ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

PREDIRE LA BORSA VALORI CON LANGUAGE MODEL NEURALI

Elaborato in
Programmazione Di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Presentata da
Gian Luca Nediani

Terza Sessione di Laurea
Anno Accademico 2020 – 2021

PAROLE CHIAVE

Natural Language Processing

Machine Learning

Deep Neural Networks

Stock Prediction

Language Model

A chiunque mi sia stato vicino,
e mi abbia aiutato a raggiungere questo traguardo.

Introduzione

La **borsa valori** esiste da secoli ed è sempre stata oggetto di grande interesse vista la possibilità di sfruttarla per guadagnare grandi somme di denaro. Al fine di guadagnare denaro tramite la borsa, è necessario comprenderla ed interpretarla, facendo **predizioni** sul suo andamento futuro. Con il progresso di discipline quali la matematica, la statistica e poi l'informatica, la predizione del mercato azionario è passata da essere basata sulla semplice intuizione degli investitori, guidati dalle proprie valutazioni soggettive, ad essere trattata con rigore scientifico.

Nonostante lo scetticismo di molti studiosi circa la possibilità di prevedere l'andamemento del mercato azionario, apparentemente stocastico, esistono svariate teorie ipotizzanti la possibilità di utilizzare le **informazioni** conosciute sull'andamento del mercato per predirne i movimenti futuri.

L'avvento, nella seconda parte dello scorso secolo, dell'**intelligenza artificiale**, ha permesso di ottenere in svariati ambiti risultati rivoluzionari, tanto che oggi tale disciplina trova ampio impiego nella nostra vita quotidiana in molteplici forme. In particolare, grazie al **machine learning**, è stato possibile sviluppare sistemi intelligenti che apprendono in maniera autonoma grazie ai dati, in grado di modellare e comprendere problemi complessi. Visto il successo di questi sistemi, essi sono stati applicati anche all'arduo compito di predire la borsa valori, dapprima utilizzando i dati storici finanziari della borsa come fonte di conoscenza, e poi, con la messa a punto di tecniche di elaborazione del linguaggio naturale umano (**NLP**), anche utilizzando dati in linguaggio naturale come il testo di notizie finanziarie o l'opinione degli investitori.

Questo elaborato di tesi ha l'obiettivo di fornire una panoramica sull'utilizzo delle tecniche di machine learning, in particolare quelle di elaborazione del linguaggio naturale, nel campo della predizione del mercato azionario, partendo dalle tecniche più elementari per arrivare ai complessi modelli neurali che oggi rappresentano lo stato dell'arte. Dopo questa panoramica, verranno formalizzati il funzionamento e le tecniche che si utilizzano in generale per addestrare e valutare i modelli di machine learning, per poi effettuare un esperimento in cui a partire da dati finanziari e soprattutto testuali si tenterà di predire correttamente la variazione del valore del titolo di borsa S&P 500 utilizzando

un **language model** basato su una **rete neurale**, ovvero un modello di machine learning ispirato al funzionamento del cervello umano.

Il lavoro di tesi è suddiviso nei seguenti capitoli:

- **Capitolo 1** - Panoramica sul dominio del mercato azionario e sulle tecnologie informatiche utilizzate per comprenderne e predirne l'andamento;
- **Capitolo 2** - Analisi delle tecnologie informatiche legate ai modelli di machine learning e al loro addestramento;
- **Capitolo 3** - Modellazione dell'esperimento di predizione dell'indice di borsa S&P 500;
- **Capitolo 4** - Implementazione dell'esperimento;
- **Capitolo 5** - Analisi dei risultati ottenuti nell'esperimento.

Indice

Elenco delle figure

Capitolo 1

Panoramica e stato dell'arte nella predizione della borsa

In questo capitolo viene introdotto il problema della predizione di valori di borsa e vengono citate le principali tecnologie informatiche utilizzate per affrontarlo, comprese quelle che oggi rappresentano lo stato dell'arte.

1.1 Contesto

La **borsa valori** è un mercato in cui è possibile effettuare la compravendita di *valori mobiliari* (*security*) riferiti a società quotate in borsa. Le security hanno un valore che fluttua durante i periodi di apertura del mercato in risposta alla domanda e all'offerta che le interessa. La principale security che viene scambiata in borsa è il *titolo di capitale*, o *azione*. Le azioni rappresentano le quote della società a cui fanno riferimento. Chi le possiede viene chiamato azionista e a tutti gli effetti possiede una parte della società pari alla percentuale di azioni che detiene rispetto al totale di azioni esistenti. Il valore delle azioni varia a seconda del valore che il mercato attribuisce alla società in questione ed è dunque influenzato da numerosi fattori.

Oltre alle azioni delle singole società, esistono *indici di borsa* che comprendono nel loro titolo le azioni di molteplici società, selezionate secondo determinati criteri e pesate secondo il loro valore.

Il valore totale di tutte le aziende quotate nelle sessanta borse mondiali supera i 90 trilioni di dollari americani, rendendo l'interesse negli investimenti al fine di lucrare tramite questo mercato estremamente alto. Nei secoli in cui la borsa valori è esistita, gli investitori hanno provato in vari modi a predire l'andamento dei vari titoli azionari al fine di guadagnare denaro, tuttavia essa è regolata da complesse dinamiche che rendono arduo ottenere costanti profitti basandosi sulla propria intuizione o su semplici metodologie empiriche. Per

questo motivo, con il progredire delle discipline scientifiche, quali la matematica e la statistica, si è provato a sviluppare tecniche rigorose che potessero modellare il mercato e dunque prevedere il suo andamento futuro.

Oltre al lucro, la predizione della borsa valori è oggetto di interesse da parte dei ricercatori di varie discipline, come la finanza e l'informatica, per via della sua elevata complessità, che permette di sperimentare nuove teorie e tecniche su un problema ostico.

La nozione di base per ottenere un guadagno tramite **compravendita** sul mercato azionario è la seguente: se si prevede che un certo titolo azionario acquisterà valore nel futuro, lo si **acquista** ora per rivenderlo poi a un prezzo più alto. Viceversa, se si prevede che un certo titolo perderà di valore, lo si **vende allo scoperto**, ovvero lo si "prende in prestito", lo si vende e lo si riacquista in seguito a un prezzo minore, per restituirlo al proprietario originale ottenendo un profitto. Queste sono le tipologie di compravendita speculativa che possono essere effettuate in borsa:

- **Long-term trading**: si acquista/vende allo scoperto un titolo con l'obiettivo di chiudere l'operazione in una giornata di trading successiva;
- **Day trading**: si acquista/vende allo scoperto un titolo all'apertura dei mercati in una data giornata con l'obiettivo di chiudere l'operazione alla chiusura delle stessa giornata;
- **Intraday trading**: si acquista/vende allo scoperto un titolo in una data giornata con l'obiettivo di chiudere l'operazione nella stessa giornata, a ore o addirittura minuti dall'inizio dell'operazione di trading. Tale strategia consente di effettuare più operazioni all'interno della stessa giornata ma richiede una modellazione più capillare delle oscillazioni del mercato

Gli studiosi scettici sulla possibilità di predire l'andamento del mercato [1] si rifanno alla teoria matematica di **random walk** [2] (passeggiata aleatoria), sostenendo che i cambiamenti nel valore di un titolo di borsa sono casuali e non riconducibili a nessun trend predicibile. In termini probabilistici, i teorici del random walk ritengono che le variazioni dei valori azionari siano variabili casuali indipendenti e dalla distribuzione uniforme. Ciò comporta che tali variabili godano di mancanza di memoria, ovvero, non siano influenzate dal passato, rendendo l'intera idea di predire il mercato non percorribile.

Nonostante un generale scetticismo, esistono nella letteratura scientifica svariate teorie che supportano la possibilità di predire l'andamento del mercato con un certo grado di precisione, permettendo dunque di ottenere profitti [3]. Tali teorie accettano solo parzialmente la teoria di random walk, sostenendo che ci sono comunque alcuni fattori prevedibili che permettono di sfruttare le informazioni passate per predire la borsa nel futuro prossimo (intraday e

day trading) e a lungo termine (long-term trading). L'esistenza di modelli in grado di predire l'andamento di indici azionari con alta precisione, sebbene in determinate condizioni, come quelli che verranno mostrati in questo capitolo, è un'ulteriore conferma della possibiltà di predire il mercato.

1.2 Formalizzazione del problema

Riconoscendo la possibilità di discuterlo come un problema di regressione (predicendo il valore esatto di un titolo), in questo elaborato il task di predire l'andamento della borsa valori viene trattato esclusivamente come un problema di **classificazione**.

Con il set di dati $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, dove $X = (x_1, x_2, \ldots, x_n) \in R$ è il vettore di dati in input e $Y \in 0, 1, ..., j$ è una variabile discreta, con j pari al numero totale di classi nel problema, l'obiettivo è costruire un modello di classificazione $C(x)$ che predìca la corretta Y. In altre parole, l'obiettivo è costruire una funzione che prendendo in input i dati disponibili, classifichi correttamente l'istanza con la corrispondente etichetta fra quelle in Y.

Nel task in questione, la variabile Y da predire è binaria e pari a 1 se all'istante su cui si effettua la predizione l'indice di borsa in analisi ha un prezzo superiore rispetto all'istante di riferimento ed è pari a 0 viceversa ($Y \in 0, 1$). Nella casistica di day trading, ovvero quella di cui si occuperà l'esperimento, la variabile binaria Y da predire è pari a 1 se in un dato giorno l'indice in analisi chiuderà a un prezzo superiore rispetto al valore di apertura, ed è pari a 0 viceversa.

1.3 Dati finanziari e indicatori tecnici

I primi dati ad essere stati presi in considerazione per modellare algoritmi in grado di predire l'andamento del mercato sono quelli **numerici** relativi all'andamento passato della borsa. Alcuni esempi di informazioni di questa tipologia, per un dato titolo di borsa e una data giornata di trading, sono ad esempio: valori di apertura e chiusura, valori di picco e di minimo quotidiani, volume delle operazioni. Questi dati sono pubblicamente disponibili per tutti i titoli quotati in borsa. Per predire un dato titolo, è anche possibile utilizzare dati passati relativi ad altri titoli, qualora vi sia correlazione.

A partire dai dati appena citati, è possibile costruirne altri detti **indicatori tecnici di performance**. Alcuni esempi di indicatori tecnici di performance sono:

- **Media mobile**;

- **Average Directional Movement Index (ADX)**, indicatore della forza di trend;
- **Bande di Bollinger**, indicatori di volatilità di un indice;
- **Relative Strength Index (RSI)**, indicatore utilizzato per segnalare la forza di un indice rispetto a un periodo prestabilito in modo tale da evidenziare le zone di ipercomprato e ipervenduto.

1.3.1 Serie temporali

I dati finanziari storici e gli indicatori tecnici possono essere concatenati temporalmente per formare delle **serie** di valori. Analizzando i trend di queste serie sia possibile modellare l'andamento futuro della borsa. L'analisi di serie temporali di dati storici per eseguire previsioni sul futuro è chiamata *time-series forecasting*.

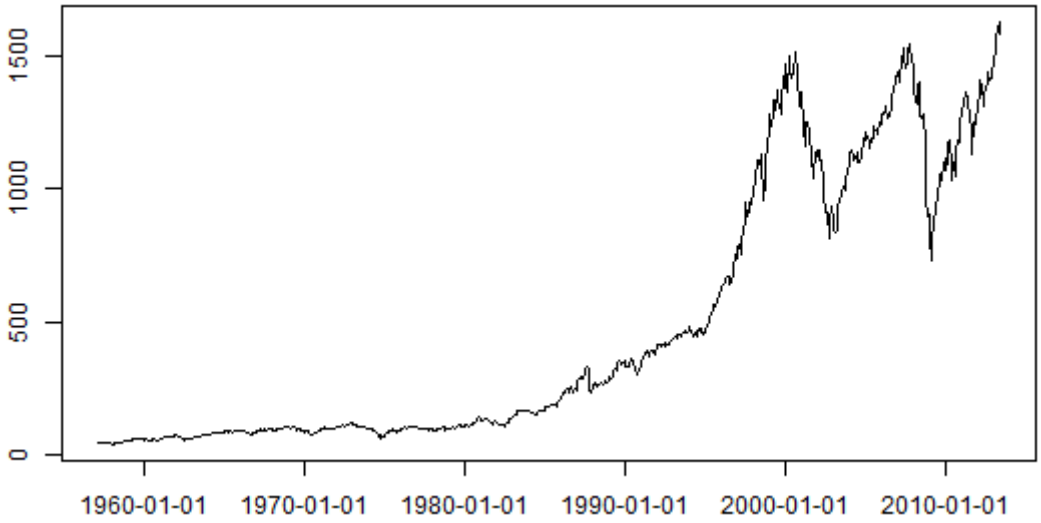

Figura 1.1: Serie temporale dei valori di chiusura dell'indice S&P 500 dagli anni cinquanta del '900

1.4 Predizione borsa valori con algoritmi di machine learning

La gigantesca mole di dati pubblicamente disponibili rendeva l'analisi di serie temporali un compito arduo da svolgere manualmente per gli esseri umani; con l'avvento nella seconda parte del ventesimo secolo delle scienze informatiche,

e di conseguenza della disciplina del **machine learning**, si è creata la possibilità di automatizzare il processo di analisi di grandi quantità di dati per eseguire previsioni.

I modelli di apprendimento automatico, o machine learning, sono algoritmi addestrati su enormi quantità di dati al fine di trovare soluzioni generali a problemi complessi. L'apprendimento di un modello avviene minimizzando la funzione di errore rispetto ai dati utilizzati durante l'addestramento. Le performance del modello vengono poi valutate su dati mai visti durante l'addestramento, al fine valutare la capacità di questo di modellare il problema generale. L'aumentare della potenza di calcolo dei calcolatori moderni ha reso possibile lo sviluppo di modelli sempre più complessi ed in grado di utilizzare sempre più dati come input, ma i primi modelli di machine learning sono basati su semplici funzioni lineari.

