio i dati potrebbero essere raccolti per minuto o

giornalmente. Per quanto riguarda i dati, solitamente quello che interessa è il prezzo dell'asset. Pertanto si può affermare che la serie finanziaria definisce l'evoluzione nel tempo del prezzo di un asset.

Il problema della previsione delle serie storiche finanziarie è dovuto a diversi fattori che intercorrono nell'evoluzione del loro trend: tra gli altri bisogna tenere in considerazione la situazione macro-economica della zona, la situazione politica, profitti e perdite della compagnia (nel caso si consideri un'azienda), tasso di interesse, eventualmente tasso di cambio, ecc. L'interesse sostanziale nel trovare la soluzione a questo genere di problema sta senza ombra di dubbio nel fine di generare profitti. Questo significa che non è solo importante prevedere il prezzo futuro di un asset, ma può rivelarsi sufficiente ottenere una previsione del movimento futuro del prezzo, per sapere se sia meglio porsi su una posizione lunga oppure su una posizione corta.

Le caratteristiche delle Reti Neurali Profonde suggeriscono di poter offrire un valido contributo nella risoluzione del problema. Le Reti Neurali aiutano a superare alcuni limiti dei metodi statistici classici:

- Generalmente è supportata la mancanza di alcuni dati nella serie;
- Grande efficacia nel captare caratteristiche altamente non lineari delle funzioni, a differenza dei metodi classici che sono essenzialmente lineari o linearizzabili;
- È supportata l'analisi multivariata;
- È possibile avere un orizzonte di previsione multi-step.

1.4 TASSONOMIA DEL PROBLEMA

Quando ci si appresta ad affrontare un problema quale la previsione delle serie storiche, ci sono molti fattori da considerare. È importantissimo comprendere quali sono le caratteristiche del problema e capire in che modo queste possano impattare nel modello, al fine di scegliere la struttura e la composizione del dataset migliore. Per tale motivo, in questo paragrafo, si tratta della tassonomia dei problemi di previsione delle serie storiche. Con tassonomia intendiamo la classificazione del problema, quindi gli aspetti rilevanti da tenere a mente durante la progettazione del framework.

Il primo aspetto da tenere in considerazione è la definizione degli Input e degli Output. Può sembrare banale, ma avere chiaro questo aspetto risulta cruciale al fine di strutturare correttamente il modello. Si sottolinea che come Input non intendiamo il set utilizzato per addestrare il modello, ma intendiamo le variabili che intendiamo

analizzare al fine di effettuare la previsione degli Output. Quindi bisogna considerare quali variabili vogliamo utilizzare come Input e quali variabili utilizzare come Output. Una stessa variabile può essere utilizzata sia come Input che come Output. Il caso più semplicistico potrebbe essere quello in cui è dato come Input al modello i dati sulle vendite mensili passate al fine di prevedere le vendite mensili future, quindi utilizzando la stessa variabile sia in Input che in Output.

Oltre a quali variabili è importante decidere da principio quante variabili utilizzare. Si dice che l'analisi è univariata se si considera una singola variabile; nel caso in cui si considerino invece più variabili si dice che l'analisi è multivariata. Questo vale sia per gli Input che per gli Output. Quindi si può avere un'analisi univariata in Input e in Output, un'analisi multivariata in Input e in Output oppure un mix tra le due, considerando una variabile singola per l'Input o per l'Output e più variabili per l'altra.

I problemi di previsione per loro natura prendono in ingresso dati passati al fine di ottenere una previsione sui dati futuri. Bisogna dunque decidere l'orizzonte di previsione, che può essere single-step oppure multi-step. Se l'analisi è multi-step significa che a partire da un certo numero di dati in ingresso si vogliono prevedere più valori futuri; viceversa se si vuole prevedere un singolo valore l'analisi è single-step. Tornando all'esempio delle vendite si potrebbe pensare di prevedere le vendite dei 3 mesi successivi a partire dai dati di vendita dei 12 mesi precedenti. In questo caso si ha un orizzonte di previsione multi-step. Se invece si vuole predire le vendite del solo mese successivo a partire dai 12 mesi precedenti allora l'orizzonte di previsione è single-step.

Bisogna poi decidere il tipo di valore che si vuole predire. Un problema di regressione consiste nel trovare un valore quantitativo; quindi nell'esempio delle serie storiche finanziarie effettuare una regressione significa prevedere il valore esatto del prezzo di un certo asset. Viceversa ci si potrebbe accontentare di risolvere un problema di classificazione, in cui l'output finale è un'etichetta o per l'appunto una classe di appartenenza. Sempre considerando l'esempio delle serie stoiche finanziarie ci si potrebbe accontentare di valutare se la fluttuazione del prezzo è positiva o negativa.

Un'altra considerazione riguarda i legami e le dipendenze che intercorrono tra le variabili. Infatti, dando per scontato che le variabili di Output hanno una qualche dipendenza con tutte le variabili di Input, le variabili di Input possono essere endogene o esogene. Sono endogene se a loro volta dipendono da altre variabili sistemiche. Sono esogene se invece non hanno alcuna dipendenza da altre variabili del sistema. Approfondire quest'aspetto può risultare utile quando non si ha certezza

della correlazione che vi è tra le variabili in gioco, che quindi può portare a sperimentazioni volte a trovare la configurazione migliore di scelta delle variabili atte a risolvere il problema che ci si è posti.

Un altro aspetto che può essere valutato è la strutturazione della serie storica. Diciamo che una serie storica è strutturata se è possibile riconoscere al suo interno dei pattern. Al fine di valutare la struttura può essere utile tracciare il grafico della variabile che si sta analizzando. Se è possibile riconoscere dei movimenti si dice che la serie storica è strutturata, viceversa si dice non strutturata.

Per quanto riguarda la dinamicità, si parla della variabilità della funzione nel tempo. Se il fenomeno è governato da una funzione tempo-variante, significa che ad ogni istante di tempo cambia la funzione di input-output che genera i valori. Viceversa tempo-invariante significa che la funzione rimane costante nel tempo, scenario ovviamente molto più semplice, ma difficilmente riscontrabile nella realtà. In quest'ultimo caso la serie si dice stazionaria. La valutazione di questo tema è molto rilevante perché impatta l'affidabilità nel tempo del sistema creato: è chiaro che nel caso in cui la funzione cambi nel tempo, un sistema tarato sui dati di oggi potrebbe non essere efficace domani; o comunque un modo di vederla diverso potrebbe essere che, se anche la struttura continuasse a risultare efficace per modellare il fenomeno in esame, resterebbe il problema di dover riaddestrare il modello con i nuovi dati quando disponibili, al fine di far captare al sistema le variazioni che ci sono state.

Infine bisogna studiare il dataset che si ha a disposizione. Se è possibile rilevare dati con cadenza temporale regolare, si dice che la serie è continua. Viceversa se bisogna accontentarsi di dati più irregolari, si dice che la serie è discontinua. Banalmente un dataset composto da una serie continua è modellabile più facilmente rispetto ad una discontinua. Purtroppo per alcuni fenomeni non sempre questo è possibile.

1.5 COME SVILUPPARE UN MODELLO DI PREVISIONE

Solitamente, un modo semplice per descrivere la fase iniziale dello sviluppo di un framework di previsione è il seguente: si parte da un dataset di partenza, che bisogna usare per prevedere i valori futuri. Come si può immaginare questa è una visione troppo semplicistica. In questo paragrafo struttureremo i passi necessari per implementare un modello di previsione.

1.5.1 Definizione dell'ambito di indagine

Per prima cosa è necessario definire l'ambito di indagine, quindi studiare a fondo il fenomeno estraendo la sua tassonomia, ovvero le caratteristiche di base, in maniera tale da poter più facilmente valutare fin da subito le risorse necessarie alla risoluzione e la tipologia e la struttura dei dati che è serve avere a disposizione.
In pratica è necessario approfondire tutto quanto spiegato fino ad ora nel capitolo, quindi stabilire con precisione in quale area ci si va a posizionare, stabilire i confini della misurazione del fenomeno ecc. In questa fase è necessario anche comprendere appieno lo scopo della previsione. Ancora bisogna decidere con quale cadenza raccogliere i dati. Infine è necessario stabilire l'orizzonte di previsione: l'accuratezza delle stime generalmente diminuisce all'aumentare dell'orizzonte di previsione.

1.5.2 Preparazione dei dati

Una parte rilevante dello sviluppo consiste nel raccogliere e preparare l'archivio dati. Infatti bisogna innanzitutto essere sicuri di avere a disposizione i dati che servono, ma non solo. Bisogna anche che questi dati continuino ad essere reperibili nel tempo: è logico che se si hanno a disposizione oggi i dati, si può addestrare il modello, ma si avrà sempre bisogno di altri dati futuri per poter testare con continuità l'accuratezza del framework nel tempo.

La preparazione dei dati quindi è importante quanto la raccolta. Si parla probabilmente dell'aspetto più delicato e forse più difficile da comprendere concettualmente. Questo perché le Reti Neurali Artificiali Profonde sono modelli altamente efficaci nel mappare le funzioni. Si capisce bene che la serie storica così come è data non è una funzione di input-output, pertanto bisogna operare delle trasformazioni al fine di renderla analizzabile da un modello Deep Learning. Si analizzerà nel dettaglio che tipo di trasformazione deve subire la serie e come è possibile l'implementazione nei capitoli successivi.

Ma la trasformazione della serie storica non è l'unica trasformazione che bisogna effettuare sui dati. Infatti anche per le Reti Neurali, così come per i classici modelli di Machine Learning, risultati empirici dimostrano come la normalizzazione dei dati su una scala comune può aiutare a migliorare la qualità del modello. Utilizzare valori più "indicativi" in luogo dei valori assoluti velocizza il processo di apprendimento.

Infine il dataset deve essere suddiviso in due gruppi: un training set e un validation set. Il training set è la parte della serie storica che verrà utilizzata per addestrare il modello, ovvero per permettergli di settare i parametri della Rete. Pertanto il training set sarà sottoposto al modello sia con gli Input che con i corrispondenti Output che

lo compongono. Il validation set, o test set, è composto da una parte di dati che il modello non ha mai analizzato e che verranno dati in pasto al modello solo come Input, in maniera tale da poter confrontare l'Output, cioè i valori attesi, con i valori reali di cui siamo a conoscenza e poter così valutare l'efficacia del modello.

In un'analisi ben fatta la preparazione dell'archivio dati avviene per raffinazioni successive. Questo perché molto spesso non si è ben consapevoli di quali variabili sono effettivamente necessarie e quali no. Esistono delle tecniche che consentono di calcolare quanto una variabile effettivamente partecipa dell'addestramento di un modello. Quindi vi è una prima raccolta e preparazione dati; si testa il modello; si analizzano i risultati; eventualmente si eliminano delle variabili considerate irrilevanti e poi si ricomincia fino a che non si ottiene il dataset migliore, ovvero quello che dà i risultati migliori con meno variabili.

1.5.3 Implementazione della Rete

La fase di implementazione è quella della costruzione della Rete. In questa fase quindi si sceglie la struttura di Rete Neurale che si vuole utilizzare e gli iper-parametri. Gli iper-parametri della Rete Neurale sono il numero di percettroni che compongono uno strato, il numero di epoche utilizzate per l'addestramento, la funzione di attivazione di ogni strato, la funzione di perdita di ogni strato, l'algoritmo di settaggio dei parametri ecc.

Purtroppo non esistono teoremi, né metodi empirici, che consentono di sapere a priori qual è la struttura e la configurazione di iper-parametri migliore per risolvere un determinato problema. Quest'aspetto si dimostra rilevante in quanto, come vedremo, il tempo necessario all'addestramento di una Rete Neurale non è trascurabile, pertanto è possibile testare solo un numero limitato di configurazioni. Questo porta ad una conseguenza per niente banale, ovvero che è possibile ottenere da una Rete Neurale risultati molto soddisfacenti o meno, ma non vi è la possibilità di sapere con certezza che non esista una configurazione della struttura e dei parametri che possano dare risultati migliori.

La fase di implementazione della rete si chiude con l'addestramento, quindi si sottopone al modello il training set tramite delle funzioni specifiche per settare i pesi che andranno a formalizzare il modello.

Tutti questi aspetti saranno analizzati nel dettaglio nei prossimi capitoli, dove sarà sviscerato tutto quello che vi è dietro le Reti Neurali, dalla genesi al funzionamento.

1.5.4 Test e valutazione delle performance

Ottenuta la Rete Neurale e addestrata al fine (si spera) di dare un risultato al problema che ci si è posto, si passa alla fase di testing.

In primis è necessario definire delle metriche di valutazione, che possono essere diverse a seconda innanzitutto del tipo di problema che si affronta, di regressione o di classificazione. Per quanto riguarda la regressione metriche molto gettonate sono il calcolo dell'RMSE e della correlazione che vi è tra i valori attesi rilevati e quelli reali. Per quanto riguarda invece la classificazione si tende soprattutto a numerare i valori assegnati alla classe corretta e i valori non correttamente assegnati. Viene in aiuto una struttura chiamata Matrice di Confusione, in cui vengono elencati questi parametri da cui è possibile calcolare diversi indicatori.

Definite le metriche si passa al testing del modello. Per valutare le performance normalmente si devono avere dei modelli di comparazione. In questi primi anni di ricerca i benchmark comparativi sono rappresentati dai modelli di previsione di machine learning più classici.

Capitolo II

RETI NEURALI ARTIFICIALI

Si tratterà ora lo strumento principale su cui si costruirà l'esperimento, le Reti Neurali Artificiali Profonde, dai fondamenti alla teoria su cui si basano questi modelli, il paradigma di apprendimento e la base matematica su cui si fondano, in particolare come evolve l'algoritmo che consente di settare "automaticamente" i weight e i bias della rete.

2.1 INTELLIGENZA ARTIFICIALE, MACHINE LEARNING E DEEP LEARNING

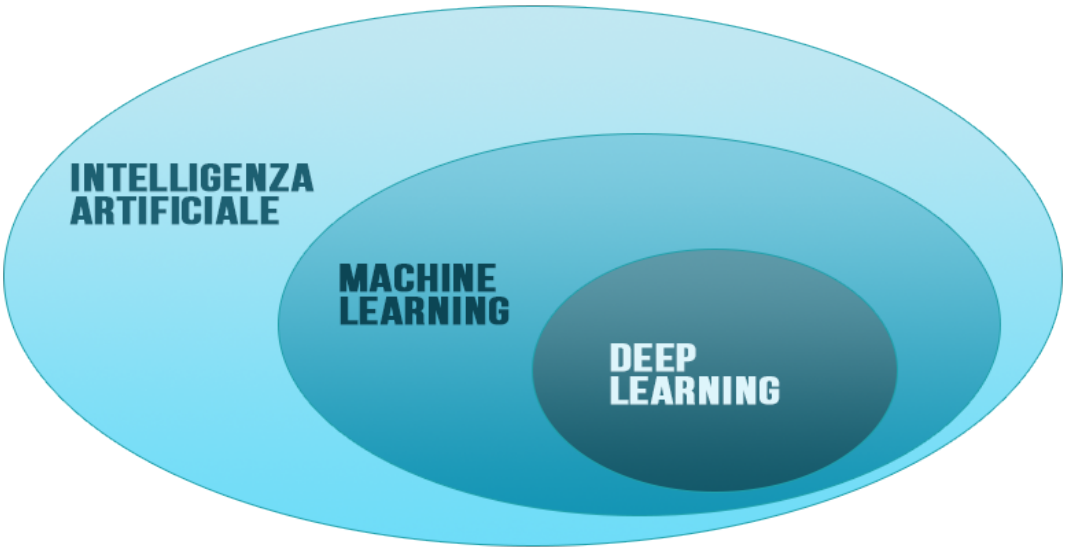
I software con cui si è abituati ad interagire sono strumenti in grado di svolgere alla perfezione una serie di operazioni in modo veloce, sicuro e affidabile. Tuttavia questi programmi non possono certamente definirsi "intelligenti", in quanto queste operazioni sono programmate ed implementate da un essere umano. Una macchina si può definire intelligente quando da sé riesce a capire come risolvere un problema, senza che un umano scriva direttamente in codice le operazioni necessarie per la risoluzione, anche se bisogna tenere a mente che è sempre il programmatore a definire l'algoritmo, anche se questo è un algoritmo di "apprendimento" (in fondo il computer è soltanto uno stupido velocista[3]). Da queste righe introduttive è già possibile iniziare a capire quali sono le caratteristiche che definiscono il mondo dell'Intelligenza Artificiale. Prima di addentrarsi nella spiegazione delle Reti Neurali, che rappresentano solo una delle possibili facce con cui si realizza l'AI, è interessante chiarire alcune terminologie di quest'area applicativa che spesso vengono utilizzate in maniera intercambiabile, ma che in realtà identificano concetti diversi, cominciando naturalmente proprio dall' Intelligenza Artificiale, che è possibile definire come la madre dei i concetti su cui si basano i modelli che si stanno per andare a sviluppare.

Di solito quando si parla di Intelligenza Artificiale il pensiero si proietta in immagini degne di film apocalittici, come Terminator o Io, Robot, in cui vi sono delle macchine che si sostituiscono in tutto (e in modo migliore) agli esseri umani. In realtà questa è una visione molto lontana dalla realtà, in quanto l'Intelligenza Artificiale abbraccia anche concetti molto più semplici di quelli espressi nei film sopra citati. In generale si può dire che l'AI incorpora tutte le azioni che possono essere svolte dall'intelletto umano, ma che sono eseguite da un calcolatore. Queste azioni possono includere operazioni come pianificazione, riconoscimento di segnali di varia natura (suoni,

immagini, testo), fino ad arrivare al cosiddetto apprendimento e successiva risoluzione di un problema. Quindi si deduce che l'AI, seguendo la definizione formale, comprende una serie di applicazioni appartenenti anche a branche che in prima battuta non si definirebbero intelligenti.

Il Machine Learning (letteralmente "apprendimento automatico") è una delle possibili strade per attuare l'Intelligenza Artificiale. Questa particolare branca include quei modelli che ricevono i dati e modificano le azioni da eseguire adattandole ai nuovi dati ricevuti, quindi in un certo senso "imparando dall'esperienza". Il Machine Learning rappresenta quindi un concetto che più si avvicina ai pensieri che comunemente si possono avere sull'Intelligenza Artificiale, ed è proprio per questa ragione che spesse volte i due termini vengono utilizzati in maniera intercambiabile. Alcuni esempi di modelli di Machine Learning sono la regressione multivariata, l'albero di decisione, reti neurali ecc.

Il Deep Learning, invece, include una classe di metodi chiamati Reti Neurali Profonde, che prendono spunto nella loro costruzione dalla struttura del cervello. Le Reti Neurali sono state nominate anche tra gli esempi di modelli Machine Learning. La differenza tra Reti Neurali e Reti Neurali Profonde è che le seconde possiedono uno o più strati intermedi nascosti. Questo dà alla rete un potere di apprendimento molto maggiore di quelle composte semplicemente da uno strato di input e da uno di output. Altri approcci di Deep Learning comprendono clustering, reti bayesiane e algoritmi di programmazione a logica induttiva.

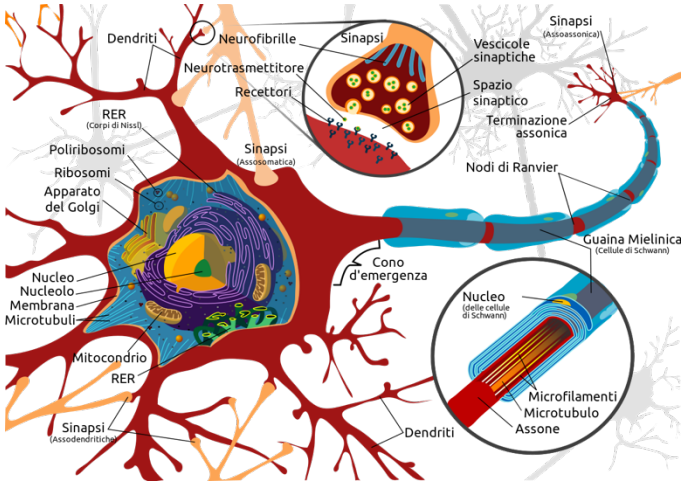


Come detto più volte anche nel capitolo introduttivo, il metodo prescelto per affrontare l'esperimento è la Rete Neurale Profonda, pertanto adesso si sviscereranno i fondamenti e la matematica su cui questo modello si poggia.

