ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI INFORMATICA-SCIENZA E INGEGNERIA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Predizione di valori azionari tramite reti neurali con memoria

Relatore:
Ch.mo Prof. Moro Gianluca

Candidato:
Fabbri Mirco

Correlatori:
Dott. Ing. Pasolini Roberto
Dott. Ing. Pagliarani Andrea

Anno Accademico 2016-2017

Indice

0 Introduzione

L'attività di seguito riportata è stata condotta con il fine di esplorare i principali concetti alla base del deep learning supervisionato ponendo particolare enfasi sulle reti neurali ricorrenti e con memoria valutandone un'applicazione concreta nel campo della previsione. L'attività è stata eseguita cercando di creare una transizione logica tra le basi dell'argomento ed i concetti rilevanti.
Nonostante al giorno d'oggi siano disponibili numerosi framework in grado di colmare l'abstraction gap generatosi dalle basi matematiche sulle quali poggia l'apprendimento neurale, sono state comunque approfondite le principali metodologie di aggiornamento dei pesi.
L'applicazione delle nozioni apprese ad un caso di studio reale ha consentito di comprendere appieno le problematiche relative alla progettazione della topologia e costruzione di reti neurali.
Nella prima di questo documento viene brevemente illustrata la storia delle reti neurali a partire dal modello lineare *McCulloch-Pitts* per giungere al percettrone multilivello. Nella seconda parte sono state approfondite le differenze tra i sistemi connessionisti artificiali e le reti neurali con memoria (ricorrenti) descrivendo alcuni dei modelli successivamente utilizzati nel progetto. In questa parte è stata descritta anche la Neural Turing Machine, innovativa rete neurale che si propone, insieme alla sua evoluzione Differential Neural Computer (DNC) di incrementare le capacità dei modelli tradizionali.
Nella terza parte viene introdotto il dominio applicativo di riferimento del progetto e si descrivono altre metodologie impiegati al suo interno. Viene quindi proposto un modello per la previsione del mercato azionario e ne vengono descritti gli esperimenti ed i risultati. In ultima analisi è stato approfondito l'emergente e promettente approccio di deep reinforcement learning, il quale consente di superare alcune delle limitazioni imposte dalla metodologia supervisionata.

1 Reti Neurali Artificiali (ANN)

1.1 Percettrone

«L'intelligenza artificiale è una disciplina che studia i fondamenti teorici, le metodologie e le tecniche che consentono la progettazione di sistemi hardware e sistemi di programmi software capaci di fornire all'elaboratore elettronico prestazioni che, ad un osservatore comune, potrebbero sembrare di pertinenza esclusiva dell'intelligenza umana» [8].

L'intelligenza artificiale nasce come diretta conseguenza dell'impulso apportato dalla teoria della computazione di Alan Turing e dalla tesi di Church-Turing secondo la quale un computer digitale è in grado di risolvere qualsiasi ragionamento formale. La principale difficoltà che si è immediatamente paventata agli studiosi verteva sull'incapacità di individuare meccanismi formali in grado di formulare un problema in maniera tale che potesse essere compreso ed eseguito da un calcolatore.
Su queste premesse, gli studiosi hanno ricercato per anni nuove metodologie che potessero conferire ad un computer capacità cognitive assimilabili a quelle umane abilitandolo alla risoluzione di problemi formulati in maniera alternativa.

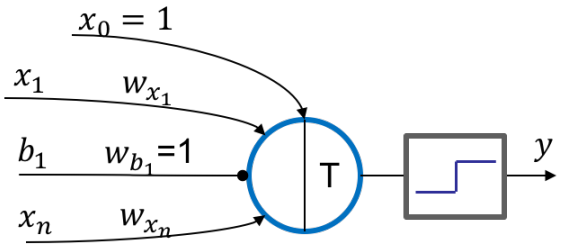


Figura 1: Percettrone

Nonostante i primi tenativi fossero puramente simboli, i quali tentavano di rappresentare qualsiasi cosa mediante simboli, il primo modello alla base delle moderne reti neurali fu formulato nel 1943 da due studiosi americani, Warren Sturgis McCulloch e Walter Pitts da cui deriva l'omonimo lavoro.
Il modello lineare McCulloch-Pitts ricalca il neurone del sistema nervoso umano il quale interagisce con altri neuroni tramite flussi ionici. Il meccanismo che consente la propagazione di un segnale elettrico all'interno del cervello è chiamato *"pompa del sodio"*. In situazione di riposo, la membrana di un neurone funge da barriera per le cariche positive espellendo nel contempo il sodio, nonostante il gradiente elettrico inverso. Un cambio repentino di voltaggio in una specifica sezione della membrana, causato da flussi ionici, porta il neurone a subire un'inversione di carica la quale verrà ripristinata dall'intervento della pompa del sodio. Questo implica una propagazione del flusso alle aree adiacente della membrana, che consegue nell'invio di un segnale da un neurone ad un altro.
Sulla base di quest'idea, il neurone McCulloch-Pitts è costituito da una serie di input $x_1, ...x_n$ (*variabili attivatrici*), $b_1, ...b_n$ (variabili inibitorie o bias), una serie di pesi relativi agli input $w_1, ...w_n$ ed una soglia T.

$$z = \sum_{i}^{n} w_{xi} x_i - \sum_{j}^{m} w_{bj} b_j$$

$$z = \begin{cases} 1, & z >= T \\ 0, & z < T \end{cases}$$

dove:

1. $x_1, ...x_n \in \{0, 1\}$
2. $b_1, ...b_n \in \{0, 1\}$, fornisce al neurone la capacità di attivarsi più o meno facilmente
3. $w_1, ...w_n \in \{1, -1\}$

Nel caso in esame la funzione di attivazione è a scalino separando linearmente gli insiemi di ingresso e suddividendo lo spazio in due partizioni. Come conseguenza, tutti gli operatori logici non lineari, quali lo XOR, non sono rappresentabili con un'unica cellula.

1.2 Percettrone multi-livello

L'evoluzione naturale del modello sopra rappresentato ha portato all'introduzione di un numero maggiore di neuroni a formare una rete di unità. E' possibile riconoscere due grandi tipologie di reti neurali:

1. Shallow: rete costituita da un livello di neuroni di input, un livello nascosto ed un livello di output

2. Deep: rete costituita da un numero di livelli nascosti maggiore di 1

Una seconda suddivisione che è possibile operare riguarda l'uscita dei neuroni:

1. Feedforward: ogni neurone è collegato a tutte le unità del livello successivo per cui non sono presenti cicli

2. Ricorrenti: ogni neurone è collegato a sé stesso ed a tutte le unità del livello successivo

La differenza tra le reti *deep* e *shallow* non è rilevante in termini computazionali in quanto per il teorema di *Kolgmogorov* una qualsiasi funzione continua

$$y = f(x) : R^n \rightarrow R^m$$

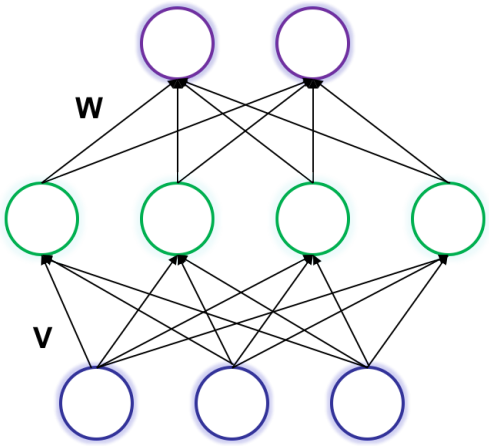
può essere risolta da un'opportuna rete ricorrente a 3 strati, con n unità nello strato di ingresso, $2n + 1$ nello strato nascosto ed m nello strato di uscita totalmente connessi tra loro. Come conseguenza di ciò, il comportamento di una rete neurale *deep* può essere approssimato tramite una rete neurale *shallow* con un numero sufficiente di unità nello strato nascosto.
La più semplice rete feedforward shallow realizzabile è costituita da 3 livelli *full connected*

$$y_k(t) = g(net_k)$$

$$net_k(t) = \sum_j w_{kj} y_j(t) + b_k$$

$$y_j(t) = f(net_j)$$

$$net_j(t) = \sum_i v_{ji} x_i(t) + b_j$$



dove v e w sono le matrici dei pesi degli archi che connettono rispettivamente il livello di input con il livello nascosto ed il livello nascosto con il livello di output.
Una rete feedforward opera quindi come un sistema combinatorio esprimibile come:

$$y(t) = G(F(x(t))) \tag{1}$$

Come risultato, il comportamento di una rete dipende esclusivamente dal valore del bias e dai pesi tra i collegamenti dei neuroni. Al crescere delle unità nascoste aumentano anche le relative connessioni rendendo pressoché impensabile una taratura manuale dei pesi e complicando l'individuazione di una topologia idonea per ogni specifico caso.
La ricerca della miglior configurazione di pesi può essere un processo automatizzato guidato dai dati e dalla tipologia di problema da affrontare:

1. **Apprendimento supervisionato**: richiede la conoscenza dell'output desiderato. Fornito in ingresso un pattern, adatta i pesi delle connessioni cercando di minimizzare le differenze fra il comportamento della rete e quello ricercato. I processi in cui è utilizzato questo approccio sono la classificazione e la regressione. Per approfondimenti sulle modalità di apprendimento vedere *Apprendimento supervisionato: aggiornamento pesi*

2. **Apprendimento non supervisionato** (o feature detection): i pesi vengono adattati ricercando un'auto-organizzazione della rete che rifletterà alcune caratteristiche del training set. La mancanza di misure di dissimilarità, come conseguenza dell'indisponibilità dell'output desiderato, complica il processo di comprensione e validazione dei risultati. Viene utilizzato nei casi di riduzione della dimensionalità, clustering, costruzione di modelli generativi e più in generale ovunque non sia disponibile una misura con cui determinare il discostamento rispetto al risultato desiderato.

3. **Apprendimento per rinforzo**: lo stato di partenza implica l'indisponibilità dei dati. Il processo consiste nella costruzione di un modello che genera dati, chiamati *azioni*, i quali possono essere basati su misurazioni o dati precedenti al fine di massimizzare una *funzione di ricompensa* (sconosciuta al modello stesso e che deve essere appresa). L'idea alla base di quest'approccio risiede nell'incapacità di generare un modello in grado di esprimere come ottenere un particolare risultato ma solamente se il risultato è corretto o sbagliato.

Nella sezione 7 verrà analizzato in maggior dettaglio il processo di apprendimento supervisionato.

2 Reti ricorrenti

La semplicità con la quale è definita una rete neurale feedforward gioca un ruolo fondamentale nelle limitazioni che essa presenta:

1. Indipendenza stocastica dell'input: gli esempi di training forniti in input sono assunti essere stocasticamente indipendenti. Tale assunzione limita le capacità della rete nell'individuazione di dipendenze temporali tra i pattern in ingresso. L'esempio più esplicativo di questa problematica ricade all'interno del NLP, in particolar modo con l'incapacità di una rete feedforward nell'imparare un modello di linguaggio. All'interno di una frase infatti, esistono *short-long long dependencies* dove termini successivi sono fortemente dipendenti da quelli precedenti. In una rete feedforward, tutti i termini sono assunti essere indipendenti gli uni dagli altri.

2. Lunghezza degli input prefissata: le dimensioni (numero di unità neurali) dello strato di input deve essere uguale alle dimensioni dell'input. Questo inibisce la possibilità di manipolare sequenze di dimensioni variabili.

A differenza di una rete feedforward, la quale opera come un sistema combinatorio (1), una rete ricorrente agisce come un *sistema sequenziale* associando un pattern in uscita ad uno in ingresso con una dipendenza da uno stato interno che evolve nel tempo in funzione degli input presentati:

$$\begin{aligned} y(t) &= G(s(t)) \\ s(t) &= F(s(t-1), x(t)) \end{aligned} \tag{2}$$

dove:

1. $y(t)$: output della rete all'istante t
2. $s(t)$: stato della rete all'istante t, il quale dipende dallo stato precedente e dall'input corrente

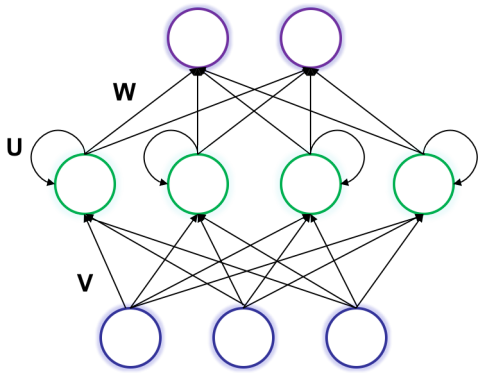
Per completezza, una rete ricorrente associa una dipendenza tra il suo stato interno iniziale ($t = 0$) e la sequenza di pattern in ingresso con la sequenza di pattern in uscita.

$$y_k(t) = g(net_k)$$

$$net_k(t) = \sum_j w_{kj} y_j(t) + b_k$$

$$y_j(t) = f(net_j)$$

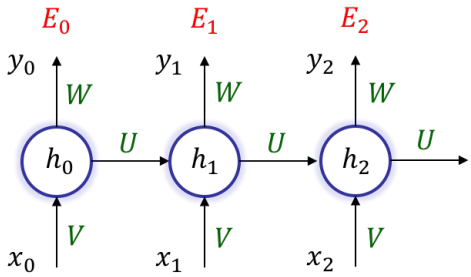
$$net_j(t) = \sum_i v_{ji} x_i(t) + u_{jj} y_j(t-1) + b_j$$



dove:

1. : U: è la *matrice di transizione*, simile alla catena di Markov. Rappresenta i pesi che connettono un neurone nello strato nascosto a sé stesso.
2. $y_j(t)$: rappresenta l'output di un neurone nello strato nascosto.
3. f, g: funzioni di attivazione (differenziabili) non lineari dei neuroni nello strato nascosto e di output. Le funzioni più utilizzate sono quella *sigmoidale* o *tanh* in quanto consentono di condensare valori molto grandi o molto piccoli in uno spazio predefinito.

E' facile notare come l'uscita di un neurone nello strato nascosto, all'istante *t*, dipenda non solo dall'input ma anche dallo stato del neurone all'istante *t-1*. Inoltre, siccome la retroazione occorre ad ogni intervallo di tempo, lo stato interno della rete all'istante *t-1* dipende da quello all'istante *t-2* che a sua volta dipende da quelli precedenti.
Tale meccanismo consente alla rete di catturare dipendenze temporali ed utilizzare informazioni su eventi passati per prevedere quelli futuri. Per meglio apprezzare il comportamento di una RNN, è possibile osservarla 'srotolata' nel tempo:



Ad ogni istante temporale (t), la rete produce un output (y_t), sulla base dell'input (x_t) e dello stato passato (h_{t-1}), a cui corrisponde un errore (C_t). Un vantaggio indiretto ottenuto da questa topologia consiste nella condivisione delle matrici *V, W, U* tra i vari istanti temporali che consente di ridurre il numero di parametri sconosciuti.

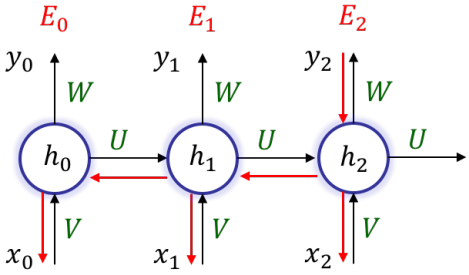
2.1 Backpropagation Through Time (BPTT)

A differenza delle reti feedforward, le quali operano in un unico istante temporale, le reti ricorrenti agiscono in periodi temporali differenti, per cui l'errore commesso al tempo t dipende anche da quello commesso all'istante *t-1* che a sua volta è funzione di quello precedente. Per tale motivo la retropropagazione dell'errore deve avvenire per tutti gli istanti temporali considerati dalla rete. Sapendo che i parametri *V, W, U* sono condivisi nel tempo:

$$\frac{\partial E}{\partial U} = \sum_t \frac{\partial E_t}{\partial U} \tag{3}$$

ovvero:

$$\frac{\partial E}{\partial U} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial U}$$



dove:

$$\frac{\partial h_t}{\partial h_k} = \prod_{k<i<=t} \frac{\partial h_i}{\partial h_{i-1}} \tag{4}$$

poiché definisce come l'errore sia trasportato nel tempo dall'istante t a quello k (con $k < t$). Tale comportamento può essere osservato graficamente ripercorrendo all'indietro tutti gli archi, che collegano temporalmente gli stati della rete $(h_0, h_1, ...)$, dall'istante t a quello k (questo equivale alla moltiplicazione delle derivate parziali che rappresentano i vari 'archi temporali'). Sapendo che il transito da un istante temporale all'altro dipende dalla matrice di transizione (U) e dall'output dei neuroni allo stato precedente:

$$\prod_{k<i<=t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{k<i<=t} U^T diag(f'(h_{i-1})) \tag{5}$$

dove:

1. U^T: è la matrice di transizione che contiene i pesi degli archi ricorrenti
2. diag: rappresenta una funzione che dato un vettore, ottenuto applicando la derivata prima della funzione di attivazione dei neuroni nello strato nascosto (all'istante precedente), lo trasforma in una matrice diagonale.

Si prega di notare che la notazione sopra espressa è in formato matriciale per cui con h_{i-1} ci si riferisce allo stato di *tutti* i neuroni nello strato nascosto all'istante $i-1$. Come risultato, il fattore $\frac{\partial h_t}{\partial h_k}$ è il prodotto di $t-k$ matrici jacobiane rappresentanti le transizioni temporali. Questo evidenzia il principale problema delle rete neurali ricorrenti ovvero il *dissolvimento / esplosione del gradiente*. Supponendo di considerare un periodo temporale sufficiente ed assumendo che i gradienti delle singole transizioni siano minore di 1, il prodotto di tutti i gradienti tenderà a 0. Viceversa, laddove i singoli gradienti siano maggiori di 1 il prodotto tenderà ad ∞. In [12] sono dimostrate le condizioni sotto le quali questo fenomeno si verifica.

2.2 Vantaggi e svantaggi

Tra i principali vantaggi vanno annoverati:

1. Matrici *U, W, V* condivise tra tutti gli istanti temporali

   Pochi parametri da imparare

   Possibilità di aggiornare i pesi 'in blocco'

2. Capacità di considerare il contesto passato sia recente che remoto

   Utile per costruire modelli di linguaggio

3. Flessibilità: potenziali impieghi ovunque sia richiesta la capacità di considerare il contesto

   Sentiment Analysis

   Traduzione automatica

   ...

Come già menzionato, lo svantaggio che rende questa rete soggetta a problemi di stabilità, in caso di contesti temporali 'lunghi', è il *dissolvimento / esplosione del gradiente*. Questo impedisce l'analisi di un contesto passato remoto seppur teoricamente possibile. Tuttavia esistono soluzioni in grado di alleviare tale problematica:

1. Esplosione del gradiente:

   Truncate backpropagation: il gradiente è calcolato all'interno di una finestra temporale fissata. Questa soluzione comporta la perdita del contesto negli istanti precedenti.

   Clip gradient: il gradiente è scalato all'interno di un range prefissato

2. Dissolvimento del gradiente:

   Utilizzo di una funzione di attivazione speciale: ReLu (rectified linear unit)

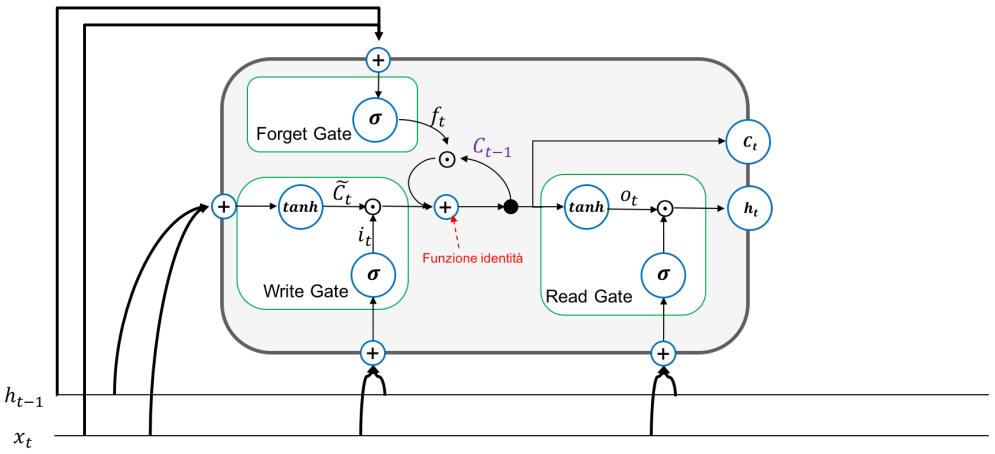
   LSTM, GRUs

2.3 Long-Short Term Memory (*LSTM*) [14]

Tra le varie soluzioni adottate per risolvere il problema del *dissolvimento / esplosione del gradiente* nelle RNN vi è l'utilizzo di una tipologia speciale di rete neurale che utilizza la *funzione identità* (la cui derivata è 1) come funzione di attivazione. Tale approccio prende il nome di *Constant Error Carousel* (CEC) e consente alle LSTM di mantenere costante l'errore durante la retropropagazione. Il solo utilizzo del CEC rende queste reti poco utili, in quanto non in grado di apprendere, per cui ad esso viene affiancato il concetto di ***gate*** definito come un'unità in grado di aprire o chiudere l'accesso al constant error flow:

1. Forget gate
2. Read / Output gate
3. Write / Input gate

Ogni unità LSTM è chiamata *memory unit* ed è costituita dai tre gates presentati in figura:



dove:

1. h_t: è lo stato nascosto del neurone, analogo a quello delle RNN. Agisce come una memoria a breve termine.
   h_{t-1} viene combinato all'interno dei vari gates al fine di determinare le informazioni utili da prelevare dalla memoria a lungo termine C_{t-1}.
2. C_t: rappresenta la memoria a lungo termine del neurone.
   C_{t-1} contiene la reale informazione a disposizione all'istante *t-1* e viene utilizzato al tempo t, insieme allo stato h_{t-1}, per determinare la nuova informazione C_t.
3. x_t: è l'input al neurone all'istante t.
4. $\tilde{C}_t$: rappresenta l'informazione apportata dall'input x_t (all'istante t) e fusa con quella proveniente dallo stato passato C_{t-1}

Il funzionamento della rete è strettamente correlato ai gates:

Forget gate: determina quale e quanta informazione proveniente dall'istante *t-1* (C_{t-1}) utilizzare all'istante *t*. Questa è poi combinata con l'input corrente.
Per far viene utilizzato un vettore (normalizzato di lunghezza n) f_t che consente di stabilire quali informazioni, provenienti dalla memoria a lungo termine all'istante *t-1*, devono essere riutilizzate al tempo corrente. Valori pari ad *1* indicano di ripresentare l'informazione passata all'istante corrente mentre valori pari a *0* indicano di scordarla completamente.
Va precisato che l'informazione reale della memory unit è contenuta all'interno di C_t. Il vettore h_t (stato del neurone) funge da selettore determinando, nei vari gates, quale informazione prelevare dalla memoria a lungo termine.

$$f_t = \sigma(x_t W_{xf} + h_{t-1} W_{hf} + b_i)$$

Write gate: indica come combinare l'informazione proveniente dalla memoria a lungo termine all'istante *t-1*, con quella in ingresso all'istante *t*. In particolar modo, a partire dall'input corrente (x_t), costruisce una proposta di informazione ($\tilde{C}_t$) che verrà successivamente fusa con quella della memoria a lungo termine dell'istante precedente (C_{t-1}). La proposta prodotta ($\tilde{C}_t$) contiene altresì informazioni irrilevanti che devono essere tralasciate. Per sopperire a tale problema, viene costruito un vettore normalizzato (i_t) il quale determina quali informazioni in ($\tilde{C}_t$) è conveniente utilizzare e quali da scartare. La moltiplicazione per componenti $i_t \odot \tilde{C}_t$ determina la nuova informazione da aggiungere alla memoria a lungo termine, proveniente dall'istante precedente, dopo che a questa è stato applicato il forget gate. C_t contiene quindi *tutta* l'informazione in output alla memory unit.

$$\tilde{C}_t = tanh(x_t W_{ci} + h_{t-1} W_{ai} + b_i)$$

$$i_t = \sigma(x_t W_{xi} + h_{t-1} W_{hi} + b_i)$$

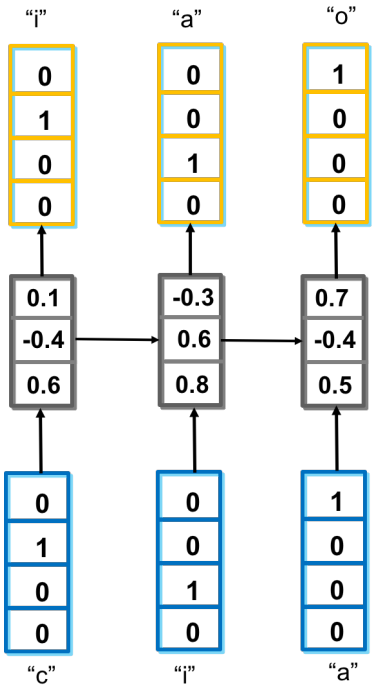
$$C_t = f_t \odot C_{t-1} \oplus i_t \odot \tilde{C}_t$$

Read gate: Aggiorna lo stato interno (h_t) della memory unit individuando in primo luogo un vettore o_t il quale indica quale parte dell'informazione contenuta in C_t, ottenuto tramite l'input gate, portare in output. Ciò consente di focalizzare l'attenzione solamente sulle informazioni immediatamente utili contenute all'interno della memoria a lungo termine tralasciando quelle che potrebbero diventarlo in futuro. Lo stato interno (h_t) è quindi aggiornato selezionando le informazioni utili da C_t, per mezzo di o_t, mediante una moltiplicazione per componenti.

$$o_t = \sigma(x_t W_{xo} + h_{t-1} W_{ho} + b_o)$$

$$h_t = o_t \odot tanh(C_t)$$

La peculiarità di questa tipologia di rete risiede nell'utilizzo di una funzione unitaria la cui derivata è 1. In questo modo il peso della ricorrenza ha valore unitario comportando la fusione (tramite moltiplicazione per componenti) dell'informazione contenuta nella memoria a lungo termine all'istante *t-1* (C_{t-1}) direttamente con l'output del forget gate. Se quest'ultimo è chiuso (f_t vicino ad 1), il gradiente non scema ma viene propagato all'indietro all'istante $t-1$.
L'introduzione del concetto di memoria a lungo termine, rispetto al solo stato nascosto già presente nelle RNN, consente quindi alle LSTM di considerare stati passati remoti (mediante C_t) fondendoli con quelli più recenti (catturati da h_t). In questo modo la rete è in grado di catturare dipendenze nel lungo periodo superando le limitazioni imposte dalla struttura delle RNN.
Una rete neurale LSTM è quindi in grado di imparare un modello di linguaggio del tipo:



Modello di linguaggio imparato:

$$P(c_t|c_{t-1}, c_{t-2}, ...c_0)$$

In un contesto reale, la probabilità condizionale mostrata in figura non sarebbe stimabile tramite un approccio completo in quanto raramente i data set hanno dimensioni tali da consentire un training soddisfacente. Soluzioni alternative sono l'utilizzo di classificatori bayesiani naive (che spesso non performano in maniera adeguata) o reti bayesiane. Queste ultime hanno la capacità, rispetto alle LSTM, di illustrare il processo che ha portato ad ottenere un determinato risultato, catturando quindi il rapporto *causa-effetto*. Il principale svantaggio risiede nella necessità di conoscere a priori il dominio applicativo all'interno del quale saranno impiegate.
Viceversa il vantaggio nell'utilizzo di una rete LSTM risiede nella flessibilità che questa fornisce in domini sconosciuti a discapito del procedimento che ha consentito di ottenere il risultato. Nei campi dell'apprendimento di modelli di linguaggio, sentiment analysis o traduzione automatica, le reti LSTM han-

no dimostrato di poter garantire ottimi risultati fornendo al contempo buona flessibilità.