1.4.1 Modello lineare

Il modello più semplice che possa predire l'andamento di un indice di borsa, dunque eseguire una classificazione binaria, è un classificatore basato su iperpiano. Questo tipo di modello classifica le istanze passate in input separandole con un piano di separazione che massimizzi la separazione tra le classi, e dunque minimizzi l'errore del modello.

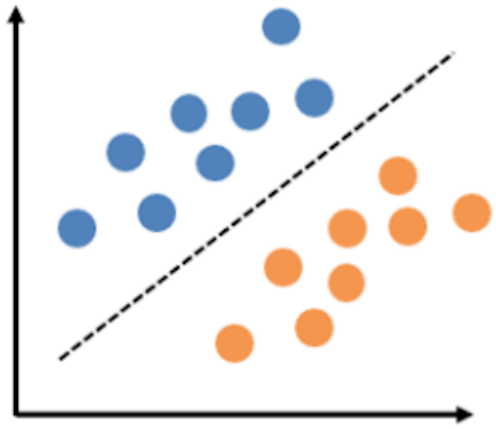

Figura 1.2: Esempio di classificatore lineare binario di dati bidimensionali

A causa della complessità delle dinamiche che influenzano l'andamento della borsa, un modello lineare come quello mostrato in figura difficilmente sarà in grado di ottenere buoni risultati.

1.4.2 Modello non lineare

Una soluzione per un problema che presenta classi non separabili linearmente è di trasformare lo spazio dei dati tramite metodi kernel [4], in modo che essi diventino linearmente separabili. Un esempio di modello di classificatore che adotta questa soluzione è **Support Vector Model (SVM)**, ed esistono in letteratura esempi dell'utilizzo di tale modello per ottenere buone performance nella predizione della borsa con abilità di generare profitto [5]. Nonostante lo sviluppo di architetture concettualmente più complesse, **SVM** continua oggigiorno ad essere utilizzato sia nella predizione del mercato che in altri task, in quanto riesce ad ottenere buoni risultati utilizzando meno tempo e risorse per l'addestramento rispetto a modelli più avanzati, come ad esempio le reti neurali.

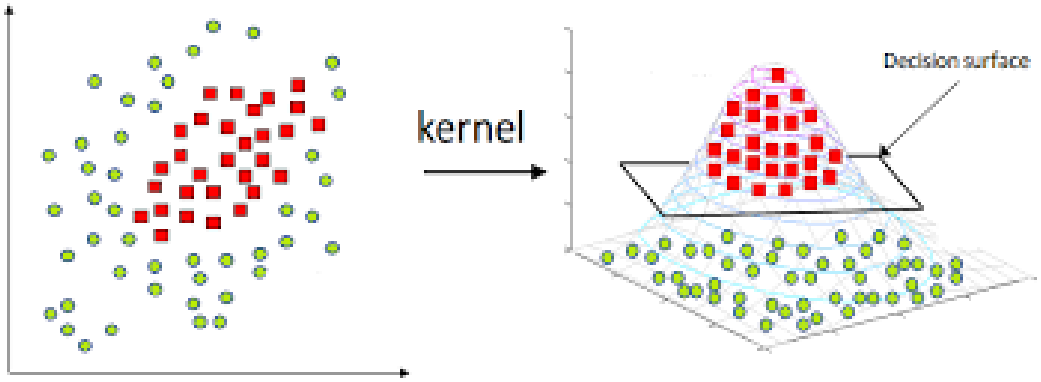

Figura 1.3: Illustrazione di kernel trick: i dati diventano separabili con un iperpiano

1.5 Predizione borsa valori con reti neurali

Con l'avvento nelle scienze informatiche dell'intelligenza artificiale, vengono introdotte le *reti neurali artificiali (ANN)*, modelli predittivi più complessi e più costosi da addestrare rispetto agli algoritmi visti in precedenza, ma con maggiori capacità di modellare relazioni complesse fra dati in input e output. Le reti neurali artificiali sono ispirate alle loro controparti biologiche e sono composte da molteplici nodi, chiamati neuroni, che contribuiscono al processo decisionale e sono organizzati in strati, chiamati layer. Se per problemi intrinsecamente più semplici e rappresentati da dati più strutturati, gli algoritmi tradizionali di machine learning avevano già ottenuto risultati rivoluzionari, è solo con lo sviluppo di reti neurali che problemi più complessi e meno strutturati

come il riconoscimento di immagini, l'elaborazione del linguaggio naturale e la guida autonoma vengono affrontati con successo dai ricercatori informatici. Ha dunque senso applicare le reti neurali a un dominio complicato come quello del mercato azionario.

1.5.1 Reti neurali feed-forward

Le reti neurali *feed-forward* sono il tipo più semplice di rete neurale, sono composte da un layer di input e uno di output, oltre che da uno o più layer interni, chiamati *hidden layers*, che processano i dati apprendendone le relazioni. Nel caso di più hidden layers, una rete neurale è definita profonda (*deep neural network*).

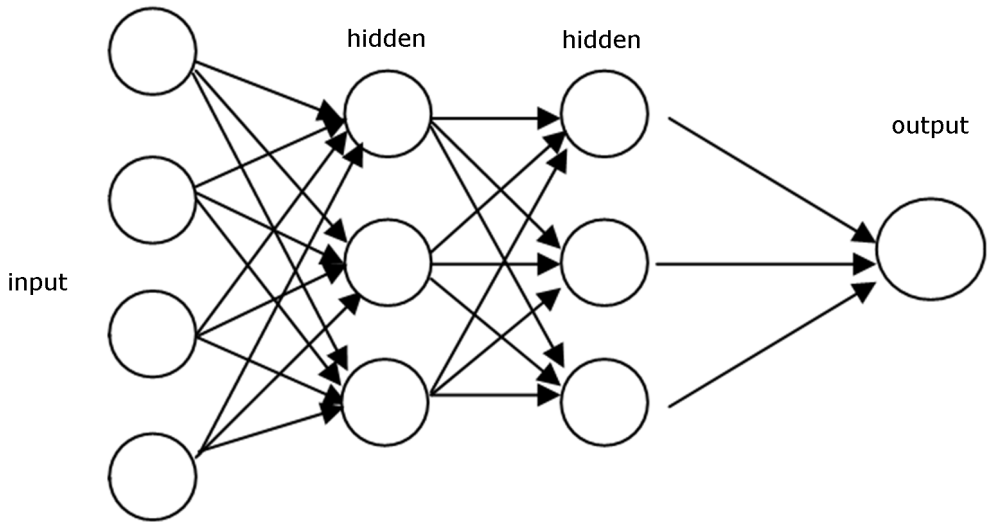

Figura 1.4: Esempio di rete neurale profonda con due hidden layer.
Fonte: www.medium.com

Come suggerisce il nome, le reti feed-forward sono caratterizzate da uno stream unidirezionale delle informazioni: tutte le connessioni fra i nodi nella rete si muovono solo in avanti: i dati scorrono dal layer di input attraverso gli hidden layer fino al layer di output in una sola direzione, senza cicli o relazioni più complesse fra i nodi.

[6] approfondisce l'applicazione delle reti neurali di tipo feed-forward al dominio della predizione finanziaria e passa in rassegna alcuni interessanti risultati: in determinate condizioni questo tipo di modello è in grado di ottenere ottimi risultati nel predire la borsa valori.

1.5.2 Reti neurali ricorrenti

Le reti neurali ricorrenti utilizzano la stessa architettura delle reti feed-forward, con la differenza che i nodi possono avere anche connessioni cicliche, oltre che in avanti. Tramite le connessioni cicliche, le reti ricorrenti possono ricevere in input dati temporali ed elaborarli sequenzialmente lungo l'asse temporale, mantenendo a ogni step uno stato di memoria sugli step precedenti. Questo tipo di rete neurale si presta dunque particolarmente a dati in input di tipo sequenziale, come è il caso nelle serie temporali di dati finanziari.

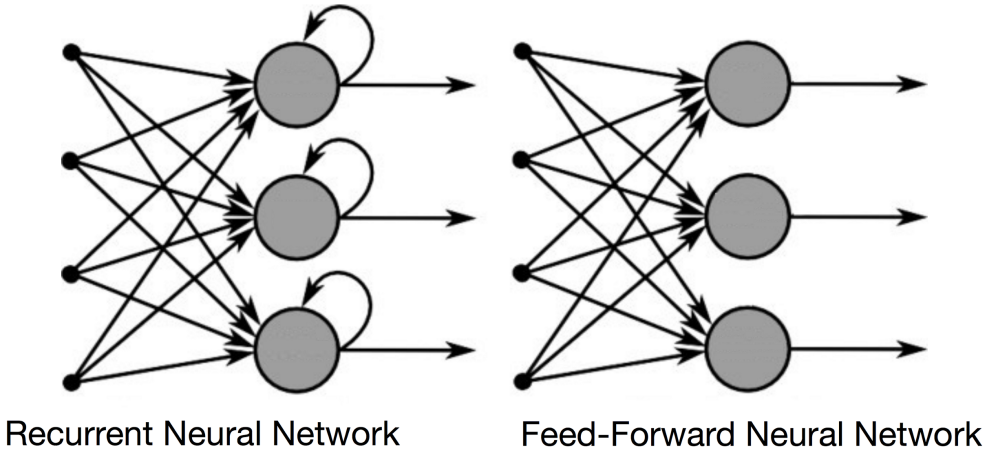

Figura 1.5: Confronto tra strato ricorrente e strato feed-forward
Fonte: www.medium.com

[7] prende in esame un set di serie temporali di dati finanziari relativi all'andamento dell'indice *Dow Jones Industrial Average (DJIA)* dal 2000 al 2017 e mostra come una architettura RNN ottenga performance abbondantemente migliori confrontata a una semplice rete feed-forward. Il modello basato su RNN riesce a ottenere una accuracy nella previsione dell'andamento di tale indice dell'83%, profitti pari a più di cinque volte l'investimento iniziale in una simulazione di mercato.

1.5.3 Reti neurali convoluzionali

Le reti neurali convoluzionali (CNN) vengono introdotte alla fine degli anni ottanta del '900 [8]. Esse si ispirano al funzionamento della corteccia visiva dei mammiferi e sono perciò pensate per processare immagini. Per fare ciò

sfruttano dei filtri di convoluzione (kernel). Nonostante la loro concezione legata a task relativi a immagini, vengono successivamente utilizzate con successo anche per input di serie temporali. Gli input temporali infatti, vengono disposti a matrice come si farebbe per i pixel di un'immagine: a ogni riga corrisponde uno step temporale e a ogni colonna una feature dei dati.

[9] confronta una architettura CNN con una rete neurale feed-forward e un modello SVC in un task di predizione a breve termine dell'indice S&P 500 e mostra come il modello convoluzionale abbia performance migliori degli altri due.

1.6 Nuove tecnologie e nuovi tipi di dati

Come visto, le reti neurali, grazie alla loro abilità di modellare relazioni complesse fra i dati ottengono risultati generalmente migliori rispetto agli algoritmi utilizzati in precedenza. Un'ulteriore possibilità offerta dalle reti neurali è la possibilità di utilizzare, oltre ai dati finanziari e tecnici, che sono dati strutturati, anche nuovi tipi di dati non strutturati. I dati storici numerici sono stati i primi ad essere utilizzati nel task, e ancora oggi sono quelli più utilizzati. Gli sviluppi nei seguenti campi hanno però permesso di integrare ai dati numerici anche nuovi dati di tipo testuale:

- **Text mining**, estrazione , spesso automatizzata, e conseguente elaborazione di grandi quantità di dati da fonti testuali. Il web è ricco di dati pubblicamente disponibili e di potenziale utilità in vari ambiti, compresa la predizione della borsa;
- **Natural language processing (NLP)**, branca dell'informatica che si occupa di realizzare sistemi intelligenti in grado di comprendere ed elaborare il linguaggio naturale umano. Tali sistemi intelligenti hanno svariati utilizzi pratici quali la traduzione, riassunzione e classificazione di documenti, la possibilità di interagire con gli esseri umani comprendendone domande e bisogni, la generazione di testo coerente, il miglioramento di motori di ricerca e molti altri. Il **NLP** interessa oltre che l'informatica, anche discipline come la linguistica e la neuroscienza: spesso i modelli sviluppati dai ricercatori informatici si ispirano ai meccanismi di elaborazione del linguaggio del cervello umano.

I dati di tipo testuale possono essere una fonte di conoscenza di valore per addestrare i modelli di predizione. Informazioni come l'opinione degli investitori, catturata su social media o blog, le news finanziarie che contengono analisi e eventi finanziari, le ricerche web degli investitori e altre informazioni di tipo

testuale possono quindi essere sfruttate per aumentare la capacità predittiva dei modelli.

1.6.1 Sentiment degli investitori

Sul web sono presenti grandi quantità di dati testuali relativi all'opinione e al sentimento degli investitori circa il mercato azionario. Le tecnologie oggi presenti nel campo del *text mining* permettono di estrarre questi dati, che possono poi essere interpretati da modelli predittivi tramite varie tecniche di *natural language processing*. L'avvento di modelli più complessi basati su reti neurali nel campo della comprensione del linguaggio naturale permette di comprendere l'opinione degli investitori anche quando questa viene espressa in maniera meno esplicita, si pensi ad esempio a forum online o social media come Twitter, dove gli utenti spesso si esprimono in maniera sintetica trascurando la grammatica.

La pratica di elaborare dati testuali per estrarre contenuti semantici in grado di esprimere l'opinione e il sentimento di individui riguardo a un certo argomento è chiamata **sentiment analysis**. L'incidenza del sentimento degli investitori sull'andamento del mercato è esplorata e verificata in [10].

Gli autori di [11] riescono ad ottenere una accuracy dell'87.6% nel predire l'antamento del Dow Jones Industrial Average (DJIA) grazie all'analisi delle serie temporali del sentiment colletivo degli utenti Twitter riguardo a tale indice.