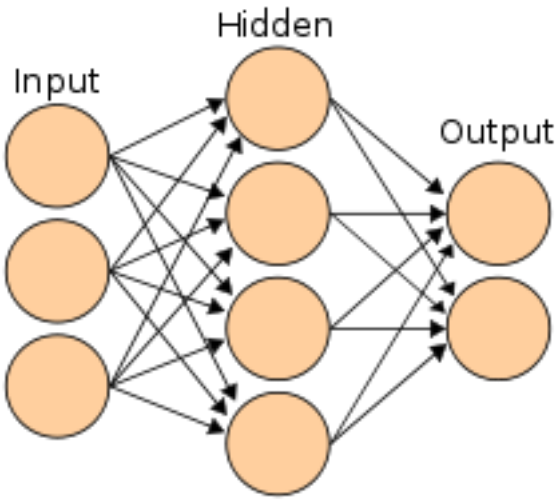
2.2 RETI NEURALI

Le Reti Neurali Artificiali definiscono dei sistemi informatici la cui struttura è ispirata dal funzionamento delle Reti Neurali biologiche. Questo non deve sorprendere in quanto nel corso del tempo vi sono state numerose similitudini tra il comportamento del cervello umano e il comportamento di un calcolatore.

Il cervello umano è composto da miliardi di cellule nervose, chiamate anche neuroni, che hanno la caratteristica di essere in connessione tra di loro tramite strutture chiamate sinapsi. I neuroni comunicano tra loro inviando impulsi elettrici che viaggiano attraverso le sinapsi, trasmettendo informazioni. I neuroni possono essere di varia natura: alcuni ricevono informazioni dall'ambiente esterno (che possono essere suoni, odori, immagini ecc.); altri neuroni invece inviano segnali che vengono trasmessi all'ambiente esterno; altre cellule nervose invece sono in connessione solo con altri neuroni, partecipando quindi esclusivamente ad elaborazioni interne alla rete. Si stima che ogni neurone è connesso a miliardi di neuroni in una rete infinita di elaborazione dell'informazione che rende l'essere umano "intelligente".



Lo scopo delle Reti Neurali Artificiali è quello di simulare il comportamento del cervello umano (anche se in maniera estremamente ridotta). Infatti creare una rete con centinaia di miliardi di percettroni (così si chiamano i "neuroni" che compongono una Rete Artificiale) e con le miliardi di connessioni che caratterizzano il cervello richiederebbe uno sforzo di elaborazione che i calcolatori, per quanto potenti, non sono in questo momento in grado di supportare. Tuttavia anche con Reti molto più piccole si sono ottenuti risultati molto soddisfacenti. Proprio come il cervello abbiamo almeno uno strato di input, che riceve i dati da analizzare e uno strato di output, che espelle l'informazione desiderata dopo l'elaborazione; poi possono esserci o meno uno o più strati nascosti (hidden layer) che partecipano solo delle elaborazioni interne. In questo caso parliamo di Reti Neurali Profonde.



L'esempio di rete schematizzata di fianco è una Fully Connected, in quanto ogni percettrone dello strato precedente invia un segnale a tutti i percettroni dello strato successivo. Inoltre è Feed Forward perché i segnali viaggiano in un solo verso, dall'inizio alla fine della rete; non c'è quindi retro propagazione di segnali. Come si vedrà alcune strutture invece prevedono la retro propagazione.

Ogni percettrone riceve quindi *n* segnali in ingresso. Quello che fa ogni percettrone della rete è assegnare un weight (o un peso) ad ogni segnale d'ingresso; dopodiché il percettrone calcolerà la somma pesata dei segnali d'ingresso, eventualmente aggiustata da una costante chiamata bias e il risultato sarà dato in ingresso ad una cosiddetta funzione di attivazione, che calcolerà l'output del precettrone e che sarà inviato a tutti i percettroni dello strato successivo (nel caso di una rete Fully Connected). Dettagli sulla funzionalità del percettrone e sull'algoritmo di settaggio di weight e bias saranno oggetto dei paragrafi successivi.

2.3 I PARADIGMI DELL'APPRENDIMENTO AUTOMATICO

Come sottolineato in precedenza il fatto che un modello sia definito intelligente non deve fuorviare. Il significato stesso della parola intelligenza non è troppo banale, in quanto può assumere diverse connotazioni a seconda del contesto in cui la si usa. Il contesto cui si fa riferimento in questa ricerca è l'intelligenza pratica, che si può definire come la capacità di un agente di affrontare e risolvere con successo situazioni e problemi nuovi o sconosciuti. Avere ben chiara la definizione è importante per capire che in realtà i modelli che si stanno descrivendo non hanno nulla di intelligente. Infatti non hanno di per sé la capacità di risolvere problemi, ma sono creati in modo da acquisire questa capacità, ovvero vengono programmati con algoritmi i cui passi consentono al modello di reagire in maniera opportuna ai dati ricevuti modificando l'algoritmo stesso di funzionamento. I paradigmi dell'apprendimento automatico comprendono le tipologie di algoritmi che consentono di realizzare questi comportamenti che possiamo definire "intelligenti"[4].

2.3.1 Apprendimento supervisionato

L'apprendimento supervisionato è un algoritmo utilizzabile nel momento in cui si dispone di un insieme di dati per l'addestramento (training set). Disporre di un

training set significa avere un esempio della funzione di input-output che si desidera mappare, con una serie di campioni di input e i corrispondenti output. Successivamente la rete viene addestrata tramite un algoritmo di back propagation, il quale usa i dati d'esempio per settare opportunamente weight e bias della rete minimizzando l'errore di previsione. Questa tipologia di algoritmo consente di risolvere problemi di regressione e classificazione.

2.3.2 Apprendimento non supervisionato

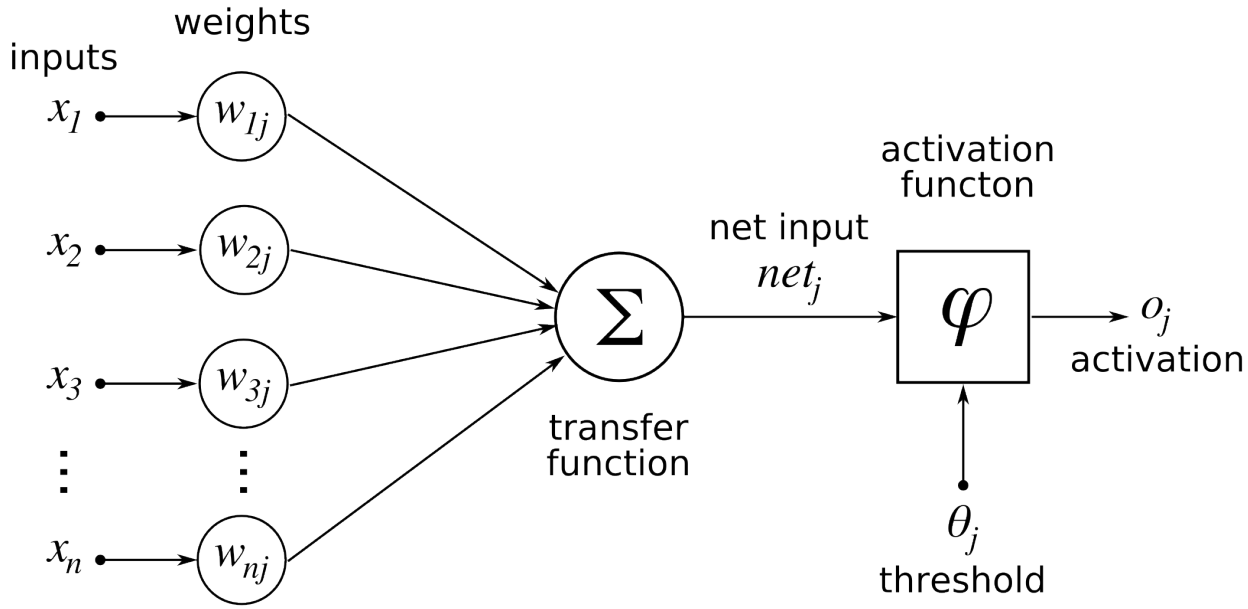
La differenza tra l'apprendimento supervisionato e il non supervisionato è che in quest'ultimo caso si hanno a disposizione soltanto gli ingressi della funzione obiettivo, senza i corrispondenti output. L'obiettivo quindi è quello di individuare delle feature che caratterizzano gruppi di dati al fine di classificarli in etichette specifiche. Per la realizzazione si fa affidamento a metodi topologici e probabilistici. Oltre alla classificazione di funzioni sconosciute, l'apprendimento non supervisionato è molto impiegato nello sviluppo di tecniche per la compressione dei dati.

2.3.3 Apprendimento per rinforzo

L'ultimo paradigma analizzato è l'apprendimento per rinforzo. Questo è molto differente dagli altri due perché non è orientato allo studio e alla categorizzazione di funzioni, ma ha lo scopo di tracciare un certo modus operandi di un processo osservato, tentando di suggerire azioni che massimizzino l'utilità. Un algoritmo per rinforzo si realizza partendo dal presupposto che le azioni hanno un impatto sull'ambiente circostante, che può essere negativo o positivo. L'algoritmo ha quindi bisogno di avere in ingresso una tabella con tutti gli incentivi e disincentivi. Si tenta poi di determinare le azioni da svolgere con l'obiettivo di massimizzare gli incentivi accumulati di volta in volta. In definitiva non si hanno informazioni da studiare per tracciare il comportamento, ma vi è un vero apprendimento basato sui dati generati dall'azione corrente e da quelle precedenti.

2.4 FUNZIONAMENTO DEL PERCETTRONE

Il percettrone è l'unità che compone una Rete Neurale. Ogni strato della rete contiene un certo numero di percettroni, che ricevono un certo numero di segnali d'ingresso e producono un'output che verrà inviato ai percettroni del layer successivo se ci troviamo in uno stadio intermedio, oppure sarà il risultato della rete se ci troviamo in uno strato di output. Obiettivo di questo paragrafo è capire come il percettrone calcola l'output e approfondire il concetto di non linearità delle Reti Neurali.



Quello riportato sopra è lo schema base di un percettrone, che riceve in ingresso un vettore di input. La prima operazione di un percettrone è quella di calcolare la somma pesata degli ingressi (l'algoritmo di settaggio dei weight e dei bias sarà approfondito successivamente).

$$net_j = (\sum_{i=1}^{n} x_i \, w_{ij}) + b_j$$

Quindi l'uscita del percettrone j-esimo è calcolata come la somma degli *n* input, ognuno dei quali moltiplicato per il corrispondente peso w_j, a cui ancora viene sommato il termine noto b_j, o bias, del percettrone.

Dallo schema si evince come questa operazione è solo il primo dei calcoli eseguiti. Il risultato infatti viene successivamente dato in ingresso ad una funzione di attivazione. La funzione di attivazione è un aspetto fondamentale di una rete neurale, in quanto è ciò che consente di riprodurre caratteristiche non lineari delle funzioni che si vuole mappare. L'uscita definitiva del percettrone quindi è la seguente:

$$y_j = \varphi(net_j)$$

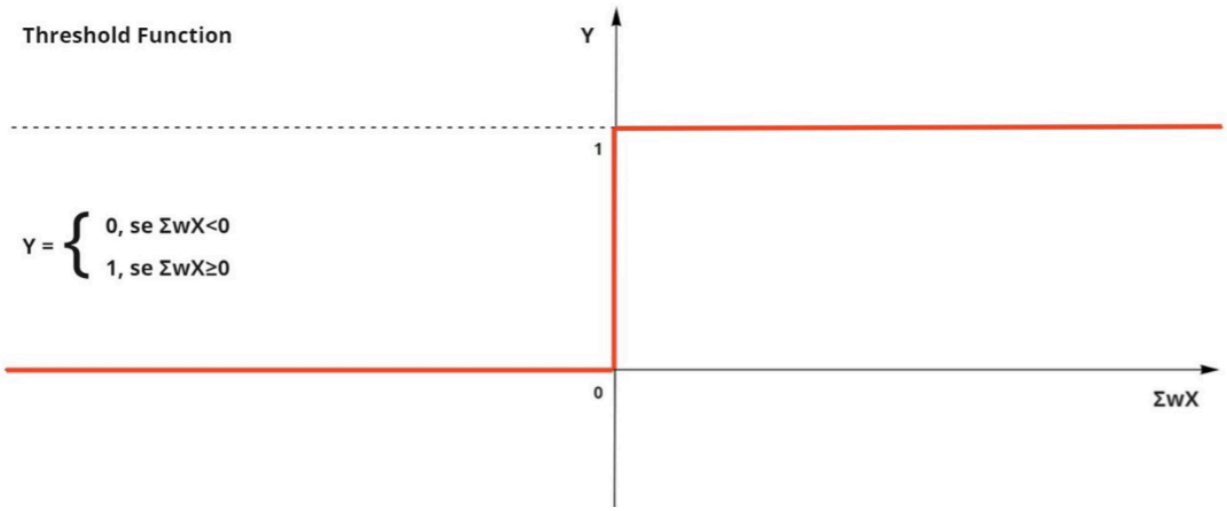
Dove φ è una funzione non lineare. La non linearità della funzione di attivazione fa in modo che y_j non sia riproducibile come una combinazione lineare degli ingressi. Per comprendere l'importanza di una funzione di attivazione adeguata basta pensare che senza di essa una Rete Neurale Profonda non sarebbe migliore di una rete composta da due soli strati, quello di input e quello di output. Le funzioni di

attivazione inoltre determinano il modo in cui la rete apprende ed anche le caratteristiche dell'output, pertanto è un parametro di scelta molto importante che caratterizza una specifica rete. Si passerà quindi in rassegna le funzioni di attivazione più utilizzate nella costruzione delle reti[5].

2.4.1 Gradino

La funzione gradino, o "soglia", è analiticamente molto semplice. Infatti restituisce il valore 1 se l'ingresso è maggiore di 0 oppure il valore 0 se l'ingresso è minore di 0.

$$\varphi(x) = \begin{cases} 1 & se \quad x > 0 \\ 0 & se \quad x < 0 \end{cases}$$

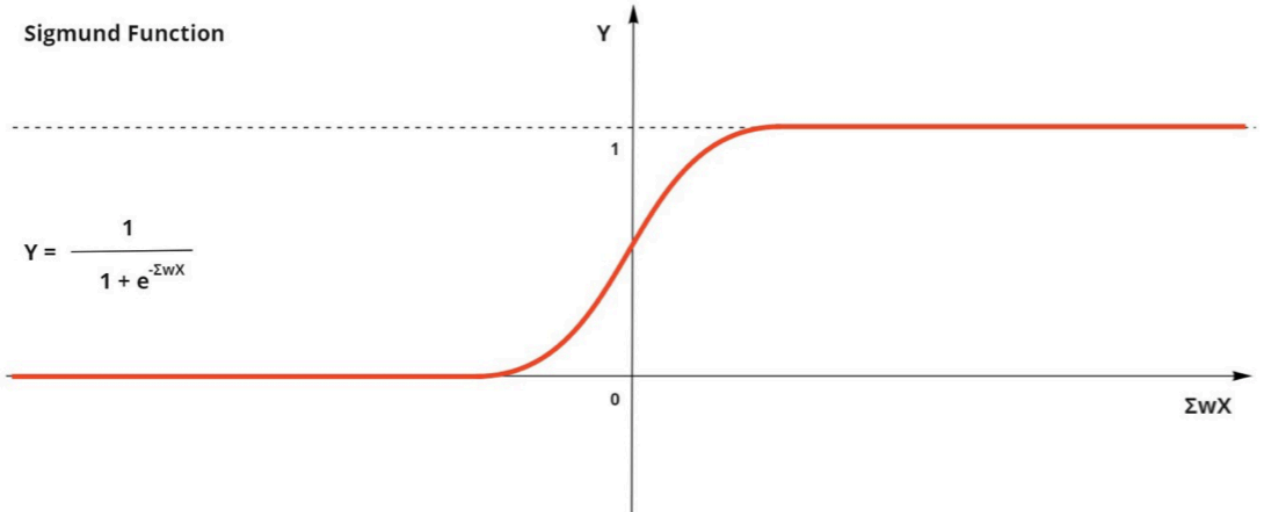


Questo tipo di funzione torna utile nel momento in cui si deve lavorare con i segnali binari, ovvero si necessita che l'uscita sia uno 0 oppure un 1. D'altro canto il fatto che la funzione non sia differenziabile crea qualche problema nell'efficienza degli algoritmi di settaggio dei pesi.

2.4.2 Sigmoide

La funzione sigmoide ha codominio in un intervallo continuo compreso tra 0 e 1. Analiticamente si esprime come segue:

$$\varphi(x) = \frac{1}{1 + e^{-x}}$$

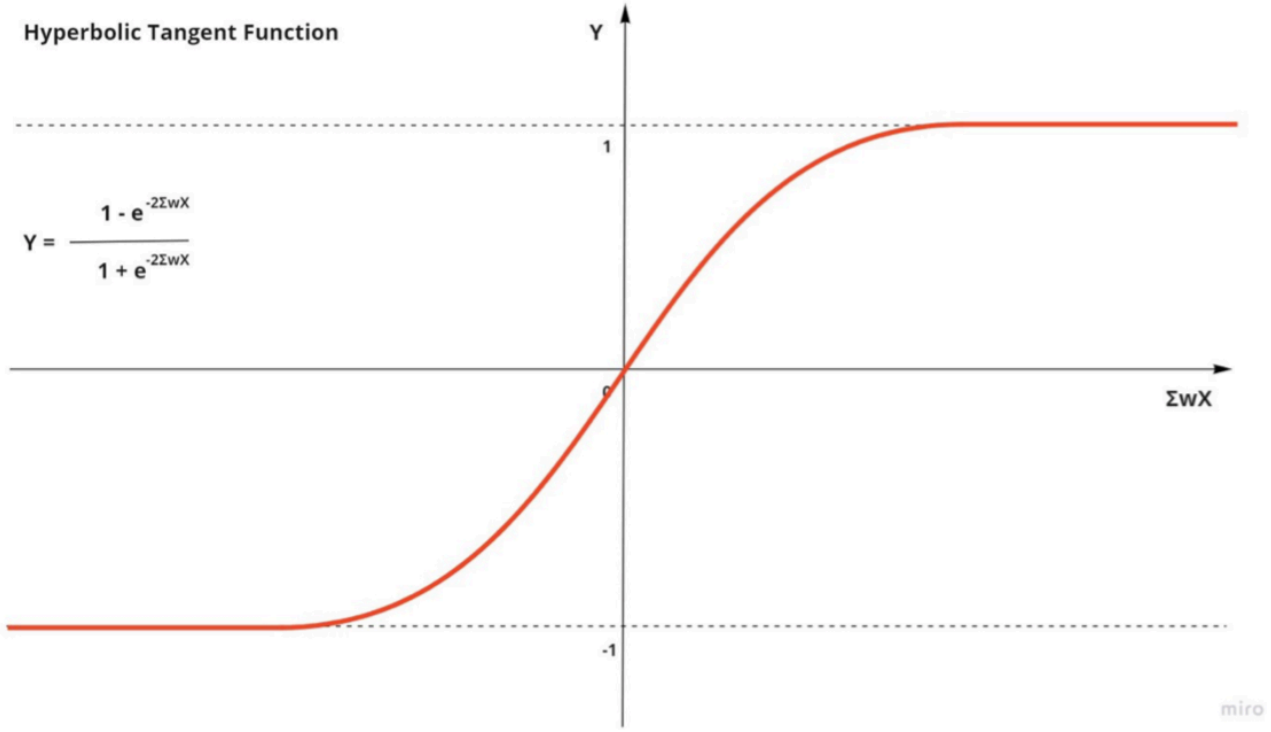


Per certi versi è simile alla funzione gradino, nel senso che può essere impiegata per identificare una variabile binaria. Però la continuità della funzione può dare un significato probabilistico all'uscita, che invece di rappresentare il valore reale può rappresentare la probabilità che il valore predetto sia pari a 1.