3 Neural Turing Machine (*NTM*) [1]

Le RNN, ed in particolar modo le LSTM, spiccano per la loro capacità nel trattare sequenze di input che si estendono prolungatamene nel tempo. A ciò si aggiunge la qualità di Turing-Completezza che consente a queste reti di risolvere qualunque problema risolubile con la macchina di Turing, a patto che si individui una topologia adeguata.
I notevoli sforzi degli studiosi rivolti al conferimento di capacità cognitive ad un pc, per mezzo di reti neurali, si sono concentrati per lungo periodo sulle attività elementari (aritmetiche) che questo è in grado di compiere, tralasciando due aspetti fondamentali nel funzionamento di un calcolatore:

1. flusso di controllo logico (branching)
2. memoria (architettura Von Neuman o Hardvard)

Soltanto negli ultimi anni si è ricercata una visione d'insieme che consentisse di espandere le potenzialità di una RNN mediante un flusso di controllo ed una memoria di grandi dimensioni, analogamente ad una Turing-Machine con nastro infinito. Il risultato di questo lavoro sono le **NTM** ovvero computer differenziabili, implementati tramite reti neurali, in grado di interagire con memorie esterne, a loro volta realizzate per mezzo di reti neurali.
Le NTM riprendono, dalla cognizione umana, il concetto di *working memory* ovvero la capacità di memorizzare informazioni per un breve periodo e di elaborarle tramite *rule-based manipulation*. In termini computazionali, tali regole sono semplici programmi i cui argomenti sono le informazioni salvate in memoria. Lo step evolutivo che è stato compiuto con le NTM consente quindi a queste reti di gestire uno spazio di memoria, allocando o recuperando informazioni da essa. Tutto ciò è possibile grazie ad un secondo concetto ripreso dalla cognizione umana ovvero il *processo attentivo* il quale consente di attuare una reazione come conseguenza di alcuni stimoli esterni tralasciandone altri.
In ultima analisi le NTM, essendo computer differenziabili realizzati per mezzo di reti neurali, sono in grado di imparare autonomamente a leggere e scrivere selettivamente da una memoria.

3.1 Struttura

Tre sono i componenti principali di una NTM:

1. Controller: è una rete neurale, feedforward o LSTM che può essere percepita come il cervello che comanda la scrittura e la lettura dalla memoria. Se questo è realizzato per mezzo di LSTM, la NTM deriva anche tutte le proprietà delle reti con memoria quali, per esempio, la capacità di accettare sequenze temporali in input.
   D'altro canto, l'utilizzo di una rete feedforward full-connected semplifica l'implementazione introducendo una consistente limitazione: il collo di bottiglia imposto dal numero di letture/scritture concorrenti. La rete sarà quindi in grado di eseguire un numero di trasformazioni concorrenti pari

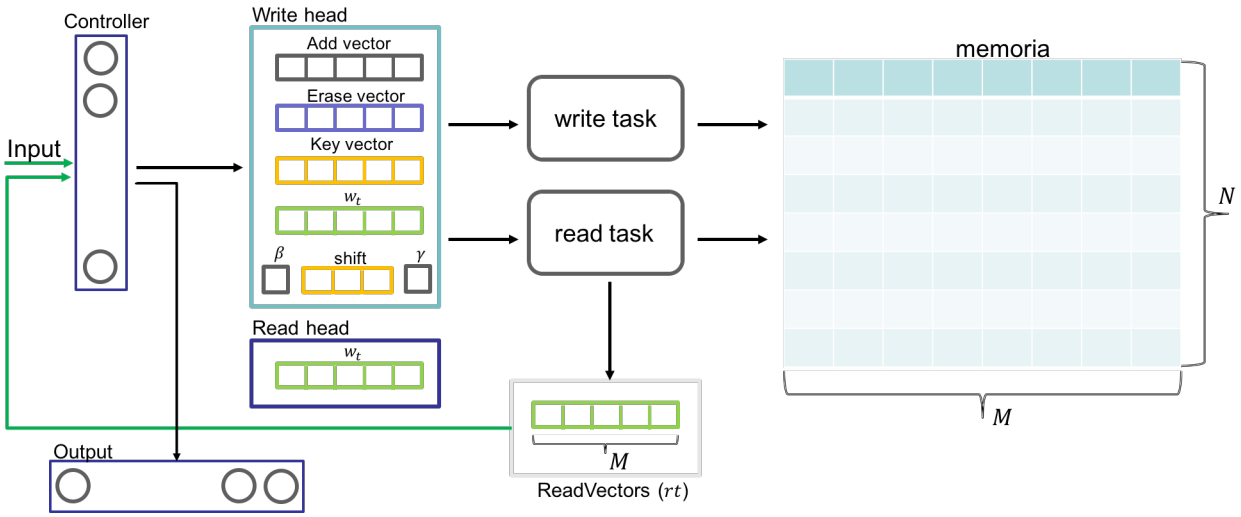


Figura 2: Struttura di una NTM

al numero di head. Per esempio con una read head sarà possibile eseguire trasformazioni unarie su un singolo vettore di memoria ad ogni istante t, mentre con due read head la rete sarà in grado di eseguire trasformazioni binarie,...
Le RNN (o LSTM) non soffrono di questo problema grazie alla capacità di 'memorizzare' i read vector all'istante *t-1*.

2. Heads: consentono l'accesso alla memoria previo comando del controller:

   - Read head: è il componente che si occupa della lettura in memoria.
   - Write head: è il componente che si occupa della scrittura in memoria.

   Possono esistere un numero arbitrario di heads e tutte quante si trovano nello stesso strato neurale.

3. Memoria: costituita da una matrice nxm di neuroni che può essere percepita come la RAM di un calcolatore. Ogni riga della matrice è chiamata *location*.

3.2 Comportamento

Il comportamento di una NTM si suddivide in tre fasi:

1. Lettura
2. Scrittura
3. Indirizzamento

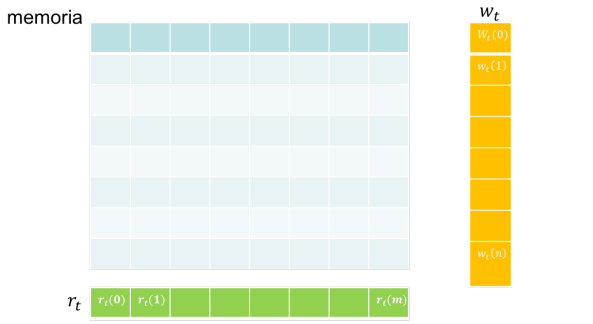


Figura 3: NTM: lettura

3.2.1 Lettura

La lettura è possibile sfruttando l'idea di processo attentivo tramite il quale un sistema è in grado di reagire a stimoli esterni tralasciandone altri. Alla base di questo processo c'è il *vettore di focusing* $w_t \in R^N$ che, per ogni location, indica quanto la lettura debba essere 'intensa':

$$r_t = \sum_i w_t(i) \odot M_t(i)$$

$$\sum_i w_t(i) = 1, \qquad 0 <= w_t(i) <= 1, \forall i$$

dove:

1. $M_t(i)$: location di memoria i (riga della matrice di memoria).
2. r_t: risultato della lettura ottenuto come combinazione convessa del vettore di focusing e la memoria

Si può facilmente notare come w_t sia delegato ad indicare su quale location di memoria concentrare la lettura. Se $w_t(0) = 1$, tutte le altre componenti del vettore sono necessariamente a *0* per cui la lettura si concentrerà esclusivamente sulla location di memoria $M_t(0)$.

3.2.2 Scrittura

L'operazione di scrittura trae ispirazione dal concetto di gate già presentato nelle LSTM. Tale operazione è scomposta in due parti:

1. Erase: analogo al gate *forget* delle LSTM, determina quale informazione presente in memoria all'istante $t-1$ debba essere preservata all'istante t:

   $$\tilde{E}_t(i) = (1 - w_t(i) \odot e_t(i))$$

   dove:

   (a) *1*: vettore di dimensioni M composto dai valori scalari *1*.

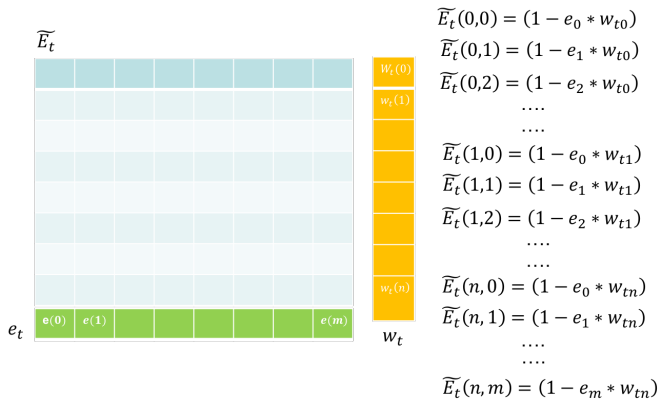


Figura 4: NTM: Erase

(b) e_t: vettore normalizzato (tipicamente tramite funzione sigmoidale) di dimensioni M che determina quanta informazione presente in memoria all'istante precedente ($M_{t-1}(i)$) debba essere preservata al tempo corrente.

L'operazione di *erase* produce una matrice $\tilde{E}_t$ (NxM) che determina l'intensità con la quale ogni cella della memoria deve dissolvere il suo valore. Moltiplicando la matrice $\tilde{E}_t$ con la matrice della memoria M_{t-1} si ottiene il nuovo stato della memoria, la quale ha 'dimenticato' alcune informazioni dello stato precedente.

2. Add: determina quanta nuova informazione debba essere aggiunta alla memoria dello stato precedente dopo che a questa è già stato applicato il processo di erase:

$$\tilde{A}_t(i) = w_t(i) \odot a_t(i)$$
$$M_t(i) = (M_{t-1}(i) \odot \tilde{E}_t(i)) + \tilde{A}_t(i)$$

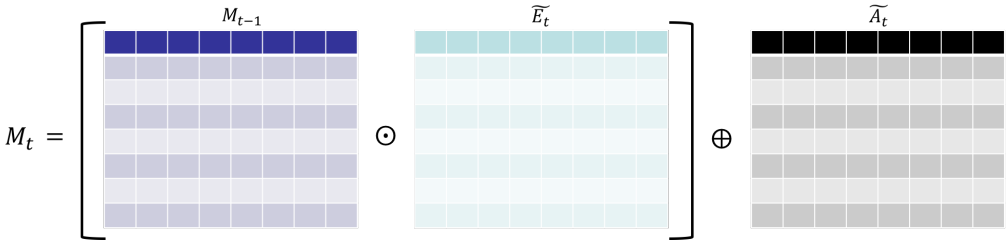
dove:

(a) a_t: vettore che rappresenta le informazioni da aggiungere alla memoria all'istante t.

(b) $\tilde{A}_t$: matrice 'add' ($\tilde{A}_t$), analoga alla matrice 'erase', che induce a focalizzare l'attenzione, in merito alle informazioni da aggiungere alla memoria, su determinate location.

(c) M_t: memoria all'istante t.

La prima equazione ha lo scopo di produrre una matrice, contenente le informazioni da aggiungere alla memoria, pesata secondo il vettore di focusing w_t. Se $w_t(0) = 1$, allora tutte le altre componenti di w_t saranno a 0 per cui verrà aggiornata la sola location $M_t(0)$ con le informazioni relative a $a_t(0)$.
La seconda equazione fonde le informazioni 'dimenticate' dallo stato precedente ($\tilde{E}_t$) con quelle da aggiungere all'istante corrente ($\tilde{A}_t$).



E' utile sottolineare che tutte le operazioni matriciali sono per-componenti. Le operazioni qui sopra illustrate descrivono il processo di lettura / scrittura per una sola head. Nel caso in cui la rete contenga più heads, i processi sono ripetuti per ognuna di queste.

3.2.3 Indirizzamento

Dalle operazioni di lettura e scrittura si evince chiaramente come la capacità della rete di accedere a determinati contenuti in memoria sia vincolata al vettore di focusing w_t. Questo infatti stabilisce su quali location la rete debba concentrarsi durante i processi di lettura / scrittura. Il vettore di focusing viene aggiornato durante la fase di *addressing* in due modi:

1. Focus by-content: l'obiettivo di questo approccio è quello di costruire una sorta di memoria associativa che consenta la ricerca di un pattern (anche approssimato).
   Ogni head (sia in lettura che scrittura) produce un vettore k_t di dimensioni M, il quale rappresenta la *chiave* (pattern approssimato) che verrà confrontata, mediante una misura di similarità (es: similarità coseno), con ogni location di memoria. Tale misura viene modulata per mezzo di un parametro β e poi normalizzato rispetto a tutte le location:

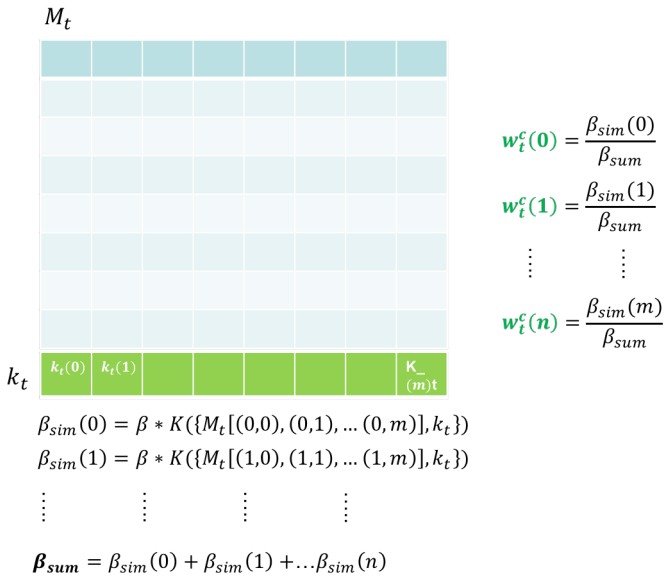
   $$w_t^c(i) = \frac{\beta_t K[k_t, M_t(i)])}{\sum_j exp(\beta_t K[k_t, M_t(j)])}$$

   dove:

   (a) β_t: è il fattore che consente di modulare il risultato della misura di similarità

   (b) K: rappresenta una misura di similarità (es: similarità coseno)

   (c) k_t: è il vettore che rappresenta il contenuto (anche approssimato) da ricercare in memoria.



2. Focus by-location: quando il contenuto da ricercare non è conosciuto a priori, l'approccio 'by-content' non è applicabile. Come alternativa viene fornita la possibilità di accedere alla memoria tramite posizione/indirizzo abilitando la rete ad eseguire salti relativi incondizionati all'interno della memoria, indipendentemente dal contenuto. Le due modalità di accesso sono quindi complementari. Il focus 'by-location' è diviso in tre fasi:

   (a) Interpolazione: il vettore di focusing all'istante *t-1* è interpolato con quello prodotto al tempo corrente dal focus 'by-content':

   $$w_t^g = g_t w_t^c + (1 - g_t) w_{t-1}$$

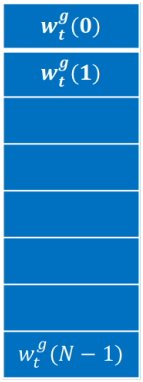
   dove g_t è il fattore di interpolazione (1 utilizza solamente il vettore di focusing prodotto al passo t, 0 solamente quello proveniente dall'istante precedente).

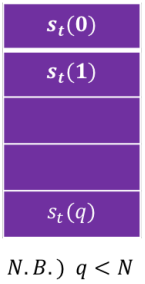
   (b) Shifting: in questa fase viene definito di quante celle debba essere traslato il focusing derivante dal passo di interpolazione. In questo modo, è possibile trasferire l'attenzione della rete alle locations adiacenti, a quella indicata dal vettore di focusing w_t^g. Per realizzare tale comportamento, ogni head emette un proprio vettore di shifting s_t che definisce una distribuzione normalizzata degli offset ammessi:

   $$\tilde{w}_t(i) = \sum_{j}^{N-1} w_t^g(j) s_t(i - j)$$

   Supponendo che gli offset ammessi siano $(-1, 0, 1)$ (ovvero è possibile traslare il vettore di focusing w_t^g indietro di una location, in avanti di una location o lasciarlo invariato), un vettore di shifting $(0.0, 0.3, 0.7)$ comporta un'intensificazione dell'attenzione della rete nei confronti della location successiva (offset $+1$) pari ad un'intensità di 0.7 ed un

incremento rispetto alla location attuale (offset 0) pari a 0.3. Il vettore di shifting indica alla rete di non porre attenzione sulla location precedente. Come risultato di ciò, la rete sarà in grado di focalizzare il proprio interesse solamente sulle locations indicate dal vettore di shifting.





$N.B.)\ q < N$

$$\begin{aligned}\tilde{w}_t(0) = {} & w_t^g(0)s_t((N+(0-0)\%N) + \\ & w_t^g(1)s_t((N+(0-1)\%N) + \\ & \ldots\ldots\ldots\ldots\ldots\ldots + \\ & \ldots\ldots\ldots\ldots\ldots\ldots + \\ & w_t^g(N-1)s_t((0+(N-1)\%N)\end{aligned}$$

(c) Sharpening: la convoluzione applicata in fase di shifting consente di focalizzare l'attenzione in principal modo sulle location desiderate ma tende anche a distribuire parte dell'attenzione su tutte le rimanenti locations. La fase di sharpening ha il compito di enfatizzare le location con pesi di focusing maggiori marcando la differenza di pesi tra le location desiderate e le altre:

$$w_t(i) = \frac{\tilde{w}_t(i)^{\gamma_t}}{\sum_j \tilde{w}_t(j)^{\gamma_t}}$$

dove γ_t è un valore scalare emesso da ogni head. Non viene specificato come tale valore debba essere calcolato ma è solamente imposto il vincolo $\gamma_t >= 1$ (spesso utilizzata la funzione $log(e^g + 1) + 1$. Il risultato dell'operazione di sharpening è il vettore di focusing aggiornato all'istante t.

N.B. Il processo di addressing può svolgersi come combinazione dell'indirizzamento 'by-content' e 'by-location':

1. solo 'focusing by-content': il vettore di focusing w_t è aggiornato utilizzando solo il contenuto

2. solo 'focusing by-location': il vettore di focusing w_t è aggiornato applicando solamente lo shifting e lo sharpening (senza interpolazione) al vettore di focus dell'istante precedente (*t-1*).

3. 'focusing by-content' seguito da 'focusing by-location': il vettore di focusing w_t prodotto dal focusing 'by-content' viene poi elaborato dal focusing 'by-location' abilitando la rete a ricercare un contenuto in memoria (es: una variabile) e spostare successivamente l'attenzione ad una location adiacente. In questo modo, una head è in grado di accedere ad un blocco di dati contiguo rispetto a quello su cui è posto il focus dal focusing 'by-content'.

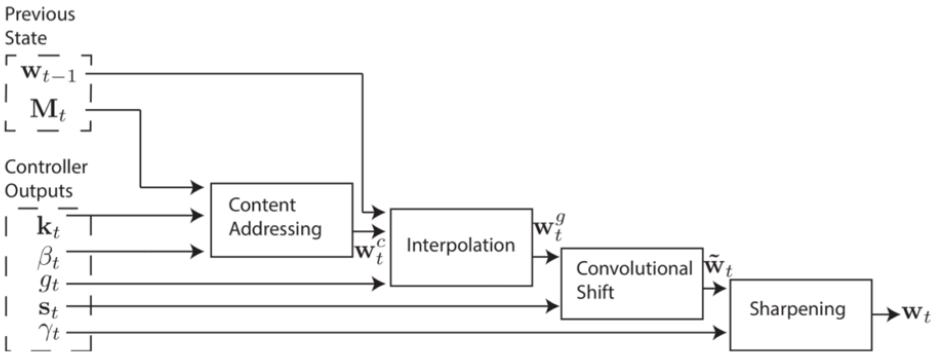


Figura 5: Comportamento NTM

4 Reti neurali artificiali per la previsione del mercato azionario

L'intelligenza artificiale ha subito una notevole diffusione negli ultimi anni grazie alla possibilità di essere sfruttata in vari contesti e domini applicativi. La sua poliedricità ha consentito di ottenere risultati positivi in svariati ambiti tra i quali il campo finanziario. In particolar modo, è fortemente attiva la ricerca nel settore dello *stock market prediction* il cui obiettivo consiste nel determinare il valore azionario di una società, o di altri strumenti finanziari, in una borsa.