1.6.2 Eventi strutturati

Testate giornalistiche specializzate nel reporting, ovvero il riportare eventi ed avvenimenti senza esprimere al riguardo una opinione, pubblicano ogni giorno migliaia di articoli che contengono avvenimenti relativi alla finanza. Questi avvenimenti influenzano l'andamento della borsa valori [12]. Se tali eventi influenzano il mercato, essi possono essere utilizzati come input in un modello predittivo. Gli autori di [13], estraggono dati testuali strutturati da decine di migliaia di articoli di reporting finanziario delle testate giornalistiche Bloomberg e Reuters e li elaborano tramite una rete neurale costruendo embedding di eventi strutturati che vengono poi dati utilizzati come input per un classificatore basato su rete neurale convoluzionale. Il loro modello utilizza per effettuare le predizioni dati a lungo, medio e breve termine ed è in grado di predire con una accuracy del 65% l'andamento del titolo S&P 500 nel periodo coperto dal dataset utilizzato, ottenendo profitti pari al 160% dell'investimento iniziale in una simulazione di 9 mesi di compravendita basata su un algoritmo che sfrutta intraday trading e day trading.

1.6.3 News finanziarie

In [13] vengono costruiti eventi strutturati di semplice interpretazione semantica per semplificare alla rete neurale di classificazione il compito di comprendere il linguaggio naturale umano. Tuttavia, modelli di NLP più complessi possono estrarre direttamente il significato dal testo non strutturato. Come nel caso della sentiment analysis degli investitori, è dunque possibile utilizzare la grande mole di news finanziarie disponibile sul web per estrarre eventi, tendenze e opinioni che influenzano l'andamento della borsa

[14] è un interessante esempio dell'utilizzo di testo nel task in questione, in quanto utilizza come dati in input, con ottimi risultati, le trascrizioni delle earning calls, ovvero le videoconferenze con cui le aziende annunciano agli investitori i risultati finanziari ottenuti nell'ultimo periodo.

Capitolo 2

Tecnologie disponibili

In questo capitolo vengono illustrate in maniera più completa le tecnologie che nel primo capitolo erano state soltanto accennate e che verranno impiegate nei capitoli successivi per svolgere l'esperimento.

2.1 Machine Learning

L'approccio classico nell'informatica per risolvere un problema è scrivere un algoritmo, ovvero una lista di regole che l'autore sa risolveranno il problema. Nel caso di problemi complessi, la lista di regole diverrebbe troppo lunga per tale approccio; inoltre non per tutti i problemi esiste una soluzione conosciuta applicabile. In questi casi si può fare uso di algoritmi di machine learning, o apprendimento automatico.

Nel machine learning, invece di descrivere una serie di regole per risolvere un problema, si addestra un modello fornendogli dei dati pertinenti al problema facendo sì che esso impari in autonomia cercando la migliore soluzione possibile. I modelli di machine learning sono stati applicati con successo in numerosi ambiti ed eccellono in contesti dove è necessario effettuare predizioni sul futuro, sulla base di dati conosciuti riguardanti il passato. Ciò permette di automatizzare o ottimizzare processi, ma anche di trovare soluzioni a problemi precedentemente irrisolti.

La larga applicazione di modelli di apprendimento automatico va di pari passo con la disponibilità sempre crescente di grandi quantità di dati, favorita dalla digitalizzazione in corso nel mondo contemporaneo. In qualsiasi ambito in cui si abbiano dati a disposizione e problemi complessi da risolvere è possibile applicare il machine learning.

2.1.1 Tipologie di apprendimento

I vari modelli di machine learning possono essere categorizzati innanzitutto in base a quanta supervisione richiedono durante l'apprendimento, e si suddividono in: **apprendimento supervisionato**, **apprendimento non supervisionato**, **apprendimento semi-supervisionato** e **apprendimento per rinforzo**.

Apprendimento supervisionato

Nell'apprendimento supervisionato, il modello da addestrare riceve in input sia i **dati a disposizione** X, con i quali dovrà effettuare la predizione, sia le **etichette** (label) Y, ovvero gli output che è chiamato a predire. Avendo a disposizione gli output attesi, il modello può confrontarli con le proprie predizioni ed apprendere in base ai propri errori, migliorando progressivamente e diminuendo l'errore. Alcuni esempi di modelli ad apprendimento supervisionato sono:

- **Modelli di classificazione**: le etichette Y sono le classi a cui appartengono le istanze
- **Modelli di regressione**: le etichette Y sono dei valori numerici in un intervallo continuo

Come spiegato nel *capitolo 1*, in questo elaborato il task della predizione della borsa valori viene affrontato come un problema di classificazione binaria: classe 1 se il valore di un titolo aumenta, classe 0 se diminuisce. Tutti i modelli mostrati nel capitolo 1 fanno uso di apprendimento supervisionato. La seguente illustrazione mostra le etichette (label) riferite a un problema di classificazione dove si vuole predire se degli studenti supereranno un esame o meno.

	X(Input)			Y(Output)
Student	Test1 marks	Test2 Marks	Study hours	Final result
1	30	35	4	Pass
2	42	45	6	Pass
3	20	17	1	Fail
4	45	48	6	Pass
5	25	22	2	Pass
6	34	40	2	Pass
7	49	47	6	Pass
8	17	10	0	Fail
9	25	20	1	Fail
10	35	38	3	Pass

Figura 2.1: Esempio di dataset per apprendimento supervisionato
Fonte: www.medium.com

Apprendimento non supervisionato

Nell'apprendimento non supervisionato i dati forniti al modello non sono etichettati, ovvero non comprendono valori di output desiderati Y. I modelli di questo tipo analizzano i dati in input per trovare delle relazioni all'interno di essi. Alcuni esempi di modelli ad apprendimento non supervisionato sono:

- **Modelli di clustering**: il modello analizza i dati e cerca di raggruppare le istanze in classi secondo delle caratteristiche comuni da esso individuate;

- **Modelli di anomaly detection**: il modello trova le istanze che sono significativamente diverse dalla massa, chiamate outliers.

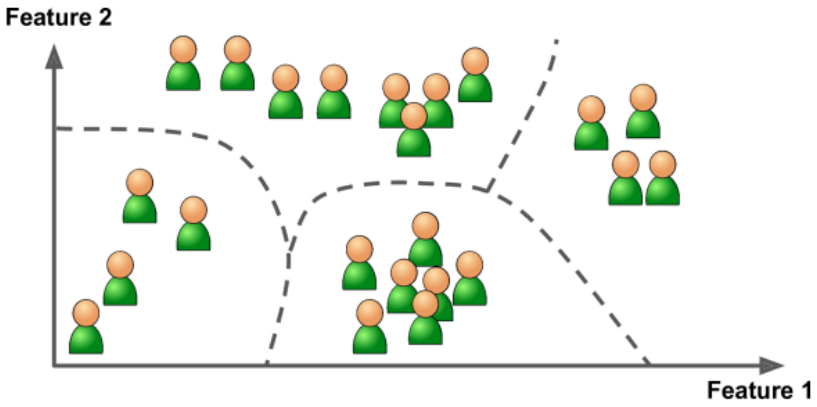

Figura 2.2: Illustrazione clustering dei dati su 2 feature
Fonte [15]

Apprendimento semisupervisionato

Essendo etichettare le istanze un compito molto dispendioso, esistono modelli ad apprendimento semi-supervisionato. In questo caso solo una piccola parte delle istanze in input è etichettata. Il modello etichetta le rimanenti istanze in maniera automatica apprendendo le caratteristiche di quelle che sono già etichettate.

Apprendimento per rinforzo

In questa casistica si ha un agente che osserva l'ambiente in cui si trova ed è libero di eseguire azioni in esso. Scegliendo le azioni corrette, l'agente viene ricompensato, viceversa scegliendo quelle sbagliate viene punito. L'agente, o modello, apprende cercando di trovare la strategia, chiamata policy, che massimizzi le ricompense.

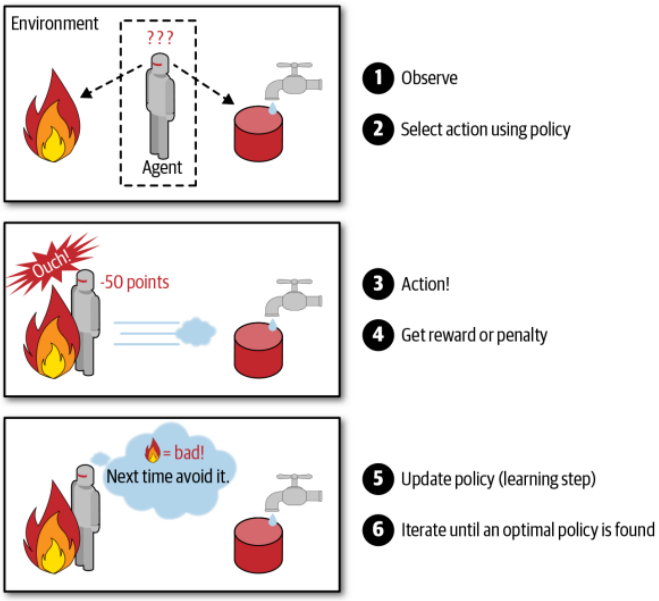

Figura 2.3: Illustrazione di apprendimento per rinforzo
Fonte [15]

Un celebre esempio di modello ad apprendimento per rinforzo è la rete neurale AlphaGo [16] sviluppata da DeepMind nel 2017. Tale rete aveva l'obiettivo di diventare capace di giocare al complesso gioco da tavolo Go ed ha ricevuto in input soltanto le regole del gioco. Giocando milioni di partite contro se stesso, la rete neurale ha appreso la policy migliore per vincere ed una volta finita la fase di apprendimento è stata in grado di battere il campione del mondo Ke Jie.

2.1.2 Minimizzare l'errore

Come spiegato sopra, nell'apprendimento supervisionato il modello ha a disposizione i valori di output desiderati e li utilizza per minimizzare l'errore che commette. Ciò richiede una misura che quantifichi l'errore, ovvero una **funzione d'errore**. Per la regressione la funzione d'errore più utilizzata è lo scarto quadratico medio; nei casi di classificazione invece si fa solitamente uso della funzione di entropia incrociata.

Un metodo matematico spesso impiegato per trovare il minimo di funzioni è la **discesa del gradiente**. L'idea di fondo di questo metodo è di modificare **iterativamente** i parametri della funzione d'errore, inizializzati a valori casuali, secondo la seguente formula, fino a convergere a un punto di minimo:

1. Si calcola la funzione d'errore ed il valore del relativo gradiente per i parametri;
2. Si ritoccano i parametri di un valore proporzionale al gradiente in direzione inversa a questo;
3. Si procede iterativamente fino alla convergenza.

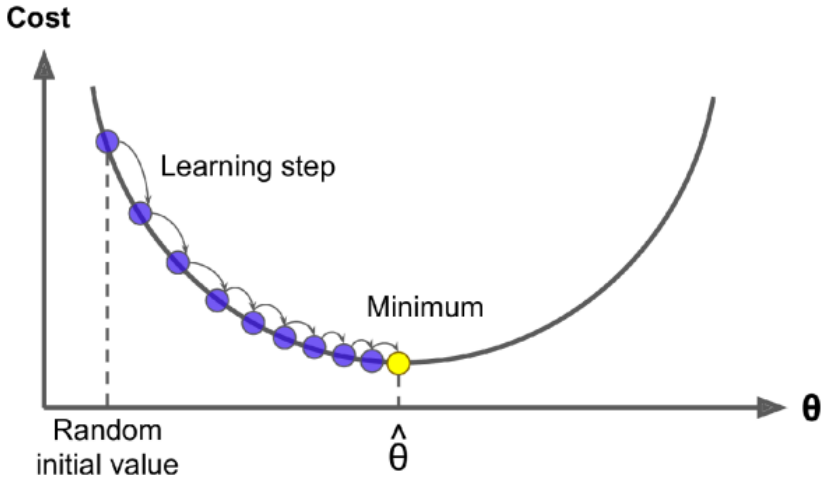

Figura 2.4: Illustrazione di discesa del gradiente
Fonte [15]

La dimensione della discesa a ogni iterazione si chiama *step size*, o *learning rate*, e costituisce un iperparametro del modello. Uno *step size* troppo piccolo farà sì che il modello impieghi molto tempo per convergere alla soluzione ottimale. D'altro canto, con uno *step size* troppo grande si corre il rischio che la discesa del gradiente "salti" il punto di minimo. è anche possibile avere un *learning rate* variabile: nelle prime iterazioni è più alto per velocizzare l'addestramento e diminuisce mano a mano che ci si avvicina al punto di minimo per non correre il rischio di oltrepassarlo.

2.1.3 Addestramento, testing e validazione

L'insieme di dati a disposizione per lavorare su un modello di machine learning, chiamato **dataset**, viene utilizzato sia per addestrare il modello che per valutarne le performance; per questo motivo viene suddiviso in **training set**, ovvero la parte di dati utilizzata per l'addestramento, e **test set**, ovvero la parte di dati utilizzata per valutare le performance del modello una volta finito l'addestramento. I dati del test set sono sconosciuti al modello durante

l'addestramento, e la valutazione delle performance avviene ad addestramento terminato, dunque il modello non ha più la possibilità di apprendere nuove informazioni; in questo modo si possono valutare le capacità del modello di risolvere il problema generale, e non solo la sua abilità di "memorizzare" le caratteristiche del training set. La ripartizione del dataset nei due sottoinsiemi è una decisione progettuale: più dati sono nel training set, maggiori dati a disposizione ha il modello per imparare, tuttavia se il test set non è sufficientemente grande, esso non sarà rappresentativo del problema generale. Nel caso di serie temporali, come nella predizione della borsa, i dati sono divisi **temporalmente**.

Ricerca degli iperparametri

L'addestramento di un modello di apprendimento automatico presenta spesso svariati **iperparametri**, ovvero parametri modificabili che controllano il processo di apprendimento. La mole di iperparametri fa sì che ci siano spesso decine di configurazioni possibili risultanti dalla loro regolazione, rendendo difficile trovare quelli ottimali manualmente. In questi casi si tiene da parte una frazione del training set, chiamata **validation set**, e si addestra ogni possibile configurazione sul training set, verificando poi le performance sul validation set. Una volta trovata la configurazione migliore, la si utilizza per addestrare il modello su tutto il training set, ottenendo così il modello ottimale, che viene poi testato sul test set.