2.4.3 Tangente iperbolica

La tangente iperbolica è nella forma simile alla sigmoide, ma il codominio è un intervallo continuo compreso tra -1 e +1.

$$\varphi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

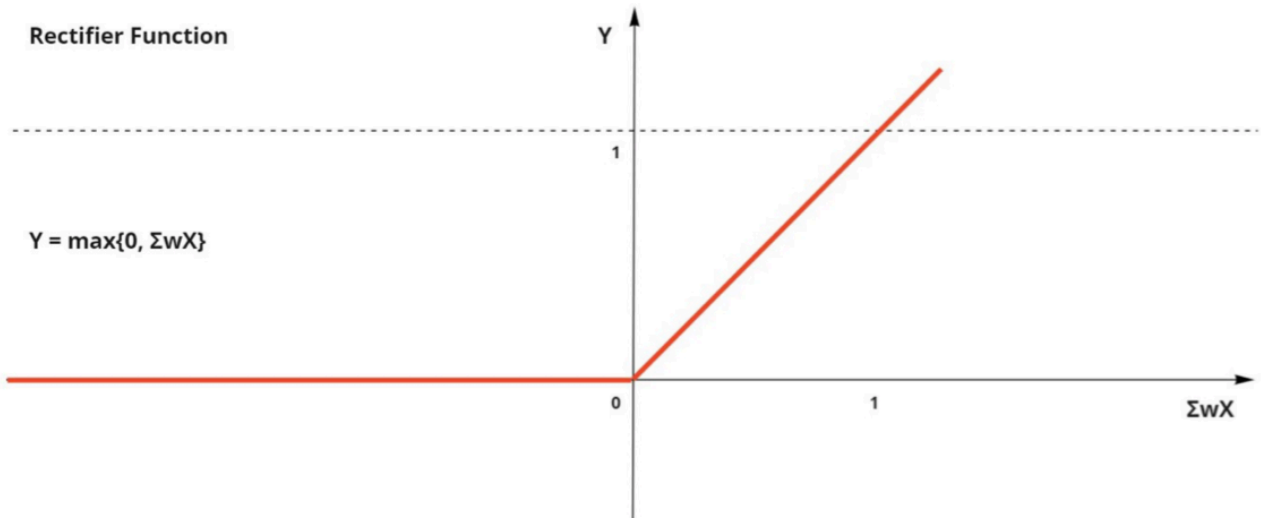


In questo caso l'uscita della funzione, compresa tra -1 e +1, potrebbe tornare utile nel caso in cui l'obiettivo di previsione sia una variazione percentuale.

2.4.4 Rettificatore

La funzione rettificatore restituisce il valore 0 se l'ingresso è minore di 0, oppure la retta bisettrice del primo quadrante se l'ingresso è maggiore di 0.

$$\varphi(x) = \begin{cases} x & se \quad x > 0 \\ 0 & se \quad x < 0 \end{cases}$$



Il rettificatore è la funzione più utilizzata nell'ambito delle reti neurali in quanto è stata empiricamente dimostrata la sua superiorità nell'algoritmo di setting dei pesi della Rete, superando il cosiddetto problema del *gradient vanish*.

2.5 ADDESTRAMENTO DELLA RETE NEURALE

La parte più affascinante della teoria delle Reti Neurali è quella dell'algoritmo che setta in modo automatico i weight e i bias di ogni percettrone al fine di rendere il modello più accurato possibile. L'algoritmo di apprendimento supervisionato che si andrà a spiegare è detto di *back propagation*, che è di gran lunga quello più utilizzato.

2.5.1 Loss function

Il primo passo da seguire per implementare un metodo d'addestramento è quello di definire una metrica che dia un'indicazione sulla bontà del modello, un indicatore i cui parametri andranno quindi modificati al fine di rendere la rete quanto più accurata possibile. Per le Reti Neurali quest'indicatore è rappresentato dalla loss function (o funzione di costo), che dato il valore predetto e il valore obiettivo restituisce la

distanza tra i due, che naturalmente si desidera più bassa possibile. Vi sono molte funzioni di costo di default che è possibile utilizzare, ma è possibile anche definirne di nuove. La loss function più utilizzata nelle Reti Neurali è il mean squared error. Ma a prescindere dalla funzione di perdita prescelta è possibile generalizzare la formula in questo modo:

$$\eta(\hat{y}, y) = \hat{y} - y = \varphi(x, w, b) - y = \eta(\varphi(x, w, b), y)$$

Dove y è il valore reale, ovvero l'obiettivo che si desidera raggiungere e $\hat{y}$ è il valore atteso. Ma il valore atteso non è altro che l'output della rete neurale che quindi possiamo dire essere dipendente dagli ingressi x, dai pesi w e dai bias b.
È stata esplicitata la funzione generale che descrive una qualsiasi funzione di perdita η al fine di mettere in evidenza la dipendenza della funzione dagli ingressi e dai parametri della rete.

Come è facile immaginare il funzionamento dell'algoritmo di addestramento si basa sulla minimizzazione della loss function. Per minimizzare la loss function non possiamo naturalmente agire né sugli input, che non sono scelti da noi, né tantomeno sulla variabile obiettivo, ovvero gli output della funzione di training. Pertanto non resta altro che la modifica dei pesi e dei bias.

2.5.2 Back Propagation

Il primo passo dell'algoritmo è quello di inizializzare i pesi della Rete. Di solito l'inizializzazione è casuale e basata su distribuzioni predefinite. I pesi non possono essere posti a 1 inizialmente perché alla rete non riuscirebbe l'apprendimento non lineare, quindi se si vuole optare per una costante bisogna utilizzare una frazione (di solito vengono utilizzate le costanti 0.1 oppure 0.01).

Il secondo passo consiste nell'effettuare un primo giro di addestramento della rete. Un ciclo completo di addestramento (significa che la rete scansiona completamente il training set) prende il nome di epoca. In realtà l'aggiornamento dei pesi non avviene necessariamente alla fine di ogni epoca ma è possibile specificarlo come parametro della rete e prende il nome di batch. In ogni caso al punto del primo aggiornamento la rete calcola il valore della funzione di costo sulla base dei valori attesi prodotti.

A questo punto bisogna calcolare la variazione dei pesi, ottenuta puntando alla minimizzazione della funzione di perdita. Essendo un problema di minimizzazione si deve calcolare la derivata parziale della funzione di perdita rispetto ad ognuno dei pesi e al bias. La derivata parziale rappresenta la direzione verso il quale bisogna

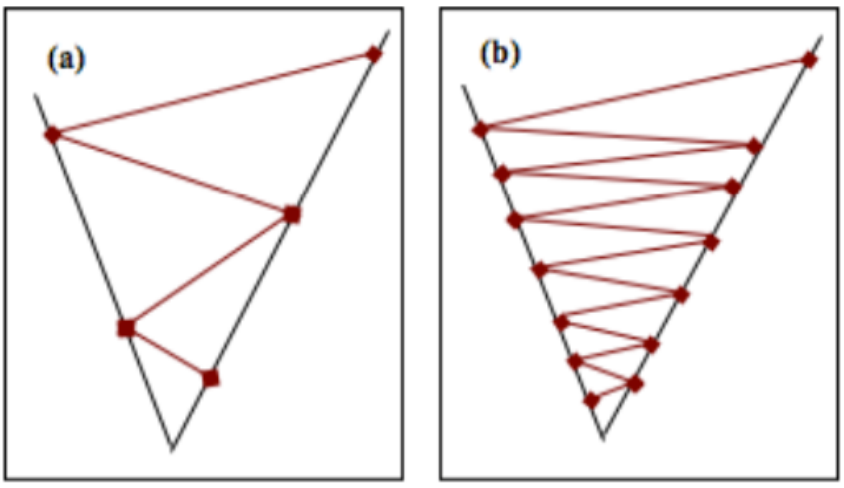
effettuare lo spostamento, in quanto rappresenta la tangente alla funzione nel punto calcolato (se la tangente è minore di 0 significa che è necessario un incremento del parametro, se è maggiore di 0 significa che il parametro ha necessità di un decremento). Il Gradient Descent è l'algoritmo utilizzato per calcolare la direzione del gradiente. In parole più semplici è il metodo con cui si calcola se la porzione di pendenza deve essere aggiunta o sottratta al valore di pesi nell'istante precedente. L'algoritmo di Gradient Descent più diffuso è lo Stochastic Gradient Descent (SGD).

Tuttavia lo scostamento non deve essere pari a tutto il valore della derivata, ma solo ad una frazione, che prende il nome di tasso di apprendimento (o learning rate). Quindi, se η è la loss function i nuovi parametri saranno calcolati come segue:

$$w_t^1 = w_{t-1}^1 - LR \cdot \frac{\partial \eta}{\partial w^1}$$
$$w_t^2 = w_{t-1}^2 - LR \cdot \frac{\partial \eta}{\partial w^2}$$
$$\dots$$
$$w_t^n = w_{t-1}^n - LR \cdot \frac{\partial \eta}{\partial w^n}$$

$$b_t = b_{t-1} - LR \cdot \frac{\partial \eta}{\partial b}$$

Il Learning Rate è un parametro della rete molto importante. La scelta determina la velocità dell'apprendimento. Un Learning Rate molto elevato infatti determina un apprendimento molto veloce; viceversa un tasso di apprendimento basso indica che la rete impiegherà più tempo a raggiungere l'obiettivo.



Il diverso comportamento derivante dalla scelta del Learning Rate è visibile nella figura sopra riportata. Infatti nella figura (a) abbiamo un LR molto elevato, in cui si raggiunge più velocemente l'obiettivo, ma le oscillazioni sono troppo ampie e di conseguenza non si ottimizza la minimizzazione della perdita. In (b) abbiamo invece un tasso di apprendimento più basso che permette alla rete di raggiungere una perdita minore, ma al costo di una più lenta velocità di apprendimento. In caso di scelta di un Learning Rate troppo basso i tempi di addestramento potrebbero essere insostenibili. Il processo di addestramento passa quindi anche dalla graduale individuazione del valore più adatto di questi parametri (lo stesso discorso vale anche per la scelta del numero di epoche e della dimensione del batch).

Proseguendo nella descrizione dell'algoritmo, una volta calcolati i nuovi parametri si ritorna al punto due, quindi ricalcolando gli output con i nuovi pesi e ricalcolando il nuovo errore, e così via in un ciclo di miglioramento fino al completamento del numero di epoche prescelto.

Capitolo III

PREPARAZIONE DEI DATI

Nei precedenti capitoli sono stati definiti nel dettaglio l'ambito di indagine e le metodologie che saranno utilizzate. Si passa ora alla parte sperimentale della tesi, ovvero la costruzione del Framework Deep Learning based per la previsione del movimento dei prezzi azionari.
In questo capitolo si affronterà la delicata problematica della raccolta e della trasformazione dei dati al fine di rendere la previsione delle serie storiche finanziarie un problema modellabile tramite una Rete Neurale Profonda.

3.1 RACCOLTA DEI DATI

Una caratteristica fondamentale che devono avere le informazioni raccolte è quella di avere una regolare frequenza di rilevazione, in modo tale da poter modificare il modello nel tempo aggiornandone i parametri. Inoltre è opportuno non addestrare la rete utilizzando periodi fortemente caratterizzati da trasformazioni strutturali di mercato. Tenuto conto di questo nell'esperimento verranno utilizzati come sottostante due indici europei su un periodo di 5 anni e mezzo: l'Euro Stoxx 50 e il FTSE Mib.

L'Euro Stoxx è l'indice azionario delle principali aziende dell'eurozona, essendo composto dalle 50 aziende europee con maggior capitalizzazione di mercato. La sua composizione viene rivista annualmente nel mese di settembre. Enel, Eni e Intesa Sanpaolo sono le rappresentanti italiane dell'indice.
Il FTSE Mib è l'indice azionario principale della borsa italiana. Racchiude nel suo paniere le 40 maggiori società per capitalizzazione di mercato quotate a Piazza Affari. La composizione viene rivista trimestralmente.

Il set informativo in ingresso verrà utilizzato per rilevare l'evoluzione della variabile di output, individuata nel valore di chiusura giornaliero degli indici. L'esperimento consiste nell'ottenere il valore di chiusura (c_i) su un orizzonte previsionale di un singolo giorno, inserendo in ingresso i dati relativi a un certo numero t di giorni precedenti. Supponendo di considerare L giorni di trading, la tabella input – output del modello sarà la seguente:

INPUT		OUTPUT
$i_{(0)}, i_{(1)}, i_{(2)}, \dots, i_{(t-1)}, i_{(t)}$	→	$c_{(t+1)}$
$i_{(1)}, i_{(2)}, i_{(3)}, \dots, i_{(t)}, i_{(t+1)}$	→	$c_{(t+2)}$
…		…
$i_{(L-t)}, i_{(L-t+1)}, i_{(L-t+2)}, \dots, i_{(L-2)}, i_{(L-1)}$	→	$c_{(L)}$

Per entrambi gli indici sono stati raccolti dati giornalieri nel periodo che va dal 5 gennaio 2015 fino al 30 aprile 2020 (i dati dei primi 4 anni, dal 05/01/2015 al 30/12/2018, saranno utilizzati per addestrare il modello, mentre i dati rimanenti, dal 02/01/2019 al 30/04/2020, saranno utilizzati in fase di testing). La base dati utilizzata per lo scarico è quella di Investing.com, un portale finanziario globale che fornisce analisi, streaming, quotazioni e grafici, dati tecnici e strumenti riguardanti mercati finanziari globali. Per ogni giorno, si ha la seguente serie di informazioni:

- *Date (d)*: data di riferimento.
- *Close (c)*: valore di chiusura dell'indice nella data corrispondente.
- *Open (o)*: valore di apertura nella data corrispondente.
- *High (h)*: valore più alto assunto dall'indice nella data corrispondente.
- *Low (l)*: valore più basso assunto nella data corrispondente.
- *Vol (v)*: volume tradato nella data corrispondente.

Quindi il dato relativo ad un singolo giorno è in realtà un vettore di *n* elementi. Riprendendo la tabella input – output descritta sopra si ha che ogni dato di ingresso *i* è un vettore di 6 elementi. Ad esempio considerando i seguenti input

$$\bar{i}_0 = (d_0, c_0, o_0, h_0, l_0, v_0)$$
$$\bar{i}_1 = (d_1, c_1, o_1, h_1, l_1, v_1)$$
$$\bar{i}_2 = (d_2, c_2, o_2, h_2, l_2, v_2)$$

significa che c_0 e l_0 sono il valore di chiusura e il valore più basso relativi al giorno *0*; h_1 e v_1 sono valore più alto e volume tradato relativo al giorno *1*; o_2 è il valore di apertura relativo al giorno *2* e così via. Quindi è possibile esplicitare la tabella input – output in questo modo, supponendo ad esempio $t = 3$:

INPUT		OUTPUT
$(d_0, c_0, o_0, h_0, l_0, v_0)$, $(d_1, c_1, o_1, h_1, l_1, v_1)$, $(d_2, c_2, o_2, h_2, l_2, v_2)$	→	$C_{(t+1)}$
$(d_1, c_1, o_1, h_1, l_1, v_1)$, $(d_2, c_2, o_2, h_2, l_2, v_2)$, $(d_3, c_3, o_3, h_3, l_3, v_3)$	→	$C_{(t+2)}$
$(d_2, c_2, o_2, h_2, l_2, v_2)$, $(d_3, c_3, o_3, h_3, l_3, v_3)$, $(d_4, c_4, o_4, h_4, l_4, v_4)$	→	$C_{(L)}$
…		…

Pertanto l'analisi è multivariata single-step. Multivariata perché vi sono in ingresso più variabili, o meglio più serie storiche parallele. Single-step perché l'orizzonte previsionale è di un singolo giorno e l'output è composto da un solo elemento, ovvero il valore di chiusura del giorno.

3.2 NORMALIZZAZIONE

Risultati empirici dimostrano come anche nelle Reti Neurali, così come in diversi altri metodi statistici, il processo di apprendimento può essere migliorato attraverso la normalizzazione dei dati su una scala comune. Infatti abbandonare i valori assoluti per variabili più "indicative" velocizza il processo di apprendimento, che risulta anche migliore, soprattutto quando lo scopo è quelle di prevedere un trand, caso del problema in esame. Esistono molti metodi di normalizzazione. Si può per esempio pensare di ridurre ogni valore compreso tra un valore minimo e un valore massimo, tramite la formula:

$$V_{norm} = L_{min} + (L_{max} - L_{min}) \cdot \frac{V - V_{min}}{V_{max} - V_{min}}$$

Dove V è il valore da normalizzare, L_{min} e L_{max} i limiti che si vuole utilizzare e V_{min} e V_{max} il valore minimo e massimo della serie.
Molto spesso la serie viene normalizzata tra i valori 0 e 1 semplificando la formula come segue:

$$V_{norm} = \frac{V - V_{min}}{V_{max} - V_{min}}$$

Un altro metodo è la normalizzazione statistica, in cui si usa la media e lo scarto quadratico medio della serie:

$$V_{norm} = \frac{V - M(\overline{V})}{\sigma(\overline{V})}$$

Per quanto riguarda questo esperimento si è optato come metodo di normalizzazione il calcolo della variazione percentuale delle variabili reali. Inoltre per cogliere la stagionalità si è tenuto conto di alcune variabili "classificatrici" per cercare di far captare alla rete se determinati trend possono essere associati al periodo in cui vengono rilevati. Pertanto, a partire dalle 6 variabili estratte dal database di Investing.com, sono state derivate le seguenti 9 variabili, che verranno utilizzate per addestrare il modello:

Nome	Range	Descrizione
month	[1,12]	codifica il mese in cui è stata fatta l'osservazione con un numero intero compreso tra 1 e 12 (1 codifica Gennaio, 12 codifica Dicembre)
day_month	[1,31]	codifica il giorno del mese in cui è stata fatta l'osservazione con un intero compreso tra 1 e 31
day_week	[1,5]	codifica il giorno della settimana in cui è stata fatta l'osservazione con un intero compreso tra 1 e 5 (1 lunedì, 2 martedì, … , 5 venerdì)
close_perc	$\mathbb{R}$	variazione percentuale del valore di chiusura tra due giorni consecutivi $close_{perc} = \frac{c_{succ} - c_{prec}}{c_{prec}}$
low_perc	$\mathbb{R}$	variazione percentuale dei valori minimi di due giorni consecutivi $low_{perc} = \frac{l_{succ} - l_{prec}}{l_{prec}}$
high_perc	$\mathbb{R}$	variazione percentuale dei valori massimi di due giorni consecutivi $high_{perc} = \frac{h_{succ} - h_{prec}}{h_{prec}}$
open_perc	$\mathbb{R}$	variazione percentuale del valore di apertura tra due giorni consecutivi $open_{perc} = \frac{o_{succ} - o_{prec}}{o_{prec}}$
vol_perc	$\mathbb{R}$	variazione percentuale del volume tradato tra due giorni consecutivi $vol_{perc} = \frac{v_{succ} - v_{prec}}{v_{prec}}$
fluctuation	$0 \vee 1$	Indica se vi è stato un aumento o un peggioramento nel valore di chiusura tra due giorni consecutivi $fluctuation = \begin{cases} 1 & se \quad close_{perc} \geq 0 \\ 0 & se \quad close_{perc} < 0 \end{cases}$

Quindi l'ulteriore passaggio di preparazione dei dati è quello di calcolare queste 9 variabili per ogni giorno di osservazione. Tuttavia in fase di testing sarà necessario capire quali di queste variabili sono davvero rilevanti, cercando di lasciare tutte quelle necessarie all'apprendimento non lineare della rete e scartando quelle irrilevanti o che addirittura possono condizionare negativamente l'apprendimento. Questo può essere ottenuto soltanto per raffinazioni successive, utilizzando un algoritmo di Feature Selection, testando quindi più volte il modello con diverse combinazioni di variabili al fine di individuare quelle che producono indicatori migliori nelle performance. Comunque questo problema sarà affrontato più dettagliatamente nel capitolo relativo alla costruzione della rete.