4.1 Stock Market Prediction

Per comprendere come le reti neurali possano essere impiegate nel campo della previsione finanziara è fondamentale individuare le principali componenti in gioco. La famosa teoria del professor Eugene Fama *Efficient Market Hypothesis* [5], secondo la quale i valori dei titoli quotati in borsa riflettono l'informazione disponibile in un preciso istante nel mercato mobiliare, afferma che sia è impossibile *"battere il mercato"* poiché tutti gli investitori in gioco dispongono delle medesime informazioni necessarie per promuovere investimenti di carattere finanziario. Tale situazione implica che, in un mercato a somma zero, un investitore potrà ottenere un profitto esclusivamente anticipando le mosse dei competitors, operando movimenti diversi, ed assumendosi di conseguenza alcuni rischi.
Questa teoria suggerisce che il cambiamento del valore di qualsiasi azione è intrinsecamente imprevedibile se non fondato su nessuna nuova informazione, in quanto l'attuale valore riflette già tutte le informazioni disponibili pubblicamente nel dato istante.
Nel 1973 Burton Malkiel nel suo lavoro *A Random Walk Down Wall Street* [4] appoggia la teoria *dell'Efficient Market* sostenendo che la previsione dei valori azionari non può essere accurata tramite la sola osservazione della storia passata. Secondo Makiel tali valori possono essere descritti tramite un processo stocastico denominato *"random walk"*, secondo cui la varianza giornaliera dei prezzi segue un andamento casuale ed indipendente l'uno dall'altra. Una prima formalizzazione di questo processo è stata fornita da *Cuthbertson e Nitzsche* nel 2004:

$$Y(t) = Y(t-1) + \epsilon(t)$$

secondo la quale una random walk è definita come una serie stocastica Y_t le cui variazioni (ϵ) hanno media 0 e sono indipendenti l'un l'altra. Il cambiamento $\epsilon(t) = Y(t) - Y(t-1)$, occorrente all'istante t, presuppone indipendenza rispetto alle variazioni passate $(\epsilon(t-1), \epsilon(t-2), ..)$. Tale teoria indica che i mercati agiscono come se fossero governati da una legge stocastica in quanto le informazioni, che fanno variare i mercati medesimi, impattano in maniera non prevedibile e quindi casuale. Di conseguenza se *Y(t)* è un processo random walk la miglior previsione del valore all'istante *t+1* è quello dell'istante *t*. Tale assunzione ha per anni fatto discutere in merito alla validità del modello ma prove empiriche hanno supportato l'affermazione dell'applicabilità generale della teoria. Come conseguenza di questa asserzione, l'andamento storico dei valori di un'azione non può essere considerato un indicatore affidabile per la previsione, a lungo termine, dei prezzi futuri. Alla luce di quanto illustrato, l'ipotesi di *Efficient Market* indica che i mercati finanziari sono già efficienti e che i relativi prezzi

rispecchiano non solo le informazioni pubbliche attualmente disponibili ma anche le aspettative future, dei traders, in merito a profitti e compensi. Secondo l'EMH un mercato rimane stabile finché non vi sono nuove informazioni di dominio pubblico disponibili ma considerando che l'immissione di tali informazioni avviene in maniera casuale e non predicibile, è possibile notare come esista una forte correlazione tra l'ipotesi di *Efficient market* e la fluttuazione stocastica dei valori azionari (*random walk*).
Esistono tuttavia diversi lavori, soprattutto a livello accademico, che sostengono che l'andamento del mercato, seppur tendenzialmente stocastico, è in parte prevedibile. Un paper particolarmente famoso è *Stock Market Prices do not follow random walks* [2] all'interno del quale gli autori (*Andrew Lo, MacKinlay*) affermano l'esistenza di relazioni tra il valore degli stock market ed alcune macrovariabili finanziarie fondamentali quali il rendimento e/o il cash flow. Studi analoghi sono quelli di *Fama, French* (1992) e *Lakonishok, Sheifer, Vishny* (1994) all'interno dei quali viene ribadita l'esistenza di alcune correlazioni già individuate da *Lo e MacKinlay*. Tali risultati non contraddicono pienamente la teoria dell'Efficient Market in quanto come illustrato da *Ferson e Harvey* la possibilità di prevedere i prezzi azionari non è necessariamente dovuta all'inefficienza del mercato ma alla predicibilità di alcune macrovariabili ad essi correlate.

4.2 Reti neurali per la stock market prediction

L'impossibilità di costruire un modello accurato con il quale prevedere l'andamento del mercato ed i recenti sviluppi nel campo dell'intelligenza artificiale, hanno spinto molti ricercatori ad impiegare diverse tipologie di reti neurali per prevedere il trend azionario. Vista la naturale propensione delle reti nel pronosticare la classe di appartenenza, piuttosto che individuare il valore esatto è ragionevole pensare che i migliori risultati possano essere ottenuti addestrando le stesse nel fornire un risultato binario ad indicare se una particolare azione deve essere comprata o venduta in un dato istante. Inoltre, in virtù di quanto illustrato nella sezione precedente, la natura stocastica del mercato azionario difficilmente consente una previsione accurata mediante l'uso esclusivo dei valori delle azioni nella storia passata.
Sulla base di quanto affermato dall'EMH, lo stato dell'arte nella previsione del mercato azionario non può prescindere dal considerare le informazioni pubblicamente disponibili in accoppiata con i valori azionari. Tali informazioni hanno tipicamente un formato non-strutturato (notizie di giornali, quotidiani,...) il quale ha da sempre presentato maggiori difficoltà di analisi rispetto alla controparte strutturata. La continua e costante crescita di disponibilità di dati non strutturati consente altresì di scongiurare problematiche tipiche dei dati strutturati quali l'overfitting, causato da data-set di piccole dimensioni.
In primo luogo, per poter analizzare informazioni non strutturate è necessario costruire una rappresentazione semantica delle stesse ed esistono diverse metodologie in grado di farlo; tra queste è possibile citare tf-idf, Latent Dirichlet Allocation, o k-means, ma lo stato dell'arte oggigiorno è paragraph-vector.

4.3 Rappresentazione semantica del testo

La maggior parte degli algoritmi di machine learning richiedono che l'input sia rappresentato come un vettore di features di dimensioni non-variabili. Un

approccio particolarmente diffuso nel campo del text-mining, che soffre del suddetto problema, è sicuramente *Bag-of-Words* (o *Bag-of-n-grams*) che consente di relazionare documenti e termini in funzione della frequenza di questi ultimi. Il principale svantaggio di questa metodologia risiede nell'incapacità di catturare la semantica dei termini e l'ordinamento degli stessi. Esistono certamente soluzioni più avanzate per l'estrazione di features quali la *Latent semantic analysis* che sfrutta la decomposizione a valori singolari per ridurre dimensionalmente lo spazio di ricerca, le cui dimensioni significative sono indicate tramite autovalori, con il fine di far emergere associazioni semantiche (latenti) di ordine superiore. Una soluzione alternativa, che ha consentito di ottenere risultati allo stato dell'arte nel campo della sentiment analysis, è stata proposta nel 2014 da *Mikolov, Le* ed è chiamata *Paragraph Vector* [11]. Tale approccio consente di ottenere una rappresentazione distribuita di un documento o di una frase, avente un numero di features variabile, superando le problematiche riscontrate con le precedenti metodologie sopraccitate. *Paragraph Vector* è quindi un algoritmo non-supervisionato, basato su reti neurali, in grado di operare su testi di dimensioni arbitrarie.
L'idea su cui poggia questo metodo è un'estensione di un precedente lavoro *Word2Vec*, il quale si prefigge lo scopo di costruire una rappresentazione distribuita di parole all'interno di un vettore, e si basa su precedente paper *A Neural Probabilistic Language Model* [17].

4.3.1 Modello di linguaggio probabilistico [17]

Bengio et al. (2003) all'interno del lavoro ha ipotizzato un nuovo modello probabilistico per l'apprendimento di un linguaggio, basato su reti neurali, in grado di risolvere una delle problematiche maggiori nel campo dell'NLP, ovvero la *curse of dimensionality*, soprattutto in presenza di variabili discrete. Nel caso di spazi altamente multi-dimensionali, infatti, le distanze (euclidea o di hamming) che intercorrono tra i vari record tendono ad appiattirsi impedendo l'individuazione di istanze "vicine" nel dominio in esame. In tali contesti sarebbe auspicabile concentrare la distribuzione di probabilità ove risulta essere più importante piuttosto che distribuirla uniformemente in tutte le direzioni vicino ad ogni training point. L'idea alla base del modello ipotizzato da *Bengio et al.* può quindi essere riassunto in tre passaggi:

1. associare ad ogni termine del vocabolario un vettore reale che rappresenta diversi aspetti (*feature*) di una parola e che può essere percepito come un punto nello spazio.

2. calcolare la probabilità condizionata di un termine nella sequenza in funzione dei vettori di *features* delle altre parole nella stessa sequenza.

3. addestrare una rete neurale affinché impari i vettori di features (rappresentanti le parole) ed i parametri della funzione di probabilità (dove il numero di features è molto minore della cardinalità del vocabolario).

Un modello statistico di linguaggio può essere descritto come:

$$\hat{P}(w_1^t) = P(w_1)P(w_2|w_1)P(w_3|w_1,w_2),...P(w_t|w_1,...w_{t-1}) \approx \prod_{t=1}^{t} \hat{P}(w_t|w_1^{t-1})$$

dove la funzione di probabilità è data dal prodotto delle probabilità condizionate dei singoli termini considerando tutte le parole facenti già parte della sequenza. Un modello del linguaggio naturale può altresì ridurre la complessità nella modellazione, sfruttando l'ordine delle parole e considerando di conseguenza solamente l'*n-gram* nell'intorno di un termine, semplificando:

$$\hat{P}(w_t|w_1^{t-1}) \approx \hat{P}(w_t|w_{t-n+1}^{t-1})$$

dove n è la dimensione scelta dell'*n-gram*.
Due sono le principali innovazioni apportate dal lavoro di *Bengio et al.*:

1. utilizzo di vettori n-dimensionali (associati a punti nello spazio) per rappresentare la distribuzione di probabilità delle sequenze di parole in un linguaggio testuale. Fino ad allora la rappresentazione di termini tramite vettori associati ad uno spazio veniva impiegata per descrivere la probabilità di co-occorrenza di più parole in uno stesso documento (es: information retrival con LSI).

2. utilizzo di una rete neurale il cui compito è quello di applicare generalizzazione rispetto alla sequenza di training. La rete "impara" quindi il modello statistico della distribuzione delle parole piuttosto che le regole dei termini nella frase.

L'obiettivo è quello di permettere alla rete di imparare un buon modello $f(w_t, ...w_{t-n+1}) = \hat{P}(w_t|w_1^{t-1})$ per il linguaggio naturale. Imponendo che l'input alla rete non sia costituito dai vettori reali indicanti le *features* delle parole, ma da vettori *one-hot* (vettori in cui una sola componente ha valore 1 e tutte le altre 0) rappresentanti gli indici delle parole nel vocabolario, è possibile calcolare la funzione $f(w_t, ...w_{t-n+1})$ in due passaggi:

1. Ognuno degli n (dove n sono le dimensioni dell'*n-gram* considerato) vettori *one-hot* i (rappresentanti gli indici delle parole nel vocabolario) fornito in input alla rete, è mappato in un vettore distribuito e reale tramite una funzione $C(i) \in R^m$, dove m è il numero di termini distinti nel corpus . Ogni vettore risultante rappresenta il vettore distribuito di features della parola i.

2. Gli n vettori di features $(C(w_{t-n+1}, ...C(w_{t-1})))$, ottenuti dal passaggio precedente, vengono mappati (tramite una funzione g) in una distribuzione di probabilità condizionate rispetto al vocabolario V per il termine successivo w_t. L'output della funzione g è un vettore in cui la componente i-esima stima la probabilità $\hat{P}(w_t = i|w_1^{t-1})$.

nota: è importante sottolineare la differenza tra l'utilizzo di vettori *one-hot* e vettori distribuiti:

1. *one-hot*: ogni vettore costituisce una relazione biunivoca tra il vettore stesso ed una parola nel vocabolario. E' il formato con cui vengono forniti in input gli indici alla rete.

2. *vettori distribuiti*: la rappresentazione di una parola non è più mappata uno-uno come con i vettori *one-hot* ma è distribuita su tutte le componenti del vettore ed ognuna di esse contribuisce a definire il significato per tutte le parole. E' il formato in output dal projection layer.

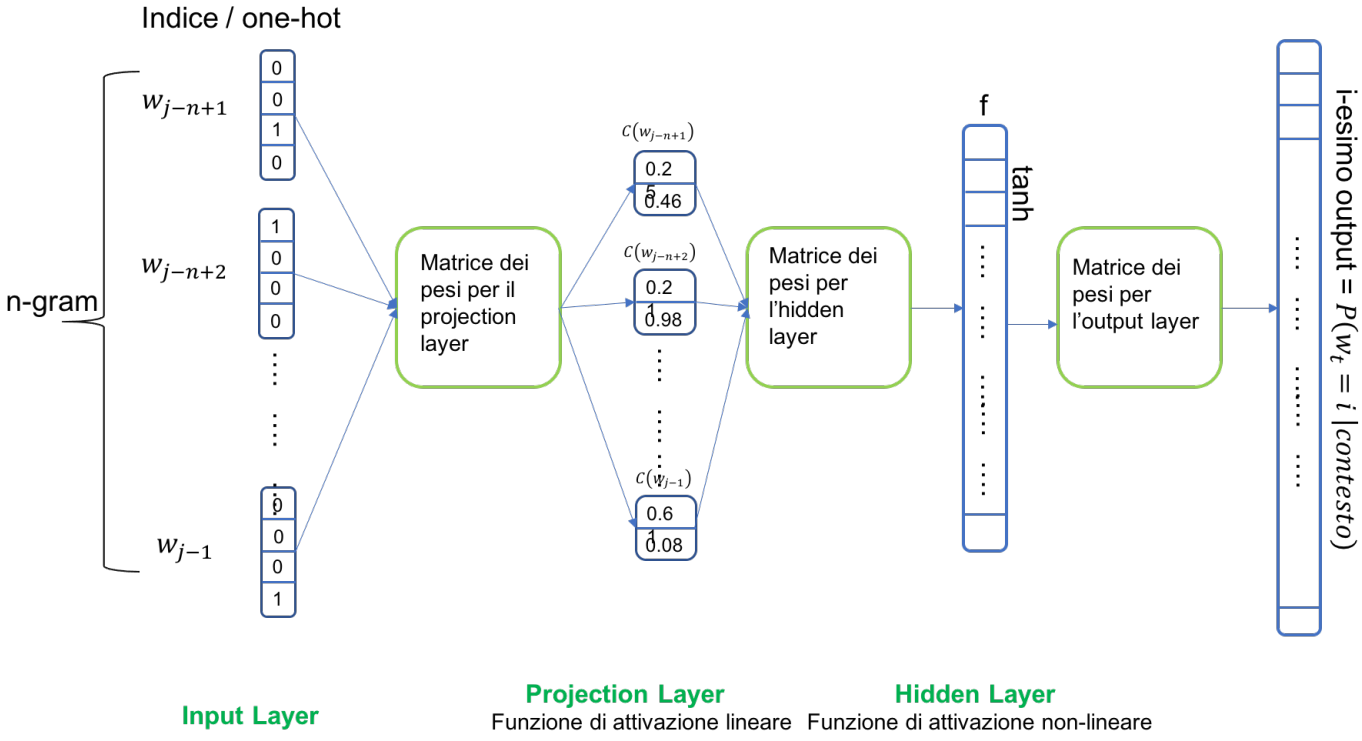


Figura 6: modello neurale

Struttura modello neurale per l'apprendimento di un linguaggio

1. *Projection layer*: l'obiettivo di questo livello è quello di mappare gl'indici (*one-hot*) dei termini in uno spazio reale ridotto (rappresentato da vettori distribuiti). A tal fine, la funzione di attivazione dei neuroni è strettamente lineare.
   nota: la matrice dei pesi utilizzata in questo livello è condivisa da tutte le parole nel contesto. Un aspetto importante da sottolineare deriva dal fatto che l'ordinamento delle parole che formano l'*n-gram* (contesto) non è catturato dalla matrice dei pesi del projection layer ma è ottenuto solamente a valle di tale livello ed è 'imparato' dall'hidden layer.

2. *Hidden layer*: il compito di questo livello è quello di calcolare la rappresentazione di probabilità condizionate per ogni possibile parola nel vocabolario conoscendo il contesto (*n-gram*) $w_{t-1}, ...w_{t-n+1}$. L'uscita di questo livello rappresenta esattamente il risultato della funzione f sopra menzionata.
   In questo layer viene utilizzata come funzione di attivazione *tanh*, la quale ha una velocità di convergenza maggiore rispetto alla funzione sigmoidale spesso utilizzata. L'Hidden layer può essere realizzato mediante una rete feed-forward o ricorrente.

3. *Output layer*: in questo livello, il vettore in output dall'hidden layer viene schiacciato mediante la funzione non-lineare *softmax*, all'interno dell'intervallo [0,1]. Come conseguenza, è possibile interpretare in maniera probabilistica il risultato associando all'indice di ogni componente del vettore in uscita, la parola (nel vocabolario) a cui è associato il medesimo indice. Il termine nel vocabolario associato all'indice della componente che presenta la più alta probabilità, sarà utilizzato come parola (w_t) successiva nel contesto.
   nota: l'output dei neuroni in questo livello è calcolato come di consueto ($y_i = \sum_j (peso_{hidden-output} * output_{hidden} + bias_{output_i})$). L'output y_i viene poi normalizzato, tramite la funzione di attivazione non lineare *softmax*, nell'intervallo [0,1] il quale fornisce un'interpretazione probabilistica del risultato.
   $$\hat{P}(w_t|w_{t-1}, ...w_{w-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$
   La componente del vettore in uscita con probabilità maggiore, sarà quella che rappresenterà la prossima parola del linguaggio.

4.3.2 Word2Vec [15]

Il modello considerato precedentemente non è lo stato dell'arte ma un lontano precursore di *paragraph vector*, viste le problematiche che presenta. La transizione dal livello di input a quello di proiezione non è particolarmente onerosa a livello computazionale grazie alla possibilità di limitare il contesto ad n parole (dove n corrisponde all'n-gram). Viceversa, a causa delle dimensioni dell'hidden layer, la transizione dal livello di proiezione a quello nascosto è risultata essere particolarmente inefficiente. Per cercare di risolvere tale problematica sono state proposte diverse varianti alcune delle quali sostituivano la funzione *softmax* in favore di *hierarchical softmax*.

Le problematiche sopraccitate sono state risolte da *Mikolov et al.* [15] nel 2013, mediante *Word2Vec*, un'evoluzione dell'approccio sopra visto. Esistono due varianti del modello presentato:

1. *Continuous Bag-of-Words*: la complessità computazionale è ridotta rimuovendo lo strato nascosto e condividendo l'intero projection layer (e non solo la matrice come visto in precedenza) tra tutte le parole. Il risultato di tale operazione consente di ottenere un unico strato nascosto, (projection layer) la cui funzione di attivazione è lineare (la somma pesata dell'input viene passata allo strato di output), abbattendo l'onere computazionale. Una seconda importante innovazione introdotta da CBOW risiede nell'utilizzare non solamente il contesto passato ma anche quello futuro: il nome *Continuous Bag-of-Words* vuole enfatizzare il fatto che il projection layer non è influenzato dall'ordine delle parole e che il comportamento è pertanto simile a quello di una Bag-of-Word tradizionale che agisce però su vettori reali.

2. *Continuous Skip-gram Model*: quest'architettura è simile alla precedente ma, invece di predire il termine corrente w_t basandosi sul contesto, cerca di predire le parole nell'intorno del termine corrente. Ogni parola w_t viene utilizzata come input per calcolare le probabilità dei termini che si trovano all'interno di un range predefinito rispetto a w_t. A termini più distanti dalla parola corrente w_t, viene assegnato un peso minore rispetto agli altri.Tale approccio fornisce risultati più accurati rispetto a CBOW presentando per@ una complessità computazionale maggiore.

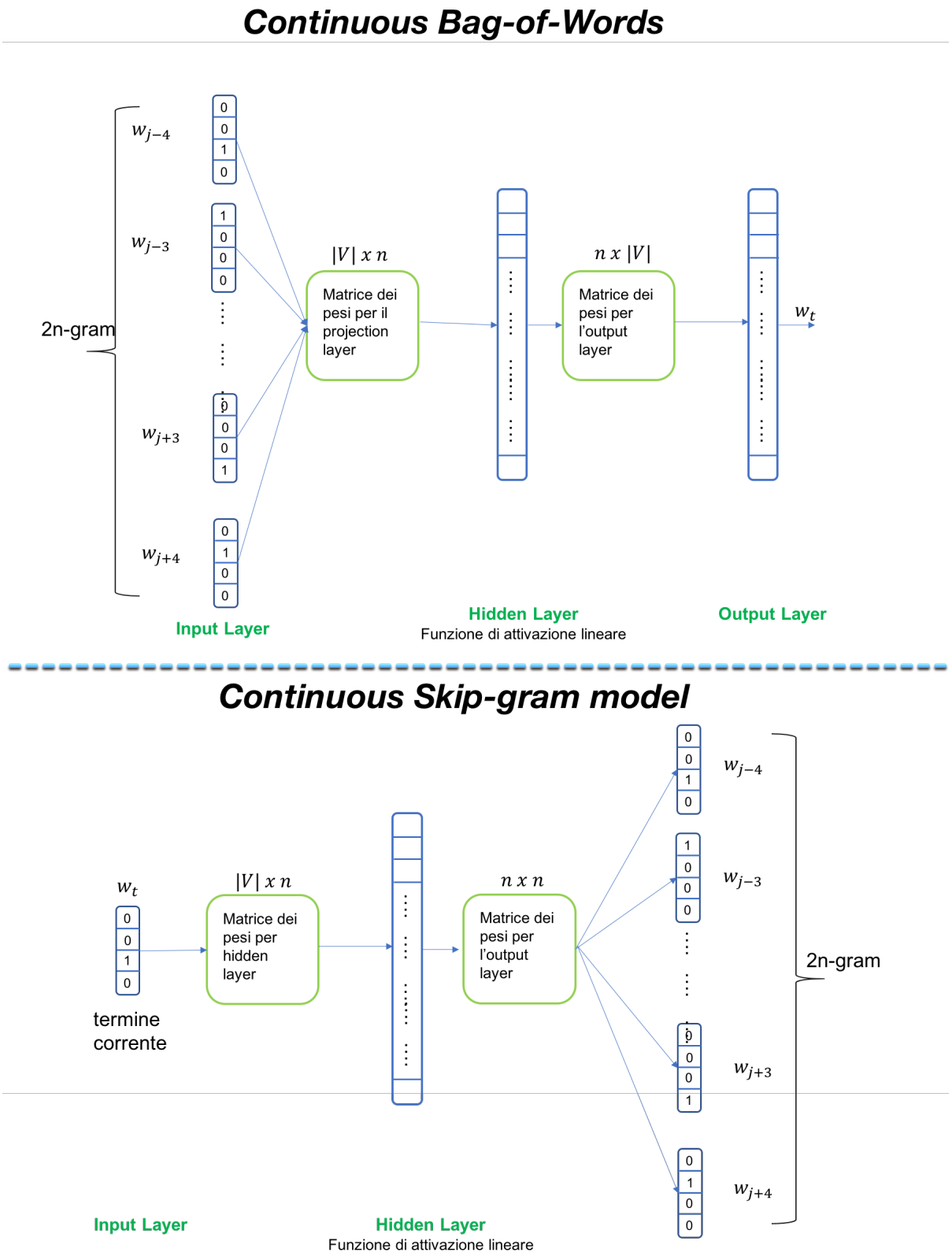


Figura 7: CBOW e CSGM a confronto

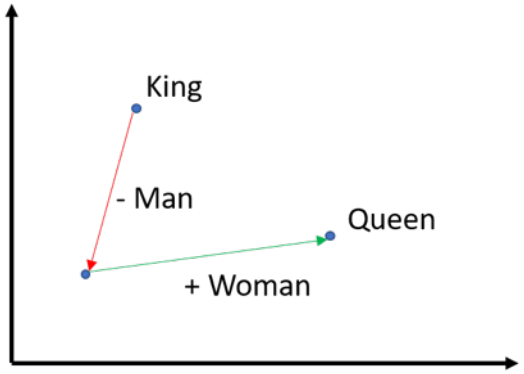


Figura 8: word2vec: rappresentazione nello spazio 2D

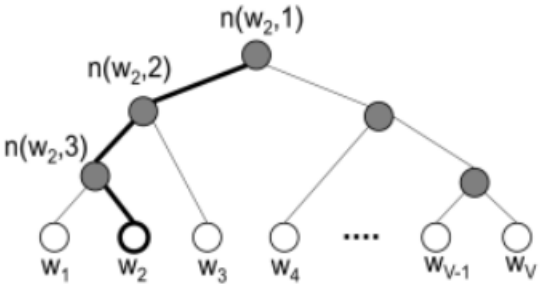


Figura 9: softmax gerarchico: i nodi bianchi rappresentano le parole mentre quelli neri sono discriminanti

La capacità del metodo appena illustrato di rappresentare un termine mediante un vettore proiettato in uno spazio n-dimensionale, consente di scoprire relazioni nascoste. Supponendo di conoscere i vettori associati ai termini *man*, *king* e *woman*, è possibile individuare la relazione che intercorre tra *king* e *woman* sottraendo il vettore *man* a *king* e sommare *woman* ottenendo il vettore *queen*.

Word2Vec: Ottimizzazioni Word2Vec è un approccio incredibilmente più efficiente rispetto al primo modello di linguaggio proposto da *Bengio et al.* [17] anche se esistono alcune possibili migliorie da applicare a quanto già visto. L'approccio utilizzato da *Mikolov et al.* sostituisce la funzione di attivazione non lineare *softmax* del livello di output, con la funzione *softmax gerachico*, la quale impiega un albero binario per rappresentare tutti i termini nel vocabolario. Le parole risiedono nelle foglie dell'albero mentre i nodi interni sono discriminanti utilizzati durante la navigazione. Esiste perciò un unico cammino, di lunghezza $\log_2 V$ che conduce dalla radice ad ogni foglia. In questo modo, invece di dover aggiornare $|V|$ vettori di termini per ogni istanza di training, è sufficiente valutare $\log_2 V$ nodi per poter calcolare la probabilità del termine corrente.