2.1.4 Problematiche del machine learning

Sebbene grazie alle tecniche di machine learning si siano raggiunti risultati rivoluzionari in svariate discipline, ci sono comunque dei limiti legati ad esso di cui bisogna tenere conto:

- **Overfitting**, questo fenomeno si verifica quando il modello impara molto bene a modellare i dati del training set, ma non è in grado di generalizzare la soluzione, mostrando quindi difficoltà su dati sconosciuti. In questo caso, si noteranno ottime performance sui dati di training e performance molto più basse sui dati di test. Quando si addestra un modello, l'obiettivo è che esso arrivi a comprendere una soluzione generale, ma spesso il modello tende a memorizzare la soluzione ottimale per performare bene sui dati di training. Per scongiurare ciò si fa uso di tecniche di regolarizzazione, che impediscono al modello di diventare troppo complesso e aderire troppo ai dati di training, e tecniche early stopping, che arrestano l'addestramento quando viene riscontrata una tendenza all'overfitting.

- **Rappresentatività dei dati a disposizione**, per ottenere buoni risultati in casi d'uso reali, è necessario che i dati utilizzati per l'addestramento e la valutazione siano rappresentativi dei dati del mondo reale. Ad esempio, se in un problema di classificazione si utilizzano dati sbilanciati, dove una classe è sovrarappresentata rispetto alla realtà, il modello tenderà a predire spesso quella classe, risultando efficace nei test, ma una volta utilizzata in un contesto reale non otterrà buoni riscontri.

- **Quantità di dati a disposizione**, i modelli di machine learning riescono generalmente a ottenere ottimi risultati, ma solo se hanno un numero sufficiente di istanze su cui imparare. La quantità di dati necessari per ottenere un buon modello aumenta con la difficoltà del problema che si vuole modellare. Se si hanno pochi dati a disposizione, il rischio è di spendere invano tempo e denaro per sviluppare un modello efficace su essi, quando in realtà sarebbe più efficiente raccogliere ulteriori dati. A dimostrazione di ciò, [17] mostra come su un difficile problema di NLP, mano a mano che aumenta la quantità di dati di addestramento anche modelli più semplici riescono a raggiungere le performance dei modelli più complessi.

2.2 Deep Learning

Le tecniche di apprendimento automatico spiegate nella sezione precedente riescono generalmente a ottenere ottimi risultati. Tuttavia, di fronte a problemi complessi, i cui dati non sono strutturati, esse hanno difficoltà. Il **deep learning** è la pratica di addestramento delle reti neurali, introdotte nel *capitolo 1*. Tali reti, se opportunamente addestrate, riescono in task complessi a ottenere performance decisamente migliori dei tradizionali modelli di machine learning. Nonostante le prime nozioni di deep learning siano state messe a punto decine di anni fa, in passato raramente se ne faceva ricorso, in quanto questo tipo di addestramento richiedeva troppe risorse computazionali rispetto al classico machine learning. Oggi invece, grazie ai seguenti fattori è molto più facile addestrare reti neurali in tempi brevi:

- Generale aumento della capacità computazionale dei calcolatori;

- Diffusione a prezzi economici delle *GPU* (Graphic Processing Unit), acceleratori grafici estremamente efficienti nella computazione di operazioni in parallelo, di cui il deep learning fa utilizzo;

- Diffusione dei servizi di *cloud computing*, che permettono di accedere in remoto all'hardware con la potenza di calcolo necessaria all'addestramento di reti neurali.

Grazie a ciò il deep learning ha oggi un abbondante impiego sia nella ricerca scientifica che in applicazioni industriali, ed ha rivoluzionato vari campi dove il machine learning presenta limitazioni, come ad esempio: visione artificiale, elaborazione di linguaggio naturale, guida autonoma e speech recognition. Visto il potenziale di architetture basate su reti neurali, non stupisce come negli ultimi anni esse siano state utilizzate anche per affrontare il task della predizione della borsa.

2.2.1 Architettura di una rete neurale

In questa sezione viene spiegata l'architettura di una semplice rete neurale feed-forward, reti neurali più complesse come le RNN e le CNN condividono la stessa architettura generale, con alcune differenze. Come accennato nel **capitolo 1**, una rete neurale è formata da vari strati di nodi, chiamati layer ed ha sempre un layer di input e uno di output, oltre che almeno un layer di elaborazione fra input e output, chiamato hidden layer. Qualora ci siano due o più hidden layer, la rete neurale viene definita *profonda* (*deep neural network*). Utilizzare più hidden layer serve nei task più complessi per modellare il problema a diversi livelli di astrazione catturando relazioni diverse tra i dati. Ogni layer è formato da più *nodi*, o *neuroni*, che sono l'unità di base delle reti neurali. Qualora tutti i neuroni di un layer siano connessi a tutti i neuroni del layer precedente, si parla di layer *fully-connected*.

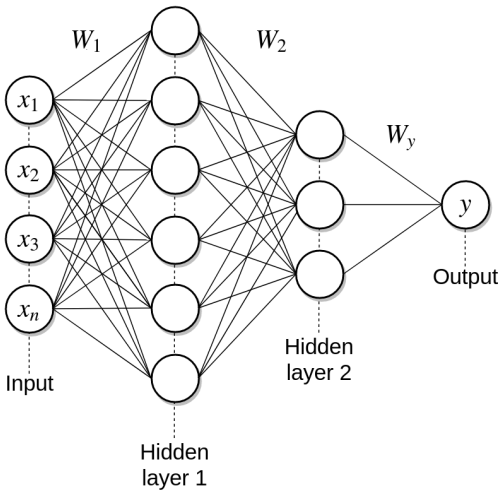

Figura 2.5: Schema di rete neurale profonda fully connected

I nodi del layer di input non fanno altro che passare al primo hidden layer i dati, e il loro numero è pari al numero di feature delle istanze del problema.

Negli hidden layer, i nodi ricevono l'input dai nodi a cui sono connessi nel layer precedente ed effettuano una **somma pesata** dei valori in input x, secondo i propri parametri w (che nel training verranno modificati a ogni step per minimizzare la funzione di errore). Al valore della somma pesata, è applicata una funzione di attivazione, che permette di modellare comportamenti non lineari. Dopodiché il valore della funzione di attivazione sulla somma di ciascun neurone è passato in output ai neuroni del layer successivo secondo le connessioni fra neuroni.

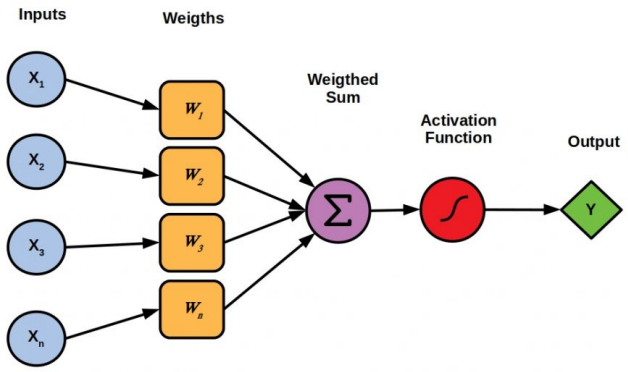

Figura 2.6: Illustrazione del funzionamento di un singolo neurone artificiale
Fonte: starship-knowledge.com

Una volta terminati gli hidden layer, le informazioni arrivano al layer di output. Qui, in caso di regressione si ha un numero di neuroni pari alle dimensioni desiderate dell'output (1 solo neurone se l'output è un singolo valore numerico), in caso di classificazione si ha invece un neurone per ogni classe del problema, in caso di classificazione binaria è sufficiente un solo neurone. Anche i nodi di output possono avere funzioni di attivazione, nel caso della regressione ciò può essere utile a far rientrare l'output in un certo intervallo, nel caso della classificazione invece la funzione di attivazione può trasformare valori numerici grezzi nelle probabilità corrispondenti alle classi, in modo che la classe a cui corrisponde la maggior probabilità sia quella predetta.

2.2.2 Addestramento di una rete neurale

Quanto spiegato per il machine learning nella sezione precedente sulle tipologie di apprendimento, su testing e valutazione e sul tuning degli iperparametri è equivalente anche nel campo del deep learning. Tuttavia, il problema di minimizzare la funzione d'errore è differente perché invece di avere una manciata di parametri, le reti neurali hanno migliaia se non milioni di pesi da aggiornare a ogni step di apprendimento. [18] introduce un algoritmo estremamente efficiente per calcolare il gradiente della funzione d'errore rispetto a ogni parametro, chiamato *backpropagation*. Fatto ciò, possono essere aggiornati tramite discesa del gradiente, o altri ottimizzatori. Per sfruttare le capacità di parallelizzazione delle GPU, gli input vengono passati alla rete neurali in piccoli gruppi, chiamati *batch*, e vengono elaborati in successione da tutti i layer fino a quello di output. A questo punto viene calcolato l'errore in relazione a tutta la batch, e tramite backpropagation vengono calcolati i gradienti e regolati i pesi di tutti i nodi di ogni layer con uno step di discesa del gradiente. L'intero training set, suddiviso in batch, viene passato alla rete neurale più volte, per aumentare la capacità di apprendimento. Ogni ciclo di addestramento sull'intero set è chiamato *epoca* di addestramento. Il numero di epoche è un iperparametro, e vi è la possibilità di arrestare l'addestramento in automatico quando il modello converge o quando va in overfitting (early stopping).

Un'altra tecnica per ridurre l'overfitting nelle reti neurali è chiamata *droput*. Per un layer, viene definito un *dropout rate*, ovvero un valore percentuale che equivale alla probabilità che l'output di ciascun neurone di tale layer venga ignorato a ogni step durante il training. Ad esempio con un dropout rate di 0.2, a ogni step ogni neurone ha il 20% di probabilità di essere ignorato. Questo viene fatto per fare in modo che la rete neurale non faccia troppo affidamento su alcuni specifici neuroni andando in overfitting. Durante la fase di valutazione del modello il dropout è disattivato e vengono utilizzati tutti i neuroni.

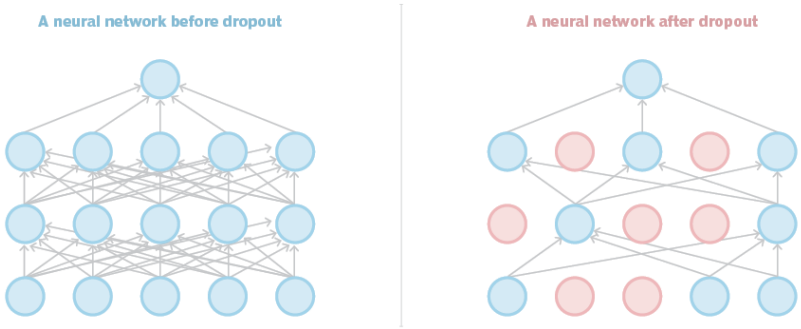

Figura 2.7: Illustrazione del dropout in una rete neurale
Fonte: techtarget.com

2.3 Natural Language Processing

Il testo in **linguaggio naturale umano** rappresenta la quantità di dati con la più larga disponibilità in rete: siti web, social media, blog, forum sono fonti inesauribili di informazioni su qualsiasi argomento con un alto potenziale per addestrare modelli di apprendimento. Questi dati infatti, possono fornire informazioni significative riguardo all'opinione generale circa un determinato argomento, prodotto commerciale, candidato politico, o nel caso della borsa valori, circa l'andamento di una azienda o di un mercato. Il problema principale legato a questi dati è la loro mancanza di struttura. Nonostante il testo segua generalmente delle regole grammaticali ben definite, vi sono numerose possibilità di ambiguità, causate dal contesto, dalle svariate possibilità espressive offerte dal linguaggio naturale, o semplicemente da errori grammaticali, nonché dall'uso di moderne forme di linguaggio come abbreviazioni, *hashtag* ed *emoji*. Nel caso dei social media poi, le regole grammaticali tendono a essere trascurate, ne è un esempio la piattaforma social *Twitter*, dove il numero di caratteri consentito per ogni post è limitato, costringendo gli utenti a omettere parole e a semplificare le strutture grammaticali. Per poter utilizzare dati in linguaggio naturale in un modello predittivo è dunque necessario fare il possibile per dare struttura ai dati (pre-processing), dopodiché bisogna estrarre da essi il significato semantico (language modeling).

2.3.1 Pre-processing del linguaggio naturale

Dal testo grezzo raccolto vanno spesso rimosse parti irrilevanti, come ad esempio elementi di markup o segni di punteggiatura. Un'altra operazione effettuata per uniformare il testo è chiamata *casefolding*, e consiste nel convertire le parole utilizzando solo lettere minuscole. Inoltre, quando la mole di dati testuali è proibitivamente grande per essere utilizzata nella sua interezza, è possibile filtrare il testo con delle keyword, per mantenere solo le frasi o i paragrafi rilevanti per il task. Vi è anche la possibilità di rimuovere parole semanticamente irrilevanti (*stopword*), come preposizioni, articoli e congiunzioni. Il testo viene poi segmentato, cioè ridotto in unità di base. Banalmente ciò può significare scomporre le frasi in singole parole, ma il testo può essere segmentato anche in sequenze più lunghe. Quando si segmenta il testo in singole parole, le parole composte vengono suddivise nelle loro componenti di base, secondo le regole grammaticali della lingua in questione (esempio la contrazione "isn't" in lingua inglese viene segmentata nei due elementi "is" e "not"). I modelli di machine learning operano con vettori, matrici e tensori numerici, ma non sono in grado di elaborare dati in formato di stringa, per questo motivo ogni segmento viene trasformato in un valore numerico. Ciò

viene fatto utilizzando un dizionario in cui a ogni parola corrisponde un numero, chiamato vocabolario. Tale conversione è chiamata *tokenizzazione* del testo. Si utilizzano anche numeri speciali riservati per indicare ad esempio l'inizio o la fine di una sequenza testuale. Nei modelli che oltre a ricevere testo in input, generano testo in output, il vocabolario servirà anche a convertire i valori numerici in output in linguaggio naturale.