3.3 TRASFORMARE UNA SERIE STORICA IN UN PROBLEMA DI APPRENDIMENTO SUPERVISIONATO

Come spiegato nei capitoli introduttivi una rete neurale si adatta molto bene a risolvere problemi appartenenti a diverse categorie (simulatori di funzioni matematiche, classificatori, memorie associative, ecc.). Quale che sia il problema che si vuole affrontare l'apprendimento supervisionato presuppone di voler mappare una funzione

$$Y = f(X)$$

Per farlo bisogna partire da un certo dataset contenente una serie di variabili di input con le corrispondenti variabili di output, che vengono date in pasto alla rete che le utilizzerà per modificare opportunamente weight e bias. Il problema delle serie storiche è che non sono in forma di funzione. Infatti non si ha altro che un insieme di dati, la cui caratteristica è quella di precedere un certo valore e seguirne un altro.

valore	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
tempo	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9

Quindi ad esempio il valore x_4 della serie storica ha la proprietà di succedere al valore x_3 e precedere il valore x_5.
Per trasformare la serie storica in una funzione input – output si può utilizzare il metodo Sliding Window[6], che consente di riorganizzare il dataset di partenza al fine di renderlo idoneo all'addestramento supervisionato. Segue un esempio considerando una serie generica:

valore	*10*	*20*	*30*	*40*	*50*	*60*	*70*	*80*	*90*
tempo	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9

Lo Sliding Window consente di mappare la serie storica in una funzione input – output mantenendo l'interdipendenza temporale delle variabili. In primis bisogna decidere il lag temporale, ovvero la numerosità del campione di input e dell'output. Se si suppone di optare per due valori in input e un valore in output, lo Sliding Window avrà numerosità 3. Per comporre la funzione bisogna porre la finestra all'inizio della serie e scrivere il primo frame, ovvero il primo campione:

	1								
valore	*10*	*20*	*30*	*40*	*50*	*60*	*70*	*80*	*90*
tempo	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9

x_1	x_2	y
10	*20*	*30*

Alla seconda iterazione si fa scorrere la finestra di un istante temporale per ottenere il secondo campione; dopodiché bisogna far scorrere di un altro istante temporale per il terzo campione e così via fino ad aver scansionato l'intera serie storica:

		2							
valore	*10*	*20*	*30*	*40*	*50*	*60*	*70*	*80*	*90*
tempo	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9

x_1	x_2	y
20	*30*	*40*

			3						
valore	*10*	*20*	*30*	*40*	*50*	*60*	*70*	*80*	*90*
tempo	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9

x_1	x_2	y
30	*40*	*50*

...

Alla fine il risultato sarà quello di avere la funzione input – output che rappresenta la serie storica.

N° Campione	INPUT		OUTPUT
	x1	x2	y
1	10	20	30
2	20	30	40
3	30	40	50
4	40	50	60
5	50	60	70
6	60	70	80
7	70	80	90

3.4 IMPLEMENTAZIONE DELLA PREPARAZIONE DEL DATASET

Nel prossimo paragrafo sarà spiegato nel dettaglio come viene realizzato nella pratica quanto descritto precedentemente, utilizzando il software Excel per la preparazione iniziale e la normalizzazione dei dati e Python per la trasformazione della serie storica in una funzione input – output.

3.4.1 Scarico dei dati

Il primo passo è quello di ottenere i dati storici relativi ai due indici. Si utilizzerà come esempio l'Euro Stoxx, mentre per il FTSE Mib bisogna seguire gli stessi passaggi. Ci si deve collegare dunque a www.investing.com e, sfruttando la casella di ricerca, si deve cercare e selezionare "Euro Stoxx 50". Si aprirà la scheda principale dell'indice, con il grafico dell'andamento. Dal menù superiore della scheda bisogna cliccare su "Dati Storici", la sezione dove è possibile scaricare quello che interessa. Nella schermata che si apre si può utilizzare il form per definire il tipo di frame desiderato (giornaliero nel nostro caso) e l'intervallo di interesse (dal 05/01/2015 al 30/04/2020), dopodiché bisogna cliccare su "Scarica dati".

I dati sono scaricati in formato .csv e contengono i 6 valori di partenza di cui si è parlato nel primo paragrafo del capitolo: data, valore di chiusura, valore di apertura, valore più basso, valore più alto, volume tradato. Questo è il file grezzo tratto dallo scarico:

	A	B	C	D	E	F
1	Date	Close	Open	High	Low	Vol
2	05/01/15	3023.14	3126.81	3150.09	3020.79	77.46M
3	06/01/15	3007.91	3033.44	3061.19	2998.53	67.02M
4	07/01/15	3026.79	3019.47	3055.87	3002.78	67.36M
5	08/01/15	3135.08	3051.26	3139.92	3051.26	76.46M
6	09/01/15	3042.90	3128.42	3128.42	3021.23	123.51M
7	12/01/15	3084.18	3055.66	3096.91	3033.34	72.87M
8	13/01/15	3133.86	3069.82	3146.54	3065.37	70.76M
9	14/01/15	3089.67	3108.34	3148.26	3077.24	86.60M
10	15/01/15	3157.36	3112.37	3169.19	3020.54	99.06M
11	16/01/15	...				
12	19/01/15	...				
13	...	...				

3.4.2 Normalizzazione dei dati in Excel

A partire da queste variabili bisogna ricavare le 9 che verranno effettivamente usate nel modello (calcolate come nella tabella pag. 31). Una possibilità sarebbe stata quella di elaborare questo file per ottenere le variabili richieste direttamente in Python. Tuttavia siccome è un'operazione standard (effettivamente le variabili grezze così come scaricate non verranno utilizzate dal programma) si è preferito modificare direttamente il file creandone uno contenente già le variabili da dare in pasto al modello in modo tale da non appesantire ulteriormente il carico di elaborazione ogni volta che si lancia il programma. Pertanto si normalizzerà il dataset sfruttando delle funzioni messe a disposizione da Excel.

Con la colonna volume si presenta già un primo problema: infatti si nota che non è presente soltanto il valore del volume giornaliero tradato, ma anche l'unità di misura. Il problema è che in questo modo il software non riconosce la variabile come numero, ma come stringa. Pertanto si deve eliminare da tutte le celle della colonna l'ultima lettera sfruttando le funzioni di Excel `SINISTRA` e `LUNGHEZZA`:

	F	G
1	Vol	Vol modificato
2	77.46M	=SINISTRA(F2; LUNGHEZZA(F2)-1)
3	67.02M	=SINISTRA(F3; LUNGHEZZA(F3)-1)
4	…	…

$\Rightarrow$

F	G
Vol.	Vol. mod.
77.46M	77.46
67.02M	67.02
…	…

Ora anche il volume è trasformato in formato di numero reale. È possibile quindi calcolare facilmente tutte le variazioni percentuali di interesse:

	A	B	C	D	E	F
1	Date	Close	Open	High	Low	Vol
2	05/01/15	3023.14	3126.81	3150.09	3020.79	77.46
3	06/01/15	3007.91	3033.44	3061.19	2998.53	67.02
4	07/01/15	3026.79	3019.47	3055.87	3002.78	67.36
5	08/01/15	…				
6	…	…				

$\Downarrow$

	A	H	I	J	K	L
1	Date	$Close_{perc}$	$Open_{perc}$	$High_{perc}$	Low_{perc}	Vol_{perc}
2	05/01/15	-	-	-	-	-
3	06/01/15	=(B3-B2)/B2	=(C3-C2)/C2	=(D3-D2)/D2	=(E3-E2)/E2	=(F3-F2)/F2
4	07/01/15	=(B4-B3)/B3	=(C4-C3)/C3	=(D4-D3)/D3	=(E4-E3)/E3	=(F4-F3)/F3
5	08/01/15	…				
6	…	…				

⇓

	A	H	I	J	K	L
1	Date	$Close_{perc}$	$Open_{perc}$	$High_{perc}$	Low_{perc}	Vol_{perc}
2	05/01/15	-	-	-	-	-
3	06/01/15	-0.005038	-0.029861	-0.028221	-0.007369	-0.134779
4	07/01/15	0.006277	-0.004605	-0.001738	0.001417	0.005073
5	08/01/15	…				
6	…	…				

Come si evince dalla tabella si perde il primo valore della serie in quanto manca l'indicazione del giorno precedente necessaria per calcolare la variazione percentuale. Il prossimo passo è calcolare le variabili relative la stagionalità. Anche in questo vengono in aiuto alcune funzioni built-in di Excel:

- `MESE(date)`: prende in ingresso una data e restituisce un intero compreso tra 1 e 12 corrispondente al mese. Ad es. `MESE(06/03/2017)` restituisce il valore 3;
- `GIORNO(date)`: al pari della funzione precedente prende in ingresso una data, ma in questo caso restituisce un intero compreso tra 1 e 31 che corrisponde al giorno del mese. Quindi `GIORNO(06/03/2017)` restituisce il valore 6;
- `GIORNO.SETTIMANA(date)`: prende in ingresso una data e restituisce un intero compreso tra 1 e 7 che rappresenta il giorno della settimana (1 per lunedì, 7 per domenica).

	A	B	C	D
1	Date	month	day_month	day_week
2	05/01/15	=MESE(A2)	=GIORNO(A2)	=GIORNO.SETTIMANA(A2)
3	06/01/15	=MESE(A2)	=GIORNO(A2)	=GIORNO.SETTIMANA(A2)
4	07/01/15	…		
5	…	…		

⟹

B	C	D
month	day_month	day_week
1	5	1
1	6	2
…		
…		

Infine bisogna calcolare la colonna della variabile fluctuation, sulla base del valore assunto da $close_{perc}$, che assume valore pari a 0 se $close_{perc}$ è negativo, 1 altrimenti. Per implementare questo comportamento si sfrutta la funzione SE di Excel, che verifica una condizione e in base al soddisfacimento o meno restituisce i valori specificati, in questo caso 0 o 1.

	F	G	M
1	Date	$Close_{perc}$	Fluctuation
2	06/01/15	-0.005038	=SE(G2<0; 0; 1)
3	07/01/15	0.006277	=SE(G3<0; 0; 1)
4	…	…	…

$\Rightarrow$

G	M
$Close_{perc}$	Fluctuation
-0.005038	0
0.006277	1
…	…

La trasformazione preliminare dei dati a questo punto è completa. Si conclude salvando il file in formato .csv per ottenere il file di partenza che sarà dato in ingresso al software della Rete Neurale.

	A	B	C	D	E	F	G	H	I	L
1	date	month	day_month	day_week	close_perc	open_perc	high_perc	low_perc	vol_perc	f
2	06/01/15	1	6	2	-0.005038	-0.029861	-0.028221	-0.007369	-0.134779	0
3	07/01/15	1	7	3	0.006277	-0.004605	-0.001738	0.001417	0.005073	1
4	08/01/15	1	8	4	0.035777	0.010528	0.027504	0.016145	0.135095	1
5	09/01/15	1	9	5	-0.029403	0.025288	-0.003663	-0.009842	0.615354	0
6	12/01/15	1	12	1	0.013566	-0.023258	-0.010072	0.004008	-0.410007	1
7	13/01/15	1	13	2	0.016108	0.004634	0.016026	0.010559	-0.028956	1
8	14/01/15	1	14	3	-0.014101	0.012548	0.000547	0.003872	0.223855	0
9	15/01/15	1	15	4	0.021908	0.001297	0.006648	-0.018426	0.143880	1
10	16/01/15	1	16	5	0.014214	0.011297	0.012341	0.037530	-0.162023	1
11	19/01/15	1	19	1	0.005827	0.019409	0.009276	0.020569	-0.301169	1
12	20/01/15	…								
13	…	…								

3.4.3 Algoritmo Sliding Window

Nei file prodotti seguendo le istruzioni precedenti si ha, per l'Euro Stoxx e per il FTSE Mib, l'elenco dei dati che serviranno per addestrare e per testare il modello, ovvero la serie storica, o meglio, le varie serie storiche parallele. Infatti ogni colonna del file non rappresenta altro che una serie storica parallela alle altre. Quindi per ogni istante di tempo vi sono *n* valori, ovvero un vettore. In uscita come detto ci si accontenta invece di un singolo valore, che è il valore di chiusura. Quindi è possibile esplicitare meglio il modo in cui deve essere realizzato l'algoritmo di Sliding Window per realizzare l'applicativo. L'obiettivo è che il framework sia in grado di prevedere il

valore di chiusura di un giorno prendendo in ingresso i dati di *t* giorni precedenti. È possibile realizzare graficamente le varie fasi dell'algoritmo supponendo *t = 2*. La finestra dello Sliding sarà di numerosità pari a 3, due vettori di input (ognuno dei quali contenente tutte le variabili nell'istante di tempo corrispondente) e un singolo valore di output, corrispondente al valore di chiusura dell'istante successivo:

month	day_month	day_week	close_perc	open_perc	high_perc	low_perc	vol_perc	f		
1	6	2	-0.005038	-0.029861	-0.028221	-0.007369	-0.134779	0	X_1	
1	7	3	0.006277	-0.004605	-0.001738	0.001417	0.005073	1	X_2	*1*
1	8	4	0.035777	0.010528	0.027504	0.016145	0.135095	1	y	
1	9	5	-0.029403	0.025288	-0.003663	-0.009842	0.615354	0		
1	12	1	0.013566	-0.023258	-0.010072	0.004008	-0.410007	1		
1	13	2	0.016108	0.004634	0.016026	0.010559	-0.028956	1		
				...						

⇓

month	day_month	day_week	close_perc	open_perc	high_perc	low_perc	vol_perc	f		
1	6	2	-0.005038	-0.029861	-0.028221	-0.007369	-0.134779	0		
1	7	3	0.006277	-0.004605	-0.001738	0.001417	0.005073	1	X_1	
1	8	4	0.035777	0.010528	0.027504	0.016145	0.135095	1	X_2	*2*
1	9	5	-0.029403	0.025288	-0.003663	-0.009842	0.615354	0	y	
1	12	1	0.013566	-0.023258	-0.010072	0.004008	-0.410007	1		
1	13	2	0.016108	0.004634	0.016026	0.010559	-0.028956	1		
				...						

⇓

month	day_month	day_week	close_perc	open_perc	high_perc	low_perc	vol_perc	f		
1	6	2	-0.005038	-0.029861	-0.028221	-0.007369	-0.134779	0		
1	7	3	0.006277	-0.004605	-0.001738	0.001417	0.005073	1		
1	8	4	0.035777	0.010528	0.027504	0.016145	0.135095	1	X_1	
1	9	5	-0.029403	0.025288	-0.003663	-0.009842	0.615354	0	X_2	*3*
1	12	1	0.013566	-0.023258	-0.010072	0.004008	-0.410007	1	y	
1	13	2	0.016108	0.004634	0.016026	0.010559	-0.028956	1		
				...						

Pertanto la funzione input – output sarà formata come segue:

n°	INPUT		OUTPUT
	x_1	x_2	y
1	(1, 6, 2, -0.005038, -0.029861, -0.028221, -0.007369, -0.134779, 0)	(1, 7, 3, 0.006277, -0.004605, -0.001738, 0.001417, 0.005073, 1)	0.035777
2	(1, 7, 3, 0.006277, -0.004605, -0.001738, 0.001417, 0.005073, 1)	(1, 8, 4, 0.035777, 0.010528, 0.027504, 0.016145, 0.135095, 1)	-0.029403
3	(1, 8, 4, 0.035777, 0.010528, 0.027504, 0.016145, 0.135095, 1)	(1, 9, 5, -0.029403, 0.025288, -0.003663, -0.009842, 0.615354, 0)	0.013566
4	...		

L' implementazione vera e propria di questo comportamento è realizzata in codice Python. Per prima cosa si deve importare il file contenente i dati preparato in precedenza. Per farlo si sfrutta la funzione `read_csv` della libreria Pandas. Inoltre i dati devono essere inseriti in una struttura dati che possa essere elaborata dalle strutture di controllo del linguaggio di programmazione, e in questo caso si può optare di utilizzare i metodi della struttura `array` contenuti nella libreria Numpy di Python.

```
# Caricamento del file nella variabile series e trasformazione in array
nomeFileCSV = "dataset/eurostoxx50_2015-2019.csv"
series = read_csv(nomeFileCSV, sep=';')
data = array(series.values)
```

Prima di dedicarsi alla funzione per il campionamento della serie e la trasformazione bisogna dividere i dati in due insiemi. I primi 1023 giorni (dal 06/01/2015 al 31/12/2018) faranno parte del training set, quindi saranno splittati e verranno dati in pasto al modello per tarare i parametri. I restanti 244 giorni (dal 02/01/2010 al 30/04/2020) saranno usati in fase di testing, il che vuol dire che verranno sottoposti al modello solo gli input, che non ha mai analizzato. Dopodiché i risultati "predetti" saranno confrontati con i valori reali, per valutare le performance della rete. Per la suddivisione in training e test set si può utilizzare lo slicing, una funzione built-in degli array di Python.

```
# estrae i campioni dall'inizio della serie fino al numero 1023
training_set = data[:1023]
# estrae i campioni dal numero 1024 fino alla fine della serie
test_set = data[1023:]
```

Per lo sliding window vero e proprio si deve costruire una funzione apposita che prende in ingresso la sequenza e il numero di step (quello che nell'esempio grafico veniva definito dalla variabile t), ovvero il numero di istanti di input utilizzati per predire il valore successivo.

```
def sliding_window(sequence, step):

   # X e y diventeranno due vettori con l'ingresso e l'output corrispondente
   X, y = list(), list()

   for i in range(len(sequence)):

   # i rappresenta l'iterazione i-esima; end_ix è quindi il numero di campioni
   # che compone l'input
      end_ix = i + step

      # questo if serve per controllare quando si arriva alla fine della
      # sequenza
      if end_ix > len(sequences)-1:
         break

      # si sfrutta ancora lo slicing degli array per tagliare solo la parte
      # della serie che ci interessa
      seq_x, seq_y = sequences[i:end_ix, 2:11], sequences[end_ix, 5]

      # seq_x e seq_y contengono l'input e l'output i-esimo. Si utilizza il
      # metodo append per aggiungerli in lista
      X.append(seq_x)
      y.append(seq_y)

   # la funzione restituirà l'insieme di input e l'insieme di output,
   # convertiti in array
   return array(X), array(y)
```

Infine non bisogna far altro che chiamare nel main del programma la funzione sul training set e sul test set, specificando il numero di step desiderati come input (l'output come detto in precedenza è sempre tarato sull'unità).
Segue l'esempio completo di questa prima parte di codice, utilizzata al fine di ottenere il training set suddiviso in sequenze di input e sequenze di output e il test set, suddiviso nella stessa maniera.

```
from pandas import read_csv
from numpy import array

def sliding_window(sequence, step):
   X, y = list(), list()
   for i in range(len(sequence)):
      end_ix = i + step
      if end_ix > len(sequences)-1:
           break
      seq_x, seq_y = sequences[i:end_ix, 2:11], sequences[end_ix, 5]
      X.append(seq_x)
      y.append(seq_y)
   return array(X), array(y)

nomeFileCSV = "dataset/eurostoxx50_2015-2019.csv"
series = read_csv(nomeFileCSV, sep=';')
data = array(series.values)

training_set = data[:1023]
test_set = data[1023:]

# split in training e in test set supponendo un numero di step pari a 10
X_train, y_train = sliding_window(training_set, 10)
X_test, y_test = sliding_window(test_set, 10)
```

Capitolo IV

IMPLEMENTAZIONE DELLA RETE

Nel capitolo precedente è stato trattato il Dataset di partenza, mettendolo in una forma idonea ad essere utilizzato dal modello che ci apprestiamo a realizzare. In questo capitolo invece si approfondirà la costruzione vera e propria del modello, dalla scelta del tipo di struttura alla definizione delle metriche di valutazione delle performance, e di come tutto questo possa essere implementato in codice Python. Infine si concluderà costruendo una struttura di controllo ciclica che permetta di testare diverse configurazioni del modello prescelto in un unico avvio del software, migliorando così l'efficienza e le tempistiche richieste in fase di testing.