4.3.3 Paragraph Vector [11]

Riprendendo ciò che è stato illustrato nella sezione 'Reti neurali per la stock market prediction', la previsione dello stock market richiede l'introduzione di ulteriori informazioni oltre ai prezzi azionari. Come già accennato, tali dati sono tipicamente non-strutturati e derivanti da sorgenti pubbliche di informazioni quali giornali, notizie web,...
Le metodologie viste fino ad ora si limitano ad estrarre una rappresentazione semantica dei termini dal contesto, per cui risultano inadatte quando l'input al modello sono frasi. Lo stato dell'arte nella rappresentazione semantica di testi è chiamato *Paragraph Vector* ed è la naturale evoluzione di *Word2Vec*.
Esistono due varianti di *Paragraph Vector* ma quella che produce un modello più accurato, a fronte di una maggiore complessità computazionale, è chiamata *Distributed Memory of Paragraph Vector (PV-DM)*. Questo modello riprende fedelmente ciò che è già stato visto in *Word2Vec* a cui aggiunge l'utilizzo dei paragrafi per il calcolo della probabilità della parola successiva w_t nel contesto. I termini vengono quindi mappati in vettori *one-hot*, così come visto nella sezione 'Word2Vec', ma oltre a questi anche ogni singolo paragrafo (o documento) è mappato in un unico vettore *one-hot*. E' importante sottolineare che i pesi dei termini e quelli dei paragrafi sono contenuti in due matrici distinte; ciò è dovuto al fatto che la matrice C, contenente i pesi dei termini, è condivisa tra tutti i paragrafi del corpus mentre la matrice D, contenente i pesi dei paragrafi, è condivisa solamente tra tutte le frasi di ogni paragrafo (due paragrafi hanno due matrici D distinte).
I vettori distribuiti derivanti dai termini, vengono poi combinati con il vettore distribuito ottenuto dal paragrafo corrente, all'interno dell'hidden layer, mediante una semplice concatenazione. I vantaggi derivanti da *Word2Vec* nell'utilizzo di un livello nascosto, la cui funzione di attivazione è data dalla somma pesata delle attività dei neuroni nello strato di input, sono le eccellenti performance garantite rispetto all'utilizzo di una funzione di attivazione non-lineare così come accadeva nel primo modello illustrato (*Bengio et al.*).
nota: l'n-gram che rappresenta il contesto, è dato da una finestra mobile che scorre all'interno del paragrafo fornito in input insieme agli *hot-vector* dei termini.

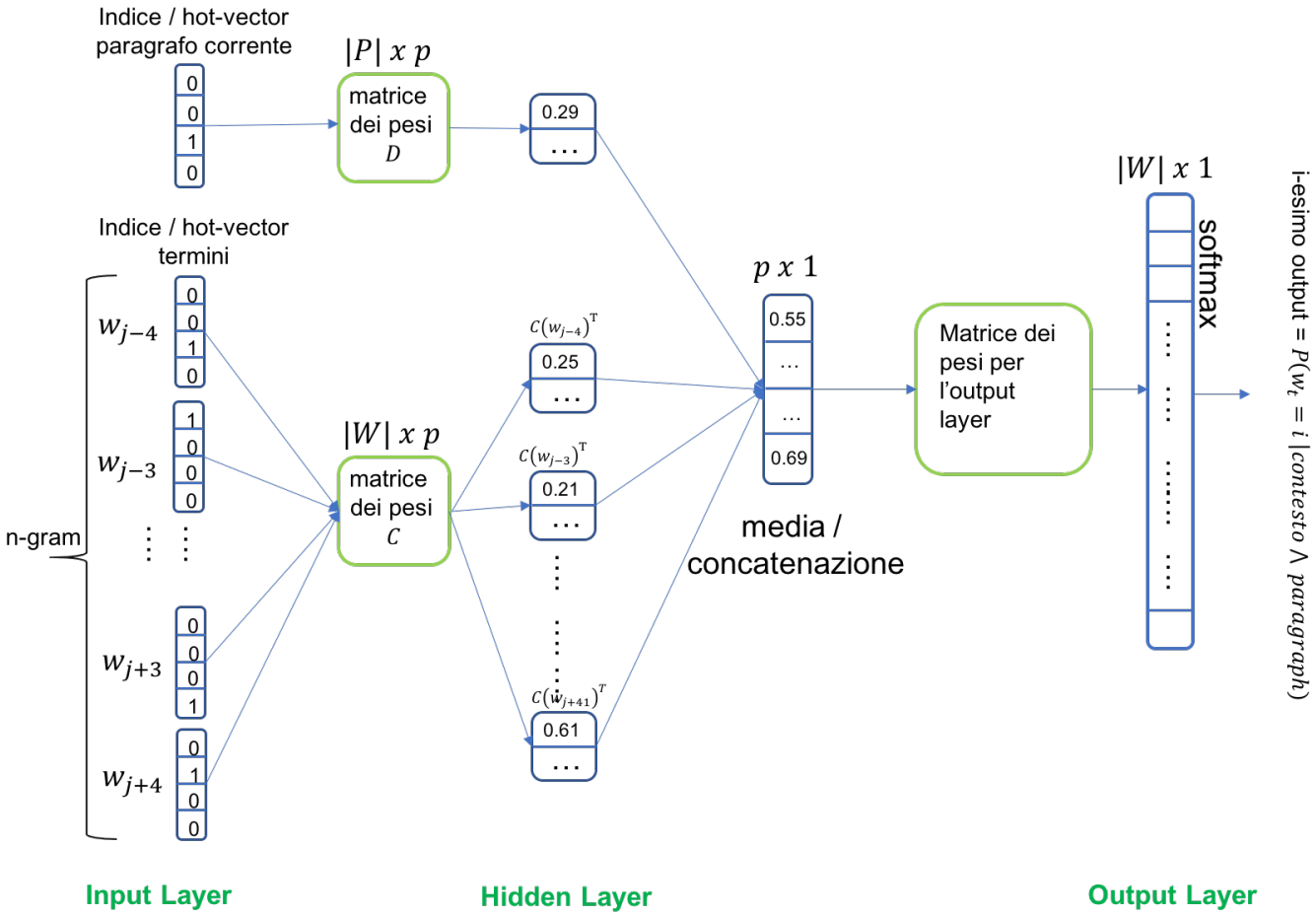


Figura 10: PV-DM

dove:

1. P: numero di paragrafi nel corpus
2. W: numero di termini nel corpus
3. p: dimensioni del livello nascosto. Il risultato del livello nascosto può essere visto come una proiezione dei termini e dei documenti in uno spazio p-dimensionale ridotto.

Il vettore di features in output dal modello può poi essere utilizzato all'interno di ulteriori algoritmi di machine learning (quali SVM, clustering K-means) come rappresentazione sintetica di paragrafi e documenti.
Riassumendo, l'algoritmo si divide in due fasi:

1. *fase di addestramento*: ad ogni step, viene selezionato il contesto (*n-gram*) all'interno del paragrafo corrente. Successivamente il contesto ed il paragrafo (codificati in vettori *one-hot*) vengono forniti in ingresso al

modello. La rete è addestrata tradizionalmente con SGD ed il gradiente, ottenuto tramite backpropagation, è utilizzato per aggiornare i parametri dell'algoritmo *softmax* e le matrici dei pesi W e D.

2. *fase di inferenza*: il modello, basandosi sui parametri calcolati durante la fase di addestramento, inferisce il vettore di features di un nuovo paragrafo mai visto prima. Ciò è possibile aggiungendo ulteriori colonne alla matrice D ed aggiornando tali colonne mediante SGD. E' importante sottolineare che tutti gli altri parametri rimangono fissi e sono quelli determinati durante l'addestramento.

Vantaggi di paragraph vector

1. *paragraph vector*, a differenza della *bag-of-words*, è in grado di estrarre la semantica dei termini

2. nel caso di PV-DM l'ordine dei termini è importante, almeno per piccoli contesti, così come avviene nel caso di modelli basati su n-gram.

3. *paragraph vector*, rispetto a *bag-of-words*, applica un'importante riduzione della dimensionalità consentendo al modello di generalizzare.

4.4 Approccio supervisionato: Stato dell'arte nella stock market prediction

Uno degli approcci alla stock market prediction che ha consentito di ottenere risultati migliori rispetto a ciò che era già presente in letteratura, è stato presentato nel 2016 da *Akita et al.* [13] e sfrutta la combinazione di dati strutturati e non strutturati. Come ampiamente discusso in precedenza, gli investitori non basano le proprie strategie esclusivamente su dati analitici quali indici o grafici ma anche sulle informazioni testuali provenienti da giornali e/o notiziari. Le aziende quotate in borsa, inoltre, hanno dipendenze dirette o indirette con competitors e partner perciò l'andamento negativo di uno di questi può generare ripercussioni sull'azienda in esame.
L'idea alla base della metodologia proposta da *Akita et al.* cerca di catturare le interrelazioni che intercorrono tra più società al fine di fornire al modello un quadro più ampio della realtà. Tale approccio combina informazioni non-strutturate e strutturate, disponibili per ogni azienda in ogni istante t, che saranno poi utilizzate per alimentare una rete neurale posta nell'ultimo stadio del modello. L'approccio presentato tenta di determinare il prezzo di chiusura delle azioni di ogni azienda sfruttando le informazioni acquisite dalla rete ad ogni istante. La possibilità di catturare dipendenze a lungo termine è garantito dall'utilizzo di una rete con memoria (tipicamente una LSTM). L'approccio può quindi essere riassunto in tre fasi:

1. nella prima fase vengono considerate separatamente informazioni testuali e non:

   (a) Informazioni non strutturate: viene costruita una rappresentazione distribuita delle informazioni non strutturate, disponibili all'istante t, mediante *paragraph vector*. Per ogni articolo riguardante una determinata azienda, viene costruito un vettore di features. Se più articoli sono presenti per una data azienda all'istante t, la semplice media aritmetica delle componenti dei vettori di features è considerata. Viceversa, se nessun articolo è disponibile, un vettore con soli 0 è utilizzato.

   (b) Informazioni strutturate: Sapendo che i valori azionari dipendono dalle dimensioni delle singole società, viene effettuata un'opera di normalizzazione al fine di ottenere prezzi nel range [-1,1]. Nominando c_n l'azienda con indice n e supponendo di essere all'istante t, la normalizzazione è eseguita come: $valore-azione-normalizzata^t_{c_n} = \frac{2 * prezzo^t_{c_n} - (max_{c_n} + min_{c_n})}{max_{c_n} - min_{c_n}}$

2. nella seconda fase i dati strutturati e non strutturati vengono combinati al fine di ottenere un unico vettore x_t con cui alimentare la rete neurale presente nell'ultimo stadio del modello. Sapendo che il vettore distribuito ottenuto delle informazioni strutturate p_t, ha un numero di dimensioni molto maggiore rispetto n_t, la combinazione non può avvenire direttamente. A tal fine è necessario ridurre la dimensionalità di p_t ed aumentare quella di n_t, mediante una seconda rete neurale. L'obiettivo è ottenere un unico vettore distribuito x_t contenente sia le informazioni strutturate che quelle non strutturate suddivise in egual misura. Nominato P_t il vettore ottenuto

da p_t a cui è stata applicata la riduzione della dimensionalità, tramite una rete neurale, e N_t il vettore ottenuto da n_t a cui è stata applicata l'aumento della dimensionalità, il vettore x_t in input alla rete può essere ottenuto come:

$$\begin{aligned} P_t &= W_p p_t + b_p \\ N_t &= W_n n_t + b_n \\ x_t &= [P_t N_t] \end{aligned} \tag{6}$$

dove:

(a) $W_p \in R^{d_x x d_p}$ è la matrice dei pesi della rete neurale utilizzata per ridurre la dimensionalità del vettore di features delle informazioni non strutturate

(b) $W_n \in R^{d_x x d_n}$ è la matrice dei pesi della rete neurale utilizzata per aumentare la dimensionalità del vettore delle informazioni strutturate

(c) b_p e b_n sono i bias rispettivamente della rete al passo *(a)* e al passo *(b)*

Sono state condotte due tipologie di esperimenti, la prima mirata ad investigare l'effettiva capacità di *Paragraph Vector* nel catturare il significato delle notizie mentre la seconda rivolta all'individuazione del classificatore in grado di fornire i migliori risultati.
L'obiettivo è descritto da:

$$r_{c_n}(t) = \frac{closing_{c_n}^{pred}(t) - opening_{c_n}^{true}(t)}{opening_{c_n}^{true}(t)}$$

$$gain_{c_n}(t) = \begin{cases} buy \rightarrow sell(r_{c_n}(t) > 0), \\ sell \rightarrow buy(r_{c_n}(t) < 0) \end{cases}$$

Il risultato degli esperimenti ha portato a confrontare il modello allenato tramite LSTM con un input basato esclusivamente sui prezzi di apertura con lo stesso modello il cui input era costituito sia dai prezzi di apertura che dalle rappresentazioni delle notizie ottenute tramite *Paragraph Vector*. Il secondo modello è stato in grado di fornire un guadagno di 1211,90 Yen rispetto a quello allenato con le sole informazioni numeriche.

5 Modello proposto per la previsione del mercato azionario

5.1 Riferimenti

All'interno del progetto sono stati utilizzati diversi riferimenti sia come spunto per la costruzione di nuovi classificatori sia come confronto in merito ai risultati ottenuti. In particolar modo alcuni spunti sono stati presi dal paper [13] (ref:4.4) per via dei buoni risultati ottenuti nonostante la classificazione non fosse orientata alla previsione dell'indice DJIA tramite tweets.
Un altro spunto importante è stato acquisito dai papers [16] e [7] per via degli approcci innovativi nel campo della sentiment analysis mediante reti neurali e *Paragraph Vector*.
I principali articoli presi come riferimento in merito ai risultati sono stati [6] e [7] per via del dominio applicativo condiviso con questo progetto.

5.2 Social Media come sorgente dati

Nell'immaginario collettivo il mercato finanziario risulta essere un ambiente asettico dove ogni scelta è basata sul raziocinio ed ogni decisione guidata da metriche analitiche. In un ecosistema di questo genere ogni trader è istruito al fine di valutare le informazioni ed i rischi il più velocemente possibile inibendo qualsiasi forma di irrazionalità e lasciando spazio solamente a dati puramente oggettivi.
La finanza comportamentale e l'economia comportamentale sono due campi di studio fortemente interconnessi che applicano la psicologia cognitiva al fine di comprendere le motivazioni che hanno spinto gli investitori ad intraprendere certe strade e come queste possano ripercuotersi sui prezzi di mercato e sull'allocazione delle risorse. Studi recenti in questi settori hanno in realtà dimostrato che qualsiasi decisione effettuata da un trader è condizionata da fattori psicologici o emotivi (intrinsecamente non predicibili) in grado di alterare le performance delle scelte, condizionando di conseguenza il valore delle azioni. Uno dei lavori più rilevanti condotto da Andrew W. Lo (professore ordinario di finanza al MIT) e Dmitry Repin (neuroscienziato e docente all'università di Boston), [3] afferma che il fattore emotivo gioca un ruolo fondamentale nella maggior parte degli investitori di Wall Street. Tale conclusione giunge al termine di uno studio che ha visto il monitoraggio di vari parametri quali la pressione del sangue, il battito cardiaco e l'attività elettrodermica di 10 traders durante una normale giornata di lavoro. Tali informazioni sono state correlate agli eventi occorsi sulle azioni manipolate dai soggetti in esame. In particolar modo sono stati analizzati tre tipi di eventi:

1. trend (positivi o negativi) costanti
2. inversione di un trend
3. rapide fluttuazioni

Lo studio ha dimostrato che sia nel caso di veterani che nel caso di investitori senza esperienza, in corrispondenza di rapide variazioni dei valori azionari, i

soggetti hanno riportato fattori emotivi tanto più evidenti quanto minore era la propria esperienza.
Altri studi hanno invece dimostrato come vi sia una certa tendenza alla razionalità in situazioni di riposo (il trader non è posto di fronte ad una scelta che comporta del rischio) che svanisce nel momento in cui un investimento "*interessante*" si prospetta di fronte ad un investitore. Ciò è dovuto al fatto che investimenti interessanti nascondono intrinsecamente una certa quantità di rischio che genera a sua volta sentimenti o emozioni.
Tutto quanto sopra riportato induce a riflettere sulla necessità di includere informazioni *emozionali* all'interno del modello previsionale. A differenza del lavoro svolto da *Akita et al.* [13], all'interno del quale venivano considerate esclusivamente informazioni derivanti da notiziari o giornali e pertanto puramente oggettive, la finanza comportamentale descrive l'andamento del mercato come soggetto a condizionamenti emotivi che perciò devono essere considerati nella previsione. L'identificazione delle emozioni e dell'umore è un task tipico della sentiment analysis, la quale si propone di caratterizzare anche la polarità di una frase o di un commento relativo ad un topic (opinion mining).
I più grandi catalizzatori di opinioni e giudizi sono senza dubbio i social media. Nel lavoro svolto, la sorgente di informazioni utilizzata è stata il social network *twitter* ed in particolar modo sono stati considerati tutti i tweet emessi durante l'anno 2008. L'individuazione dell'umore non è stata eseguita in maniera esplicita ma si è lasciato che le polarità venissero incapsulate all'interno della rappresentazione semantica dei tweets. Ciò è stato possibile grazie all'estrazione di vettori di features mediante *paragraph vector*. In questo modo ad ogni tweet corrisponde un vettore *n-dimensionale* (al quale può essere associata una coordinata spaziale) rappresentante l'esatta semantica della frase. Come conseguenza, frasi semanticamente simili e con analoga polarità, si posizioneranno in uno spazio prossimo mentre tweets con polarità differente e con diverso significato, avranno vettori completamente differenti.

5.3 Dataset e pre-processing

In accordo con quanto esposto nel paragrafo precedente, l'obiettivo di questo lavoro consiste nell'impiego delle informazioni contenute all'interno di un social network al fine di prevedere l'andamento dell'indice azionario della borsa di New York (DJIA) mediante un modello neurale che fa uso di diverse tecniche illustrate nei paragrafi 2.3,2.3, 4.
Sono stati utilizzati due dataset, contenenti tweets riferiti a due periodi temporali distinti (entrambi costituiti da *id-utente*, *data*, *tweet*):

Dataset 1 $\sim$ 10.000.000 tweets - multilingua				
Training set	01/01/2008 - 31/07/2008	$\sim$ 360.000 tweets	Buy: 77	Sell: 70
Validation set	01/08/2008 - 31/09/2008	$\sim$ 300.000 tweets	Buy: 19	Sell: 23
Test set	01/10/2008 - 20/12/2008	$\sim$ 9.400.000 tweets	Buy: 38	Sell: 28
Dataset 2 $\sim$ 200.000.000 tweets - in lingua inglese				
Training set	08/06/2009 - 30/09/2009	$\sim$ 130.000.000 tweets	Buy: 83	Sell: 59
Validation set	01/08/2009 - 31/09/2009	$\sim$ 30.000.000 tweets	Buy: 20	Sell: 10
Test set	01/10/2009 - 20/12/2009	$\sim$ 40.000.000 tweets	Buy: 22	Sell: 7

Il primo dataset è il medesimo utilizzato in [6] (utilizzato nella prima parte degli esperimenti) mentre il secondo è una versione ridotta di quello utilizzato in [10] (utilizzato in accoppiata con un modello neurale revisionato). In considerazione di quanto illustrato in 5.2, si è deciso di eliminare tutti i tweets non scritti in lingua inglese (dataset 1) e carenti di informazioni relative agli stati d'animo degli utenti che li hanno emessi. In particolar modo sono stati considerati solamente quei tweet contenenti le seguenti parole: *I am feeling, I feel, I don't feel, I'm, makes me*. Inoltre per rimanere coerenti con quanto illustrato in [6] sono stati eliminati anche tutti i riferimenti a pagine web al fine di minimizzare la probabilità di spam.
Di seguito vengono riportate le dimensioni dei training set dopo l'operazione di pulizia:

1. (dataset 1) 01/01/2008 - 31/07/2008 $\sim$ 260 tweets
2. (dataset 2) 08/06/2009 - 30/09/2009 $\sim$ 14.000.000 tweets

In considerazione del fatto che il dataset 1 conteneva tweets anche in lingua non inglese, l'applicazione dei filtri su di esso ha portato ad una riduzione di informazioni troppo consistente lasciando poche centinaia di tweets a disposizione per la classificazione. Per questo motivo il dataset 1 è stato utilizzato esclusivamente nella prima parte del progetto al fine di validare il modello su informazioni eterogenee (non solamente in lingua inglese) mentre il secondo (filtrato) è stato impiegato per il vero e proprio addestramento del classificatore. In entrambi i casi sono state eseguite le seguenti operazioni di pre-processing:

1. Sostituzione di emoticons con placeholder
2. Rimozione punteggiatura, accenti e caratteri speciali
3. Rimozione stopwords

N.B. Non sono state applicate metodologie tipiche del text-mining quali stemming, lemmatizzazione, ecc.. in quanto al fine di determinare una rappresentazione semantica del testo è stato utilizzato *paragraph vector* il quale è immune a tali operazioni.
Al fine di poter tradurre efficacemente una frase in un vettore distribuito, sono stati costruiti tre diversi modelli PV (per ogni dataset) ognuno con un numero differente di features:

1. 50 features
2. 200 features
3. 300 features: consigliato da *Mikolov et al* [11]

La figura 11 mette a confronto i tre modelli PV (addestrati rispettivamente con 50, 200 e 300 features (in 50 iterazioni)) utilizzati per ottenere una rappresentazione semantica di 1500 tweets tramite inferenza. In particolar modo l'immagine mostra le similarità coseno dei vettori distribuiti ottenuti tramite inferenza con i rispettivi vettori reali ottenuti durante l'addestramento.

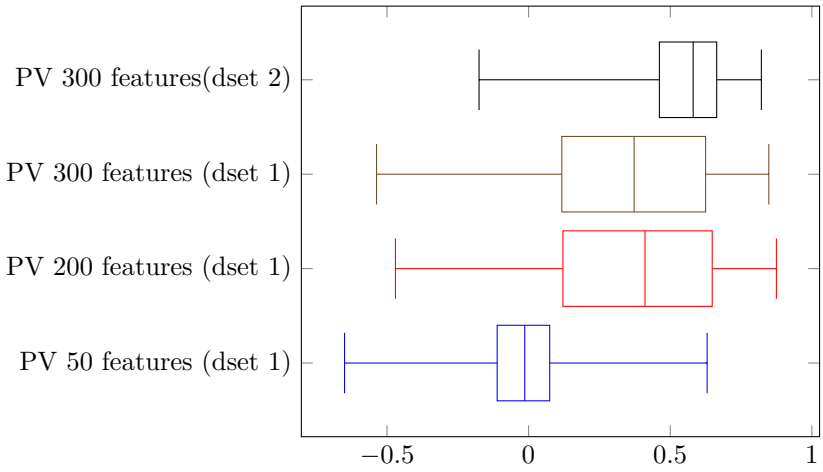


Figura 11: BoxPlot Similarità coseno dataset 1 e 2

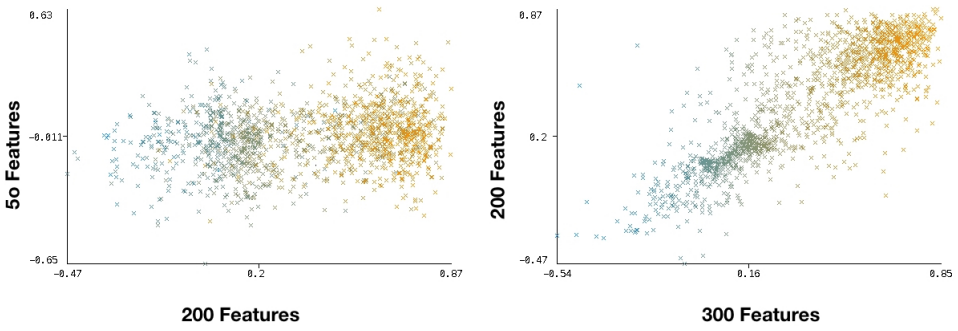


Figura 12: Correlazione fra le similarità coseno dei modelli (dataset 1)

Dalla figura 11 è possibile notare come lo scarto interquartile del modello con 50 features sia particolarmente concentrato a ridosso del valore *0*. Ciò significa che il vettore generato non presenta alcuna correlazione con il tweet originale, conducendo alla conclusione che il modello non è stato in grado di apprendere realmente la semantica della frase.
Con l'incremento del numero di features, la mediana tende a muoversi verso il valore *1* denotando una migliore predisposizione del modello all'apprendimento. E' utile sottolineare inoltre come, nonostante il paper originale *Mikolov et al* [11] indicasse come 300 il numero ideale di features, il modello a 200 features presenti performance leggermente superiori rispetto a quello con 300 (nel caso del dataset 1). Ciò è probabilmente dovuto alle limitate dimensioni delle frasi (140 caratteri) ed all'elevato rumore presente all'interno dei dati.
In ultima analisi, il ridotto scarto interquartile del modello a 300 features relativo al dataset 2, conferma indirettamente la migliore qualità dei dati del secondo database rispetto al primo (il quale è multi-lingua). Le conclusioni sopra riportate sono parzialmente confermate dall'immagine 12 dove è possibile notare una buona correlazione tra i risultati ottenuti dai modelli con 200 e 300 features ed

una totale assenza della stessa tra quelli con 50 e 200 features.
Al fine di determinare il miglior modello tra quelli selezionati, ogni vettore distribuito, ottenuto tramite inferenza, è stato utilizzato come una query in un approccio di information retrival rank-based selezionando solamente il primo documento nella lista. Ogni documento inferito risultante più simile a sé stesso che agli altri, è stato classificato come *TP* (true positive) mentre tutti gli altri sono stati marcati come *FN* (false negative). Sono quindi state costruite le *confusion matrix* tramite le quali calcolare le $F_1 measure$:

200 features

		True labels	
		Positivo	Negativo
Predicted labels	Positivo	1008	0
	Negativo	472	0

300 features

		True labels	
		Positivo	Negativo
Predicted labels	Positivo	1046	0
	Negativo	444	0

300 features

		True labels	
		Positivo	Negativo
Predicted labels	Positivo	0	0
	Negativo	0	0

300 features

		True labels	
		Positivo	Negativo
Predicted labels	Positivo	1426	0
	Negativo	74	0

$$F_1 measure = \frac{2 * (Precision * Recall)}{Precision + Recall} =$$
$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} =$$
$$Recall = \frac{TP}{TP + FN} \rightarrow$$

$$Recall_{200 features} = 0,672$$
$$Recall_{300 features} = 0,697 \quad \text{(dataset 1)}$$
$$Recall_{300 features} = 0,95 \quad \text{(dataset 2)}$$

Vista la sola presenza di documenti *TP* o *FN*, il calcolo della $F_1 measure$ equivale al calcolo dell'accuratezza che a sua volta equivale alla *recall*. E' possibile notare come il modello a 300 features abbia recuperato un maggior numero di documenti validi rispetto a quello da 200 features ed è per tale motivo che è stato utilizzato per entrambi i dataset, così come consigliato da *Mikolov et al* [11].