2.3.2 Language modeling

Per task semplici, una volta strutturati i dati testuali con le tecniche viste sopra, sarà sufficiente applicare algoritmi che conteggiano il numero di occorrenze di certe parole, o che confrontano documenti verificando la similarità dei vocaboli utilizzati. Qualora invece si sia interessati ad affrontare task complessi, sarà necessario interpretare la **semantica** del testo, per avere una comprensione completa del suo significato. Solo in questo modo sarà possibile ottenere buoni risultati in problemi avanzati come la riassunzione e traduzione di documenti, il question answering, o il sentiment analysis. In **NLP**, un modello di comprensione semantica del linguaggio naturale è chiamato **language model**. I language model tradizionali fanno uso di database lessicali per classificare i segmenti testuali nelle rispettive classi grammaticali (nomi, verbi, aggettivi, ecc.) e per comprendere le relazioni semantiche tra diversi termini. Attraverso grandi set di sinonimi, contrari, implicazioni e altre relazioni semantiche, questi database lessicali riescono a fornire ai language model una base semantica con la quale interpretare il linguaggio naturale. I language model più semplici, basati solo su basi di conoscenze esterne che contengono set di relazioni semantiche hard-coded hanno però varie limitazioni: hanno difficoltà nei casi di ambiguità, dove uno stesso termine può avere molteplici significati, trattano ogni termine come un elemento a sé e non lo considerano in relazione ai termini ad esso vicini, spesso non modellano la posizione dei termini all'interno del testo. Per una interpretazione più profonda del testo, necessaria per ottenere una comprensione completa, si fa perciò uso di modelli che simulino il ragionamento umano, fra tutti le reti neurali. I language model basati su reti neurali, chiamati **language model neurali**, possono essere addestrati su grandi quantità di dati testuali non strutturati per apprendere autonomamente le complesse relazioni semantiche del linguaggio naturale, arrivando a un livello di comprensione linguistica impareggiabile dai modelli tradizionali. Tale comprensione del linguaggio può poi essere sfruttata per generare a partire dai dati testuali delle rappresentazioni numeriche ad elevata dimensionalità che rappresentino in maniera completa significati, relazioni, posizioni all'interno di un testo e innumerevoli altre feature che sarebbe impossibile ottenere manualmente per testi lunghi.

2.3.3 Modelli encoder-decoder

Nei task che richiedono, a partire dalla comprensione di un testo, di generare in output altro testo, come ad esempio la traduzione o la riassunzione, i modelli con performance migliori utilizzano un'architettura **encoder-decoder** [19]. L'idea di questa architettura è di avere un encoder che prende in input una sequenze di simboli di qualsiasi lunghezza, e ne restituisce un encoding numerico vettoriale a lunghezza fissa e prestabilita che ne rappresenti il significato semantico; a questo punto l'encoding viene dato in input al decoder che lo utilizza per restituire un output simbolico di lunghezza variabile. Nel caso di un sistema di traduzione, l'input e l'output saranno stringhe di testo. I modelli di questo tipo basano generalmente sia l'encoder che il decoder su reti neurali ricorrenti, viste le performance di queste reti su dati sequenziali, come è il caso di dati testuali. Nei modelli encoder-decoder, al primo step l'input del decoder è un token di inizio sequenza, dopodichè a ogni step successivo l'input è l'output del modello allo step precedente, tale modello è illustrato nella figura 2.8 e si dice autoregressivo.

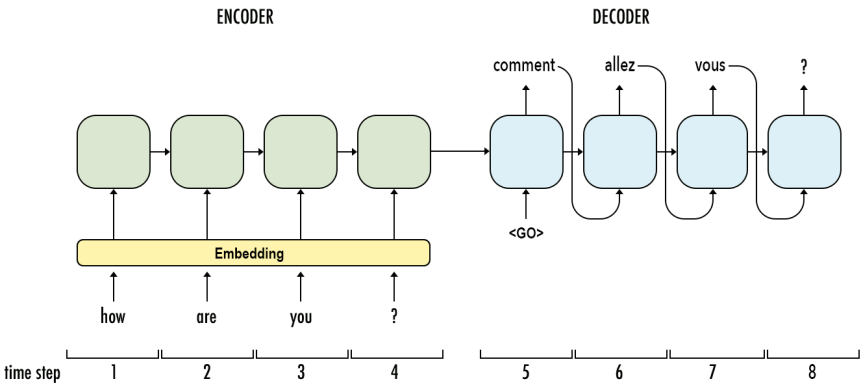

Figura 2.8: Architettura encoder-decoder autoregressiva per traduzione inglese-francese

Fonte: www.towardsdatascience.com

Tali modelli possono essere utilizzati anche per task che non prevedono la generazione di testo; in tal caso si usa semplicemente l'encoder, che fornirà in output delle rappresentazioni numeriche del testo in input. Queste potranno poi essere utilizzate come input per modelli di classificazione o regressione. I modelli encoder-decoder, che hanno ottime performance per sequenze brevi, hanno però difficoltà e limitazioni di memoria per sequenze più lunghe.

Attention

Per alleviare le limitazioni su input lunghi, viene introdotto in [20] il concetto di *attention*, grazie al quale è possibile per il decoder ricevere in input un encoding non più a lunghezza fissa, bensì a lunghezza variabile. Nel caso delle RNN, ciò significa fornire al decoder gli hidden state di tutti gli step, e non solo l'ultimo come avveniva senza attention nei modelli encoder-decoder. Grazie al meccanismo di attention, implementato aggiungendo alla rete neurale un attention layer, anch'esso addestrato assieme al modello, a ogni step di decoding, il decoder sarà in grado di concentrarsi solo sulla parte di input più rilevante, calcolando uno score di "allineamento" rispetto allo step di decoding attuale. L'introduzione di questo meccanismo rende più performanti i modelli encoder-decoder che lavorano su dati testuali, specialmente all'aumentare della lunghezza delle sequenze.

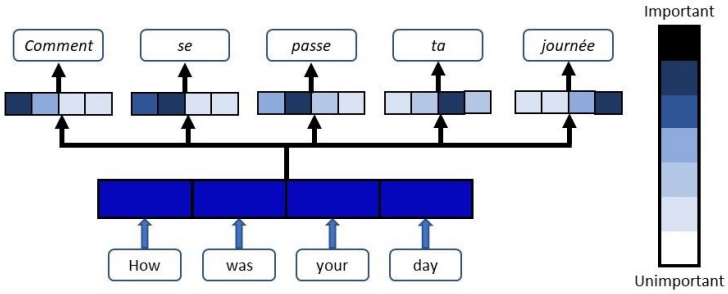

Figura 2.9: Visualizzazione del meccanismo di attention

2.3.4 Modello Transformer

L'aggiunta di un layer di attention ai modelli encoder-decoder basati su reti neurali ricorrenti ne migliora le performance, specialmente su sequenze lunghe. [21] introduce invece **Transformer**, una rivoluzionaria architettura encoder-decoder basata interamente sul concetto di attention. Oltre ad essere utilizzato per allineare encoder e decoder, nel modello Transformer l'attention è utilizzata anche all'interno dell'encoder e del decoder per modellare le relazioni tra parole nel testo, al fine di ottenere una rappresentazione semantica più ricca, oltre che più efficiente a livello computazionale rispetto alle RNN.

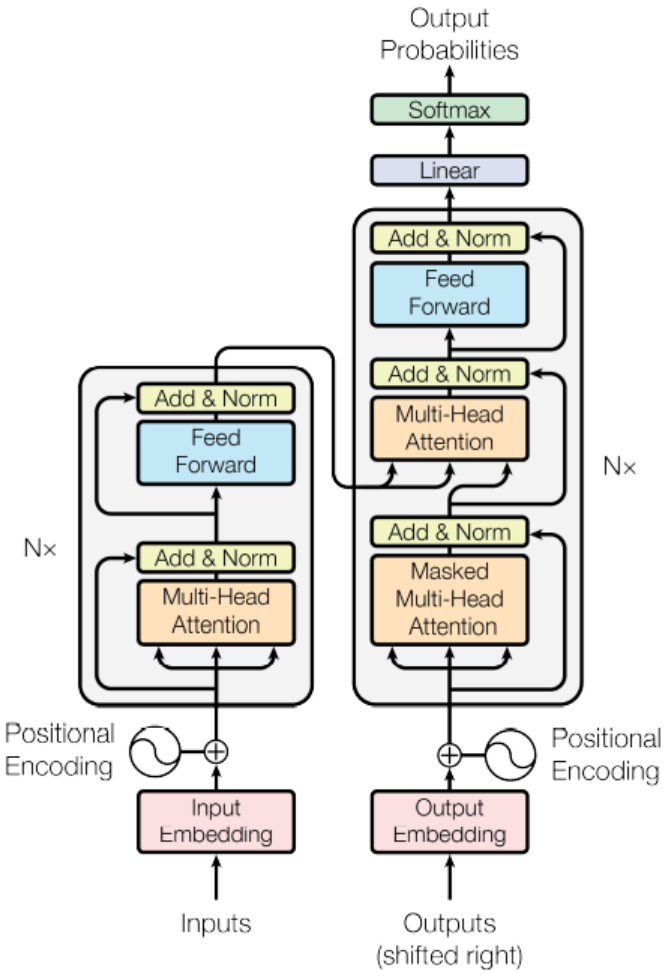

Figura 2.10: Architettura Transformer

Il tipo di attention utilizzata nel modello Transformer viene chiamato *scaled dot-product attention* ed è descritto come:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_{keys}}})V$$

dove:

- Q è la matrice delle query;
- K è la matrice delle chiavi;
- V è la matrice dei valori;

- QK^T è un prodotto scalare che calcola lo score di similarità per ogni coppia query-chiave;

- $softmax$ è una funzione che trasforma gli score di similarità in probabilità, la somma degli score fa dunque 1;

- $\sqrt{d_{keys}}$ è un fattore di scaling per evitare di saturare la funzione softmax.

L'output della funzione di attention sarà dunque la matrice dei valori V pesata per lo score di similarità calcolato con prodotto scalare.

Come mostrato nella figura 2.10, Transformer fa uso del meccanismo di attention in tre diverse circostanze:

1. Nell'interazione tra encoder e decoder. In questo caso le query dell'operazione sono gli output del decoder, mentre le chiavi e i valori sono gli output dell'encoder;

2. All'interno dell'encoder per modellare le relazioni tra parole nella stessa sequenza. In questo caso, le query, le chiavi e i valori sono lo stessa matrice, ovvero l'output del precedente layer dell'encoder o gli embedding iniziali se si tratta del primo layer dell'encoder. Quando il meccanismo di attention è applicato in questa maniera per modellare relazioni interne a una sequenza, viene chiamato self-attention;

3. All'interno del decoder, self-attention allo stesso modo del punto 2. In questo caso però l'attention non può essere bidirezionale in quanto una parola non può avere informazioni relative alle parole che la seguono nella sequenza (le parole a destra). Per rendere l'attention unidirezionale e solo da sinistra, a ogni step le parole successive a quella attuale sono mascherate.

Self-attention

A differenza di quanto fatto in precedenza con layer convoluzionali o ricorrenti, tramite self-attention nel modello Transformer il meccanismo di attention viene utilizzato anche per mappare le relazioni all'interno delle stesse sequenze. Questo viene fatto per i seguenti motivi:

- Minore complessità computazionale;

- Possibilità di sfruttare le GPU parallelizzando il calcolo dell'attention in quanto i dati da elaborare non sono sequenziali;

- Maggiori performance con self-attention nel modellare le relazioni all'interno di una sequenza, specialmente qualora questa sia particolarmente lunga;

- Maggiore interpretabilità del modello: le RNN e CNN sono di difficile interpretabilità, mentre il calcolo di self-attention può essere facilmente visualizzato e interpretato come mostrato nella seguente figura, dove è possibile vedere in maniera esplicita come in questo particolare layer di encoding il modello riesce a comprendere la relazione tra la parola "making" e le parole "more difficult".

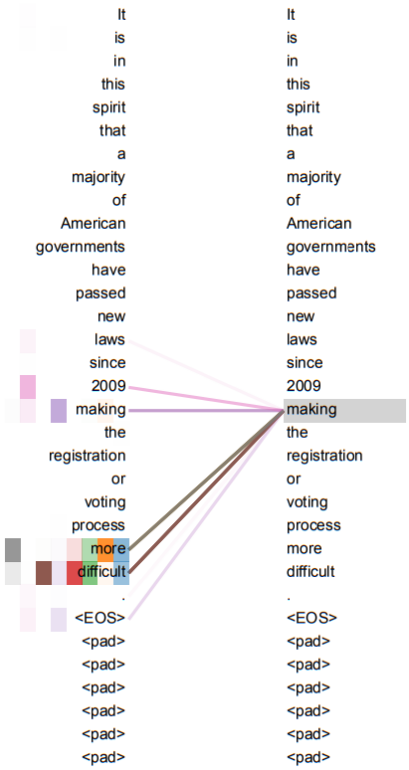

Figura 2.11: Visualizzazione del meccanismo di self-attention

Multi-head attention

Generalmente l'attention viene calcolata unitamente su tutte le dimensioni di query, chiavi e valori. Nel caso di Transformer invece, si fa uso di un concetto chiamato **multi-head attention**, dove invece che calcolare una singola funzione di attention su tutte le dimensioni, l'input viene trasformato con h diverse proiezioni lineari ottenendo così h variazioni delle stesse query, chiavi e valori.

A questo punto l'attention viene calcolata in parallelo su tutte le h variazioni e i valori risultanti vengono concatenati per poi essere trasformati linearmente nella dimensione iniziale. In questo modo, si possono modellare h diverse sfaccettature delle relazioni fra elementi di una sequenza.