4.1 RETI NERUALI NELL'AMBITO DELLA PREVISIONE DELLE SERIE STORICHE

La previsione delle serie storiche, in particolare quelle finanziarie, è un problema particolarmente impegnativo rispetto ad altre tipologie di problemi di classificazione e regressione in quanto, oltre alle difficoltà intrinseche della previsione, si aggiunge la complessità della dipendenza inter-temporale tra le variabili. Come detto nel I capitolo, il problema della previsione delle serie storiche era limitato a metodi essenzialmente lineari o linearizzabili, come ad esempio l'ARIMA. Ma i metodi classici hanno delle limitazioni:

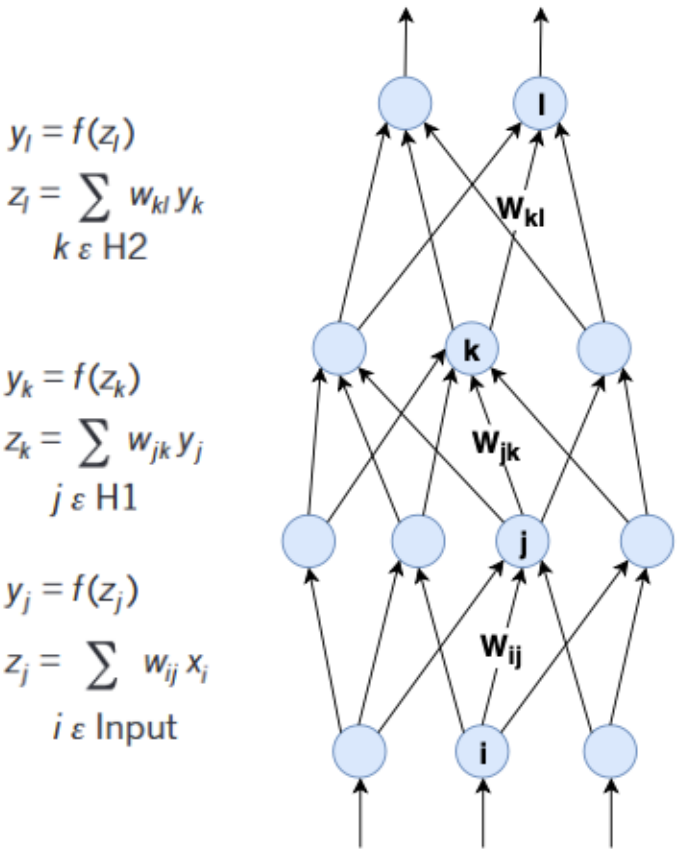
- Non supportano la mancanza di dati all'interno della serie, e nel caso è quindi necessario utilizzare metodi di interpolazione per ovviare al problema;
- Difficilmente riescono a captare le relazioni non lineari;
- Supportano modelli univariati (mentre la maggior parte delle analisi sono multivariate);
- Sono spesso limitate ad un orizzonte previsionale single-step.

Le Reti Neurali Profonde, grazie alle loro peculiarità e caratteristiche, riescono meglio a sopperire a queste mancanze che affliggono i metodi classici, sfruttando le loro superiori capacità nel mappare funzioni input – output, anche se queste sono altamente non lineari. Delle varie tipologie di Reti ce ne sono alcune che meglio di altre riescono ad affrontare le caratteristiche del problema relativo alla previsione delle serie storiche finanziarie: Multilayer Perceptron, Reti Neurali Ricorrenti, Reti

Neurali Convoluzionali. Verranno analizzate nel dettaglio pro e contro di queste strutture per scegliere quella più opportuna da utilizzare nell'esperimento[7].

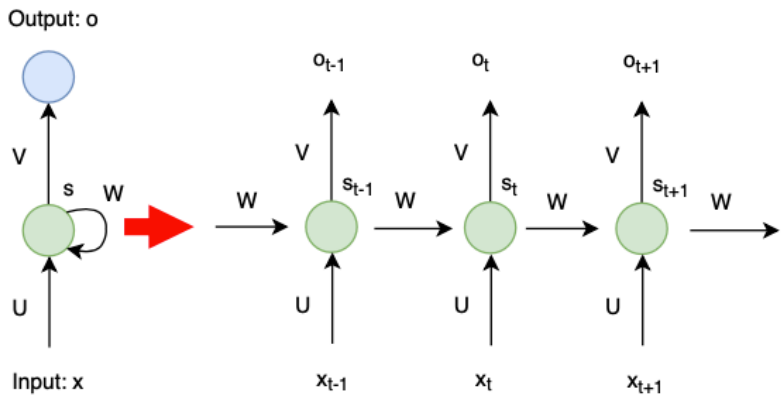
4.1.1 Multilayer Perceptron

Il Multilayer Perceptron (in acronimo MLP) è un modello Feedforward Fully Connected. Feedforward perché i dati viaggiano lungo la rete in un solo verso, quindi senza mai essere retro-propagati; Fully Connected perché ogni percettrone di uno strato precedente è connesso a tutti i percettroni dello strato successivo. È una rete in grado di mappare in maniera molto precisa e accurata una funzione di input – output, anche non lineare, anche se gli input e gli output sono multivariati e supporta un orizzonte di previsione multi-step. Nonostante la grande efficacia nella fase di mapping delle funzioni le MLP soffrono di alcune limitazioni per quanto riguarda i problemi relativi alle serie storiche, in quanto non hanno una struttura idonea per cogliere le dipendenze intertemporali che le caratterizzano, il che le rende non particolarmente efficaci in questo tipo di contesto.



4.1.2 Reti Neurali Ricorrenti

Le Reti Neurali Ricorrenti (in acronimo RNN) sono un tipo di Rete Profonda non Feedforward, (quindi c'è retro-propagazione di alcuni segnali). È il tipo di Rete più utilizzata in letteratura per trattare i dati sequenziali come le serie storiche (ma anche file multimediali e i data-mining). La caratteristica di questa tipologia di rete, che la rende adatta a trattare le sequenze, è l'immagazzinamento in uno degli hidden layer di una variabile di stato, in modo da tenere conto della storia dei dati passati durante il mapping della funzione. Nella figura seguente vi è uno schema base di una RNN in differenti istanti di tempo:



Come si evince dalla figura, in ogni istante di tempo l'uscita dell'hidden layer è dipendente dall'uscita dell'hidden layer dell'istante precedente (che viene appunto retro-propagata) secondo la funzione:

$$h^{(i)} = \sigma(Ux^{(i)} + Wh^{(i-1)} + c_1^{(i)})$$

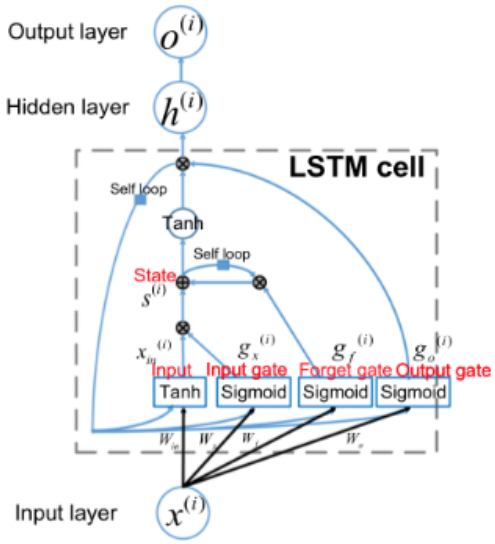
Dove $\sigma()$ è la funzione di attivazione, $x^{(i)}$ è l'ingresso corrente, U e W sono rispettivamente i weight dell'ingresso e dell'hidden layer, $h^{(i-1)}$ è l'uscita dell'hidden layer dell'istante precedente, $c_1^{(i)}$ è il bias. L'output della Rete è invece la seguente:

$$o^{(i)} = \sigma(Vh^{(i)} + c_2^{(i)})$$

Con $\sigma()$ funzione di attivazione, V il peso dell'output layer, $h^{(i)}$ l'uscita dell'hidden layer così come l'abbiamo calcolata sopra, $c_2^{(i)}$ è il bias.

La tipologia più diffusa di RNN è la Long Short Term Memory (LSTM). Come il nome suggerisce la struttura più complessa di questa Rete Profonda consente di tenere traccia durante l'addestramento sia di dati più recenti (Short Term) che di dati

collocati su istanti temporali più lontani (Long Term). La differenza sostanziale è il modo più complesso con cui viene calcolata l'uscita dell'hidden layer. L'hidden layer è infatti composto da strutture chiamate celle, in cui vengono retro-propagati in appositi percettroni il segnale di input e di output oltre che il segnale di forget, ovvero quello che misura il peso degli stati più remoti. Questi percettroni interni alla cella prendono il nome di gate che, tramite la funzione sigmoide, estraggono il valore dell'ingresso corrente rispetto al passato della serie. Infine la funzione tangente iperbolica restituisce l' $h^{(i)}$ definitivo che sarà dato in ingresso al percettrone a valle per calcolare l'output. Segue lo schema di una cella elementare della struttura LSTM:



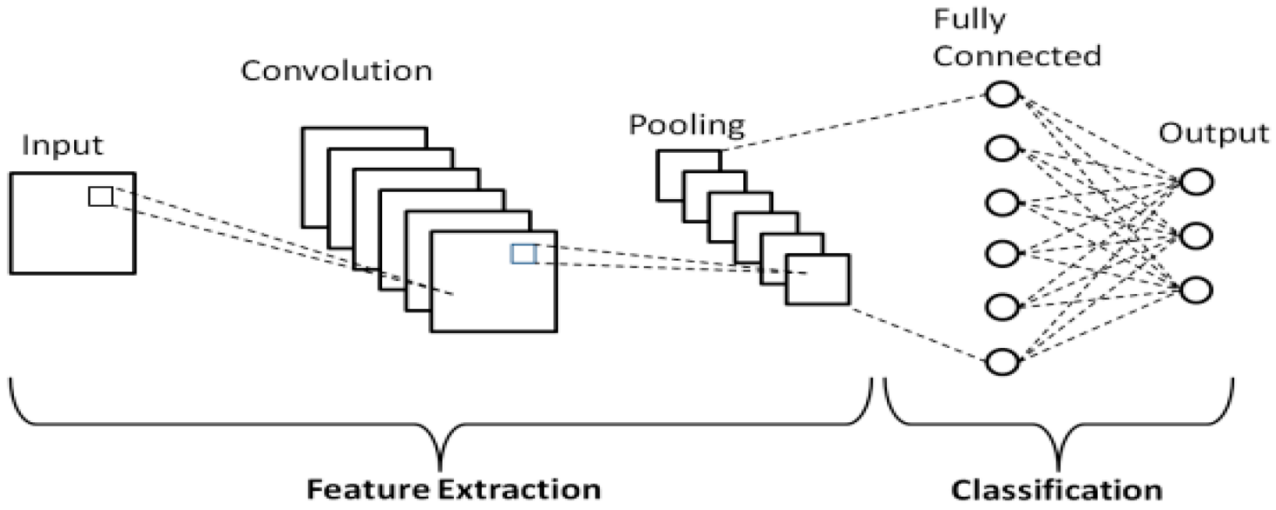
4.1.3 Reti Neurali Convoluzionali

Le Reti Neurali Convoluzionali sono una classe di reti Feedforward, nate con lo scopo di affrontare problemi di Computer Vision e pre-processamento delle immagini, ma che negli ultimi anni stanno trovando applicazione in molti altri campi, con ottimi risultati. Si basano sul prodotto di convoluzione, che matematicamente è così definito:

$$f(t) * g(t) = \int_{-\infty}^{+\infty} f(\tau)\, g(t-\tau)\, d\tau$$

In parole semplici eseguire il prodotto di convoluzione significa far scorrere una funzione sopra un'altra facendone l'integrale. La prima funzione può rappresentare un segnale che si vuole analizzare, per esempio un'immagine o, nel nostro caso, la serie storica. Sopra gli si fa scorrere una seconda funzione, che prende il nome di filtro, che ha la funzione di identificare particolari strutture o pattern all'interno del

primo segnale. Quindi la funzione di questa rete è appunto quello di riconoscere dei pattern ricorrenti all'interno del segnale d'ingresso, ed è basandosi su ciò che è possibile sfruttarla nella previsione delle serie storiche.



In figura è presentato lo schema di una CNN semplice. Il primo layer è quello di convoluzione, che si occupa di elaborare una porzione limitata dell'input (che nell'esempio è un'immagine bi-dimensionale, ma lo si può adattare ad una serie storica immaginandola come un'immagine mono-dimensionale). Il filtro di convoluzione scansiona l'intera immagine (con un algoritmo simile allo Sliding Window utilizzato per la creazione del training set visto nel capitolo precedente), e le singole porzioni elaborate vengono poi date in ingresso ad un layer di Pooling, che si occupa di aggregare l'input tramite un sotto-campionamento. Il Pooling, così come il layer Convoluzionale, agisce tramite un kernel che scansiona le sotto-porzioni dell'ingresso originale, riducendolo tramite l'operazione di media (Average Pooling) o estraendo i valori massimi (Max Pooling). Infine, a valle della rete, si trova un layer Fully Connected che sintetizza le elaborazioni dei singoli filtri in un risultato finale.

In questo paragrafo sono state prese in rassegna le strutture più interessanti che è possibile usare nell'affrontare il problema posto. Tutte sono state utilizzate in diversi lavori di ricerca con risultati molto validi. La struttura più utilizzata è la LSTM, essendo tarata proprio per affrontare problemi aventi sequenze come input; tuttavia in tempi recenti (precisamente negli ultimi 3 anni) molti lavori di ricerca si sono dedicati all'adattamento delle CNN per risolvere problemi di previsione delle serie storiche sfruttando la loro caratteristica nel riscontrare pattern ricorrenti all'interno dei segnali di input, con risultati in molti casi migliori delle LSTM. Pertanto in questo lavoro si è deciso di seguire questa strada più innovativa.

4.2 COSTRUZIONE DELLA RETE IN PYTHON

Per realizzare modelli di Deep Learning in Python sono state utilizzate le funzionalità di Keras[8]. Keras è una libreria open-source ed è la più gettonata al momento nella risoluzione dei problemi di apprendimento supervisionato e reti neurali. Supporta tutte le tecnologie back-end più diffuse, come Tensorflow e Theano. Per l'esperimento si è optato per utilizzare quella di default, Tensorflow.

Come spiegato nel paragrafo precedente le Reti Neurali Convoluzionali nascono per l'elaborazione delle immagini o, più in generale, di file bi-dimensionali. Per sfruttarla su una serie storica bisogna modificare l'hidden layer convoluzionale affinché possa lavorare su un input mono-dimensionale. In realtà più che altro è sulla modifica del dato che si deve lavorare, in quanto Keras offre già due strutture diversificate, una per lavorare su input bi-dimensionali e l'altra per gli input mono-dimensionali. Quello che invece è necessario operare sul dato è una modifica della dimensionalità dell'input. Infatti, oltre al numero di campioni e alla loro lunghezza (numero di time-step), che per costruzione sono informazioni già incorporate nel dato (lunghezza e larghezza della sequenza di input), bisogna incorporare un'ulteriore informazione sul numero di caratteristiche, ovvero sul numero di variabili di input, essendo questa un'analisi multivariata (vedi cap. III). Questo perché è necessario inviare come parametro al layer convoluzionale il numero di caratteristiche.

Per creare una Rete Neurale Profonda si utilizza il metodo `Sequential()`, che è il costruttore di un'istanza della Rete Neurale. Dopodiché con il metodo `add` della classe `Sequential` è possibile aggiungere nuovi layer. I layer che verranno utilizzati appartengono alle seguenti classi:

- `Conv1D`: è la classe del layer convoluzionale per strutture mono-dimensionali. Gli attributi da specificare sono `filters` (numero di percettroni dello strato), `kernel_size` (intero che specifica la dimensione del kernel), `activation` (funzione di attivazione dei percettroni del layer), `input_shape` (parametro importante in quanto specifica il numero di time step del campione e il numero di caratteristiche).
- `MaxPooling1D`: la classe del layer di Pooling che estrae i valori massimi delle caratteristiche per ridurre la quantità di informazioni da elaborare e inviare al layer successivo. In questo caso è necessario specificare la dimensione del kernel del layer tramite il parametro `pool_size`.

- `Dense`: è la classe che realizza un classico Fully Connected Layer. I parametri da specificare sono il numero di percettroni che compongono lo strato, e la funzione di attivazione, rispettivamente tramite i parametri `filters` e `activation`.
- `Flatten`: è un layer utilizzato tra lo strato convoluzionale e lo strato Fully-Connected che riduce le caratteristiche in uscita ad un vettore mono-dimensionale.

Il modello che si andrà ad implementare è composto da due strati convoluzionali seguiti da uno strato di pooling, seguito da un ulteriore strato convoluzionale, seguito da un ulteriore strato di pooling; a valle della rete, inseriamo lo strato di flatten per la riduzione delle caratteristiche, prima di due strati Fully Connected che produrranno il risultato finale. Le righe di codice per implementare la struttura sono le seguenti:

```
model = Sequential()
model.add(Conv1D(filters=num_filters_1, kernel_size=dim_kernel_1,
                         activation='relu', input_shape=(n_steps, n_features)))
model.add(Conv1D(filters=num_filters_2, kernel_size=dim_kernel_2,
                                                        activation='relu'))
model.add(MaxPooling1D(pool_size=dim_pool_1))
model.add(Conv1D(filters=num_filters_3, kernel_size=dim_kernel_3,
                                                        activation='relu'))
model.add(MaxPooling1D(dim_pool_2))
model.add(Flatten())
model.add(Dense(num_filter_dense_1, activation='relu'))
model.add(Dense(num_filter_dense_2))
```

`num_filters_1`, `dim_kernel_1`, `num_filters_2`, `dim_kernel_2`, `dim_pool_1`, `num_filters_3`, `dim_kernel_3`, `dim_pool_2`, `num_filter_dense_1`, `num_filter_dense_2` sono variabili che contengono valori interi. Non sono specificati i valori perché la determinazione degli iper-parametri migliori per il modello è un aspetto importante che verrà discusso nei prossimi paragrafi.

Una volta costruito il modello, su di esso si deve invocare il metodo `compile`, i cui parametri `optimizer` e `loss` permettono di specificare l'algoritmo che si vuole applicare per ridurre la perdita ad ogni iterazione e la loss function che si vuole utilizzare per la riduzione.

```
# In questo esempio è stato scelto l'algoritmo di Discesa del Gradiente
# (optimizer='SGD') come ottimizzatore e
# l'MSE come funzione di perdita (loss='mse')

model.compile(optimizer='SGD', loss='mse')
```

Per concludere si addestra il modello utilizzando il metodo `fit`, che prende come parametri il traning set (diviso per le componenti di ingresso `X` e le componenti corrispondenti di output `y`), e altri due parametri che sono il numero di epoche (tramite il parametro `epochs`) e la dimensione del batch (tramite il parametro `batch_size`). Si ricorda che nella fase di addestramento il numero di epoche indica quante volte deve essere applicato l'intero training set al modello, mentre il batch rappresenta il numero di campioni che devono essere elaborati prima di aggiornare i pesi dei percettroni.

```
# X_train e y_train sono due vettori di uguale dimensione che contengono i
# campioni di ingresso e i rispettivi output.
# Nel seguente esempio ci sono anche due variabili che contengono interi
# rappresentanti il numero di epoche che si vuole utilizzare e la dimensione
# del batch (rispettivamente num_epochs e num_batch).

model.fit(X_train, y_train, epochs=num_epochs, batch_size=num_batch)
```

Le variabili `X_train` e `y_train` rappresentano rispettivamente l'input e l'output del training set. Della creazione (traimte l'implementazione della funzione `sliding_window`) è stato discusso in dettaglio nel capitolo III, a cui si rimanda per approfondimenti.

4.3 DEFINIZIONE DELLE METRICHE DI VALUTAZIONE

Scelto il modello con cui si vuole affrontare il problema, un passo altrettanto importante è definire il modo in cui il modello deve essere valutato. Questo è un problema particolarmente importante nell'ambito del Deep Learning, in quanto non vi sono a priori teoremi e metodologie per scegliere la struttura e gli iper-parametri più efficienti ed efficaci. Infatti, essendo le Reti Neurali assimilabili a Black Box, non si ha modo di risalire agli intricatissimi calcoli interni che portano alla tara dei weight e dei bias. Pertanto si procede per tentativi, testando il modello con diverse strutture e iper-parametri e valutando le performance, scegliendo poi la migliore. Ed ecco il motivo per cui la definizione delle metriche di valutazione assume un ruolo molto importante. Il modello verrà testato in due approcci differenti: regressione e classificazione.