5.4 Tecnologie e metodologie impiegate

L'implementazione dei vari modelli testati è basata esclusivamente su *Lasagne*, una tecnologia che agisce in accoppiata con *theano*, una libreria per la valutazione di espressioni matematiche simboliche, che abilita lo sviluppo di reti neurali artificiali in python. Il principale costo computazione nell'addestramento di modelli neurali, tanto maggiore quanto più profonda è la rete, risiede nella retropropagazione dell'errore. Tale fattore limitante è attenuato dalla capacità di valutare espressioni matematiche, tramite *theano*, direttamente sulle GPGPU, le quali consentono di abbattere di diversi fattori il tempo di addestramento. In questo progetto specifico è stata utilizzata una GPU Nvidia 1070.
La costruzione dei modelli di *Paragraph Vector* è basata su *gensim*, una libreria originariamente sviluppata per la consultazione di variabili latenti (LSA, LDA, PCA,...) al fine di operare LSI su dati non strutturati. Nel corso del tempo, sono state rese disponibili metodologie basate anche se reti neurali quali *Word2Vec* e *Doc2Vec*.
Tra tutte le reti presentate nelle sezioni 1, 2.3, 2.3, 3, 4.4 sono state impiegate tutte tranne le *Neural-Turing Machine*. Questa scelta è dovuta alla necessità di individuare potenziali correlazioni esistenti nel tempo che solamente una rete con memoria è in grado di identificare. Sono inoltre stati impiegati *Word2Vec* (4.3.2) e *Paragraph Vector* (4.3.3) al fine di catturare rappresentazioni semantiche di dati non strutturati.
In ultimo luogo, è stato seguito un approccio similare a quanto descritto in [13] con alcune variazioni dovute al differente dominio di riferimento.

5.5 Performance 'di base'

In prima analisi, per poter comprendere appieno le performance nella previsione delle serie storiche, sono stati eseguiti diversi esperimenti su un classificatore il cui unico input era costituito dai prezzi di apertura dei giorni precedenti. Il modello utilizzato era composto da due reti neurali GRU poste in cascata dove il numero di neuroni della seconda era esattamente la metà di quelli della prima:

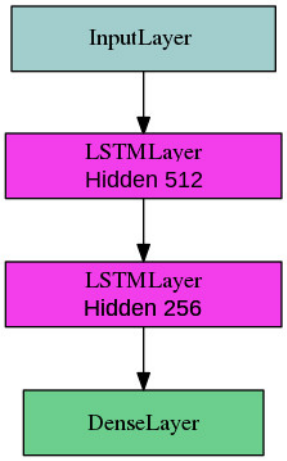


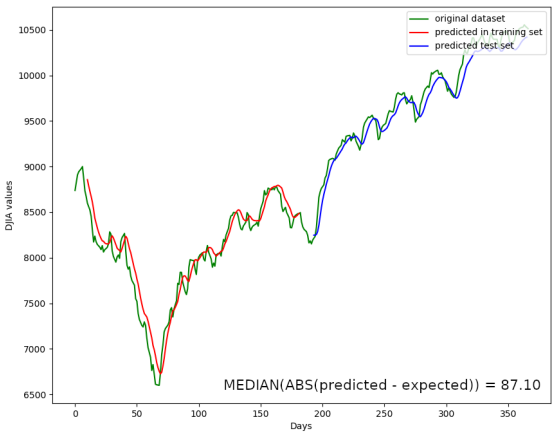
Figura 13: GRU con i soli prezzi di apertura in input

Il modello è stato testato sullo storico dei prezzi dell'anno 2009 relativo all'indice DJIA. Nei giorni in cui tali valori non risultavano disponibili a causa delle festività, è stata applicata ricorsivamente una funzione convessa dove i nuovi prezzi sono stati ottenuti come $\frac{x_{prev} + x_{last}}{2}$ dove x_{prev} è l'ultimo valore disponibile mentre x_{last} è il primo prezzo disponibile dopo l'intervallo dei valori mancanti.
Il dataset è stato suddiviso in due parti identiche dove la prima è stata utilizzata per l'addestramento mentre la seconda per la convalida. Gli esperimenti hanno riguardato l'individuazione della configurazione in grado di approssimare al meglio la distribuzione dei prezzi di chiusura dati quelli di apertura dei k giorni precedenti.

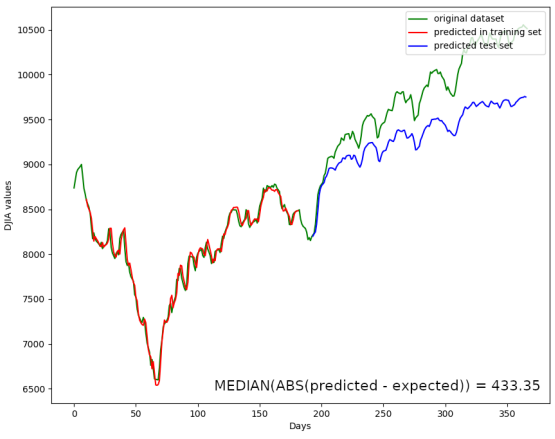
Accuratezza in testing									
	Adagrad			**Momentum**			**Rmsprop**		
	50 epoche	**100 epoche**	**200 epoche**	**50 epoche**	**100 epoche**	**200 epoche**	**50 epoche**	**100 epoche**	**200 epoche**
2 giorni	s: 39% m: 41% l: 84%	s: 40% m: 81% l: 80%	s:64% m: 79% l: 47%	s: 69% m: 78% l: 87%	s: 71% m: 74% l: 70%	s: 72% m: 80% l: 83%	s: 72% m: 71% l: 68%	s: 68% m: 61% l: 79%	s: 70% m: 69% l: 70%
5 giorni	s: 52% m: 74% l: 69%	s: 79% m: 59% l: 80%	s: 66% m: 71% l: 79%	s: 68% m: 68% l: 71%	s: 68% m: 74% l: 74%	s: 71% m: 78% l: 83%	s: 55% m: 73% l: 67%	s: 51% m: 70% l: 68%	s: 68% m: 68% l: 66%
10 giorni	s: 68% m: 70% l: 80%	s: 68% m: 78% l: 86%	s: 65% m: 54% l: 45%	s: 69% m: 71% l: 80%	s: 73% m: 73% l: 79%	s: 71% m: 74% l: 75%	s: 68% m: 68% l: 68%	s: 64% m: 68% l: 79%	s: 66% m: 68% l: 64%
s: 32 neuroni nel livello 1, 16 nel livello 2 m: 128 neuroni nel livello 1, 64 nel livello 2 l: 512 neuroni nel livello 1, 256 nel livello 2									

Tabella 1: Risultati della previsione con il modello 5.5

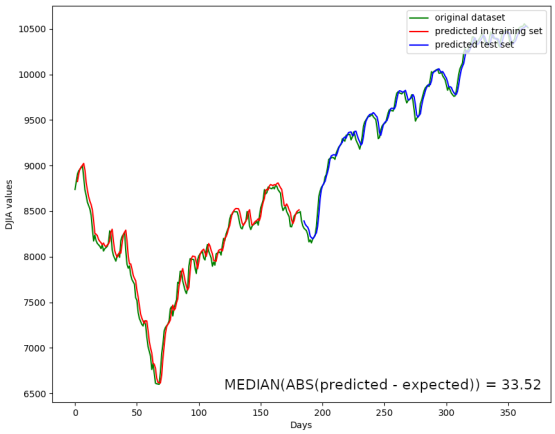
Dalla tabella 1 risulta piuttosto evidente che i migliori risultati sono stati ottenuti con reti composte da una quantità relativamente elevata di neuroni indipendentemente dal numero di giorni aggregati della storia passata. In realtà osservando le figure 15, le quali riportano i risultati ottenuti in regressione, è possibile notare come la varianza tra il valore atteso e quello predetto è tanto più grande quanto maggiore è il contesto passato considerato.



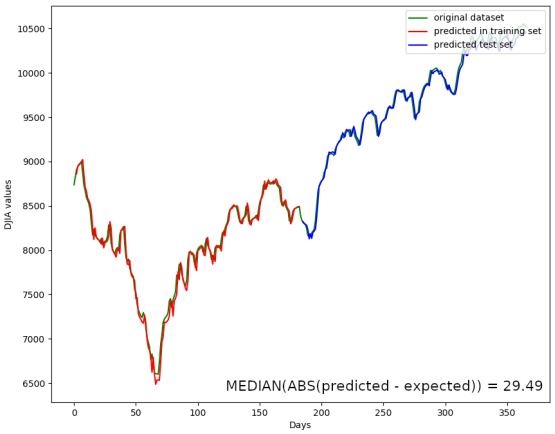
(a) 10 giorni, large, 50 epoche, momentum



(b) 10 giorni, large, 100 epoche, rmsprop



(c) 12 giorni, large, 50 epoche, momentum



(d) 2 giorni, large, 100 epoche, rmsprop

Figura 15: Predizione prezzi chiusura in regressione

La linea verde rappresenta il prezzo di chiusura atteso, quella rossa il prezzo di chiusura predetto in fase di training mentre quella blu rappresenta il prezzo di chiusura predetto in fase di testing.

La variazione tra i prezzi attesi e quelli ottenuti, in fase di testing, è stata calcolata come $MEDIAN(ABS(predicted - exptected)))$. Come era facilmente prevedibile, la varianza cresce al crescere del contesto passato considerato. Utilizzando una storia passata limitata, la variazione di prezzo che intercorre tra un giorno e l'altro è anch'essa molto limitata mentre aumenta all'aumentare del contesto considerato. In figura 15 è possibile notare come utilizzando una piccola finestra temporale (2 giorni), l'errore commesso in fase di testing tenda a circa 30$ mentre nel caso di una finestra maggiore (10 giorni) l'errore cresca notevolmente.
Ciò suggerirebbe di utilizzare i prezzi di apertura, ed eventualmente di chiusur,) di pochi giorni passati al fine di prevedere quelli di chiusura dei giorni a venire. Questo comportamento, però, consente esclusivamente di approssimare i valori di uscita dei giorni futuri impedendo l'individuazione di rapide e consistenti variazioni. Inoltre, il prezzo di chiusura di un'azione potrebbe dipendere non solo dalle informazioni introdotte a ridosso del giorno da prevedere ma anche da informazioni emesse diversi giorni prima. Per tale motivo, l'obiettivo è quello di costruire un modello in grado di ottenere performance similari a quelle in figura 15c e 15d ma considerando un contesto passato superiore.

5.6 Modello

L'idea alla base dell'approccio utilizzato, illustrato anche in [6], consiste nell'addestrare un classificatore affinché preveda la classe di appartenenza dell'indice DJIA del giorno t date le informazioni dei giorni $[t-1, t-k]$. Si tratta quindi di risolvere è quindi un problema di *classificazione multi-classe* con le seguenti classi:

1. acquisto: se *prezzo di apertura - prezzo di chiusura* > 0
2. vendita: se *prezzo di apertura - prezzo di chiusura* < 0

Per via della maggiore facilità di apprendimento, è stata preferita la modellizzazione del problema tramite classificazione piuttosto che per regressione. La figura 16 descrive sommariamente la metodologia adottata. La figura 17 è la

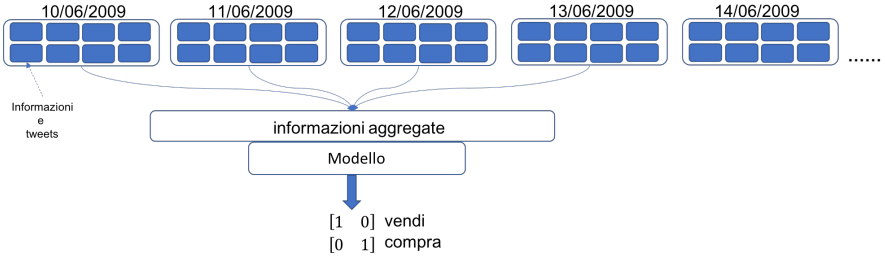


Figura 16: Rappresentazione dell'approccio utilizzato

trasposizione dell'immagine 16 ad un livello di dettaglio più elevato. Il primo approccio al problema ha portato a sviluppare un modello in grado di sfruttare la rappresentazione semantica dei tweet in congiunzione con il prezzo di apertura del giorno da prevedere.

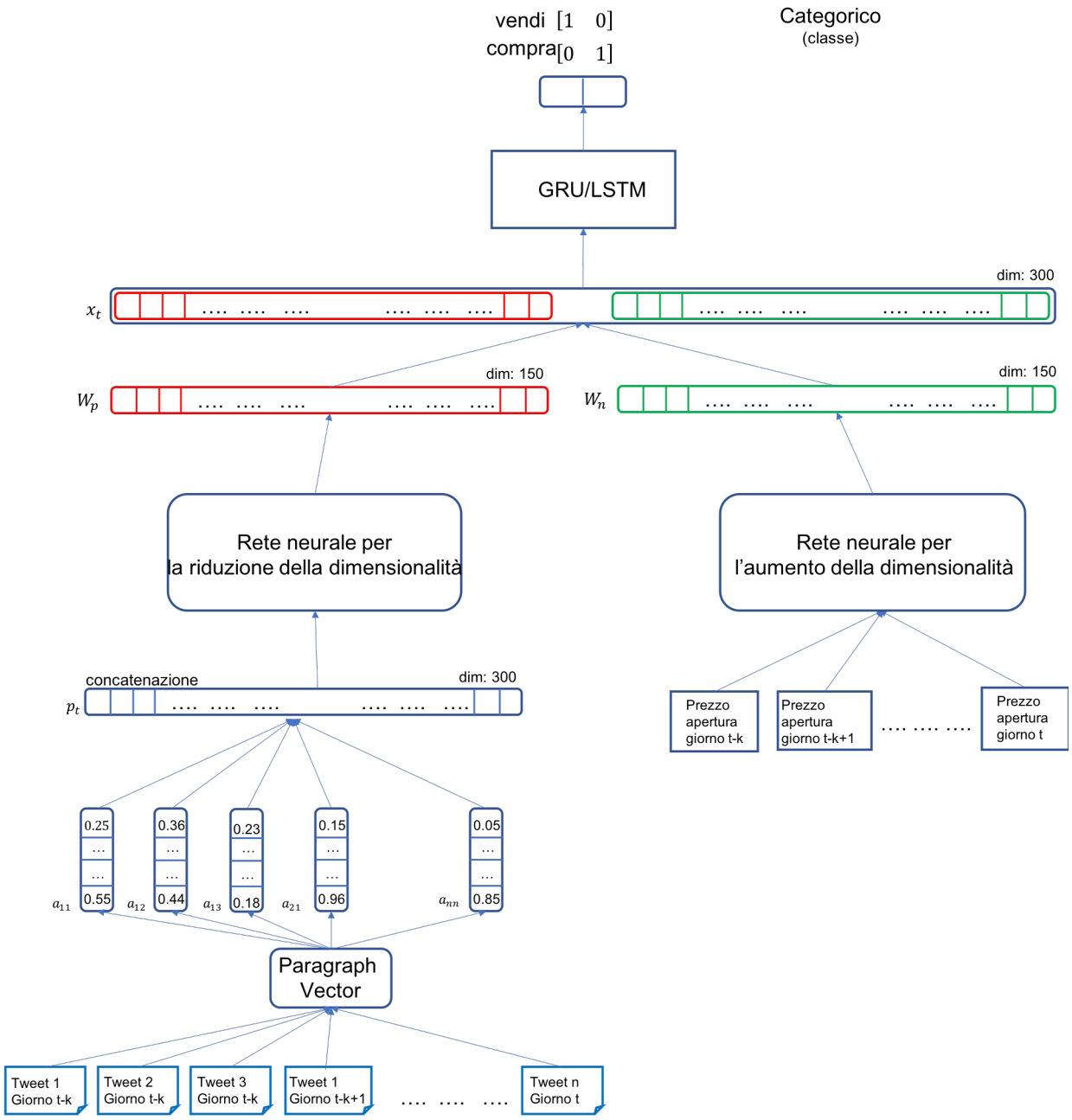


Figura 17: Modello utilizzato per la previsione del mercato azionario

Il modello in figura 17 sfrutta due reti neurali FF parallele al fine di ridurre/aumentare la dimensionalità dell'input adottata anche in [13], in modo da eliminare lo squilibrio presente tra il numero di features dei vettori distribuiti ed i prezzi di apertura (stock). Le informazioni normalizzate sono quindi utilizzate per addestrare il modello finale costituito da una rete GRU. Si è deciso di utilizzare una rete GRU piuttosto che una LSTM, o più in generale qualsiasi altra rete ricorrente, per via delle migliori performance di apprendimento. L'idea alla base delle reti GRU è comune a quella delle LSTM dove il problema dell'esplosione/svanimento del gradiente è risolta per mezzo di gates. La differenza sostanziale tra le due reti consiste nell'assenza della *memory unit* nelle GRU,utilizzata per il controllo del flusso di informazioni con una conseguente riduzione del numero di parametri e migliori performance in fase di allenamento. Tuttavia le reti GRU hanno tipicamente una minore capacità di apprendere sequenze di grandi dimensioni laddove le LSTM riescono a considerare un contesto di maggiori dimensioni.
Con l'obiettivo di superare tale limitazione, è stata utilizzata una rete bidirezionale composta da due GRU ognuna delle quali svolge una funzione differente:

1. forward GRU: fornito un input, il contesto "passato" tenta di prevedere il contesto "futuro"

2. backward GRU: agisce esattamente in maniera opposta rispetto alla rete forward, ovvero dato il contesto "futuro" tenta di prevedere "passato"

Ciò è possibile in quanto non esiste interazione diretta tra le due reti ma gli strati nascosti di entrambi i modelli sono collegati insieme in output. In questo modo vi è un incremento delle informazioni a disposizione coadiuvando la rete nella comprensione dell'intero contesto.
L'addestramento è stato eseguito tramite mini-batches dove ogni batch è nella forma:

$$(dimensioni\_batches, numero\_sequenze, numero\_features)$$

dove:

1. $dimensioni\_batches$: rappresenta il numero di campioni contenuti in un batch.

2. $numero\_sequenze$: rappresenta le dimensioni del vettore x_t, ovvero l'insieme concatenato degli input di n giorni.

3. $numero\_features$: rappresenta il numero di features (300) di ogni esempio di training in input alla rete.

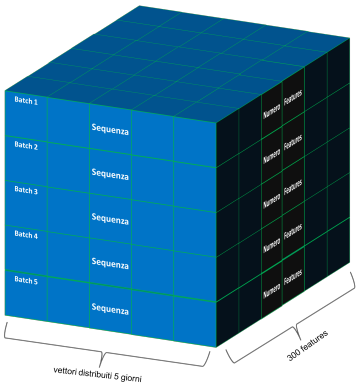


Figura 18: struttura mini-batches

Per ridurre il rischio di overfitting, è stato adottato un approccio basato su cross-validation dove l'insieme dei dati disponibili è stato suddiviso in sottoinsiemi i quali sono poi stati forniti in input al modello. L'impiego di questa tecnica ha permesso di superare la limitazione dovuta alla limitata disponibilità di dati e di labels (pochi giorni da prevedere), attraverso l'addestramento della rete su sottoinsiemi di tweets di ogni giorno, simulando in questo modo l'esistenza di un numero maggiore di labels rispetto a quelli effettivamente utilizzabili. Inoltre a ciò sono stati introdotti alcuni livelli di *dropout*, i quali disattivano alcuni neuroni limitando l'espressività della rete e riducendo di conseguenza i gradi di libertà della rete stessa. Le probabilità di dropout selezionate (iper-parametri, probabilità di settare l'uscita di un neurone a 0) sono state rispettivamente 0.3 e 0.5 nei livelli di più interni ed esterni. Tali valori, relativamente alti, sono dovuti alla necessità di limitare l'influenza del rumore presente nel dataset sulla classificazione finale. La figura 19 illustra il modello dettagliato utilizzato.

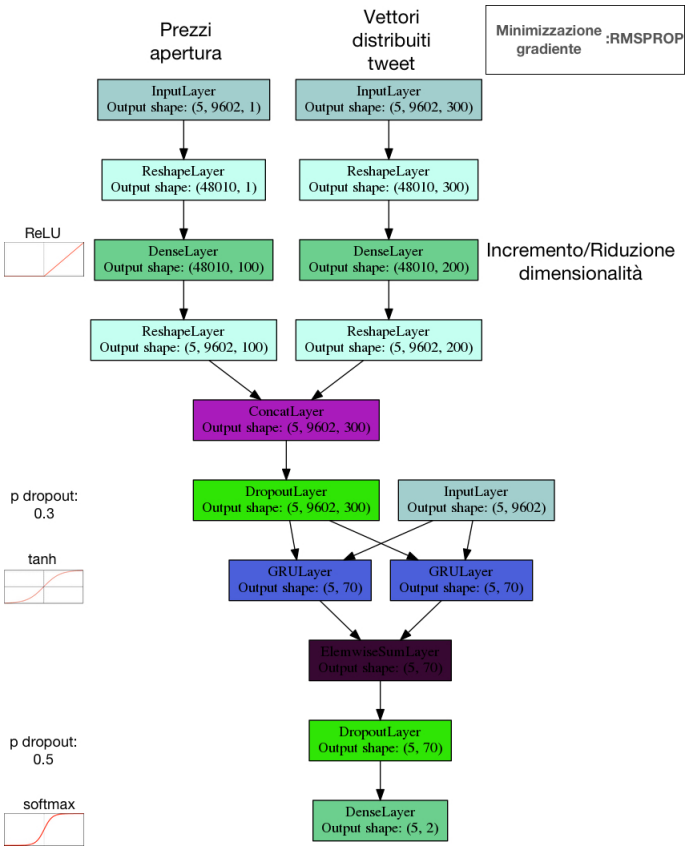


Figura 19: dettaglio del modello utilizzato

N.B. La lunghezza della sequenza (*9602*) è un vincolo tecnologico derivante dalla necessità di fornire alla rete, input di lunghezza fissa. Ciò ha comportato la sottomissione della più lunga sequenza di vettori di features (individuata tra tutte le sequenza di 5 giorni nel training-set) insieme ad una maschera indicante gli input validi per ogni esempio di training.
La funzione di attivazione delle due reti feedforward, utilizzate per l'incremento/riduzione della dimensionalità, è la funzione a rampa (ReLU) e la sua scelta è dovuta alla volontà di ottenere in output dalle due reti delle features pesate secondo la loro importanza. Dato che la topologia del modello vede la combinazione delle features ottenute dai tweets e quelle ottenute dai prezzi di apertura in modo tale che entrambe abbiano dimensioni equivalenti, le due reti avranno il compito di pesare tali features a seconda della rilevanza che queste ricoprono nell'esecuzione della previsione finale.
I livelli *Dropout* rappresentano l'omologa tecnica illustrata in [9] finalizzata a ridurre/prevenire overfitting nel qual caso la quantità di dati a disposizione sia limitata o il numero di epoche di allenamento sia molto elevato.