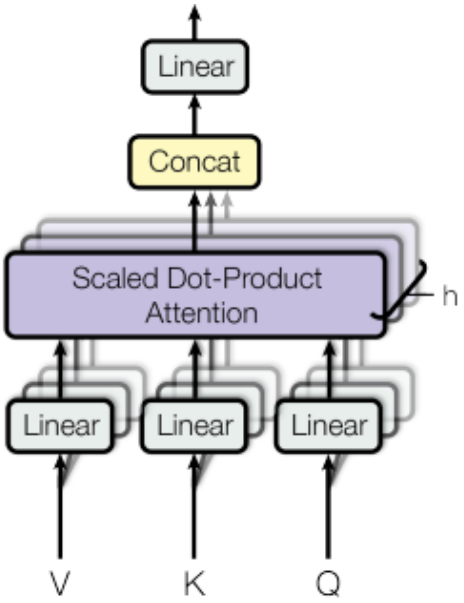

Figura 2.12: Illustrazione multi-head attention

Embedding e encoding della posizione

Ogni token passato in input sia all'encoder che al decoder viene trasformato da un layer di embedding in un vettore a 512 dimensioni (la *hidden size* scelta nel modello) che ne rappresenti le feature semantiche. Visto che Transformer non fa uso di layer ricorrenti o di convoluzione e visto che gli embedding di input vengono passati in parallelo, il modello sarebbe privo della nozione di posizione delle parole all'interno della sequenze. Per aggiungere tale nozione, all'embedding degli input viene sommato un vettore, anch'esso a 512 dimensioni, calcolato tramite funzioni sinusoidali e chiamato positional encoding. In questo modo gli embedding in input, oltre a rappresentare la semantica delle parole, ne rappresentano anche la posizione nella frase.

2.3.5 Modelli pre-addestrati e fine-tuning: BERT

L'architettura Transformer rappresenta oggi, per le prestazioni e per le possibilità di parallelizzazione delle operazioni, lo stato dell'arte in molti task di

NLP. Architetture full-attention come Transformer sono state inoltre utilizzate con successo in altri ambiti, come la visione artificiale [22] e la generazione di musica [23]. Inoltre, utilizzando solo l'encoder si possono generare encoding di dati testuali per task di classificazione o regressione. Nonostante il successo, Transformer presenta ancora delle limitazioni, fra tutte la principale è il proibitivo costo di addestramento del modello; l'addestramento per ottenere i risultati citati in [21] ha impiegato 12 ore utilizzando congiuntamente 8 GPU ad alte prestazioni. Tali requisiti rendono l'addestramento arduo, ogni azienda o organizzazione dovrebbe riaddestrare la rete neurale per ogni specifico task. Per ovviare a questa problematica, [24] propone **Bidirectional Encoder Representations from Transformers (BERT)**. Tale language model neurale, segue in larga parte l'architettura dell'encoder Transformer, seppur con alcune differenze. La grande novità di BERT è il pre-addestramento: il language model è addestrato in maniera non supervisionata su enormi quantità di dati testuali (800 milioni di parole provenienti da BookCorpus [25] e 2.5 miliardi da *wikipedia.com*), dopodichè i pesi del modello addestrato sono stati rilasciati pubblicamente. In questo modo chiunque voglia caricare il modello già addestrato, ovvero già in grado di modellare il linguaggio naturale, potrà farlo in pochi secondi, scaricando dalla rete i pesi e caricandoli sul modello. Il pre-addestramento del modello, chiamato **pre-training**, avviene in maniera non supervisionata con un task di language modeling, in cui una porzione delle sequenze in input viene mascherata, e l'obiettivo del modello è completare le sequenze predicendo le parti mascherate (masked language modeling), e un task di next sentence prediction, dove il modello riceve due sequenze testuali ed è chiamato a predire se la seconda segua la prima in un testo o meno. Il pre-training è bidirezionale, ovvero ogni elemento delle sequenze la self-attention è calcolata sia sugli elementi precedenti che quelli successivi, potendo così modellare il contesto e le relazioni di ogni parola rispetto a tutto il resto della frase.

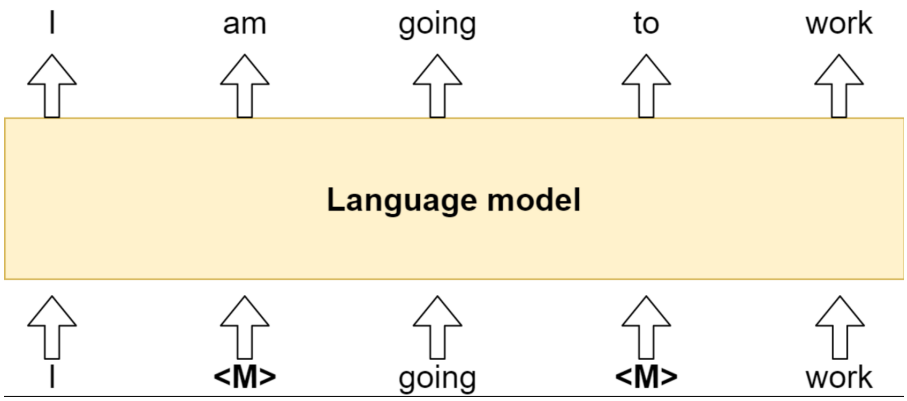

Figura 2.13: Esempio di masked language modeling
Fonte: www.machinecurve.com

Una volta caricato il language model pre-addestrato, sarà necessario adattarlo al task obiettivo. Ciò viene fatto aggiungendo al modello uno o più layer finali, a seconda del task. Ad esempio in un task di regressione sarà sufficiente aggiungere un layer di output con un neurone per ottenere un valore numerico, mentre in un caso di classificazione sarà sufficiente aggiungere un layer di output con un neurone per ogni classe. Adattata la rete neurale al task aggiungendo uno o più layer alla fine, si effettua ciò che viene chiamato **fine-tuning**, ovvero si effettuano alcune epoche di addestramento supervisionato utilizzando i dati etichettati del task obiettivo: in questo modo i pesi del language-model vengono ritoccati a seconda delle specifiche necessità del task, mentre i pesi dei nuovi layer appena aggiunti vengono addestrati. L'approccio appena illustrato di fine-tuning di un modello pre-addestrato, oltre ad essere efficiente ed accessibile, ottiene notevoli risultati in 11 task di natural language processing.

Capitolo 3

Modellazione del progetto

In questo capitolo viene esposta la modellazione dell'esperimento svolto facendo riferimento al progetto a cui questo si ispira.

3.1 Progetto di riferimento

Come spiegato nel capitolo 1, [13] estrae dalla rete decine di migliaia di news finanziarie dalle testate giornalistiche Bloomberg e Reuters e servendosi dei titoli di esse costruisce degli eventi strutturati, ovvero delle tuple composte da un attore, un'azione e un oggetto. Ad esempio dal titolo di notizia "Microsoft sues Barnes & Noble", viene creata la tupla (Attore = Microsoft, Azione = sues, Oggetto = Barnes & Noble).

Da questi eventi vengono costruiti degli embedding tramite una rete neurale addestrata in modo che eventi simili abbiano rappresentazioni simili anche se utilizzano parole diverse. Gli embedding vengono poi forniti in input a una rete neurale di classificazione per effettuare previsioni sull'andamento dell'indice di borsa S&P 500, un indice rappresentativo delle performance delle 500 aziende più quotate nella borsa statunitense.

Il classificatore è basato su layer convoluzionali e le istanze fornitegli in input comprendono gli embedding a lungo termine, che fanno riferimento ai 30 giorni precedenti quello della predizione, embedding a medio termine, che fanno riferimento ai 7 giorni precedenti quello della predizione e embedding a breve termine, che fanno riferimento al giorno precedente quello della predizione. Dei layer convoluzionali modellano l'impatto a lungo e medio termine delle news finanziarie, e queste informazioni vengono poi concatenate a quelle del giorno precedente al giorno obiettivo per effettuare una predizione binaria: se il modello prevede per il giorno obiettivo una chiusura in rialzo rispetto al valore di apertura dell'indice S&P 500, restituirà il valore 1, diversamente restituirà il valore 0.

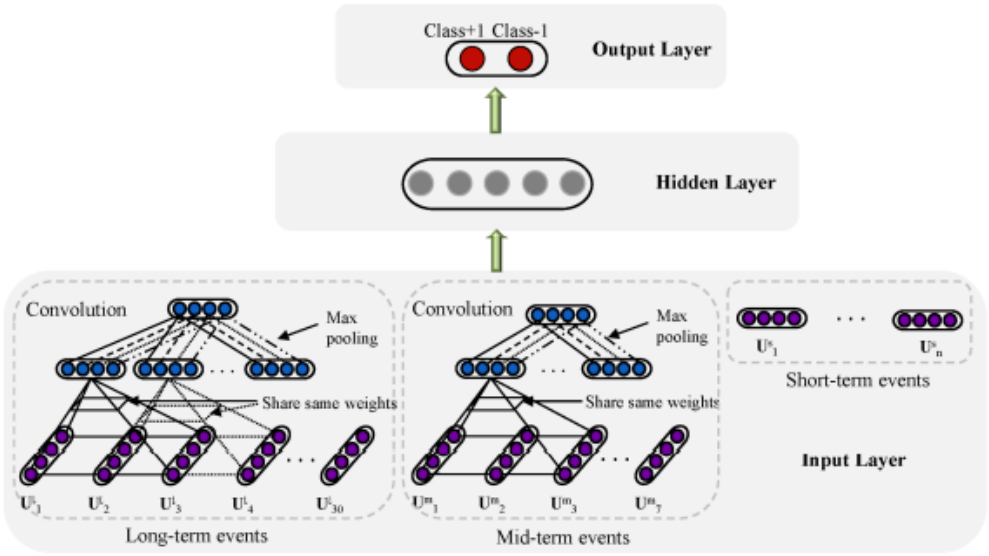

Figura 3.1: Architettura del classificatore utilizzato in

Il modello illustrato è addestrato su news che fanno riferimento al periodo 2007-2012 circa, ed è testato su news che fanno riferimento al 2013, ottenendo una accuracy del 65%.

L'obiettivo di questo esperimento è di seguire l'approccio generale del paper appena citato, con una sostanziale differenza. Invece di strutturare i dati testuali come eventi e di elaborarli con una rete neurale ad hoc per creare degli embedding, si vuole utilizzare l'encoder BERT pre-addestrato per effettuare direttamente l'encoding del testo non strutturato delle news. Al testo verranno poi aggiunte le etichette relative all'andamento del titolo S&P 500, per poter addestrare una rete neurale di classificazione in maniera supervisionata al fine di predire l'andamento di tale indice. Come nel paper si utilizzerà per la classificazione una rete con layer convoluzionali.

Occorre precisare che il paper sopracitato non è accompagnato dal codice, perciò l'esperimento mostrato in seguito è solo ispirato a tale paper, ma implementato da zero basandosi sulle indicazioni generali in esso contenute.

3.2 Fasi di sviluppo

Vengono qui elencate le fasi di sviluppo che si dovranno seguire per realizzare l'esperimento:

1. Pulizia dei dati

2. Raggruppamento delle news finanziarie secondo il giorno di pubblicazione
3. Filtering del dataset tramite keyword per estrarre solo le news rilevanti all'andamento di S&P 500
4. Pre-processamento del testo
5. Calcolo degli encoding di ciascun titolo di notizia
6. Pooling degli encoding relativi a uno stesso giorno in un unico vettore
7. Aggiunta al dataset dei dati storici finanziari e creazione delle etichette
8. Creazione delle istanze per la rete neurale di classificazione
9. Creazione e addestramento della rete neurale di classificazione
10. Valutazione dei risultati con metriche di performance e simulazione di compravendita

Capitolo 4

Sviluppo

In questo capitolo viene documentato lo sviluppo dell'esperimento introdotto nel capitolo precedente.

4.1 Setup dell'ambiente di sviluppo

L'addestramento di modelli complessi richiede l'utilizzo di hardware potente, in particolare di GPU in grado di eseguire in maniera efficiente e parallela operazioni su matrici e tensori, per questo motivo per eseguire l'esperimento si è ricorso a *Google Colaboratory*, un'applicazione web che permette di creare ambienti di sviluppo basati su *Jupyter Notebook* e di utilizzare hardware di ultima generazione tramite *cloud computing*.

L'intero esperimento è realizzato utilizzando **Python 3** come linguaggio di programmazione. Questo linguaggio è estremamente versatile per la prototipazione e può contare su numerose librerie che coprono tutto il range di necessità della data science, dalla rappresentazione di dati fino alla creazione e l'addestramento di modelli di deep learning. Per questi motivi è diventato lo standard *de facto* nel mondo della data science. Di seguito vengono mostrate le librerie utilizzate nell'esperimento.

```
import pandas as pd # data manipulation
import numpy as np # efficient matrix operations
import datetime # operations with dates
import transformers # library for pre-trained language models by HuggingFace
import sklearn # data preprocessing
import yfinance as yf # retrieve financial historical data
import torch # deep learning library
```

La libreria scelta per la parte di deep learning è PyTorch, una libreria open-source multipiattaforma per la creazione e l'addestramento di reti neurali, sviluppata dall'AI research lab di Facebook.