Approcciando con una regressione significa tentare di predire il valore esatto del closing value giornaliero dell'indice. La metrica per valutare il modello sarà l'RMSE, definito come la radice dell'errore quadratico medio, a sua volta definito come il quadrato della differenza tra il valore stimato e il valore obbiettivo:

$$SE = \left(\hat{\vartheta} - \vartheta\right)^2$$

Calcolando tutti gli errori quadratici medi di una serie si può calcolare L'Errore Quadratico Medio (MSE):

$$MSE = \frac{\sum_{i=0}^{n} SE_i}{n}$$

La radice dell'errore quadratico medio è la radice di MSE e ha il vantaggio di avere la stessa unità di misura del valore che si vuole stimare.

$$RMSE = \sqrt{MSE}$$

Per quanto riguarda l'approccio basato su classificazione, si procederà convertendo il $close_{perc}$ assegnandolo ad una classe, un'etichetta. In particolare sarà assegnato alla classe denominata "1", se $close_{perc}$ è maggiore di 0, sarà assegnato invece alla classe "0" se minore di 0. Quindi appartenere alla classe "1" significa che vi è stato un incremento positivo del valore di chiusura giornaliero rispetto al giorno precedente; appartenere alla classe "0" significa invece che vi è stato un decremento del valore di chiusura rispetto al giorno precedente. In pratica esattamente come è stata calcolata la variabile *fluctuation*. Per la valutazione di questo approccio si utilizzerà una matrice di confusione, che restituisce una rappresentazione dell'accuratezza di una classificazione statistica. Nella matrice di confusione si mettono sulle righe i valori predetti, mentre sulle colonne i valori reali.

		Valori predetti		
		n'	***p'***	**totale**
Valori Reali	***n***	Veri negativi	Falsi positivi	N
	p	Falsi negativi	Veri positivi	P
	totale	N'	P'	

Quindi nella diagonale principale vi sono i valori correttamente predetti, mentre sull'anti-diagonale quelli non correttamente predetti. A partire dai valori contenuti nella matrice di confusione è possibile calcolare i seguenti indicatori:

- $accuratezza\ (Acc) = \frac{VP+VN}{VP+VN+FP+FN}$
- $recall = \frac{VP}{VP+FN}$
- $specificity = \frac{VN}{VN+FP}$
- $precision = \frac{VP}{VP+FP}$
- $NPV\ (Negative\ Predictive\ Value) = \frac{VN}{VN+FN}$

Il passo mancante è quello di implementare il calcolo di questi indicatori in Python. Esistono molte alternative; la libreria utilizzata in questo esperimento è Sklearn. Infatti vi sono contenuti due metodi, `mean_squared_error` e `confusion_metrix`, rispettivamente per calcolare L'errore quadratico medio e la matrice di confusione.

```
import math as mt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import confusion_matrix

# Il metodo mean_squared_error prende in ingresso due parametri. Il primo è
# la sequenza dei valori reali o target, la seconda è la sequenza delle
# previsioni. Per ottenere l'RMSE ne calcoliamo la radice quadrata
# sfruttando il metodo sqrt() della libreria math

MSE = mean_squared_error(target, forecast)
RMSE = mt.sqrt(MSE)

# Anche in questo caso il metodo confusion_matrix prende in ingresso come
# primo parametro la sequenza dei valori reali e come secondo parametro
# i valori previsti. La variabile conf_matrix contiene quindi una matrice.
# Il metodo ravel() serve ad estrarre i 4 valori dalla matrice.

conf_matrix = confusion_matrix(classification_target, classification_forecast)
true_neg, false_pos, false_neg, true_pos = conf_matrix.ravel()

# Ora è possibile calcolare anche gli indicatori per valutare il problema in
# termini di classificazione

acc=(true_pos+true_neg)/(true_pos+true_neg+false_pos+false_neg)
recall=(true_pos)/(true_pos+false_neg)
specificity=(true_neg)/(true_neg+false_pos)
precision = (true_pos)/(true_pos+false_pos)
NPV = (true_neg)/(true_neg+false_neg)
```

4.4 IL PROBLEMA DELLA CONFIGURAZIONE DEGLI IPER-PARAMETRI

Un aspetto importante, di cui non si era tenuto conto fino a questo momento, è la scelta degli iper-parametri di ogni layer, ovvero numero di percettroni che lo compongono, caratteristiche della funzione di attivazione e della funzione di perdita e, nel caso dei layer convoluzionali e di pooling, la dimensione del kernel di scansione. Purtroppo non esistono teoremi o metodi che permettono di conoscere i parametri ottimali di una rete, essendo una Rete Neurale assimilabile ad una black box di cui è complicatissimo risalire ai calcoli che il modello effettua per fissare i Weight e i Bias dei nodi. Pertanto si deve procedere per tentativi, provando diverse configurazioni e scegliendo quelle che danno i risultati migliori.

Considerando che il tempo per l'addestramento della rete e il calcolo dei risultati, utilizzando un computer casalingo medio, sono molto elevati, si è optato per l'implementazione di una struttura automatica che permettesse di provare diverse combinazioni di parametri avviando una sola volta il programma, con un evidente vantaggio in termini di comodità e efficienza.

Il tutto viene realizzato creando una variabile vettore per ogni parametro, che si andrà a riempire con la lista dei valori che si vuole testare.

```
# kernel_config è il vettore che contiene la dimensione del kernel del primo,
# del secondo e del terzo layer convoluzionale
kernel_config = [7, 5, 3]

# n_features è un intero che indica il numero di caratteristiche prese in
# considerazione
n_features = 5

# filter_conv_config contiene il numero di percettroni del primo, del secondo e
del # terzo layer. La struttura è un array di array ed è utilizzata per provare
più # configurazioni contemporaneamente
filter_conv_config = (
   (128, 128, 64),
   (64, 64, 32)
)

# filter_dense_config rappresenta il numero di percettroni del penultimo Fully
# Connected layer. L'ultimo per default ha un solo percettrone in quanto è il
layer # di output che per costruzione è composto da un singolo valore
filter_dense_config = [50, 100, 150]

# n_steps_config contiene il numero di time-step, ovvero la numerosità di
# osservazioni (per ogni caratteristica) di ogni campione di input
n_steps_config = [10, 15, 20]
```

```
# optimizer_config contiene gli algoritmi di ottimizzazione che si vogliono
# testare
optimizer_config = ['Adam', 'SGD']

# epochs_config il numero di epoche che si vogliono utilizzare per i test
epochs_config = [300, 500, 1000]

# batch_config contiene quale dimensione di batch si vuole utilizzare per i
# test.
batch_config = [16, 32, 64]
```

Per realizzare quest'automatismo è stata realizzata una struttura di `for` in cascata, implementata nel modo seguente:

```
for i in range(len(filter_conv_config)):
   for j in range(len(filter_dense_config)):
      for k in range(len(n_steps_config)):
         for t in range(len(optimizer_config)):
            for p in range(len(epochs_config)):
               for x in range(len(batch_config)):

                  # split in training set e test set
                  data_previsioni_train = data[n_steps_config[k]:1023, 1]
                  data_previsioni_test = data[(1023 + n_steps_config[k]):, 1]

                  X_train,y_train=split_sequences(trainin_set,n_steps_config[k])
                  X_test, y_test = split_sequences(test_set, n_steps_config[k])


print("============================================================")

print("============================================================\n")

print("\nConfigurazione: ", count_config)
print("Model: ", filter_conv_config[i], filter_dense_config[j])
print("n_step: ", n_steps_config[k])
print("Optimizer: ", optimizer_config[t])
print("Epoche: ", epochs_config[p])
print("batch_size: ", batch_config[x])
print("Kernel: ", kernel_config)
print("Feature: ", n_features)


print("\n============================================================\n")

model = Sequential()
model.add(Conv1D(filters=filter_conv_config[i][0], kernel_size=kernel_config[0],
                 activation='relu', input_shape=(n_steps_config[k], n_features)))
model.add(Conv1D(filters=filter_conv_config[i][1], kernel_size=kernel_config[1],
                                                              activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=filter_conv_config[i][2], kernel_size=kernel_config[2],
                                                              activation='relu'))
model.add(MaxPooling1D(pool_size=2))
```

```
model.add(Flatten())
model.add(Dense(filter_dense_config[j], activation='relu'))
model.add(Dense(1))

model.compile(optimizer=optimizer_config[t], loss='mse')

model.fit(X_train, y_train, epochs=epochs_config[p], batch_size=batch_config[x])
```

Il codice sviluppato permette di lanciare il programma e fargli testare diverse configurazioni della rete in un solo avvio. Definire quindi i parametri in vettori come descritto in precedenza

```
filter_conv_config = (
   (128, 128, 64),
   (64, 64, 32)
)
filter_dense_config = [50, 100, 150]
n_steps_config = [10, 15, 20]
optimizer_config = ['Adam', 'SGD']
epochs_config = [300, 500, 1000]
batch_config = [16, 32, 64]
```

significa che all'avvio del programma testeremo un numero di configurazioni diverse pari al prodotto della numerosità dei vettori rappresentanti gli iper-parametri.

Avviando il programma con questi dati di esempio il programma testerà le seguenti configurazioni diverse:

- configurazione iterazione 1:
  - N° percettroni primo layer convoluzionale = 128;
  - N° percettroni secondo layer convoluzionale = 128;
  - N° percettroni terzo layer convoluzionale = 64;
  - N° percettroni layer fully connected = 50;
  - Time-step = 10;
  - Algoritmo di discesa del gradiente = "Adam";
  - N° epoche = 300;
  - Dimensione batch = 16;

- configurazione iterazione 2:
  - N° percettroni primo layer convoluzionale = 128;
  - N° percettroni secondo layer convoluzionale = 128;
  - N° percettroni terzo layer convoluzionale = 64;
  - N° percettroni layer fully connected = 50;
  - Time-step = 10;
  - Algoritmo di discesa del gradiente = "Adam";
  - N° epoche = 300;
  - Dimensione batch = 32;

- configurazione iterazione 3:
  - N° percettroni primo layer convoluzionale = 128;
  - N° percettroni secondo layer convoluzionale = 128;
  - N° percettroni terzo layer convoluzionale = 64;
  - N° percettroni layer fully connected = 50;
  - Time-step = 10;
  - Algoritmo di discesa del gradiente = "Adam";
  - N° epoche = 300;
  - Dimensione batch = 64;

- configurazione iterazione 4:
  - N° percettroni primo layer convoluzionale = 128;
  - N° percettroni secondo layer convoluzionale = 128;
  - N° percettroni terzo layer convoluzionale = 64;
  - N° percettroni layer fully connected = 50;
  - Time-step = 10;
  - Algoritmo di discesa del gradiente = "Adam";
  - N° epoche = 500;
  - Dimensione batch = 16;

- configurazione iterazione 5:
  - N° percettroni primo layer convoluzionale = 128;
  - N° percettroni secondo layer convoluzionale = 128;
  - N° percettroni terzo layer convoluzionale = 64;
  - N° percettroni layer fully connected = 50;
  - Time-step = 10;
  - Algoritmo di discesa del gradiente = "Adam";
  - N° epoche = 500;
  - Dimensione batch = 32;

- configurazione iterazione 6:
  - N° percettroni primo layer convoluzionale = 128;
  - N° percettroni secondo layer convoluzionale = 128;
  - N° percettroni terzo layer convoluzionale = 64;
  - N° percettroni layer fully connected = 50;
  - Time-step = 10;
  - Algoritmo di discesa del gradiente = "Adam";
  - N° epoche = 500;
  - Dimensione batch = 64;

- configurazione iterazione 7:
  - N° percettroni primo layer convoluzionale = 128;
  - N° percettroni secondo layer convoluzionale = 128;
  - N° percettroni terzo layer convoluzionale = 64;
  - N° percettroni layer fully connected = 50;
  - Time-step = 10;
  - Algoritmo di discesa del gradiente = "Adam";
  - N° epoche = 1000;
  - Dimensione batch = 16;

E così via finché non saranno state provate tutte le combinazioni possibili, che nel nostro esempio ammontano a 324. È chiaro con questo esempio come sia possibile risparmiare molto tempo, visto che con una singola esecuzione del programma è possibile testare 324 combinazioni di parametri diverse.

Capitolo V

TESTING E VALUTAZIONE DELLE PERFORMANCE

Nei precedenti capitoli si sono viste le fasi di preparazione del modello, la teoria, i problemi da affrontare durante la raccolta dei dati e l'implementazione della rete. Ora si arriva al nocciolo della questione. La preparazione del modello è ultimata, quindi non resta che testarlo per valutarne l'effettiva efficacia.

5.1 APPROCCI DI VALUTAZIONE

Come detto in precedenza le performance della rete sono state testate utilizzando i dati di due indici azionari europei: l' Euro Stoxx 50 e il FTSE Mib 40. L'archivio dati è composto dai dati giornalieri dei due indici dal 05/01/2015 al 30/04/2020[9]. Il primo aspetto è stata la suddivisione del Dataset tra training e testing set. Il training set è la parte dei dati che viene utilizzata per addestrare il modello, pertanto viene fornito alla rete completo dei campioni di input e dei corrispondenti output al fine di tarare i weight e i bias della rete con lo scopo di replicare la funzione. Il testing set ha invece lo scopo di verificare la bontà del framework. Infatti il modello ha lo scopo di fornire previsioni, pertanto utilizziamo dati già conosciuti, questa volta fornendo alla rete solo i campioni di input. Gli output sono le previsioni fatte dal modello, ma su eventi che sono già conosciuti. Questo dà la possibilità di confrontare le previsioni del modello con dati reali e quindi di testare le performance. Nell'esperimento sono stati utilizzati come training set l'insieme dei dati dei primi 4 anni, dal 05/01/2015 al 31/12/2018, mentre come testing set i dati dal 01/01/2019 al 30/04/2020.

La rete è stata testata utilizzando sia un approccio a regressione che un approccio a classificazione. Come regressione si ha l'obiettivo di prevedere il valore esatto assunto dalla variabile obiettivo a partire dai dati di un certo numero di giorni precedenti. Come classificazione ci si accontenta di valutare l'incremento o il decremento della variabile obiettivo rispetto al giorno precedente, sempre partendo dai dati di un certo numero di giorni precedenti.

La valutazione avviene inserendo nella rete delle variabili di input e valutando l'output che il modello restituisce. Ovviamente, potenzialmente, potremmo inserire qualsiasi dato in input, purché costruito con una struttura compatibile. Questo significa che se si decide di prevedere il closing value a partire, per esempio, dai dati dei 10 giorni precedenti, il modello si aspetta in ingresso una sequenza di 10 numeri. Ma la variabile che prende in ingresso è semplicemente un array, un vettore, il modello non

ha idea del fatto che quella sia una sequenza. La differenza la fa il modo in cui implementiamo il software che crea il modello. Pertanto potremmo inserire qualunque vettore con 10 numeri casuali e il modello restituirà lo stesso un output. Questo è importante perché volendo si potrebbero anche fare delle simulazioni di comportamento creando scenari ad hoc che si vogliono studiare sottoponendoli al modello. Fatta questa precisazione, sono 3 gli approcci di valutazione che sono stati testati:

a. Test sul training set: questo è l'approccio più arduo da comprendere. Fare test sul training set significa che, dopo aver addestrato il modello, quindi fornendo alla rete i campioni di input e i corrispondenti output che compongono il training set, si fa il testing fornendo al framework soltanto gli input che compongono lo stesso training set, al fine di valutare gli output che ne derivano. La ratio di quest'approccio sta nell'interesse di valutare come il modello riesce ad interpretare i dati su cui si è addestrato.
b. Test sui dati del 2019: il secondo approccio consente di valutare effettivamente le performance di previsione del modello, essendo dati che la rete non ha mai avuto modo di analizzare precedentemente. In questo caso forniamo alla rete i dati di input solo dell'anno 2019.
c. Test sul testing set completo: l'ultimo approccio è quello completo, in cui vengono sottoposti al modello i campioni di input dell'intero testing set. La suddivisione tra quest'approccio e quello precedente sta nel fatto che il 2019 è stato un anno sostanzialmente privo di shock significativi, mentre i primi mesi del 2020 sono stati caratterizzati dagli shock derivanti dalla crisi covid. Pertanto questa differenza è interessante per valutare come il modello performa diversamente con e senza shock del mercato.

5.2 TESTING IN PYTHON

Anche per quanto riguarda il testing della rete vengono in aiuto delle funzioni implementate nella libreria Keras, usata per costruire il modello. In questo caso si utilizza un metodo della classe `Sequential()`, che prende il nome di `predict()`. Il metodo `predict()` prende in ingresso un campione di input ed altri parametri di configurazione opzionali.

```
from keras import Sequential

# istruzioni base per la costruzione di un modello
model = Sequential()
model.compile(optimizer=optimizer_config, loss='mse')
model.fit(X, y, epochs=num_epochs, batch_size=num_batch)
```

```
# il metodo predict prende in ingresso il campione i-esimo di cui si vuole
# ottenere la previsione e inserisce il risultato nella variabile posta a
# sinistra, in questo caso forecasting
forcasting = model.predict(xi)
```

Come si può notare la funzione fornita da Keras consente di testare soltanto un input per volta. Lo scopo è invece quello di ricreare i valori previsti giorno per giorno in maniera tale da costruire l'andamento grafico dei prezzi futuri attesi. Pertanto, per ovviare al problema, bisogna costruire una funzione che prenda in ingresso la sequenza di input e restituisca la sequenza di output. Segue pertanto l'implementazione della funzione `predict_function`, che prende in ingresso l'istanza del modello che deve fare la previsione, contenuto nella variabile `model` e la sequenza dei campioni di input di cui si vuole ottenere la previsione, contenuta nella variabile chiamata appunto `input`.

```
def predict_function(model, input):
    # si inizializza una struttura in cui si allocherà la sequenza delle
    # previsioni, ovvero degli output
    forecast_array = list()

    # questo ciclo scorre la variabile input, che contiene la
    # sequenza dei campioni di input
    for i in range(len(input)):

        # per ogni singolo campione di input è necessaria l'aggiunta di una
        # dimensione per renderlo compatibile con la funzione predict
        input_i = input[i].reshape(1, input.shape[1], input.shape[2])

        # per ogni campione viene effettuata la previsione e salvato il valore
        # nella variabile forecast. Poi il valore contenuto in forecast viene
        # aggiunto in coda alla variabile di output forecast_array
        forecast = model.predict(input_i, verbose=0)
        forecast_array.append(forecast)

    return array(forecast_array)
```

Con questa funzione si è arrivati ad ottenere la sequenza dei valori previsti, che graficamente rappresenta l'andamento del valore dei prezzi futuri attesi. Si hanno quindi valori numerici reali, che rappresentano dunque la soluzione al problema di regressione che si era posto. Per quanto riguarda il problema di classificazione è necessario realizzare un'altra funzione, che prende in ingresso la sequenza di output ottenuta dalla funzione predict_function, e restituisce una sequenza in cui nell' i-esima posizione si ha 1 se il corrispondente valore è maggiore di 0, 0 se il corrispondente valore è minore di 0.

```
def classification(forecast_array):

   forecast_classification_array = list()

   for i in range(len(forecast_array)):
      if forecast_array[i] >= 0:
         forecast_classification_array.append(1)
      else:
         forecast_classification_array.append(0)

   return array(forecast_classification_array)
```

Tramite le due funzioni predict_function e classification si sono ottenuti i risultati desiderati. Nel main del software potranno essere richiamate queste due funzioni per ottenere i risultati delle metriche che descrivono i modelli, come descritto nel capitolo precedente.

Per ottenere i grafici invece possiamo seguire due strade equivalenti. La prima è sfruttare le funzioni implementate nella libreria `matplotlib` di Python. La seconda è scaricare un file contenente i dati e tracciare i grafici sfruttando Excel. Per il primo approccio è necessario inserire il codice seguente:

```
   import matplotlib.pyplot as plt

   # la funzione plot prende in ingresso le sequenze che si vuole plottare e
   # altri valori di configurazione opzionali, come ad esempio il colore o
   # un'etichetta tramite l'attributo label
   plt.plot(forecast_array, 'b', label='Previsioni')
   plt.plot(y_testing_set, 'g', label='Valori Reali')

   # attivando la funzione legend si specifica che bisogna inserire la leggenda
   # nel grafico
   plt.legend()

   # show mostra il grafico a video
   plt.show()
```

Per percorrere la seconda strada va importata la libreria os, che contiene delle funzioni con cui il programma può interagire col sistema operativo. In questo caso interessano le funzioni che consentono di modificare il file system.

```
   import os

   # in dirName si inserisce la stringa contenente il percorso della cartella in
   # cui si vuole salvare il file. Se la cartella non esiste verrà creata
   dirName = "/Users/Desktop/OUTPUT/"
   if not os.path.exists(dirName):
      os.mkdir(dirName)
```

```
# si crea una struttura a matrice in cui si inseriscono i dati che si vuole
# salvare
data_to_file = {
   "valori reali": y_testing_set,
   "previsioni": forecast_array,
}

# con la funzione DataFrame si trasforma la matrice in una struttura
# scrivibile in un file
data_to_file = DataFrame(data_to_file)

# infine si utilizza la funzione to_csv per salvare i dati in un file .csv
# nella cartella specificata dal percorso contenuto in dirName
data_to_file.to_csv(dirName+"/forecast.csv")
```

Una volta eseguita la funzione si ritroverà il file .csv con le informazioni salvate al percorso specificato dalla variabile `dirName`.