L'ultimo strato della rete è costituito da 2 neuroni aventi come funzione di attivazione la funzione non-lineare *softmax* ($\mathbb{R}^k \to [0,1]^k$)(7) la quale consente di comprimere un vettore k-dimensionale di valori reali in un vettore k-dimensionale di valori compresi nell'intervallo *[0, 1]*. L'output è pertanto interpretabile come

Tabella 2: Risultati ottenuti dopo 100 epoche di allenamento ($\sim$ 20.000 giorni) per ogni contesto considerato

numero di test: 10		
3 giorni	**4 giorni**	**5 giorni**
51,10%	54,40%	55,20%
52,70%	52,78%	51,00%
42,20%	51,00%	58,30%
46,10%	62,20%	50,00%
48,40%	51,70%	51,20%
48,00%	51,10%	56,70%
53,90%	60,50%	56,10%
51,10%	55,50%	50,50%
45,60%	58,90%	55,00%
48,90%	56,70%	53,30%

una distribuzione di probabilità dove ogni neurone rappresenta la probabilità dell'i-esimo evento.
L'obiettivo della rete è minimizzare l'errore derivante dalla funzione di perdita *cross entropia* (8) la quale viene utilizzata per quantificare la distanza tra una distribuzione (discreta) reale ed una predetta (output della rete).

$$\sigma(q)_j = \frac{e^{q_j}}{\sum_{k=1}^{K} e^{q_k}} \; con \; j = 1, \ldots, K \tag{7}$$

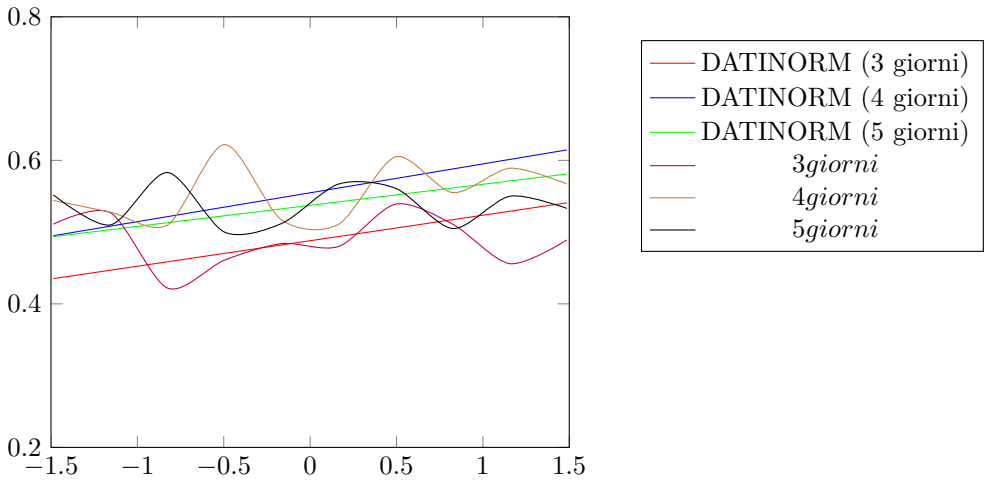
$$H(q,p) = -\sum_{x} p(x) log \; q(x) \tag{8}$$

Dove q è la distribuzione predetta dalla rete mentre p è la distribuzione reale codificata in un vettore one-hot. Il modello cercherà quindi di minimizzare la differenza tra le due distribuzioni.

5.7 Risultati e discussioni

L'addestramento del modello è stato eseguito (sul dataset 1) più volte considerando contesti differenti al fine di individuare l'esistenza di correlazioni tra le dimensioni del contesto ed i risultati ottenuti. Tali risultati sono poi stati confrontati con quelli ottenuti all'interno del paper [6]. La tabella 2 illustra l'accuratezza ottenuta in fase di testing su 10 esecuzioni differenti.

Successivamente, è stato eseguito un test di differenze fra medie al fine di determinare quale contesto fornisca performance migliori. In primo luogo i risultati sono stati confrontati con le distribuzioni normali degli stessi:



dove *DATINORM* rappresentano le distribuzioni dei dati se i risultati fossero normali. Il grafico consente quindi di stabilire se la distribuzione dei dati approssimi quella normale abilitando l'esecuzione di un test parametrico oppure se sia necessario condurre test non parametrici. Osservando che le tre linee, rappresentanti i risultati, seguono l'andamento delle distribuzioni normali è possibile applicare il ***t-test*** (test parametrico):

contesti 3 - 4 giorni

	3 giorni	4 giorni
Media osservazioni	0.488	0.55478
Varianza osservazioni	0.001263333	0.0016171
Media ipotizzata (ipotesi nulla)	0	
α	5%	
Osservazioni	10	
Gradi libertà	9	
t	4.244914925	
t critico (una coda)	1.833112933	
t critico (due code)	2.262157163	

contesti 4 - 5 giorni

	4 giorni	5 giorni
Media osservazioni	0.55478	0.5373
Varianza osservazioni	0.001617151	0.0008609
Media ipotizzata (ipotesi nulla)	0	
α	5%	
Osservazioni	10	
Gradi libertà	9	
t	0.994360695	
t critico (una coda)	1.833112933	
t critico (due code)	2.262157163	

Tabella 3: Differenza fra medie

Dalla tabella 3 si evince come il modello che considera 4 giorni come contesto sia decisamente migliore, con confidenza al 95%, in quanto il valore di t è molto maggiore rispetto al t *critico*, sia che il test sia eseguito su una che su due code. E' pertanto possibile rigettare l'ipotesi nulla, concludendo che c'è una probabilità minore del 5% che la differenza di performance tra il modello con 4 giorni di contesto e quello con 3 giorni sia dovuta al caso.
La stessa conclusione non è affermabile nel caso dei contesti con 4 e 5 giorni. In questo caso il valore t è minore del t *critico*, per cui non è possibile determinare con certezza quale dei due modelli sia più performante.
I risultati sono poi stati confrontati con quelli ottenuti da due papers che hanno utilizzato lo stesso dataset:

Tabella 4: Confronto risultati ottenuti

accuratezza in validazione			
4 giorni	**5 giorni**	**Paper [6]**	**Paper [7]** (decision-tree)
55,5%	54,40%	86,7%	88,9%

Nel paper [7] viene utilizzato un training set particolarmente ridotto al fine di poter validare con maggiore confidenza il risultato finale. In questo paper viene proposto un modello basato sul text-mining e data-mining tradizionale (Bag-of-words) per l'individuazione e rimozione del rumore all'interno del dataset. In questo progetto è stato utilizzato volutamente lo stesso training set ridotto

pur sapendo che una rete neurale ha in genere bisogno di un'elevata quantità di labels da predire. Per ovviare a tale problema è stata eseguita una campionatura casuale delle informazioni disponibili in ogni giorno al fine di aumentare la quantità di dati a disposizione.
I risultati illustrati nella tabella 4 sono verosimilmente dovuti alla quantità di informazioni rumorose ed irrilevanti per la previsione.

5.8 Incremento espressività classificatore

Le performance non molto positive, specialmente se confrontate con quelle ottenute in 5.5, ottenute sul dataset composto da elevato rumore hanno portato ad indagare sulle motivazioni che non hanno consentito al classificatore di performare adeguatamente. In particolar modo ci si è soffermati maggiormente sul contesto a 3 giorni in quanto è quello che ha fornito le performance peggiori nonostante la limitata finestra temporale.

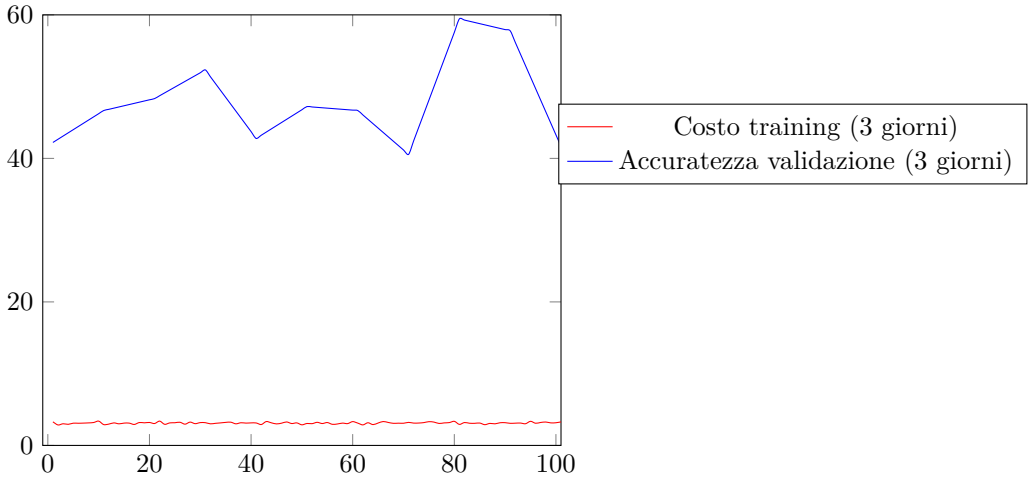


Figura 20: Accuratezza/Errore modello 3 giorni

La figura 20 illustra chiaramente come l'accuratezza tenda a fluttuare a fronte di un errore sostanzialmente invariato. Tale comportamento può essere dovuto in particolar modo a due fattori:

1. presenza di artefatti (quali termini con molte lettere ripetute, presenza di rumore, tweet scorrelati rispetto al topic finanziario,...) e limitata quantità di informazioni disponibili.

2. underfitting (elevato bias), che porta il modello a non apprendere neppure in fase di training.

In considerazione della qualità dei dataset illustrata in 5.3, la problematica 1 è stata risolta utilizzando una seconda sorgente di informazione, la quale è risultata essere decisamente migliore rispetto a quella utilizzata nel paragrafo precedente. Vista la migliore capacità del modello *Paragraph vector* di catturare la semantica dei nuovi dati, è stata eseguita un'operazione di visualizzazione del training set al fine di individuare informazioni aggiuntive utili all'addestramento.

La proiezione del dataset costituito da $\sim$ 14.000.000 di twees, ognuno rappresentato da 300 features, in uno spazio a due dimensioni, è stato possibile mediante una prima riduzione della dimensionalità tramite PCA. Essendo quest'ultimo metodo basato su *SVD* (Singular value decomposition), le dimensioni maggiormente significative sono quelle ottenute in corrispondenza degli autovalori che catturano la maggiore variazione. Tali informazioni sono poi state fornite in input ad un secondo metodo di riduzione della dimensionalità, *t-SNE*. Il risultato, illustrato in figura 21, non evidenza cluster particolarmente pronunciati anche se potrebbe essere possibile identificare funzioni di kernel in grado di enfatizzare l'esistenza di agglomerati.

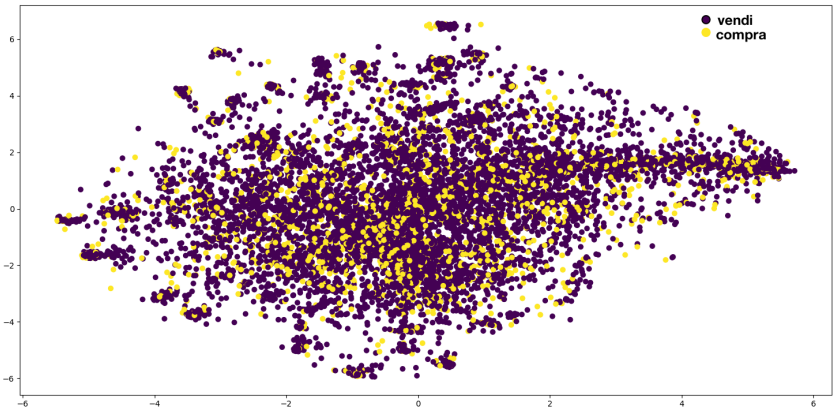


Figura 21: Visualizzazione del training set (dataset 2). Ogni punto nel grafico rappresenta un tweet

A riprova di quanto appena affermato, l'addestramento dello stesso modello, spogliato dei livelli di regolarizzazione (*dropout*), su un sottoinsieme particolarmente (20 giorni) ridotto del dataset 2, mostra l'elevata velocità con la quale l'overfitting viene raggiunto. Dopo sole 30 epoche, in corrispondenza di un errore che tende asintoticamente a 0, il modello entra in fase di sovra-allenamento con conseguente crollo dell'accuratezza in validazione. Questo comportamento è ovviamente dovuto al limitato training set fornito in input ma dimostra che il modello, con il corretto numero di parametri in funzione delle dimensioni dell'input, è in grado di fornire risultati incoraggianti. La seconda problematica è stata affrontata incrementando i gradi di libertà della rete includendo ulteriori caratteristiche ed aumentando di conseguenza l'espressività del modello.

Una prima revisione ha condotto alla somministrazione alla rete di tutta la serie storica dei prezzi di apertura e chiusura (normalizzati tramite *z-score*) a partire dal primo giorno disponibile nel dataset fino a quello target da prevedere. In questo modo l'input non è limitato alle sole informazioni nel batch corrente ma comprende l'andamento dei prezzi nell'intera storia passata; inoltre l'eventuale esistenza di patterns in specifiche configurazioni, composte dal prezzo di apertura/chiusura ed i tweets, potrebbe essere individuata dal modello. Si è inoltre deciso di fornire maggiore rilevanza alle rappresentazioni dei tweets, rispetto a quelle dei prezzi, per via dell'elevata differenza dimensionale tra le due informazioni.

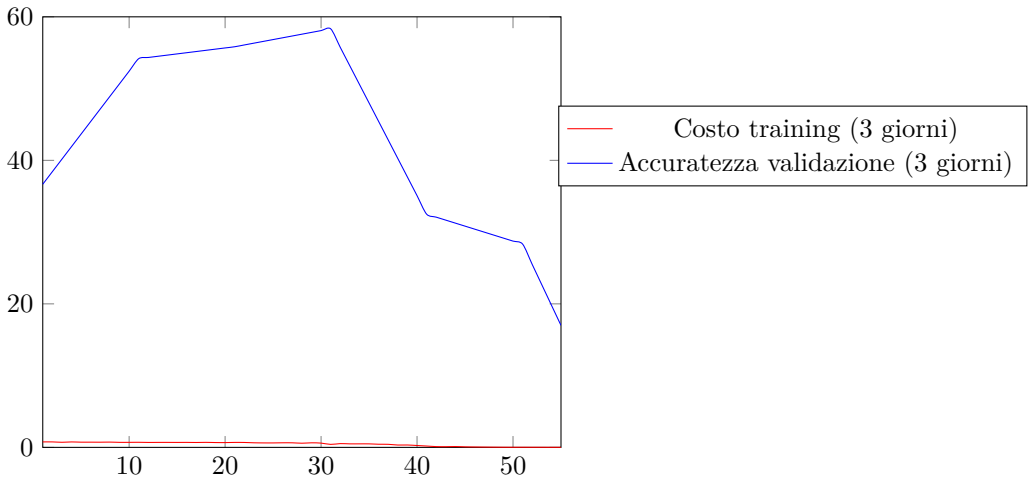


Figura 22: Accuratezza/Errore modello 3 giorni su sottoinsieme del dataset 2

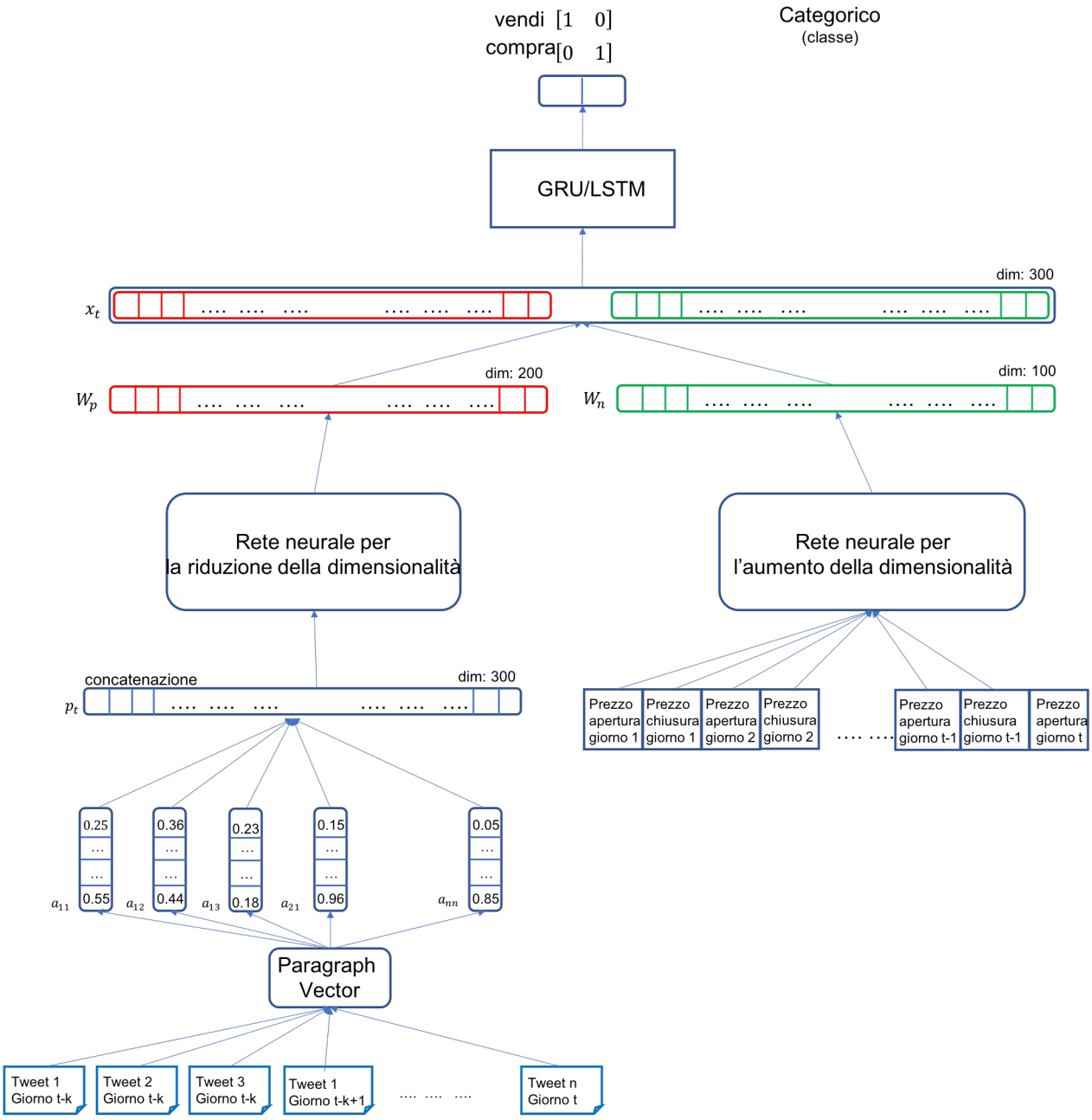


Figura 23: Prima revisione del modello

Tutti i classificatori illustrati fino ad ora cercavano di apprendere affidandosi esclusivamente alle rappresentazioni dei tweets ottenute tramite *paragraph vector* le quali cercano di descrivere, in maniera più o meno attinente con la realtà, il significato intrinseco di un dato non strutturato correlandolo con il contesto. In considerazione di quanto descritto in 5.2, nei modelli utilizzati fino ad ora vi è un'evidente carenza informativa che ha portato ad escludere informazioni

potenzialmente rilevanti quali lo stato d'animo dell'utente che ha emesso un tweets. Si è quindi deciso di includere all'interno del patrimonio informativo della rete anche tali informazioni derivandole tramite alcune tecniche di sentiment analysis.
Una delle principali metodologie adottate in text mining per la rilevazione della polarità, nel caso di approccio supervisionato, consiste nel calcolo della rilevanza di ogni termine tramite *tf-idf* che può poi essere combinata con la ricerca di co-occorrenza di pattern (*n-gram*), selezionati da una lista specifica di termini marcati esplicitamente come positivi o negativi. Il risultato è poi utilizzato per allenare un classificatore al fine di prevedere le polarità rispetto a dati di test mai visti dal modello. Questa soluzione è però limitata in quanto assume indipendenza tra i termini assegnando la rilevanza in base alla frequenza con cui essi appaiono nel testo senza però considerare i contesti. La presenza di un modello *paragraph vector* pre-calcolato ha consentito di sfruttare una seconda metodologia in grado di assegnare rilevanza ai termini sulla base non solo della frequenza con cui essi appaiono ma anche del contesto di riferimento.
In considerazione del fatto che l'approccio *paragraph vector* è una mera estensione di quello *word embedding* (Word2Vec), il modello pre-allenato ha comunque consentito l'accesso ai vettori di features dei singoli termini. Ricordando che la somma di questi ultimi produce un ulteriore vettore rappresentante il significato di entrambe le parole, è stata sfruttata tale caratteristica per derivare un piccolo lexicon contenente i termini positivi e negativi relativi al testo in esame. In particolar modo è stata adottata una versione modificata dell'approccio descritto in [16], la cui idea è quella di sfruttare un famoso test psicometrico (*POMS*, Profile of mood states) per derivare 7 classi di umore (prestabilite) riferite al testo in esame (tensione-ansia, depressione-sconforto, rabbia-ostilità, vigore, fatica-inerzia, confusione-perplesità, stima).
A partire dalle *seed-words* indicate in tabella 5, le quali sono state utilizzate come termini di partenza anche nel paper [16], è stata sfruttata la rappresentazione word-embedded dei termini al fine di ricercare tutte le parole più simili alle combinazioni di 2 o 3 seed-words. Tale operazione è stata eseguita con lo scopo di espandere il lexicon di partenza identificando tutti i termini semanticamente simili a quelli contenuti nella lista delle seed-words. Ogni nuova parola individuata è stata reimmessa all'interno della lista delle seed-words e l'operazione è stata ripetuta finché è stato possibile estrarre termini con similarità coseno superiore a 0.7.

Dati i termini t_1, t_2 e t_3 rispettivamente più simili ai vettori di parole v_1, v_2 e v_3 (appartenenti ad una delle categorie descritte nella tabella 5), l'estrazione di una nuova parola simile a t_1, t_2 e t_3 potrebbe portare ad ottenere una nuova voce vicina (tramite similarità coseno) ai termini di partenza ma irrilevante per la categoria a cui le seed-words originarie appartenevano. Per tale motivo nella combinazione di termini è sempre stato incluso un vocabolo appartenente alla categoria di POMS da cui la ricerca è stata originata.
L'operazione ha portato ad espandere la lista di seed-words generando un insieme di 626 parole, identificate come le migliori in grado di descrivere (semanticamente) le categorie illustrate nella tabella 5.