4.2 Pulizia dei dati

Le notizie di *Reuters* sono già catalogate in file *TSV* per ogni giorno nel dataset, inoltre è mostrato solo il titolo e non il corpo della notizia. Come nel paper di riferimento, in questo esperimento si utilizzeranno solo i titoli delle notizie e non l'intero corpo, in quanto il titolo è considerato la parte più efficiente e siginificativa del testo. Oltre al titolo ogni istanza ha un timestamp temporale e un link web alla notizia. Per i file TSV di ogni giorno dunque, vengono eliminati i link in quanto non necessari, e i timestamp vengono ridotti alla sola data, in quanto in questo esperimento l'unità è il giorno e all'interno dello stesso giorno tutte le notizie vengono utilizzate allo stesso modo senza distinzioni di orario, perciò l'orario è irrilevante.

```
ts      title   href
20070222 11:58 PM EST   Allergan exec says deal to bolster obesity biz  http://www.reuters.com/article/ousiv/idUSN2248584420070223
20070222 11:55 PM EST   Roddick and Murray close on semi-final showdown http://www.reuters.com/article/sportsNews/idUSL2209135620070223
20070222 11:51 PM EST   NYMEX planning large stock offering - WSJ       http://www.reuters.com/article/newIssuesNews/idUSN2219977220070223
20070222 11:48 PM EST   China zoo tiger kills six-year-old girl http://www.reuters.com/article/worldNews/idUSPEK20791920070223
20070222 11:36 PM EST   North Korea accuses United States of hostility, lies    http://www.reuters.com/article/topNews/idUSN2216690220070223
20070222 11:36 PM EST   North Korean nuclear envoy to visit U.S.: report        http://www.reuters.com/article/topNews/idUSSEO20725020070223
20070222 11:23 PM EST   Chlorine bombs mark new guerrilla tactics: U.S  http://www.reuters.com/article/topNews/idUSKRA14854020070223
20070222 11:23 PM EST   Cheney issues rallying call not to abandon Iraq http://www.reuters.com/article/newsOne/idUSSYD9257420070223
```

Figura 4.1: Esempio di notizie finanziarie Reuters nel dataset

Per quanto riguarda le notizie di *Bloomberg* invece, è fornito un file testuale per ciascuna notizia. In questo caso per ogni notizia è incluso il titolo, l'autore, il timestamp e il corpo dell'articolo. Viene mantenuto solo il titolo e la data dal timestamp.

```
-- Norilsk Slashes Borrowing Costs as Nickel Price Soars
-- Todd Prince
-- 2007-02-22T14:56:23Z
-- http://www.bloomberg.com/news/2007-02-22/norilsk-slashes-borrowing-costs-as-nickel-price-soars-update1-.html


            OAO GMKN  Norilsk Nickel , the world's
largest producer of nickel, slashed its borrowing costs by a
third as surging prices for the metal boost the company's profit.
 Norilsk today signed a $450 million five-year revolving
credit with a group of banks led by  Barclays Capital ,  Societe
Generale SA (GLE)  and ING Groep NV, the Moscow-based company said in a
press release. The loan was increased from $300 million amid
demand by banks.
 Russia's largest mining company will pay interest of 42.5
basis points more than the London interbank offered rate on the
```

Figura 4.2: Esempio di notizia finanziaria Bloomberg nel dataset

4.3 Raggruppamento delle news finanziarie

Tutte le notizie nel dataset sono corredate di timestamp che ne indica la data di pubblicazione. I titoli delle notizie contenuti nei file testuali vengono quindi passati in rassegna e catalogati secondo il giorno di pubblicazione. Per questa operazione si fa uso di un *Dataframe* Pandas dove gli indici sono le date a cui corrispondono le liste di titoli di notizie.

4.4 Filtering del dataset tramite keyword

Il totale delle notizie nel dataset ammonta a più di 550000. Il numero di notizie è molto alto ed utilizzarle tutte potrebbe rendere l'addestramento troppo lungo. Inoltre, è probabile che molte notizie siano irrilevanti al fine di predire l'andamento di S&P 500. Per questo motivo viene creata una lista di *keyword*, ovvero di parole di interesse relative al titolo in questione, e vengono confrontati tutti i titoli con le parole nella lista: solo i titoli che contengono almeno una keyword sono mantenuti, gli altri sono scartati. La lista delle keyword è ottenuta estraendo da una tabella su *Wikipedia* i nomi delle aziende che fanno parte dell'indice S&P 500, oltre ai nomi delle 500 aziende vengono aggiunte altre keyword rilevanti.

```
html_path = 'https://en.wikipedia.org/wiki/List_of_S\%26P_500_companies'
# extract tables from wikipedia page
table = pd.read_html(html_path)
# select first table
df = table[0]
# retain only company names as list
companies = pd.concat([df['Security']]).tolist()
# other relevant keywords
keywords = ['S&P', 'S&P500', 'SP', 'SP500', 'Standard & Poor\'s',
            'AMEX', 'NYSE', 'Nasdaq', 'US market']
# merge
keywords += companies
```

4.5 Pre-processamento del testo

Per poter essere fornito in input all'encoder BERT, il testo deve essere segmentato e trasformato in token. Inoltre, visto che l'encoder riceve input di lunghezza fissa, la lunghezza dei titoli viene uniformata con dei token di padding. Per indicare all'encoder quali sono i token su cui calcolare l'attention e quali

invece sono token di padding, viene creato un vettore della stessa lunghezza del vettore dei token con degli 1 nelle posizioni dei veri token e degli 0 nelle posizioni dei token di padding. Tale vettore di valori binari si chiama attention mask e viene passato all'encoder in modo che venga calcolato lo score di attention solo per le posizioni che contengono veri token. Sia per realizzare il vettore dei token che per realizzare l'attention mask si fa uso di un tokenizzatore pre-costruito importato tramite la libreria Python *Transformers* di *HuggingFace*, che fornisce per ogni language model pre-addestrato anche il relativo tokenizzatore. Il tokenizzatore realizza anche operazioni di pre-processing quali il casefolding per uniformare il testo, spiegato nel capitolo 2.

4.6 Calcolo degli encoding

L'encoder utilizzato è il modello BERT pre-addestrato come mostrato in [24]. Il modello è stato caricato tramite la libreria *Transformers*. Il modello pre-addestrato utilizzato fa uso della configurazione di base mostrata nel paper e gli encoding in output hanno 768 dimensioni. Per ciascuna notizia, i token del testo e l'attention mask vengono passati all'encoder, che restituirà un encoding sotto forma di una matrice di dimensione $n_{token} \times 768$. Questa matrice passa per un layer di pooling che ne restituisce un vettore. Per ogni titolo si avrà quindi una rappresentazione vettoriale a 768 valori, uno per ogni dimensione.

Il codice qui riportato mostra la classe utilizzata per istanziare un generatore che per ogni notizia ne genera token e attention mask e da questi l'encoding.

```
class EncodingGenerator():
  def __init__(self, encoder, tokenizer, max_len):
    self.encoder = encoder
    self.tokenizer = tokenizer
    self.max_len = max_len

  def tokenize(self, text):

    tok_out = self.tokenizer.encode_plus(text, add_special_tokens=True,
                                          max_length=self.max_len,
                                          pad_to_max_length=True,
                                          return_attention_mask=True,
                                          return_tensors="pt")

    return tok_out['input_ids'].to(device), tok_out['attention_mask'].to(device)

  def encode(self, text):
```

```
    ids, att_mask = self.tokenize(text)
    output = self.encoder(ids, att_mask).pooler_output
    return output # pooled 1x768 output
```

4.7 Media degli encoding

Per ogni giorno nel dataset si avrà ora un vettore corrispondente a ogni titolo. I giorni hanno un numero di notizie variabili e alcune notizie hanno più di 100 notizie. Per uniformare le istanze viene calcolata una media di ciascuna delle 768 dimensioni: in questo modo ogni giorno ha una rappresentazione 1×768 che contiene una media di tutti gli encoding relativi a tutte le notizie pubblicate in quel giorno.

4.8 Aggiunta al dataset dei dati storici finanziari e creazione delle etichette

Oltre ai dati relativi alle news finanziarie si vogliono utilizzare i dati storici finanziari. Per ottenere tali dati si fa uso della libreria Python *yfinance*, che scarica da *Yahoo! Finance* i dati finanziari relativi a un indice di borsa e un periodo selezionato sotto forma di Dataframe Pandas. Per ogni giorno le feature utilizzate sono: valore di apertura, valore di chiusura, valore di minimo, valore di picco e volume delle operazioni.

```
ticker = '^GSPC' # ticker for S&P 500 index
start_date = "2007-01-01"
end_date = "2013-12-01"
# retrieval of financial data through yfinance
stock = yf.download(ticker, start=start_date, end=end_date)
# drop of non-relevant column
stock.drop(labels=['Adj Close'], axis=1, inplace=True)
```

Date	Open	High	Low	Close	Volume
2007-01-03	1418.030029	1429.420044	1407.859985	1416.599976	3429160000
2007-01-04	1416.599976	1421.839966	1408.430054	1418.339966	3004460000
2007-01-05	1418.339966	1418.339966	1405.750000	1409.709961	2919400000
2007-01-08	1409.260010	1414.979980	1403.969971	1412.839966	2763340000
2007-01-09	1412.839966	1415.609985	1405.420044	1412.109985	3038380000
...	...	...	...	...	...
2013-11-22	1797.209961	1804.839966	1794.699951	1804.760010	3055140000
2013-11-25	1806.329956	1808.099976	1800.579956	1802.479980	2998540000
2013-11-26	1802.869995	1808.420044	1800.770020	1802.750000	3427120000
2013-11-27	1803.479980	1808.270020	1802.770020	1807.229980	2613590000
2013-11-29	1808.689941	1813.550049	1803.979980	1805.810059	1598300000

1741 rows × 5 columns

Figura 4.3: Dataframe dei dati finanziari S&P 500

Da tali dati si ottiene una ulteriore feature per ciascun giorno, ovvero la differenza tra valore di chiusura e valore di apertura, denominata Delta. Questa feature verrà sia utilizzata per l'addestramento che per creare le etichette. Binarizzando il valore Delta di ogni giorno utilizzando come soglia il valore 0 si ottengono delle etichette binarie: 1 quando S&P 500 chiude in rialzo rispetto all'apertura, 0 viceversa. Le etichette sono contenute nella colonna Target.

```
def binarize(x):
  if x > 0:
    return 1
  return 0

stock['Delta'] = stock['Close'] - stock['Open']
stock['Target'] = stock['Delta'].apply(binarize)
```

Date	Open	Close	Delta	Target
2007-01-03	1418.030029	1416.599976	-1.430054	0
2007-01-04	1416.599976	1418.339966	1.739990	1
2007-01-05	1418.339966	1409.709961	-8.630005	0
2007-01-08	1409.260010	1412.839966	3.579956	1
2007-01-09	1412.839966	1412.109985	-0.729980	0
...	...	...	...	...
2013-11-22	1797.209961	1804.760010	7.550049	1
2013-11-25	1806.329956	1802.479980	-3.849976	0
2013-11-26	1802.869995	1802.750000	-0.119995	0
2013-11-27	1803.479980	1807.229980	3.750000	1
2013-11-29	1808.689941	1805.810059	-2.879883	0

1741 rows × 4 columns

Figura 4.4: Nuove colonne Delta e Target

Le feature finanziarie vengono normalizzate e raggruppate in un unico vettore. Ci si trova ora con un dataframe che contiene per ogni data la rappresentazione vettoriale dei relativi titoli di notizie, e un altro dataframe che per ogni data contiene le feature finanziarie e l'etichetta: viene quindi effettuato il merge di questi Dataframe.

4.9 Creazione delle istanze per la rete neurale di classificazione

Come spiegato, per la predizione di un giorno si vogliono utilizzare i dati relativi ai precedenti 30 giorni, perciò le istanze in input al classificatore dovranno contenere: la data utilizzata come indice , la serie temporale contenente i 30 vettori di encoding dei titoli dei 30 giorni precedenti, la serie temporale contenente i 30 vettori dei dati finanziari dei 30 giorni precedenti e l'etichetta binaria. Inoltre, viene utilizzata come ulteriore feature il valore di apertura del

giorno della predizione, visto che la predizione viene fatta proprio all'apertura del mercato. Con uno script python si costruisce il Dataframe contenente questi dati a partire dal Dataframe ottenuto in precedenza.

Il Dataframe viene ora suddiviso in training set per l'addestramento e test set per la valutazione. Come spiegato nel capitolo 2, per dati temporali la divisione avviene utilizzando una data come soglia, i dati prima di quella data vengono utilizzati per il training e quelli dopo per il testing. I dati vanno passati sequenzialmente e non possono essere mescolati altrimenti si fornirebbero al modello dati futuri rispetto alla previsione attuale e si perderebbe così la coerenza temporale. La data soglia per la suddivisione temporale viene indicata tramite l'iperparametro SPLIT DATE: i dati dal 2007 al 2012 vengono utilizzati per l'addestramento e quelli del 2013 per testare il modello. Semplicemente cambiando questo iperparametro sarà possibile sperimentare con ripartizioni diverse dei dati in addestramento/valutazione.

```
# date string format: yyyy-mm-dd, latest date in dataset is 2013-11-29
SPLIT_DATE = "2013-01-01"

def split_dataset(dataframe, split_date):
    is_train = dataframe['Date']<datetime.date(int(split_date[:4]),
                                               int(split_date[5:7]),
                                               int(split_date[8:10]))
    df_train = dataframe[is_train]
    df_test = dataframe[~is_train]

    return df_train, df_test

df_train, df_test = split_dataset(final_df, SPLIT_DATE)
```

4.10 Creazione e addestramento della rete neurale di classificazione

Viene ora definita la rete neurale che prende in input per ogni istanza i dati descritti ed effettua la predizione binaria. Il classificatore neurale è composto da due blocchi convoluzionali: uno per processare i dati a lungo termine (30 giorni prima) e una per quelli a medio termine (7 giorni prima). Nei blocchi convoluzionali viene eseguita una convoluzione, seguita da normalizzazione, funzione di attivazione Tanh e dropout; infine è posto un layer di pooling per estrarre i dati più significativi. Gli output dei due blocchi convoluzionali sono due vettori 1x768, che vengono concatenati col vettore 1x768 dei dati a

breve termine (1 giorno prima) e il valore in apertura. Si ottiene dunque un vettore a 2323 dimensioni, che viene passato al layer in output per eseguire la classificazione binaria.

```
class Classifier(nn.Module):
  def __init__(self):
        super(Classifier, self).__init__()

        self.cnn_long = self.convolutional_block(c_in=1, c_out=8, dropout=0.3,
                                                 kernel_size=(3, 1),
                                                 stride=(3, 1))
        self.maxpool_long = nn.MaxPool3d(kernel_size=(8, 10, 1))

        self.cnn_mid = self.convolutional_block(c_in=1, c_out=8, dropout=0.3,
                                                kernel_size=(3, 1),
                                                stride=(3, 1), padding=(1, 0))
        self.maxpool_mid = nn.MaxPool3d(kernel_size=(8, 3, 1))

        self.out = nn.Linear(2323, 1)

  def forward(self, data_long, data_mid, data_short, open_value):
        x = self.cnn_long(data_long)
        x = self.maxpool_long(x).squeeze(1)

        y = self.cnn_mid(data_mid)
        y = self.maxpool_mid(y).squeeze(1)
        # concat of long, mid and short data into single vector of shace 1x2323
        concat = torch.cat([x.squeeze(1), y.squeeze(1),
                            data_short, open_value.unsqueeze(1)], dim=1)

        return self.out(concat)

  def convolutional_block(self, c_in, c_out, dropout, **kwargs):
        block = nn.Sequential(
            nn.Conv2d(in_channels=c_in, out_channels=c_out, **kwargs),
            nn.BatchNorm2d(num_features=c_out),
            nn.Tanh(),
            nn.Dropout2d(p=dropout)
        )
        return block
```

Viene costruita una classe Dataset e poi il relativo Dataloader Pytorch per fornire al classificatore le singole istanze di training e test set, raggruppate in batch. Ricapitolando, per ogni istanza i dati forniti al classificatore per predire l'etichetta binaria sono:

- Dati a lungo termine (matrice 30x774), concatenazione di dati testuali, 768 dimensioni, e finanziari, 6 dimensioni;
- Dati a medio termine (matrice 7x774), concatenazione di dati testuali, 768 dimensioni, e finanziari, 6 dimensioni;
- Dati a breve termine (vettore 1x774), concatenazione di dati testuali, 768 dimensioni, e finanziari, 6 dimensioni;
- Valore di apertura del giorno da predire.