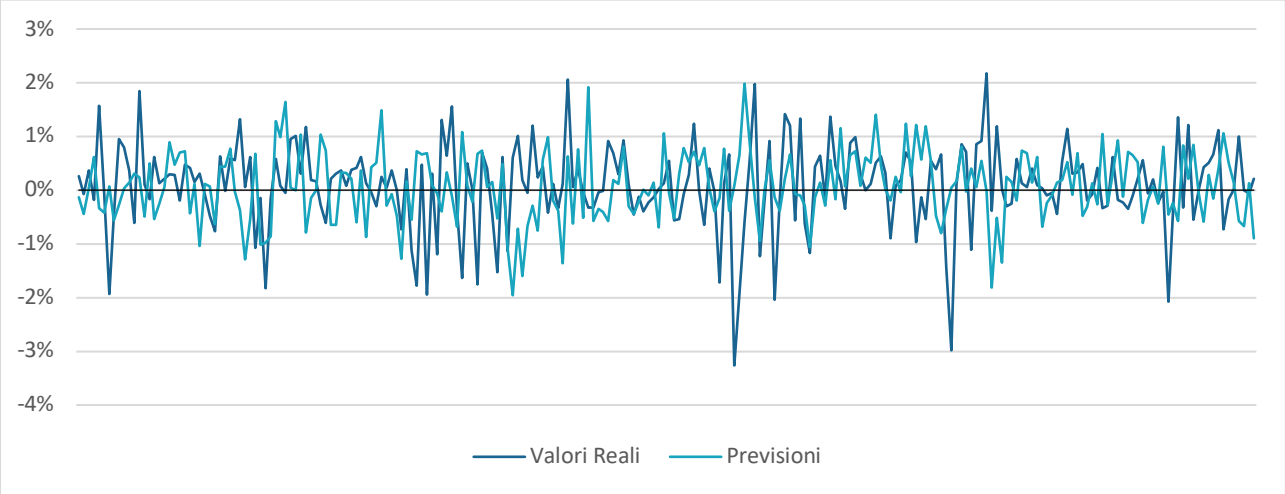
Ultimati tutti i passaggi di preparazione dell'archivio dati e di implementazione del modello, è finalmente possibile lanciare il software ed ottenere i risultati. Nei prossimi paragrafi si elencheranno e analizzeranno i risultati traendo conclusioni sull'efficacia del modello, sui limiti e sui possibili miglioramenti.

5.3 PERFORMANCE SULL' EURO STOXX 50

Si inizia dall'analisi dei risultati ottenuti sull' Euro Stoxx 50, valutando gli output dei test effettuati sui diversi insiemi descritti all'inizio del capitolo, sia per l'approccio a regressione che per quello a classificazione.

5.3.1 Valutazione sui dati del 2019

Il primo test viene effettuato sui primi 234 campioni del dataset, per i dati giornalieri che vanno dal 02/01/2019 al 27/12/2019.



Nel grafico si esplicita l'andamento delle previsioni nel tempo (in verde) sovrapposto all'andamento reale dei prezzi dell'indice (in blu). Partendo dalla valutazione delle metriche, per quanto riguarda l'approccio a regressione, si calcola un RMSE di 0.99; È chiaro che, visto la previsione in termini percentuali, l'RMSE è molto elevato per poter considerare la previsione accurata. Tuttavia, in termini empirici, si è ottenuto comunque un errore minore rispetto ad altri modelli Machine Learning precedenti le Reti Neurali Artificiali Profonde. Analizzando il grafico si evince come il modello riesce a replicare l'andamento e l'ampiezza dei valori reali, senza tuttavia riuscire a cogliere con precisione il valore esatto.

Per la valutazione dell'approccio a classificazione si va ad analizzare i valori contenuti nella confusion matrix.

		Valori predetti	
		0	1
Valori Reali	0	46	50
	1	62	76

Nella matrice di confusione si hanno sulle righe i valori reali e sulle colonne i valori predetti. Agli incroci vi sono quindi nell'ordine i "veri negativi", i "falsi positivi", i "falsi negativi" e i "veri positivi". In pratica sulla diagonale principale si hanno i valori correttamente predetti mentre sull'anti diagonale i valori non correttamente predetti. La somma di tutti i valori da proprio 234, ovvero il numero di campioni analizzati, pertanto si ha un risultato per ognuno di loro.

A partire dai valori contenuti nella matrice di confusione è possibile calcolare gli indicatori per la valutazione dell'approccio a classificazione.

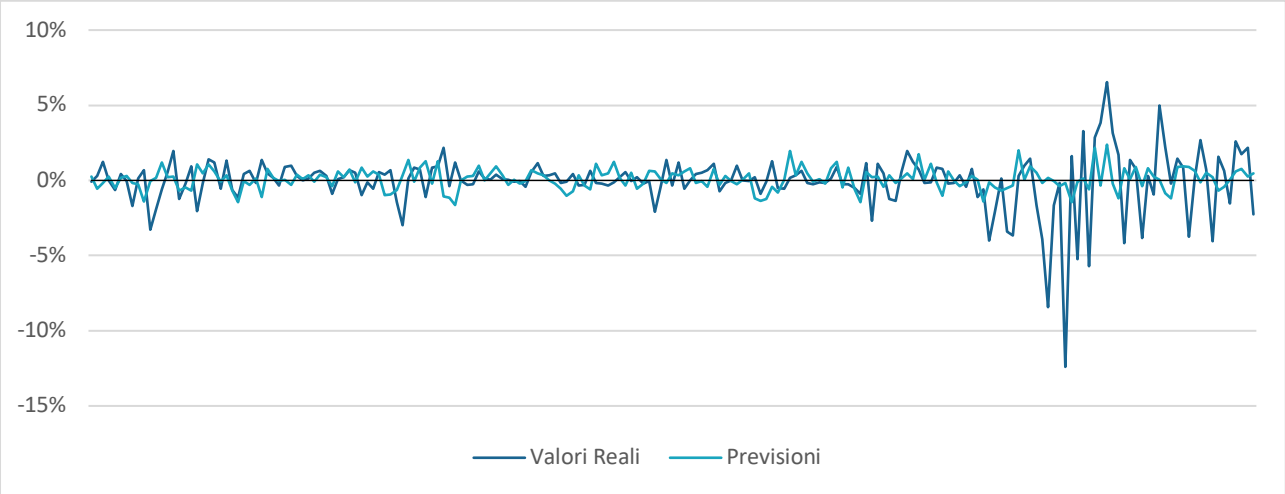
- $accuratezza = 52.14\ \%$
- $recall = 55.07\ \%$
- $specificity = 47.92\ \%$

- $precision = 60.32\ \%$
- $NPV\ (Negative\ Predictive\ Value) = 42.59\ \%$

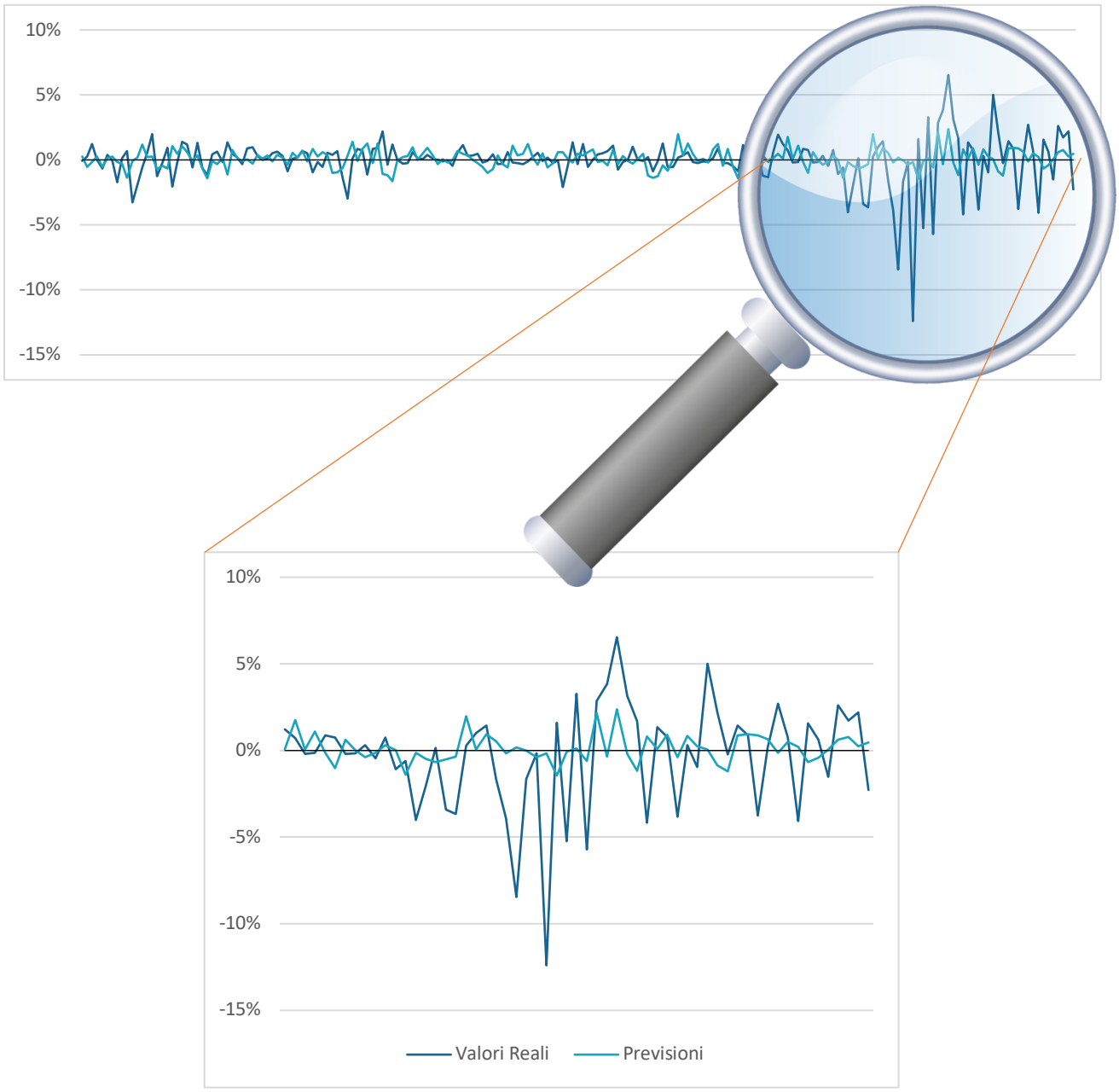
L'indicatore più importante, che dà l'indicazione generale della bontà di funzionamento del modello è l'accuratezza, che è del 52.14 %. Quindi soltanto poco più della metà delle fluttuazioni sono state correttamente identificate. Approfondendo gli indicatori recall e specificity, che indicano la corretta identificazione rispettivamente delle fluttuazioni positive e negative, notiamo una percentuale di accuratezza che pende in favore dei movimenti positivi (55 % contro 48 %). È possibile spiegare questa differenza analizzando il training set, composto per la maggior parte da fluttuazioni positive rispetto a quelle negative. Pertanto il modello potrebbe aver appreso meglio come riconoscere le prime rispetto alle seconde. Anche la differenza tra gli indicatori precision e NPV può essere spiegata allo stesso modo.

5.3.2 Valutazione sul testing set completo

Il secondo test effettuato comprende i campioni dell'intero testing set, quindi i dati giornalieri che vanno dal 02/01/2019 al 30/04/2020, per un totale di 319 campioni. Di seguito il grafico con l'andamento delle previsioni e dei valori reali, rispettivamente in verde e in blu.



La prima cosa che si osserva è l' RMSE, che balza a 1.56, un valore molto più elevato rispetto allo 0.99 riscontrato nell'analisi precedente. Infatti il motivo per cui il testing set è stato suddiviso in un insieme contenente i soli dati del 2019 e in un insieme che contiene anche quelli del 2020 è che nel 2019 non vi sono stati shock significativi di mercato, mentre nei primi mesi del 2020 i mercati sono stati afflitti dalla crisi derivante dal covid.



La porzione di grafico evidenziata è quella relativa al mese di marzo, dove è possibile notare le profonde oscillazioni dei prezzi reali (grafico in blu) causate dallo shock dovuto alla crisi Covid, che hanno il picco nella data del 12/03, dove l'indice ha fatto registrare un decremento del 12 % rispetto al giorno precedente. Il problema è che il training set non possiede degli andamenti così ampi e repentini, pertanto il modello non è in nessun modo in grado di replicarli. Ma anche nel caso in cui il training set avrebbe contenuto delle oscillazioni causate da una crisi poteva verificarsi il problema contrario, ovvero quello dell'overfitting, in cui il modello avrebbe potuto considerare gli andamenti da crisi non come uno shock, ma come una fase di normalità. Un modo per tentare di ovviare a questo problema è quello di aggiungere delle variabili di classificazione che possono indicare al modello se il campione i-esimo fa parte di un

periodo di cui tener conto oppure no. Tuttavia non si ha la certezza dell'efficacia del metodo.

Per quanto riguarda la classificazione, si esplicita la confusion matrix:

		Valori predetti	
		0	1
Valori Reali	0	66	72
	1	78	103

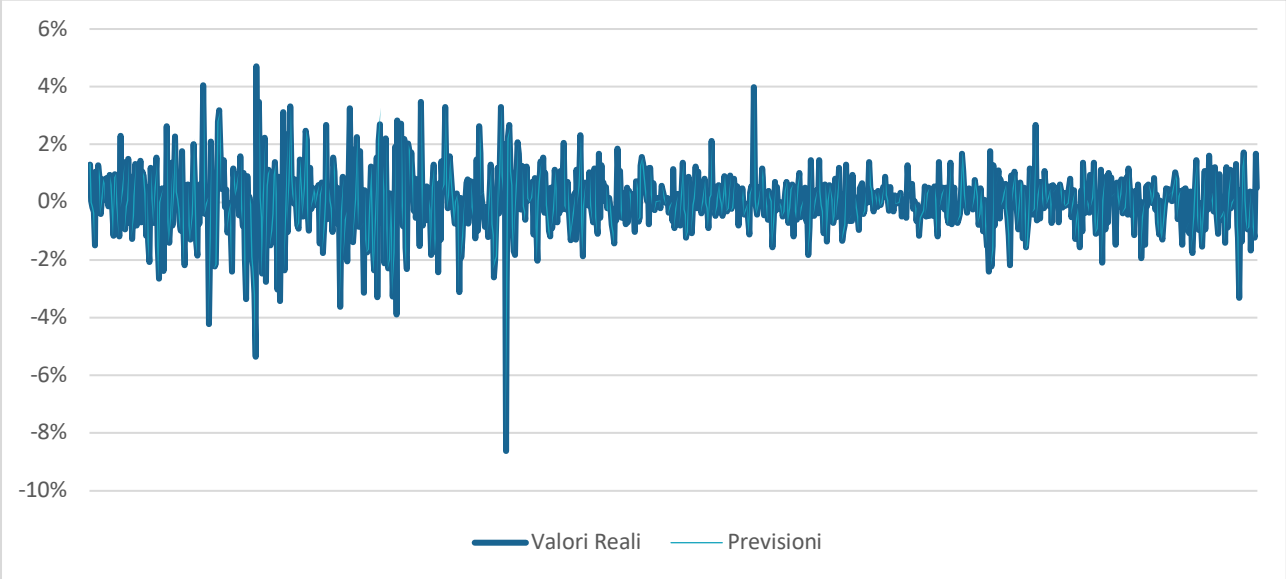
Nella matrice si notano dei valori in linea con i risultati precedenti, cosa che denota ancora la minor complessità nel rilevamento delle sole fluttuazioni rispetto alla predizione dei valori reali, come si evince anche dal calcolo degli indicatori:

- $accuratezza = 52.98\ \%$
- $recall = 56.91\ \%$
- $specificity = 47.83\ \%$
- $precision = 58.86\ \%$
- $NPV\ (Negative\ Predictive\ Value) = 45.83\ \%$

Infatti gli indicatori sono simili rispetto al test precedente; non vi è stato nell'approccio a classificazione il peggioramento che invece c'è stato nella valutazione dei valori numerici, ancora a riprova del fatto che il modello riesce comunque a cogliere in parte l'andamento dei prezzi.

5.3.3 Valutazione sul training set

Per concludere l'analisi sull' Euro Stoxx 50, si traccia il grafico dell'andamento ottenuto inserendo gli stessi dati utilizzati per l'addestramento.



I valori previsti sono quasi completamente sovrapposti ai valori reali. Infatti per quanto riguarda gli indicatori abbiamo un RMSE pari a 0.06 e un'accuratezza del 98 %. Questo a riprova dell'enorme efficacia che le Reti Neurali Artificiali dimostrano nel mapping delle funzioni, anche se queste sono altamente complesse e non lineari come lo è la funzione dell'andamento dei prezzi.

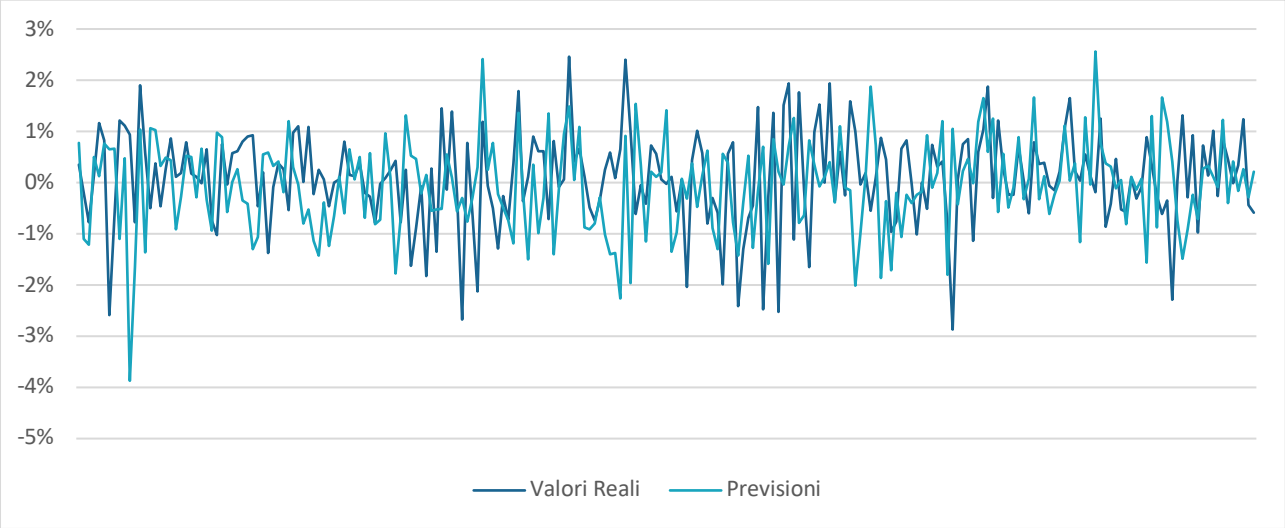
I risultati non altrettanto buoni ottenuti negli altri test sono da ricondurre al fatto che i prezzi futuri non dipendono solo dai dati dei giorni precedenti, ma hanno a che fare con fattori molto più complessi e inaspettati come la situazione macro-economica della zona, la situazione politica, il tasso di interesse ecc. I modi con cui potrebbe essere possibile migliorare il modello tenendo conto anche di questi aspetti saranno analizzati nei capitoli conclusivi.

5.4 PERFORMANCE SUL FTSE MIB

Si passa ora all'analisi dei risultati ottenuti dai test effettuati sul secondo indice che è stato preso in considerazione per l'esperimento, il FTSE Mib.

5.4.1 Valutazione sui dati del 2019

Anche in questo caso si parte dal test effettuato sui dati del solo 2019, quindi su 231 campioni (abbiamo qualche valore in meno rispetto all' Euro Stoxx in quanto nel Dataset vi sono alcuni giorni di trading mancanti per il FTSE Mib). Per l'analisi valutiamo sempre l'andamento del grafico delle previsioni (in verde) sovrapposto a quello dei prezzi reali (in blu).