Successivamente tutti i termini in ogni tweet sono stati confrontati tramite similarità coseno con ogni parola nella lista appena generata. Sia $v_1 = \{word_1, word_2, ...\}$ il vettore dei termini appartenenti al tweet 1 e $candidate = \{S_1, S_1, ...S_7\}$ il vettore contenente la lista dei termini appartenenti alle categorie

N	**Categoria POMS**	**seed words**
1	Tensione	Agitated, Anxious, Disturbed, Nervous, Edgy, Antsy, Uneasy, Concerned, Tense, Apprehensive, Strained, Impatient, Insecure, Afraid, Distressed
2	Rabbia	Annoyed, Enraged, Heated, Resentful, Indignant, Outraged, Sullen, Bitter, Furious, Irate, Irritable, Offended, Mad
3	Fatica	Bored, Bushed, Disgusted, Exhausted, Fatigued, Overworked, Jaded, Drained, Weakened, Disabled, Sleepy, Overtired
4	Depressione	Desperate, Forlorn, Helpless, Sad, Tragic, Dejected, Discouraging, Gloomy, Pathetic, Wretched, Useless, Despairing, Depressed, Dismayed, Pessimistic
5	Vigore	Animated, Bright, Optimistic, Lively, Rosy, Chirpy, Cheery, Perky, Merry, Buoyant, Active, Zealous, Potent, Dynamic, Brisk, Spirited, Vital, Peppy, Frisky
6	Confusione	Addled, Puzzled, Perturbed, Distracted, Dazed, Bewildered, Befuddled, Confused, Disorganized, Forgetful, Inattentive
7	Stima	Ashamed, Abashed, Apologetic, Embarrassed, Humbled, Honored, Noble, Satisfied, Appreciative, Content, Competent, Qualified, Decent

Tabella 5: seed words

illustrate in 5, dove S_1 è l'insieme delle parole che identificano lo stato d'animo 'tensione' ed S_7 quello che identifica la 'stima', è stata calcolata la similarità tra tutti i vettori v e *candidate* al fine di determinare la correlazione tra ogni tweet ed i relativi stati d'animo. In particolar modo si è deciso di selezionare la massima similarità coseno per ogni tweet ed ogni categoria in 5. Dato l'elevato costo computazionale, l'intero procedimento è stato eseguito su GPU abbattendo di circa 10.000 volte il tempo di calcolo. L'operazione ha portato ad ottenere un vettore $W_i = \{w_1, w_2, ...w_7\}$ contenente la rilevanza di ogni termine rispetto al relativo stato d'animo. Ogni vettore W_i è poi traslato all'interno del range [0,1] in modo tale che i valori siano sulla stessa scala delle altre informazioni fornite in input alla rete. Infine, una semplice somma per elementi è stata eseguita per tutti i termini di ogni tweet [16]:

$$W_i = \frac{W_i - W_i^{max}}{(W_i^{max} - W_i^{min})} * (1 - 0) + 0$$

$$V_{tweet} \in V^7 = \sum_i W_i$$

La figura 24 illustra graficamente il procedimento descritto:

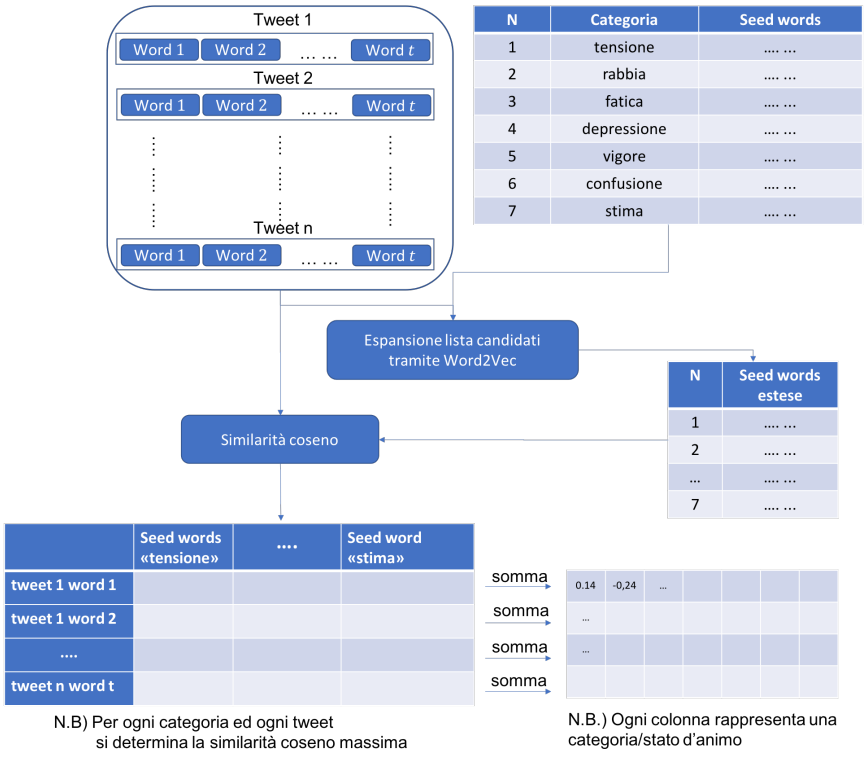


Figura 24: Procedimento seguito per la determinazione degli stati d'animo

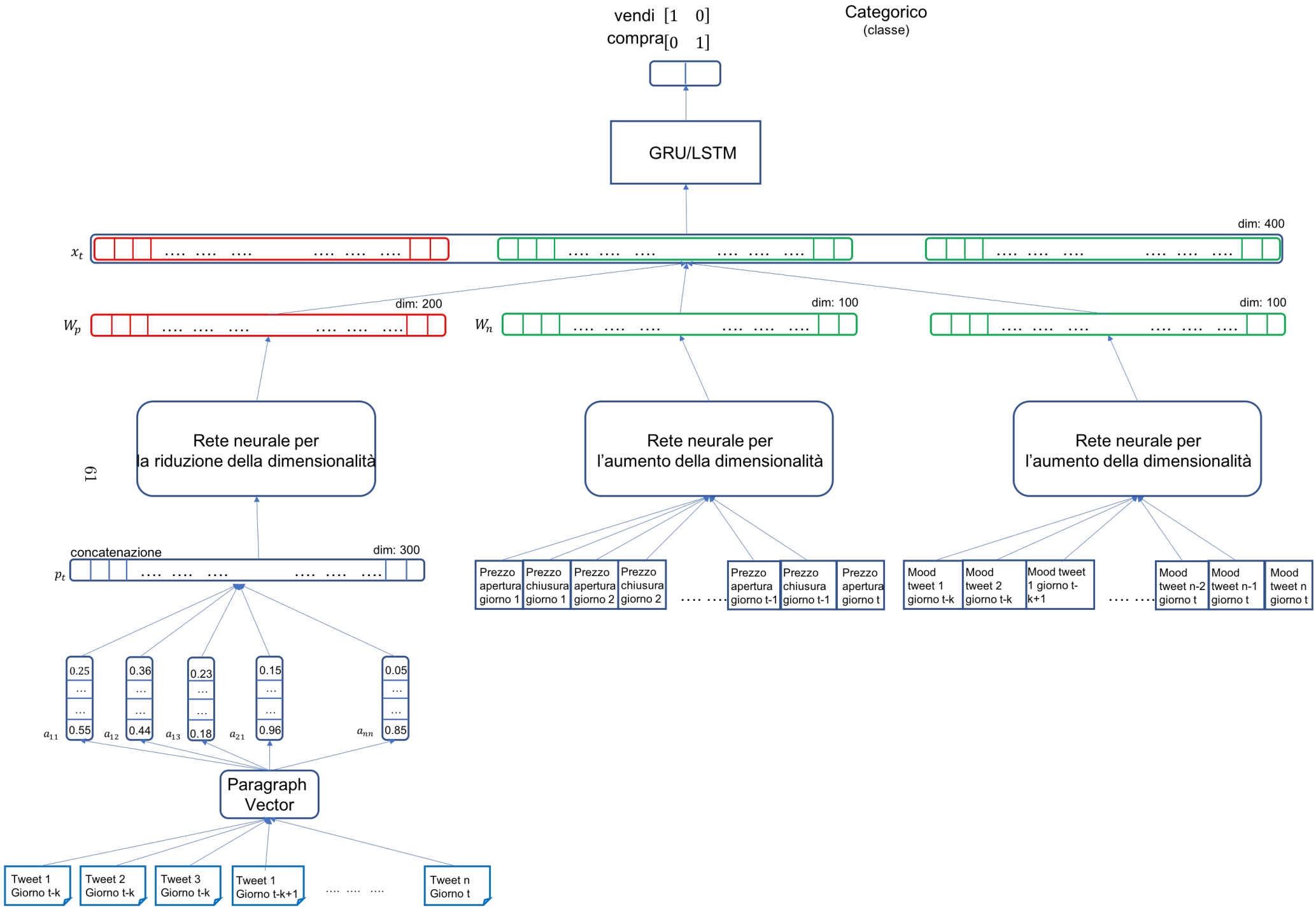


Figura 25: Seconda revisione del modello

La figura 25 illustra l'architettura generica utilizzata mentre l'immagine 5.8 mostra nel dettaglio il modello utilizzato per gli esperimenti.

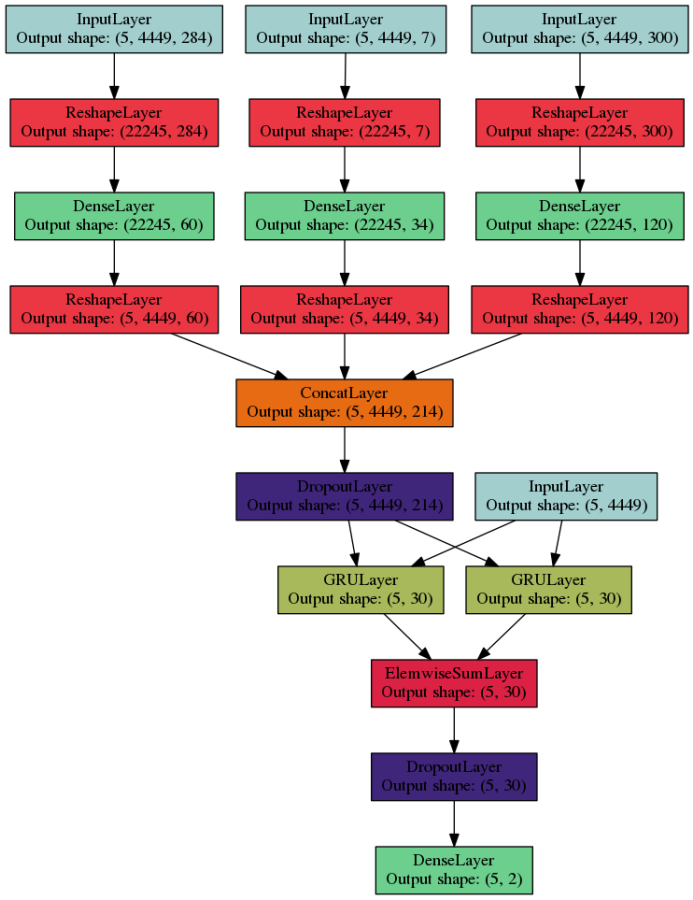


Figura 26: dettaglio del modello utilizzato

5.9 Risultati e discussioni

Gli esperimenti sono stati condotti in due passaggi. In primo luogo è stato adottato un approccio similare a quello riportato in [13] dove il modello illustrato in 25 è stato addestrato fornendo in input la storia passata dei prezzi e la **media dei vettori di features** in ogni fold (di ogni giorno). Questa scelta è ragionevole in quanto, dal punto di vista concettuale, la media di due vettori di features (ottenuti tramite PV) rappresenta la congiunzione del significato di entrambi i vettori (ref:8). Ciò significa che la media dei vettori ottenuti tra due tweets rappresenta esattamente la congiunzione del significato di entrambi. Sono quindi state eseguite diverse sperimentazioni tutte comprese di dropout:

Le stesse sperimentazioni sono poi state replicate considerando l'intero insieme dei tweets disponibili, campionandone una piccola quantità per ogni giorno. In questo caso l'input fornito alla rete era composto dalla storia passata dei prezzi

Tabella 6: Risultati con media dei vettori di features dei tweets

accuratezza in validazione (100 epoche)			
	20 neuroni GRU	**50 neuroni GRU**	**512 neuroni GRU**
3 giorni	52%	54,2%	53,98%
4 giorni	52%	55,4%	53,3%
5 giorni	53,40%	52,13%	51%

Tabella 7: Risultati considerando tutti i tweets disponibili

accuratezza in validazione (100 epoche, momentum, con dropout)			
	20 neuroni GRU	**50 neuroni GRU**	**512 neuroni GRU**
3 giorni	50,83%	52,50%	50,83%
4 giorni	50,00%	55,00%	50,00%
5 giorni	53,33%	52,50%	50,83%

di apertura e chiusura, l'insieme dei vettori di features ed i moods associati ai relativi tweets (come descritto in 24):

I risultati ottenuti dai due esperimenti riportati in tabella 6 e 7 sono pressoché similari ed entrambi piuttosto deludenti, specie se confrontati con quelli ottenuti con il modello (1) il cui unico input era costituito dalla serie storica dei prezzi. Osservando le due tabelle, è possibile notare come la fornitura di tutte le informazioni disponibili (tabella 7) produca, nel complesso, risultati leggermente inferiori rispetto a quelli ottenuti con in input la media dei vettori di features. Considerando che in quest'ultimo caso le informazioni fornite in ingresso alla rete sono sensibilmente inferiori, è ragionevole attribuire l'ulteriore degrado di performance al rumore presente all'interno del dataset.
In ogni caso, vista l'impossibilità nel notare particolari differenze nei risultati ottenuti al variare delle topologie (incremento neuroni negli stati nascosti) ed osservando il notevole gap in termini prestazionali rispetto alla rete 5.5, è possibile concludere che il modello presentato è caratterizzato da un bias (varianza) troppo elevato che non consente di apprendere in maniera soddisfacente.
Dal punto di vista dell'apprendimento supervisionato, una possibile soluzione potrebbe risiedere nell'individuazione e rimozione del rumore (tweets non inerenti a situazioni finanziare) mediante approcci similari a quello presentato in [6].
Esistono tuttavia alcune metodologie alternative ed in forte crescita che verranno illustrate nella sezione seguente.

6 Possibili approcci alternativi

Nonostante il modello utilizzato nella sezione 5.9 fosse ritenuto valido, i risultati hanno smentito tale ipotesi sollevando la necessità di individuare soluzioni alternative. Una di queste, facente parte del filone dell'intelligenza artificiale in forte crescita negli ultimi anni, è l'approccio basato sull'apprendimento per rinforzo. L'idea alla base di questa metodologia risiede nel superamento del vincolo imposto dall'apprendimento supervisionato i cui risultati sono vincolati alla conoscenza delle labels da predire. Con questa nuova tecnica un modello è in grado di apprendere conoscendo esclusivamente il modello dell'ambiente in cui si trova ad operare.
Nelle successive sezioni viene illustrata la formulazione del problema e le caratteristiche di base di questa nuova metodologia.

6.1 Apprendimento non-supervisionato

Il maggior vincolo derivante dall'utilizzo di approcci supervisionati risiede nella necessità di conoscere a priori le labels tramite le quali addestrare il modello. Ciò implica la presenza di almeno un esperto di dominio che analizzi il problema ed estragga i risultati attesi per ogni campione fornito in input alla rete. Tale assunzione non sempre può essere considerata attuabile in contesti reali. Inoltre è doveroso notare che l'approccio supervisionato è intrinsecamente debole nel caso in cui il problema sia modellabile con un processo non-markoviano.
L'impossibilità di istruire un modello decisionale per mezzo dell'apprendimento supervisionato, a causa dell'assenza delle labels associate ai campioni sottoposti alla rete, ha costretto alla ricerca di nuove forme di apprendimento. Un approccio che ha suscitato un importante fermento nell'ambito della ricerca è *l'apprendimento per rinforzo*. Alla base di questa tecnica c'è un agente (sistema) immerso in un ambiente che può cambiare stato. L'apprendimento per rinforzo mira a consentire al sistema di osservare l'ambiente ed adattarsi alle sue mutazioni compiendo azioni che saranno poi valutate tramite una funzione *"di ricompensa"*, la quale rappresenta la bontà delle decisioni tratte.

6.2 Apprendimento per rinforzo

Per comprendere a fondo l'idea dell'apprendimento per rinforzo, è necessario illustrare la definizione di processo markoviano: *Un* ***processo markoviano*** *è un processo aleatorio nel quale la probabilità di transizione, che determina il passaggio ad un nuovo stato del sistema, dipende esclusivamente dallo stato immediatamente precedente e non dagli stati passati.*
Formalmente:

1. $S \in S$ = Spazio degli stati (set di tutti i possibili stati del sistema)
2. T = Funzione di transizione.
3. $M = S, T$

Dove M è il modello. Un processo markoviano può essere descritto per mezzo della proprietà di markov (*assenza di memoria*) la quale indica che se un evento

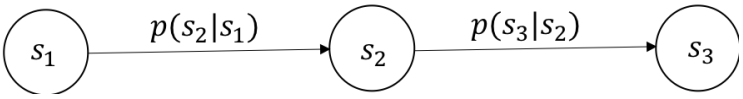


Figura 27: Catena di Markov

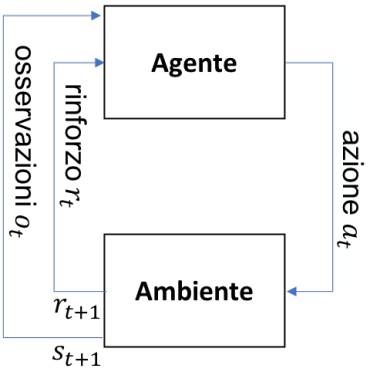


Figura 28: Processo decisionale di Markov

non è ancora accaduto, non sono disponibili informazioni sul fatto che questo accadrà in futuro:

$$\overrightarrow{\mu}_{t+1} = p(s_{t+1} = i|s_t = j)\overrightarrow{\mu}_t \tag{9}$$

dove $\overrightarrow{\mu}_(t+1)$ e $\overrightarrow{\mu}_t$ sono i vettori di probabilità rispettivamente all'istante t e $t+1$. Come è possibile notare, lo stato $t+1$ dipende esclusivamente dallo stato all'istante t e non da tutta la storia passata. La proprietà di Markov assume un ruolo fondamentale all'interno di un processo decisionale di Markov (MDP). Nel contesto dell'apprendimento non-supervisionato assumono una forte rilevanza i **processi decisionali di Markov finiti** (FMDP), i quali hanno stati ed azioni finiti.
In un processo decisionale markoviano un agente compie azioni (in ogni stato) al fine di adattarsi ai cambiamenti dell'ambiente circostante:

1. $S \in S$ = Spazio degli stati (set di tutti i possibili stati del sistema)
2. $T : S \times A \to \pi(S)$ = Funzione di transizione che assegna ad ogni coppia stato-azione una distribuzione di probabilità in S
3. $A \in A$ = Spazio delle azioni (set di tutte le possibili azioni che l'agente può compiere)
4. $r : S \times A \to \mathbb{R}$ = funzione di ricompensa la quale assegna un valore numerico ad ogni possibile transizione
5. $M = S, A, T, r$

L'apprendimento per rinforzo può essere percepito esattamente come un MDP dove, ad ogni istante t, l'agente percepisce l'ambiente circostante tramite osservazioni o_t sulla base delle quali decide di attuare un'azione a_t. L'effetto di tale azione cambia lo stato dell'ambiente s_{t+1} e genera una ricompensa r_{t+1}. Il fine

ultimo dell'agente è quello di massimizzare il *rinforzo atteso*, ovvero la funzione di rinforzo r nel lungo periodo. Ne esistono diverse formulazioni:

1. $E[\sum_{k=0}^{\infty} r_{t+k+1}]$: non attuabile nella pratica in quanto la serie è illimitata
2. $E[\sum_{k=0}^{T} r_{t+k+1}]$: limita l'orizzonte temporale al più a T istanti
3. $E[\sum_{k=0}^{\infty} \gamma^k \times r_{t+k+1}]$: è chiamata *infinite discounted horizon* ed è la soluzione più comunemente utilizzata. γ (***fattore di rinforzo***) rappresenta l'interesse dell'agente rispetto ai rinforzi che potrà ricevere in futuro.
   Dato che $0 < \gamma < 1$, se $\gamma \sim 0$ l'agente privilegerà i rinforzi nel breve futuro mentre penalizzerà quelli a lungo termine, se invece $\gamma \sim 1$, accadrà l'esatto opposto.

La funzione di ricompensa r enunciata nella formalizzazione del MDP è chiamata ***politica*** all'interno del contesto RL ed è indicata con π. Tale politica (o strategia) è una funzione $\pi : S \times A \to \mathbb{R}$ che associa la probabilità di eseguire un'azione a nello stato s (conoscendo lo stato s, probabilità condizionata).
Al fine di poter massimizzare la funzione di rinforzo, l'agente ha bisogno di conoscere in ogni istante t la *valutazione del rinforzo atteso* per un certo stato s_t ed una certa azione a_t. Tale funzione è chiamata ***value function*** e può non essere unica:

1. $V : S \to \mathbb{R}$: è chiamata *value function* ed associa il valore del rinforzo atteso agli stati del sistema.
2. $Q : S \times A \to \mathbb{R}$: è chiamata *action-value function* ed associa il valore del rinforzo atteso ad ogni coppia *stato-azione* (s_t, a_t) se in s_t si verifica l'azione a_t.

A questo punto è possibile determinare quanto sia "conveniente" per l'agente intraprendere una determinata azione (in un certo stato) sulla base della politica π e della value funtion (V o Q):

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}[\sum_{k=0}^{\infty} \gamma^k \times r_{t+k+1} | s = s_t] \\ Q^{\pi}(s,a) &= E_p i[\sum_{k=0}^{\infty} \gamma^k \times r_{t+k+1} | s = s_t \wedge a = a_t] \end{aligned} \tag{10}$$

Le due equazioni qui sopra rappresentano il ***valore di una politica***. Tale concetto è ciò che guida il comportamento di un agente nell'ambiente. Un agente, infatti, potrebbe sapere esattamente quale dovrebbe essere l'azione successiva da eseguire al fine di massimizzare la funzione di rinforzo ma potrebbe anche verificarsi il caso in cui debba scegliere in maniera casuale in quanto non riesce ad individuare l'operazione migliore.
Il valore di una politica, che descrive il comportamento di un agente, è composta da:

1. $\pi : S \times A \to \mathbb{R}$: la politica che definisce la strategia per la selezione delle azioni. Rappresenta la probabilità condizionata $\pi_\theta(a_t|s_t)$, ovvero la probabilità di eseguire un'azione a nello stato s.

2. V o Q: value-function che rappresenta il valore dell'azione a nello stato s.

E' utile sottolineare, inoltre, che è possibile determinare una relazione di ordinamento fra tutte le politiche:

$$\pi <= \pi' \iff V^{\pi}(s) <= V^{\pi'}(s) \forall s \in S$$

da cui è possibile comprendere come esista almeno una ***politica ottima*** π^* a cui viene associata una value function ottima V^* (o Q^*):

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$
$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

6.3 Apprendimento per rinforzo: policy iteration

Considerando ciò che è stato illustrato nella sezione precedente, il problema dell'apprendimento per rinforzo può essere percepito come un problema di ottimizzazione multimodale dove l'obiettivo è individuare la policy ottima (la quale massimizza la funzione di rinforzo) che però è implicitamente definita dal suo valore (value-function). L'obiettivo ultimo è quello di individuare una successione monotona crescente di politiche π_t, a cui corrisponde una stessa successione monotona crescente di value-functions. Tale fine può essere raggiunto secondo l'algoritmo ***policy iteration***:

1. sia π_0 la politica iniziale
2. determina la value-function (di π_0) V^{π_0} o Q^{pi_0}.
3. ricerca una politica $\pi_1 : V^{\pi_1} > V^{\pi_1}$
4. itera dal passo 2 finché una politica migliore dell'attuale esiste

Il risultato di tale algoritmo produce una sequenza alternata di passi di valutazione (V) e ricerca della politica migliore (R):

$$\pi_0 \rightarrow^V V^{\pi_0} \rightarrow^R \pi_1 \rightarrow^V V^{\pi_1} \dots \pi^* \rightarrow^V V^*$$

In ogni stato, l'agente ha la facoltà di assegnare una valutazione della bontà di uno stato s_t sulla base delle esperienze passate. Tale assegnazione avviene mediante un filtro complementare:

$$v_{k+1} \leftarrow (1-\alpha)v_k + \alpha\Delta_{t+1} \tag{11}$$

dove:

1. v_{k+1}, v_k: rappresentano rispettivamente lo stato successivo e corrente
2. $[0 <= \alpha <= 1]$: rappresenta il *learning rate*
3. Δ_{t+1}: rappresenta la valutazione dell'azione corrente effettuata dalla policy (premio o penalità)

Osservazione Tutti gli algoritmi adottati nell'apprendimento non-supervisionato, richiedono che l'agente conosca come l'ambiente reagirà alle sue azioni (parametro T), ovvero deve avere un modello dell'ambiente.

6.4 Apprendimento per rinforzo: Q-learning

Uno dei più conosciuti algoritmi di apprendimento per rinforzo, che si basa su ciò che è stato illustrato nelle sezioni precedenti, è l'algoritmo del *Q-learning*. L'obiettivo di quest'ultimo è quello di consentire ad un sistema di adattarsi all'ambiente che lo circonda cercando di massimizzare il valore del premio dell'istante successivo **senza che esso conosca il modello dell'ambiente**.
L'idea alla base dell'apprendimento per rinforzo, e quindi anche dell'algoritmo del *Q-learning*, è quella di generalizzare che, a differenza dell'apprendimento supervisionato, consiste nel costruire una *action-value function* (Q) in grado di associare il rinforzo atteso ad ogni possibile coppia *stato-azione*. Considerando che tipicamente il numero di possibili stati (S) in cui un agente può trovarsi è molto elevato, va da sé che il costo computazionale affinché possa avvenire l'apprendimento è altrettanto elevato. In particolar modo, l'assegnazione del rinforzo atteso ad ogni possibile coppia *stato-azione* richiede grandi quantità di memoria (la memorizzazione dell'*action-value function* richiede $|S|x|A|$) ed elevato tempo di elaborazione, in quanto ogni coppia *stato-azione* deve essere visitata un numero teoricamente infinito di volte.
Considerati i problemi sopra riportati, la capacità di un modello di apprendere, e quindi di generalizzare, risiede nella possibilità di costruire una *rappresentazione dell'action-value function* (un valore di rinforzo atteso per ogni coppia *stato-azione*) *compatta*.
La formulazione dell'algoritmo continua dall'equazione 11 dove si suppone che l'agente, precedentemente nello stato s_t, abbia agito con un'azione a_t ricevendo una ricompensa r_{t+1} ed una sensazione s_{t+1}. Di conseguenza:

$$Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha\Delta_{t+1}$$

dove:

1. $Q(s_t, a_t)$: è il rinforzo atteso associato alla coppia *stato-azione* (s_t, a_t). Tale valore è stato 'calcolato' nella storia passata ed è ciò che l'agente conosce rispetto alla coppia (s_t, a_t).

2. Δ_{t+1}: è la funzione deputata a valutare la bontà dell'azione $t+1$ intrapresa dall'agente (premio o penalità) e determina come il 'vecchio' valore $Q(s_t, a_t)$ verrà aggiornato.