Vista la rapidità dell'addestramento di modelli basati su layer convoluzionali, la ricerca degli iperparametri viene effettuata manualmente testando le combinazioni di iperparametri più promettenti. L'iter di addestramento prevede per ogni epoca di addestrare il classificatore sul training set, e testare poi le performance a quell'epoca sul test set. Alla fine di tutte le epoche si mantengono i pesi relativi all'epoca con le migliori performance sul test set, e li si utilizzano per l'analisi finale.

Capitolo 5

Valutazione dei risultati

In questo capitolo vengono analizzati i risultati ottenuti dal modello costruito ed addestrato nel capitolo precedente.

5.1 Iperparametri utilizzati

Vengono qui elencati gli iperparamtri utilizzati per ottenere i risultati:

- **Batch size**: 64;
- **Numero di epoche**: 200;
- **Dropout rate**: 0.3;
- **Ottimizzatore**: Adam [26];
- **Learning rate**: 0.001;
- **Weight decay (regolarizzazione)**: 0.01.

5.2 Valutazione con matrice di confusione

La **matrice di confusione** è una tabella di cui si fa uso per **valutare modelli di classificazione**. Su un asse mostra i valori predetti dal modello, sull'altro quelli reali. Nel caso binario ha due righe e due colonne.

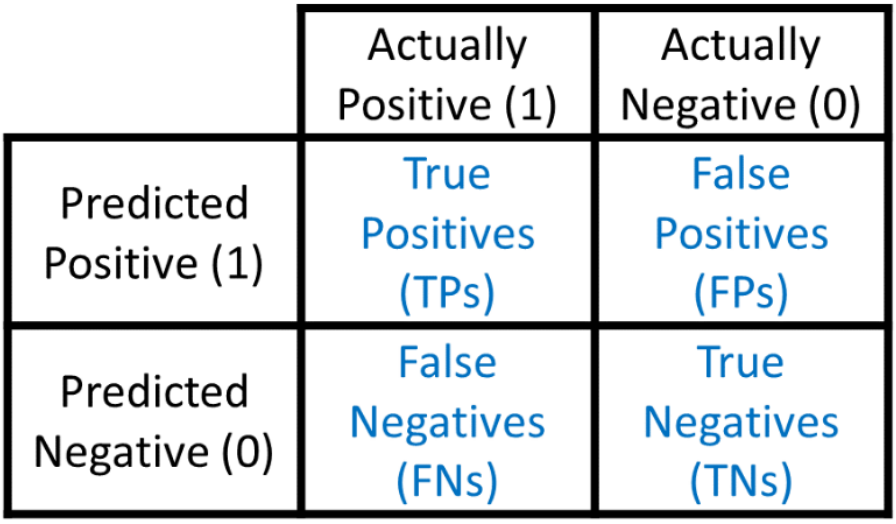

Figura 5.1: Matrice di confusione binaria

Nel caso della predizione del rialzo o ribasso di indice di borsa, i valori 1 (True) corrispondono al rialzo del valore dell'indice nella giornata, mentre i valori 0 (False) corrispondono al ribasso, dunque la tabella è da interpretarsi nella seguente maniera:

- **True positives**, numero di rialzi dell'indice predetti correttamente;
- **False positives**, numero di volte in cui il modello ha predetto un rialzo; dell'indice quando questo è invece diminuito
- **False negatives**, numero di volte in cui il modello ha predetto una diminuzione dell'indice quando questo è invece aumentato;
- **True negatives**, numero di ribassi dell'indice predetti correttamente;

Nella figura seguente viene mostrata la matrice di confusione del modello sul test set.

	Actually positive (1)	Actually negative(0)
Predicted positive (1)	True positives: 85	False positives: 48
Predicted negative (0)	False negatives: 23	True negatives: 25

Figura 5.2: Matrice di confusione del modello sviluppato

La matrice di confusione mostra che il modello è "ottimista", prediligendo predizioni di rialzo rispetto al ribasso. Ciò è in linea con la tendenza nel dataset e in generale dell'indice S&P 500 di aumentare di valore nel tempo. Si può anche notare come il modello sia più preciso nell'indovinare le predizioni positive rispetto a quelle negative.

5.3 Valutazione con metriche di performance

A partire dalla matrice di confusione è possibile estrarre varie metriche di performance per valutare un modello. Nel caso in questione quelle utilizzate sono due: **accuracy** e **punteggio MCC**.

L'accuracy è calcolata come

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN}$$

e non è altro che la percentuale delle predizioni corrette effettuate dal modello sul test set. L'accuracy ottenuta dal modello è pari a 60.77%.

L'indice di correlazione di Matthews, o MCC, è un punteggio che si utilizza per valutare la bontà globale di un modello di classificazione. La formula utilizzata è

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

e prende in considerazione tutte e quattro le categorie della matrice di confusione, perciò soltanto un modello bilanciato e che ottiene buoni riscontri in tutti e 4 otterrà un buon punteggio MCC. Il punteggio va da -1, che corrisponde a una previsione perfettamente inversa, a +1, che corrisponde a un modello che classifica perfettamente le istanze del test set. Il punteggio MCC ottenuto nell'esperimento è 0.143

5.4 Valutazione del modello in una simulazione di compravendita in borsa

Gli strumenti di valutazione appena mostrati sono comuni a tutti i domini applicativi quando si progetta un modello di classificazione, tuttavia, nel caso della borsa valori, al fine rendersi conto della effettiva utilità del modello creato è possibile simulare delle operazioni di compravendita per comprendere se il classificatore è effetivamente in grado di generare profitti comprendendo il mercato azionario. La simulazione avviene nel periodo coperto dai dati del test set: quando il modello prevede per una giornata un rialzo dell'indice S&P 500, acquista il titolo all'apertura dei mercati e lo rivende a fine giornata; viceversa

se il modello prevede un ribasso dell'indice, lo vende allo scoperto (naked short selling) a inizio giornata e lo ricompra a fine giornata.

Il guadagno indica il guadagno assoluto in dollari realizzato dal modello nel periodo coperto dal test set: il modello opera ogni giorno acquistando o vendendo allo scoperto una unità del titolo S&P 500. Si noti che i guadagni assoluti realizzati dal modello aumenterebbero se venisse investito più denaro. Se si operasse per esempio ogni giorno con 10 unità del titolo, il guadagno sarebbe moltiplicato di 10 volte.

Poichè il guadagno dipende dalla quantità di denaro investito, una misura più indicativa della efficacia pratica del modello è il *return of investment (ROI)*. Il ROI non è altro che rapporto tra il guadagno e la cifra investita. In questo caso si considera la cifra investita come la media del valore di apertura di S&P 500 nel periodo preso in esame.

$$ROI = \frac{guadagno}{average(apertura)}$$

Nella simulazione sui dati coperti dal test set il modello ottiene un ROI del 18%, evidenziando discrete capacità di ottenere profitti in un caso reale.

5.5 Confronto con il modello di riferimento

Il modello sviluppato ha una accuracy del 60.77%, rispetto a quella di [13], che è pari a 65.08%. Anche nel caso dello score MCC si ottiene un punteggio minore: 0.143 contro 0.435. La differenza più grande si vede invece nella simulazione di trading dove il paper di riferimento ottiene oltre il 160% di return of investment, contro il 18% del modello sviluppato.

Per quanto riguarda il ROI, va tuttavia notato che l'esperimento di questo elaborato utilizza una semplice routine che prevede di acquistare/vendere allo scoperto a inizio giornata e concludere l'operazione alla fine della giornata, nel paper invece il protocollo di compravendita utilizzato nella simulazione prevede di vendere anche all'interno della giornata non appena sia possibile realizzare un piccolo profitto (intraday trading). Per questo motivo non è possibile confrontare con precisione i risultati in una simulazione di mercato.

5.6 Conclusioni

In questo elaborato di tesi è stato dapprima illustrato il problema della predizione di borsa valori, mostrando i principali approcci nell'ambito del machine learning, compresi quelli che oggi rappresentano lo stato dell'arte. Sono state poi elencate le tecnologie utilizzate per creare ed addestrare dei

modelli di apprendimento automatico, facendo riferimento anche a quelli che sfruttano l'elaborazione e la comprensione del linguaggio naturale. Dopodiché, è stato modellato un esperimento con l'obiettivo di predire correttamente l'andamento dell'indice di borsa S&P 500 servendosi di un language model neural ed una rete neurale di classificazione, ispirandosi a un progetto simile facente però uso di tecnologie NLP differenti. L'esperimento è stato poi implementato, mostrando i passaggi chiave ed infine sono stati analizzati i risultati ottenuti servendosi sia delle tradizionali metriche di performance di modelli predittivi, sia simulando uno scenario reale con la compravendita di titoli sul mercato. Nelle valutazioni del modello, esso ha evidenziato discrete capacità di modellare l'andamento dell'indice e quindi di generare profitti economici in una simulazione di compravendita sul mercato azionario, sebbene i risultati ottenuti non siano paragonabili a quelli del progetto di riferimento.

Il modello ottenuto mostra in ogni caso come sia possibile adattare un language model neurale pre-addestrato come BERT al task della predizione della borsa valori, sfruttando con facilità informazioni di tipo testuale come le notizie, e lascia aperte le porte ad utilizzare questa metodologia per ottenere risultati migliori, ad esempio servendosi di più dati al fine di fornire più informazioni al classificatore durante l'addestramento, e per ottenere risultati più attendibili nella valutazione.

Ringraziamenti

Il primo ringraziamento va al relatore di questo lavoro, il Prof. Gianluca Moro, che ha reso possibile tutto ciò e ha acceso il mio personale interesse nei confronti di questa splendida disciplina.

Il ringraziamento più grande va alla mia famiglia per il supporto offertomi durante tutto il percorso di studi.

Bibliografia

[1] Burton G. Malkiel and Eugene F. Fama. Efficient capital markets: A review of theory and empirical work*. *The Journal of Finance*, 25(2):383–417, 1970.

[2] KARL PEARSON. The problem of the random walk. *Nature*, 72(1865):294–294, Jul 1905.

[3] Burton G. Malkiel. The efficient market hypothesis and its critics. *Journal of Economic Perspectives*, 17(1):59–82, March 2003.

[4] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3), Jun 2008.

[5] Yuling Lin, Haixiang Guo, and Jinglu Hu. An svm-based approach for stock market trend prediction. In *The 2013 International Joint Conference on Neural Networks, IJCNN 2013, Dallas, TX, USA, August 4-9, 2013*, pages 1–7. IEEE, 2013.

[6] Sneha Soni. Applications of anns in stock market prediction : A survey. 2011.

[7] Mirco Fabbri and Gianluca Moro. Dow jones trading with deep learning: The unreasonable effectiveness of recurrent neural networks. In Jorge Bernardino and Christoph Quix, editors, *Proceedings of the 7th International Conference on Data Science, Technology and Applications, DATA 2018, Porto, Portugal, July 26-28, 2018*, pages 142–153. SciTePress, 2018.

[8] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Handwritten digit recognition with a back-propagation network. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pages 396–404. Morgan Kaufmann, 1989.

[9] Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Forecasting stock prices from the limit order book using convolutional neural networks. In Peri Loucopoulos, Yannis Manolopoulos, Oscar Pastor, Babis Theodoulidis, and Jelena Zdravkovic, editors, *19th IEEE Conference on Business Informatics, CBI 2017, Thessaloniki, Greece, July 24-27, 2017, Volume 1: Conference Papers*, pages 7–12. IEEE Computer Society, 2017.

[10] PAUL C. TETLOCK. Giving content to investor sentiment: The role of media in the stock market. *The Journal of Finance*, 62(3):1139–1168, 2007.

[11] Johan Bollen, Huina Mao, and Xiao-Jun Zeng. Twitter mood predicts the stock market. *J. Comput. Sci.*, 2(1):1–8, 2011.

[12] David Cutler, James Poterba, and Lawrence Summers. What moves stock prices? NBER Working Papers 2538, National Bureau of Economic Research, Inc, 1988.

[13] Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. Deep learning for event-driven stock prediction. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2327–2333. AAAI Press, 2015.

[14] William Yang Wang and Zhenhao Hua. A semiparametric gaussian copula regression model for predicting financial risks from earnings calls. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 1155–1165. The Association for Computer Linguistics, 2014.

[15] Aurlien Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1st edition, 2017.

[16] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016.

[17] Michele Banko and Eric Brill. Scaling to very very large corpora for natural language disambiguation. In *Association for Computational Linguistic, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference, July 9-11, 2001, Toulouse, France*, pages 26–33. Morgan Kaufmann Publishers, 2001.

[18] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

[19] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014.

[20] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[22] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[23] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, and Douglas Eck. An improved relative self-attention mechanism for transformer with application to music generation. *CoRR*, abs/1809.04281, 2018.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[25] Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *CoRR*, abs/1506.06724, 2015.

[26] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.