Il modello è riuscito a replicare in parte l'andamento e l'ampiezza delle funzioni dei prezzi, anche se l'RMSE ottenuto, pari ad 1.18, è più alto rispetto a quello ottenuto nel pari test effettuato sull' Euro Sotxx. Questo si può spiegare col fatto che il FTSE Mib è un indice più instabile dell' Euro Stoxx e più sensibile al peso delle banche.

Nella valutazione dell'approccio a classificazione si può notare una cosa interessante:

		Valori predetti	
		0	1
Valori Reali	0	56	44
	1	60	71

Come ci si aspetta la somma di tutti i valori della matrice di confusione è pari a 231, come il numero di campioni analizzati.

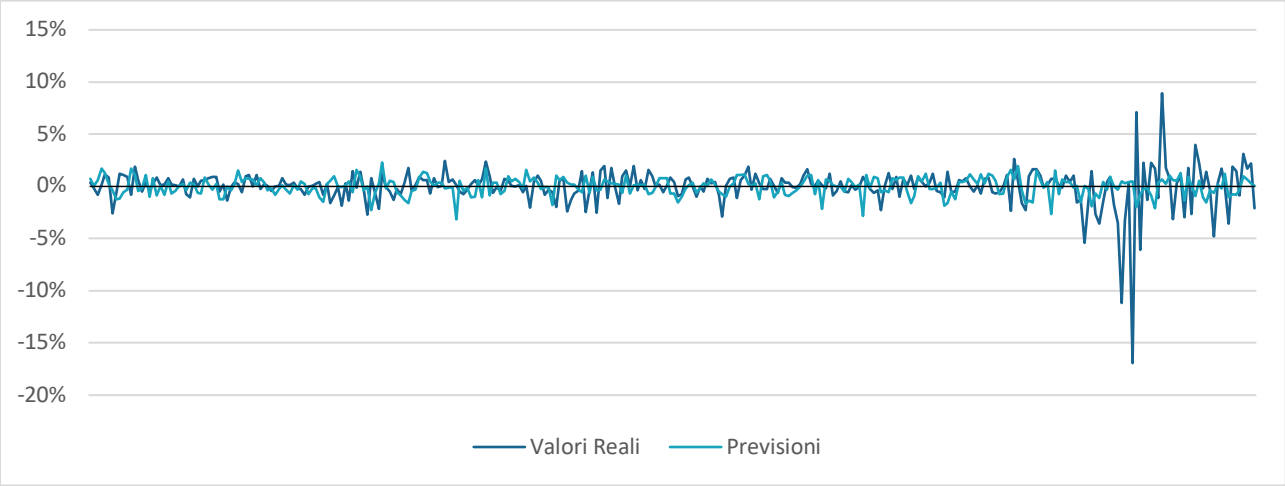
- $accuratezza = 54.98\ \%$
- $recall = 54.20\ \%$
- $specificity = 56\ \%$
- $precision = 61.74\ \%$

- $NPV\ (Negative\ Predictive\ Value) = 48.28\ \%$

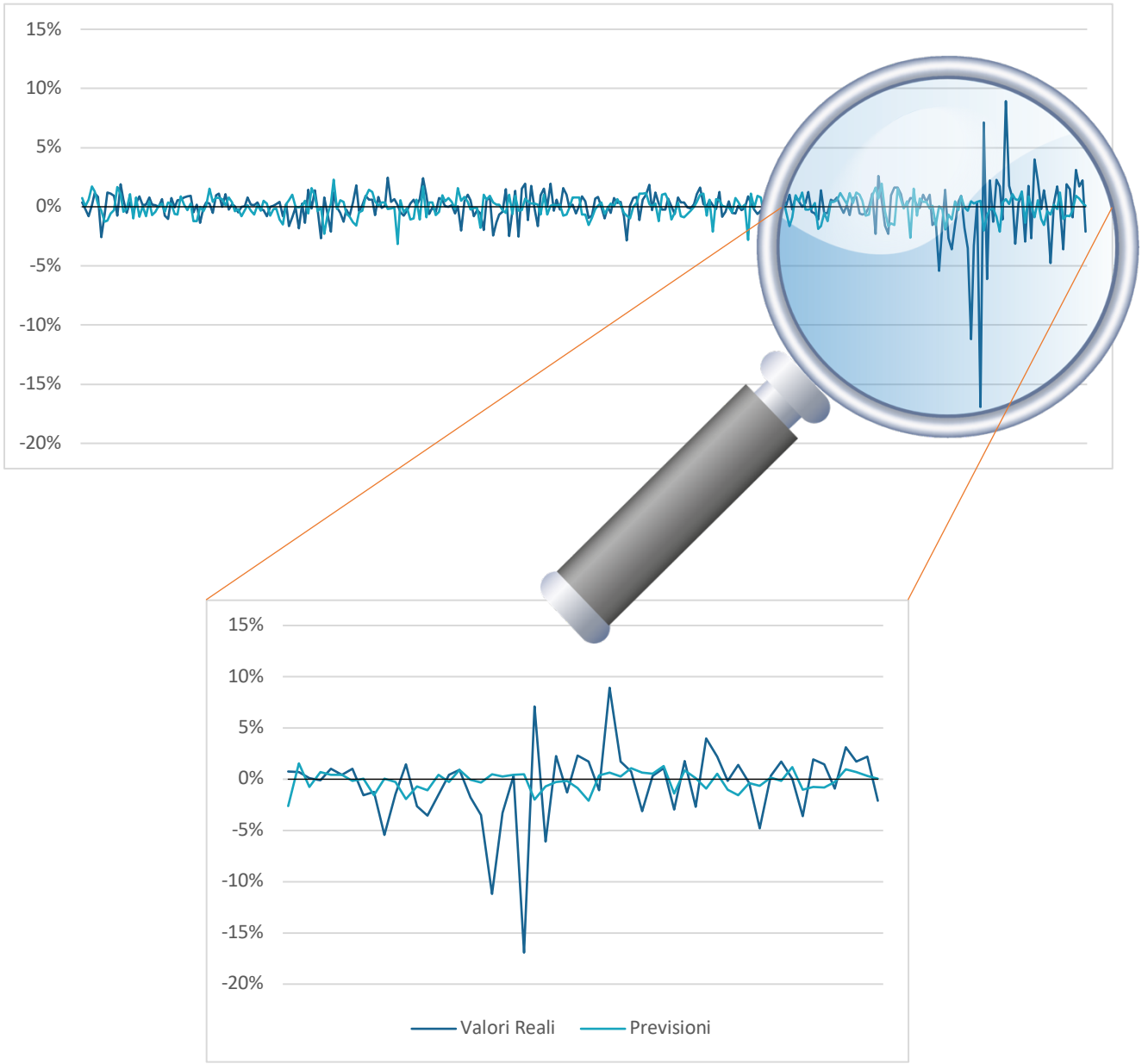
La cosa interessante sta nel fatto che nel FTSE Mib abbiamo un'accuratezza leggermente migliore di quella riscontrata nel pari test dell' Euro Stoxx, ma soprattutto il fatto che i valori di recall e specificity sono più allineati rispetto a quelli dell'altro indice. È possibile spiegare questa cosa con il fatto che il training set del FTSE Mib ha fluttuazioni positive e negative più simili in numero, rispetto all' Euro Stoxx in cui invece vi era una pendenza in favore delle positive che aveva portato il modello a riconoscere meglio quest'ultime rispetto alle fluttuazioni negative. Un altro aspetto è che anche in questo caso l'approccio a classificazione ha dato risultati migliori rispetto alla regressione, cosa che ancora una volta denota la minor complessità nel riconoscere l'andamento delle fluttuazioni rispetto al riconoscimento dei valori esatti dei prezzi.

5.4.2 Valutazione sul testing set completo

Il secondo test effettuato comprende i campioni dell' intero testing set, quindi i dati giornalieri che vanno dal 02/01/2019 al 30/04/2020, per un totale di 316 campioni (anche in questo caso il numero è leggermente inferiore rispetto all'Euro Stoxx per la mancanza dei dati di tre giornate di trading nel Dataset del FTSE Mib). Di seguito il grafico con l'andamento delle previsioni e dei valori reali, rispettivamente in verde e in blu.



In questo secondo test l' RMSE balza da 1.18 a 1.82. Si possono fare le stesse considerazioni fatte per l'Euro Stoxx. Il problema è la presenza della crisi covid, che il modello non riesce a replicare non essendoci andamenti del genere all'interno del training set.



Lo zoom, anche in questo caso, corrisponde al periodo di marzo, dove è possibile vedere le profonde oscillazioni causate dalla crisi covid, che hanno il loro picco nella giornata del 12/03 dove l'indice ha fatto registrare una perdita del 17 % rispetto al giorno precedente. I prezzi attesi non riescono neanche lontanamente a replicare in questo caso l'andamento dei prezzi reali, proprio per mancanza nel training set di andamenti del genere, ed anche se fossero stati presenti, comunque si correva il rischio che questi movimenti potevano essere considerati normali, come spiegato qualche paragrafo fa nell'analisi dell' Euro Stoxx.

Analizzando gli indicatori calcolati tramite la matrice di confusione, vediamo che i valori non sono molto diversi da quelli del test precedente, ad ulteriore prova del fatto che la previsione delle fluttuazioni è meno complessa rispetto alla previsione del valore esatto del prezzo dell'indice. Anche in questo caso, il decremento di

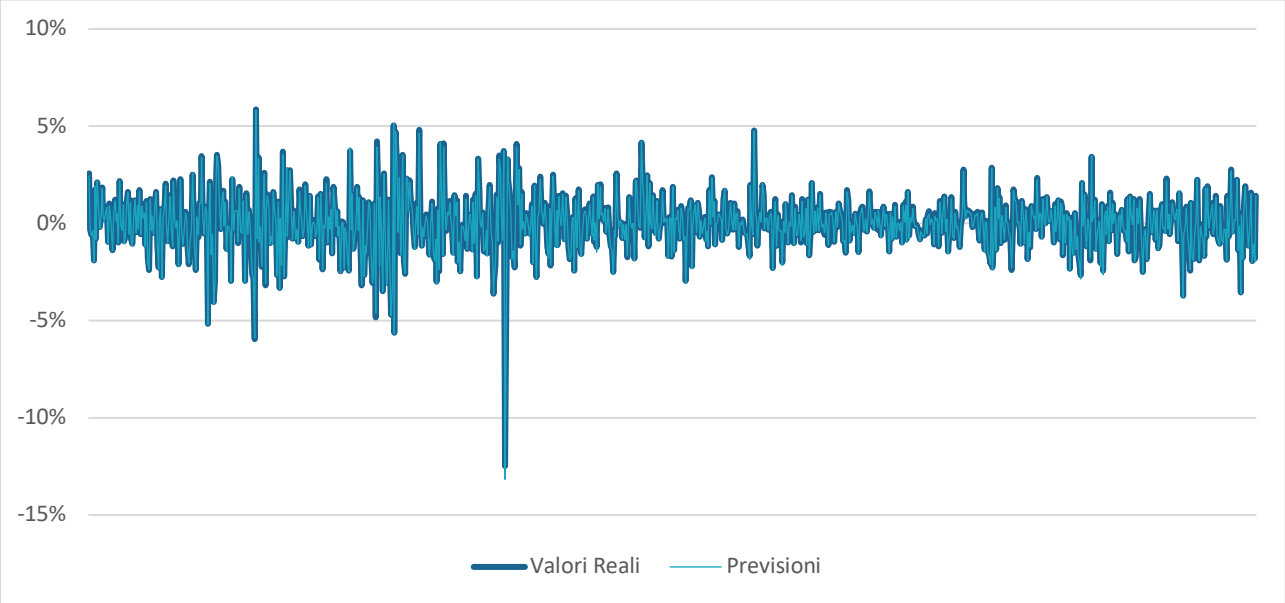
performance nell'approccio a regressione, non si è verificato per l'approccio a classificazione.

		Valori predetti	
		0	1
Valori Reali	0	70	67
	1	77	102

- $accuratezza = 54.43\ \%$
- $recall = 56.98\ \%$
- $specificity = 51.09\ \%$
- $precision = 60.36\ \%$
- $NPV\ (Negative\ Predictive\ Value) = 47.62\ \%$

5.4.3 Valutazione sul training set

Si conclude l'analisi sul FTSE Mib tracciando il grafico dell'andamento ottenuto inserendo gli stessi dati utilizzati per l'addestramento.



Le due funzioni dell'andamento dei prezzi reali e dei prezzi attesi sono quasi completamente identiche, un ulteriore riprova del fatto che le Reti Neurali Artificiali sono straordinarie nel mapping di funzioni molto complicate e altamente non lineari. Per quanto riguarda le cause per cui il modello funziona perfettamente con i dati di addestramento ma non con dati di testing diversi, valgono le stesse considerazioni fatte nel paragrafo precedente per l' Euro Stoxx 50.

Capitolo VI

CONCLUSIONI

Conclusa anche la fase di testing, è giunto il momento di trarre le conclusioni finali sull'esperimento. Si metteranno in luce punti di forza e limiti riscontrati, e il modo in cui potrebbe essere possibile ottenere dei miglioramenti dal modello.

6.1 PUNTI DI FORZA DEL MODELLO

Di certo è possibile affermare che non è stato risolto il problema della previsione dei prezzi azionari, ma in ogni caso sono stati ottenuti risultati interessanti che difficilmente sono stati raggiunti utilizzando altri modelli machine learning più classici. È risaputo come ripetuto più volte nel corso della trattazione, che i prezzi azionari dipendono da fattori più complicati rispetto alla semplice serie passata dei prezzi. Tuttavia la ratio dell'approccio utilizzato è basato sui fondamenti dell'analisi tecnica, secondo cui si crede che è possibile ottenere una stima dei prezzi futuri analizzando l'andamento dei prezzi passati perché quest'ultimi già incorporano il set informativo del mercato. A prescindere dalla visione che si predilige, la natura dei risultati ottenuti, non molto accurata, può dipendere dal fatto che la funzione dei prezzi sia tempo-variante. Per inciso una funzione è detta tempo-invariante (o stazionaria) se questa rimane costante nel tempo. In parole più semplici se consideriamo una funzione $f(x)$ e un ingresso x^*, applicando l'ingresso in qualunque istante di tempo la funzione restituirà sempre lo stesso risultato. Viceversa la tempo-variante è una funzione che muta in ogni istante di tempo. Pertanto dato un ingresso x^* questo può restituire un output diverso a seconda dell'istante di tempo in cui si applica. Il fatto che la funzione del prezzo sia tempo-variante comporta che il modello setta weight e bias sulla base di una funzione che muta potenzialmente istante per istante. Pertanto una rete che va bene oggi, può dover essere ricalibrata domani sulla base dei nuovi valori.

Resta il fatto che il modello ha restituito risultalti a dir poco strabilianti nel mappare la funzione input-output che gli è stata sottoposta. Questo può risultare non troppo utile per il problema che abbiamo affrontato (a causa della tempo-varianza della funzione) però lascia spiragli importanti per altri casi di studio finanziari che riguardano il mapping di funzioni, come ad esempio il pricing di opzioni americane, in particolare quelle esotiche.

6.2 LIMITI DELLE RETI NEURALI

I limiti del modello sono riconducibili di fatto alla struttura stessa delle Reti Neurali, quindi si parla di difetti che per costruzione e caratteristiche sono insiti in queste strutture.

Il limite principale di questa classe di modelli è il fatto che non esistono teoremi o metodi empirici che suggeriscano quali siano la struttura e i parametri ottimali che la rete deve avere per risolvere un determinato problema. La realizzazione della rete è stata fatta tramite esperimenti: si è scelta una struttura e sono state provate diverse configurazioni di iper-parametri e alla fine è stata scelta la rete che dava i risultati migliori. Tuttavia le combinazioni possibili sono praticamente illimitate: si parte dalla scelta della struttura, quindi rete neurale convoluzionale, fully connected o meno, rete neurale ricorrente di tipo LSTM, di tipo bidirezionale, può anche essere un misto di strutture; poi c'è da scegliere il numero di layer e per ogni strato bisogna valutare gli iper-parametri, quindi numero di percettroni che lo compongono, algoritmo di ottimizzazione, funzione di attivazione e quant'altro; oltre ai parametri che compongono la rete bisogna aggiungere ancora i parametri di addestramento, quindi numero di epoche, dimensione del batch ecc., tutte variabili che possono determinare risultati finali diversi. Lo svantaggio principale di questo limite è il fatto che non è possibile affermare che la struttura e la configurazione di parametri scelti siano i migliori per risolvere un problema. E questo vale sempre. È possibile solo ottenere dei risultati ottimi, ma per quanto buoni possano essere potrebbe sempre esserci una rete tra le praticamente infinite configurazioni possibili che ne dia di migliori.

Il secondo limite è il fatto che le Reti Neurali sono una classe di modelli assimilabili a Black Box, ovvero gli si dà un input e viene restituito un output, ma il modo in cui l'output viene generato è sconosciuto. Questo perché è impossibile risalire agli intricatissimi calcoli interni che portano al settaggio dei pesi e dei bias dei precettroni. Non è possibile in sintesi analizzare le elaborazioni che determinano i risultati.

Infine un ulteriore limite è il tempo di elaborazione necessario per l'addestramento della rete. Infatti il motivo per cui le Reti Neurali, la cui teoria e il loro potenziale era ben noto fin dagli anni '60, hanno iniziato a diffondersi solo in anni recenti, è perché solo in anni recenti i calcolatori hanno raggiunto una potenza tale da poter supportare lo sforzo di elaborazione richiesto. Questo diventa un doppio problema se si pensa alla correlazione con il primo limite, in cui si è detto che la rete va costruita per esperimenti. È chiaro che si hanno più probabilità di ottenere risultati soddisfacenti quante più configurazioni di rete si riescono a provare, pertanto il problema del tempo di elaborazione diventa in questo senso molto rilevante.

6.3 SVILUPPI FUTURI

Si conclude la tesi provando ad immaginare cosa può riservare il futuro, ovvero quali miglioramenti possono essere apportati al modello per ottenere dei risultati migliori.

Nell'esperimento si è provato a prevedere il prezzo futuro sulla base della serie storica dei prezzi passati, basandosi sui fondamenti dell'analisi tecnica economica secondo cui le informazioni di mercato sono già inglobate all'interno della serie passata dei prezzi. D'altro canto i sostenitori della teoria dei mercati efficienti asseriscono che non è possibile prevedere i prezzi futuri solo dall'analisi dei prezzi passati.

Partendo da questa considerazione possiamo immaginare di fare un merge con altre applicazioni finanziarie in cui il Deep Learning si sta facendo strada con risultati interessanti, in particolare il text mining e la sentiment analysis applicati in ambito finanziario. Infatti è possibile costruire una rete che scansioni siti di notizie finanziarie, come ad esempio Bloomberg, e sulla base di queste tracciare un profilo sentimentale degli investitori.

Quindi, alla struttura implementata durante questo lavoro, si potrebbe aggiungere un'ulteriore rete basata su text mining finanziario e sentiment analysis finanziaria al fine di aggiungere, oltre all'analisi della serie storica dei prezzi, anche l'analisi delle possibili intenzioni degli investitori, per riuscire ad ottenere dei risultati più accurati.

BIBLIOGRAFIA

[1]Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. Omer Berat Sezer, Mehmet Ugur Gudelek, Ahmet Murat Ozbayoglu.

[2]Deep Learning for Financial Applications: A Survey. Ahmet Murat Ozbayoglua, Mehmet Ugur Gudeleka, Omer Berat Sezera.

[3]Cit. Umberto Eco.

[4]Reti Neurali Artificiali: Teoria ed Applicazioni Finanziarie. Crescenzo Gallo, Università degli studi di Foggia.

[5]Immagini tratte da https://medium.com/@paolopiacenti1/le-4-funzioni-di-attivazione-neuronale-più-usate-negli-artificial-neural-network-41096953b85c

[6]Deep Learning for Time Series Forecasting. Jason Brownlee.

[7]Forecasting Fluctuations in the Financial Index Using a Recurrent Neural Network Based on Price Features. Yu-Fei Lin, Tzu-Ming Huang, Wei-Ho Chung and Yeong-Luh Ueng.

[8] https://keras.io/guides/functional_api/

[9]Dati tratti da https://www.investing.com