E' quindi necessario definire la funzione Δ_{t+1} che dipende:

1. dal rinforzo immediato r_{t+1} ricevuto

2. dallo stato raggiunto s_{t+1} a seguito dell'azione intrapresa

Supponendo di utilizzare una funzione di rinforzo atteso con finestra temporale infinita, il rinforzo all'istante t dipende da tutti i successivi fini all'istante ∞:

$$E[\sum_{k=0}^{\infty} \gamma^k \times r_{t+k+1}] = r_{t+1} + \gamma E[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2}] \tag{12}$$

La funzione 12 è conosciuta come **equazione di Bellman** dove r_{t+1} è la ricompensa che l'agente ha ricevuto per essere entrato nello stato corrente mentre la seconda parte dell'equazione rappresenta le ricompense future (scontate) che

l'agente riceverà entrando nello stato successivo. Dato che r_{t+1} è conosciuto, è possibile esprimere la seconda parte dell'equazione in funzione di Q:

$$E[\sum_{k=0}^{\infty} \gamma^k x r_{t+k+2}] = \max_{a \in A} Q(S_{t+1}, a)$$

ovvero si ricerca l'azione che massimizza la ricompensa dello stato successivo s_{t+1}.
La formula finale è quindi:

$$Q(s_t, a_t) \leftarrow (1 - \alpha) Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a)) \tag{13}$$

dove:

1. $Q(s_t, a_t)$ è il 'vecchio' rinforzo già conosciuto dall'agente
2. $r_{t+1} + \gamma \max_{a \in A} Q(s_{t+1}, a)$ è il nuovo valore di rinforzo:
   (a) r_{t+1} è la ricompensa ricevuta dall'agente per essere entrato nello stato corrente
   (b) $\max_{a \in A} Q(s_{t+1}, a)$ è la massima ricompensa ottenuta nell'attuazione di un'azione a a partire dallo stato corrente.

7 Appendice: Apprendimento supervisionato (aggiornamento pesi)

Nell'apprendimento supervisionato, lo stato di partenza implica la conoscenza dell'output desiderato da parte della rete. In questo caso diventa possibile automatizzare il processo di aggiornamento dei pesi ricercando la configurazione che consenta di minimizzare l'errore commesso rispetto al comportamento atteso. Una delle prime proposte a tal fine fu la *legge di Hebb* secondo la quale *'se due unità sono attive nello stesso istante, il peso della relativa connessione deve essere incrementato'*. Questo modello nacque per giustificare l'apprendimento nelle reti neurali biologiche ma presentava alcuni svantaggi tra cui la crescita illimitata dei pesi nel caso di grandi training set.
La principale difficoltà nell'individuazione di una soluzione di pesi soddisfacente, risiede nelle modalità con le quali questi vengono aggiornati all'interno degli strati nascosti. Considerando una rete shallow e fissato $w \in R^n$ ad indicare i parametri incogniti, la ricerca della migliore configurazione può essere descritta come un problema di ottimizzazione del tipo:

$$\min_w E(w) = \frac{1}{2P} \sum_{p=1}^{P} E_p(w) \tag{14}$$

dove:

1. E(w): è la funzione obiettivo da minimizzare
2. $E_p(w)$: rappresenta l'errore commesso dalla rete con la configurazione w in presenza del pattern in ingresso p. Tipicamente viene utilizzato l'errore quadratico.

Tuttavia il problema appena formulato è computazionalmente oneroso per via di diversi fattori quali la possibilità di minimi locali e l'elevata dimensionalità di w.

7.1 Backpropagation

Tra i vari approcci proposti rivolti all'aggiornamento automatizzato dei pesi, quello maggiormente utilizzato è chiamato *metodo di retropropagazione* il quale ricerca la configurazione che minimizza l'errore commesso mediante la *discesa del gradiente*. Questa metodologia si basa sull'idea di differenziare automaticamente la funzione, operazione a cui ben si presta un calcolatore, a patto che quest'ultima sia differenziabile (ovvero che tutte le derivate parziali calcolate nel punto esistano).
Sfruttando l'**anti-gradiente** è possibile individuare la direzione lungo la quale la funzione di costo decresce più velocemente con il conseguente aggiornamento dei pesi come:

$$\Delta w = -\lambda \nabla E(w^t) \tag{15}$$

dove:

1. $\Delta(w)$: è la variazione da apportare alla configurazione dei pesi.
2. λ: rappresenta il **tasso di apprendimento** che indica la velocità con la quale i pesi vengono aggiornati e di conseguenza con la quale la rete apprende.

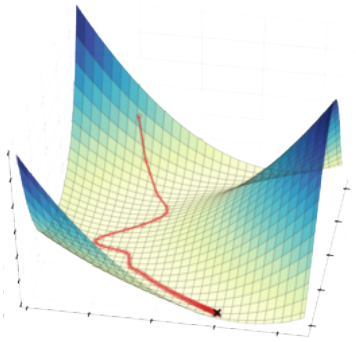


Figura 29: Rappresentazione grafica del metodo di discesa del gradiente

3. $-\nabla E(w^t)$: rappresenta la direzione lungo la quale transitare al fine di ridurre maggiormente l'errore commesso con configurazione dei pesi all'istante t.

L'aggiornamento dei pesi di una rete avviene in due fasi:

1. Nella prima fase l'input della rete è propagato *in avanti* tra i vari strati di neuroni
2. Nella seconda fase, è calcolato l'errore commesso, rispetto al comportamento desiderato, e propagato *all'indietro* al fine di aggiornare i pesi in maniera consistente con la contribuzione all'errore.

Ciò significa che, durante la retropropagazione, i pesi che hanno contribuito maggiormente alla formazione dell'errore subiranno una variazione maggiore rispetto agli altri. Indicando con W la matrice dei pesi che collegano uno strato di neuroni ad un altro e con E la funzione di costo da minimizzare, l'aggiornamento dei pesi può essere riscritto come:

$$\Delta W = -\lambda \nabla \frac{\partial E}{\partial W} \tag{16}$$

dove, supponendo di utilizzare la differenza quadratica (SSE) come funzione di costo:

$$\frac{\partial E}{\partial W} = \frac{\partial \sum \frac{1}{2}(y - \hat{y})^2}{\partial W} \tag{17}$$

con:

1. y: è l'output desiderato dalla rete
2. $\hat{y}$: è l'output ottenuto dalla rete
3. $\sum$: rappresenta la sommatoria di tutti i neuroni che sono in input allo strato di neuroni in esame

L'aggiornamento dei pesi avviene in maniera differente a seconda che si consideri i collegamenti tra gli strati *nascosto-uscita* e *input-nascosto*.
Per semplicità, la matrice dei pesi W viene suddivisa in due matrici distinte $W^{(1)}$ e $W^{(2)}$ contenenti i pesi tra i vari strati.

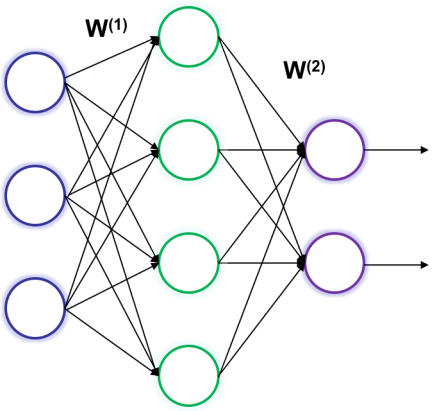


Figura 30: Matrici dei pesi tra i vari strati

Nelle sezioni 7.2 e 7.3 vengono forniti approfondimenti sulla retropropagazione dell'errore nelle reti feedforward. Nella sezione 7.4 vengono invece illustrate alcune varianti del metodo di backpropagation che cercano di superarne le limitazioni.

7.2 Aggiornamento pesi strato *nascosto-uscita*

Nel caso in cui l'output desiderato della rete sia disponibile, è possibile determinare l'errore commesso al fine di determinare le connessioni che hanno contribuito maggiormente alla sua formazione. Tali connessioni riceveranno un aggiornamento più consistente rispetto a quelle che hanno partecipato in minor parte.

$$\frac{\partial E}{\partial W^{(2)}} = \frac{\partial \sum \frac{1}{2}(y - \hat{y})^2}{\partial W^{(2)}} \tag{18}$$

Per semplicità, verrà momentaneamente tralasciata la somma degli errori dei neuroni in uscita considerando esclusivamente l'errore commesso da un solo neurone. Inoltre, viene operata una suddivisione della connessione tra lo strato nascosto e lo strato in output:

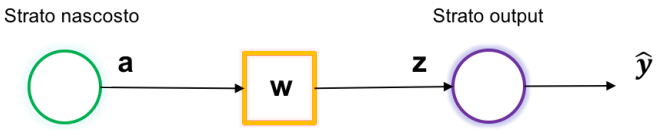


Figura 31: Modello nascosto-output considerato

dove:

1. a: è l'output del neurone nello strato nascosto.
2. w (o $w^{(2)}$): è il peso che connette il neurone dello strato nascosto con quello dello strato di output.
3. z (o $z^{(3)}$): è l'input del neurone nello strato di output, calcolato come aw.

Differenziando con riferimento a (18) (applicando la *regola della catena*), si ottiene:

$$\frac{\partial E}{\partial w} = -\frac{2}{2}(y - \hat{y})\frac{\partial (y - \hat{y})}{\partial w} = -(y - \hat{y})\frac{\partial (y - \hat{y})}{\partial w} \tag{19}$$

L'output desiderato dalla rete (y) può essere considerato come una costante, in quanto indipendente dalla funzione di attivazione del neurone, e pertanto tralasciato. Al contrario $\hat{y}$ dipende esclusivamente dalla funzione di attivazione del neurone e da ciò che gli viene fornito in ingresso:

$$\hat{y} = f(z) \tag{20}$$

dove f rappresenta la funzione di attivazione del neurone. Applicando (20) a (19) otteniamo:

$$\frac{\partial E}{\partial w} = -(y - \hat{y})\frac{\partial f(z)}{\partial w} = -(y - \hat{y})\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w} \tag{21}$$

dove:

1. $\frac{\partial \hat{y}}{\partial z}$: descrive il comportamento della funzione di attivazione che, dato in ingresso y, restituisce in uscita z. La sua differenziazione stabilisce "la quantità" dell'errore totale è imputabile al neurone in esame consentendo una retropropagazione consistente:

   $$\frac{\partial \hat{y}}{\partial z} = f'(z) \tag{22}$$

   dove $f'(z)$ è la derivata della funzione di attivazione calcolata in z (ingresso al neurone).

2. $\frac{\partial z}{\partial w}$: descrive la relazione tra z e w

Per maggiore chiarezza, è inoltre possibile introdurre:

$$\delta = -(y - \hat{y})f'(z) \tag{23}$$

che applicato a (21) porta a riscrivere:

$$\frac{\partial E}{\partial w} = \delta \frac{\partial z}{\partial w} \tag{24}$$

La differenziazione $\frac{\partial z}{\partial w}$ risulta essere molto semplice in quanto è possibile notare che esiste una relazione lineare tra il valore in ingresso al neurone dello strato di output (z) ed il peso (w) data da:

$$z = aw$$

da cui si ottiene:

$$\begin{aligned} \frac{\partial E}{\partial w} &= \delta a \\ \delta &= -(y - \hat{y}) f'(z) \end{aligned} \tag{25}$$

Ovviamente nell'esempio è stato considerato un solo neurone dello strato nascosto in input allo strato di output. Nel caso di reti con più unità in input allo strato di uscita sarà necessario considerare:

1. W: come matrice dei pesi delle connessioni tra tutti i neuroni dello strato nascosto e quelli dello strato di uscita.
2. A: come matrice contenente gli output dei neuroni dello strato nascosto.
3. z: come vettore contenente l'input del neurone dello strato di output.

L'aggiornamento dei pesi avverrà pertanto come:

$$w^{(2)} = w^{(2)} + \lambda \frac{\delta E}{w^{(2)}}$$

7.3 Aggiornamento pesi strato *input-nascosto*

Il procedimento è similare a quello già visto nella sezione precedente con l'unica eccezione che la differenziazione avviene rispetto alla matrice dei pesi dei collegamenti tra gli strati di input e nascosto.
Considerando che il numero dei livelli delle connessioni tra gli strati è maggiore di 1, indichiamo con:

1. x: l'input fornito alla rete.
2. $w^{(1)}$ il peso che collega il neurone nello strato di input con quello nello strato nascosto e con $w^{(2)}$ il peso che collega il neurone nel livello nascosto con quello nello strato in output.
3. $z^{(2)}$ e $z^{(3)}$ gli input rispettivamente al neurone nello strato nascosto e a quello nel livello di output.
4. a l'output del neurone nello strato nascosto.

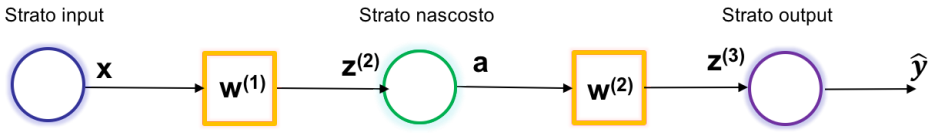


Figura 32: Modello input-nascosto considerato

Dove l'obiettivo è ottenere:

$$\frac{\partial E}{\partial w^1}$$

In maniera similare a quanto già visto:

$$\frac{\partial E}{\partial w^{(1)}} = -(y - \hat{y})\frac{\partial \hat{y}}{\partial w^{(1)}} \Rightarrow -(y - \hat{y})\frac{\partial \hat{y}}{\partial z^{(3)}}\frac{\partial z^{(3)}}{\partial w^{(1)}} \tag{26}$$

come in precedenza, $\frac{\partial \hat{y}}{\partial z^{(3)}} = f'(z^{(3)}$ per cui:

$$\frac{\partial E}{\partial w^{(1)}} = -(y - \hat{y})f'(z^{(3)})\frac{\partial z^{(3)}}{\partial w^{(1)}} \tag{27}$$

ponendo $\delta^{(3)} = (-y - \hat{y})f'(z^{(3)})$ otteniamo:

$$\frac{\partial E}{\partial w^{(1)}} = \delta^{(3)}\frac{\partial z^{(3)}}{\partial w^{(1)}} \Rightarrow \delta^{(3)}\frac{\partial z^{(3)}}{\partial a}\frac{\partial a}{\partial w^{(1)}} \tag{28}$$

Anche questa volta è possibile notare come esista una relazione lineare $z^{(3)} = a^{(2)}w^{(2)}$:

$$\frac{\partial E}{\partial w^{(1)}} = \delta^{(3)}w^{(2)}\frac{\partial a}{\partial w^{(1)}} \tag{29}$$

Differenziando un'ultima volta $\frac{\partial a}{\partial w^{(1)}}$ si ottiene:

$$\frac{\partial E}{\partial w^{(1)}} = \delta^{(3)}w^{(2)}\frac{\partial a}{\partial z^{(2)}}\frac{\partial z^{(2)}}{\partial w^{(1)}} => \delta^{(3)}w^{(2)}f'(z^{(2)})\frac{\partial z^{(2)}}{\partial w^{(1)}} \tag{30}$$

in quanto $\frac{\partial a}{\partial z^{(2)}}$ rappresenta la funzione di attivazione del neurone nello strato nascosto. Inoltre è possibile notare come esista una relazione lineare tra $z^{(2)}$ e $w^{(1)}$ che dipende dall'input x:

$$\begin{aligned} \frac{\partial E}{\partial w^{(1)}} &= x\delta^{(2)} \\ \delta^{(2)} &= \delta^{(3)}w^{(2)}f'(z^{(2)}) \end{aligned} \tag{31}$$

dove il peso è aggiornato come:

$$w^{(1)} = w^{(1)} + \lambda\frac{\partial E}{\partial w^{(1)}} \tag{32}$$

Notare che, per semplicità, è stato considerato un solo neurone per ogni strato. Nel caso di più unità per livello, è necessario considerare tutti gli input, output e pesi in formato matriciale.

7.4 Backpropagation: varianti

Il metodo di retropropagazione si è rivelato essere uno dei primi approcci per l'aggiornamento automatizzato dei pesi negli strati interni di una rete neurale. Tale metodo si fonda interamente sulla tecnica di discesa del gradiente, la quale presenta alcuni limiti:

1. non minimizza la funzione di costo globalmente ma riduce, sulla base del gradiente, il contributo all'errore degli esempi.
2. è soggetta a minimi locali per cui difficilmente individua quelli globali.
3. è un processo intrinsecamente lento che convergerà con velocità pari alla *ripidità* della discesa.

Il metodo di backpropagation illustrato in 7.1 è la versione *batch* (classica) che richiede di calcolare i gradienti per l'intero dataset prima di poter eseguire un solo aggiornamento dei pesi. Tale approccio risulta essere molto costoso specialmente per grandi dataset.
Esistono tuttavia alcune varianti:

1. ***Discesa del gradiente stocastica*** (SGD): a differenza della versione batch, i pesi vengono aggiornati dopo aver calcolato il gradiente di ogni training sample nel data set. Questo approccio si è dimostrato molto più veloce rispetto al precedente ma anche caratterizzato da una forte variabilità dovuta all'utilizzo di un solo esempio di training per volta. Tale proprietà risulta non essere necessariamente negativa in quanto potrebbe consentire di uscire da situazioni di minimi locali 'saltando' in una nuova posizione. D'altra parte la forte variabilità riduce le probabilità di convergenza verso un minimo locale. Un compromesso tra i due approcci consiste nell'utilizzo di *mini-batch* dove il gradiente è calcolato per ogni gruppo di esempi di training. Un ulteriore miglioramento consiste nell'utilizzo di un *learning rate adattivo* (*Adagrad*) che consenta di rallentare la discesa nell'intorno di un minimo e di accelerarla negli altri casi.
2. ***Momentum***: cerca di velocizzare la convergenza di SGD limitandone variabilità ed oscillazioni. L'idea alla base di questo approccio è la stessa che vede l'energia cinetica accumularsi quando una palla è spinta giù da una collina. Se la palla segue una direzione ripida, accumulerà velocemente molta energia mentre quando incontra una resistenza la perderà. Analogamente il momentum cresce in base alle direzioni dei gradienti passati. Se questi hanno tutti la stessa direzione, il momentum tende a crescere, altrimenti cala.

$$\begin{aligned} w &= w - v_t \\ v_t &= \gamma v_{t-1} + \lambda \frac{\partial E}{\partial W} \end{aligned} \tag{33}$$

   dove γ è il valore di momentum. Tale approccio consente di ottenere una buona rapidità di convergenza limitando la variabilità di SGD.
3. ***Adagrad***: nasce sulla base di SGD con il fine di ridurre la forte variabilità e lentezza di quest'ultimo. Adagrad utilizza un *learning rate adattivo*

per ogni parametro così da consentire un aggiornamento "fine" per quelli frequenti ed un più grossolano per quelli meno frequenti. Sia:

$$\nabla_{t,i} = \nabla E(w_i) \tag{34}$$

il gradiente della funzione obiettivo all'istante t per il parametro i, l'aggiornamento dei parametri avviene come:

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_t(i,i) + \epsilon}} \nabla_{t,i} \tag{35}$$

dove:

(a) $G_t(i,i) \in R^{dxd}$ è una matrice diagonale dove ogni elemento contiene la somma dei quadrati di tutti i gradienti, per il parametro w_i, dall'istante 0 fino all'istante t.

$$G_t = \begin{bmatrix} \nabla^2_{t,w_0} + \nabla^2_{t-1,w_0} + \cdots + \nabla^2_{0,w_0} & 0 & 0 & \dots & 0 \\ 0 & \nabla^2_{t,w_1} + \nabla^2_{t-1,w_1} + \cdots + \nabla^2_{0,w_1} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

(b) ϵ: è un parametro che evita la divisione per 0

Come conseguenza, il learning rate viene aggiustato automaticamente compiendo piccoli aggiustamenti per parametri frequenti e grandi modifiche per quelli infrequenti. Si è inoltre notato che le prestazioni dell'algoritmo decadono qualora non venga utilizzato il quadrato dei gradienti nella matrice $G_t(i,i)$. Questa è anche la principale debolezza di Adagrad in quanto i quadrati dei gradienti tendono ad accumularsi al denominatore portando il learning rate ad assumere un valore infinitesimamente piccolo, annullando le proprietà discusse.

4. ***RMSprop***: è un approccio sviluppato indipendentemente da *Adagrad* ma che lo completa superando il limite del learning rate tendente a 0. L'idea alla base di questo metodo risiede nell'accumulo dei gradienti passati, per ogni parametro w_i, ristretti all'interno di una finestra temporale q. RMSprop è definito in maniera similare ad *Adagrad* ma differisce per come i gradienti vengono mantenuti all'interno della matrice $G_t(i,i)$.
Per ridurre la crescita esponenziale del denominatore, come visto in *Adagrad*, ogni elemento diagonale contiene il gradiente quadratico all'istante t interpolato con la media dei gradienti quadratici all'istante *t-1*:

$$G_t = \begin{bmatrix} \gamma M[\nabla^2]_{t-1,w_0} + (1-\gamma)\nabla^2_{t,w_0} & 0 & 0 & \dots & 0 \\ 0 & \gamma M[\nabla^2]_{t-1,w_1} + (1-\gamma)\nabla^2_{t,w_1} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

dove:

(a) $M[\nabla^2]_{t-1,w_0}$: è la media dei gradienti, per il parametro w_0, definiti all'istante *t-1*. Notare che questa è ricorsiva in quanto definita in funzione della media dei gradienti all'istante *t-2* che a sua volta è definita in maniera ricorsiva.

(b) ∇^2_{t,w_0}: è il gradiente quadratico all'istante t.

(c) γ: è il fattore di interpolazione che determina quanta informazione mantenere del gradiente all'istante *t-1* e quanta acquisirne da quello all'istante *t*.

L'aggiornamento del peso avviene esattamente come in *Adagrad*:

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_t(i,i) + \epsilon}} \nabla_{t,i} \tag{36}$$

8 Conclusioni

L'obiettivo di questo documento verteva sull'approfondimento di alcune delle principali tecnologie oggi utilizzate nel campo dell'intelligenza artificiale e machine learning. In particolar modo si è cercato di seguire un filo conduttore al fine di individuare i legami esistenti tra le varie tecnologie e le motivazioni che hanno spinto gli studiosi ad individuare nuove soluzioni. Ci si è focalizzati quindi sul metodo di apprendimento supervisionato (sezione 7) con l'intento di esplorare le reali motivazioni che portano ad un modello neurale volto ad apprendere illustrando successivamente l'evoluzione dei vari approcci.
Le nozioni apprese sono state poi impiegate in un caso di studio reale come quello dello stock-prediction al fine di comprendere i comportamenti delle reti neurali in tutte le loro sfacettature. Nonostante i non buoni risultati relativi al progetto eseguito, sono stati illustrati nuovi ed emergenti approcci che ci si propone di applicare in casi di studio reali similari a quello scelto.

Riferimenti bibliografici

[1] I. D. Alex Graves, Greg Wayne. Neural turing machine, 2014. https://arxiv.org/pdf/1410.5401.pdf.

[2] A. C. M. Andrew W. Lo. Stock market prices do not follow random walks. http://www.ekonometria.wne.uw.edu.pl/uploads/Main/1988.RW.Lo.MacKinlay.VRTest.pdf.

[3] D. V. R. Andrew W. Lo. The psychophysiology of real-time financial risk processing, 2002. http://alo.mit.edu/wp-content/uploads/2015/06/PsychophysFinRiskProcessing2002.pdf.

[4] M. Burton. A random walk down wall street: time-tested strategy for successful investing, 1973.

[5] E. F. Fama. Efficient capital markets: A review of theory and emppirical work, 1970. http://efinance.org.cn/cn/fm/Efficient%20Capital%20Markets%20A%20Review%20of%20Theory%20and%20Empirical%20Work.pdf.

[6] A. P. R. P. Giacomo Domeniconi, Gianluca Moro. Learning to predict the stock market dow jones index detecting and mining relevant tweets, 2017. https://www.researchgate.net/publication/321072395_Learning_to_Predict_the_Stock_Market_Dow_Jones_Index_Detecting_and_Mining_Relevant_Tweets#pf7.

[7] X.-J. Z. Johan Bollen, Huina Mao. Twitter mood predicts the stock market, 2010. https://arxiv.org/pdf/1010.3003.pdf.

[8] S. V. Marco Somalvico, Amigoni Franceso. Intelligenza artificale, 2003.

[9] A. K. I. S. R. S. Nitish Srivastava, Geoffrey Hinton. Dropout: A simple way to prevent neural networks from overfitting, 2014. http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf.

[10] X. L. A. N. Quanzhi Li, Sameena Shah. Data sets: Word embeddings learned from tweets and general data, 2017. https://www.aaai.org/ocs/index.php/ICWSM/ICWSM17/paper/download/15672/14889.

[11] T. M. Quoc Le. Distributed representations of sentences and documents. https://cs.stanford.edu/%7Equocle/paragraph_vector.pdf.

[12] Y. B. Razvan Pascanu, Tomas Mikolov. On the difficulty of training recurrent neural networks, 2013.

[13] T. M. K. U. Ryo Akita, Akira Yoshihara. Deep learning for stock prediction using numerical and textual information. http://ieeexplore.ieee.org/abstract/document/7550882/?reload=true.

[14] J. S. Sepp Hochreiter. Long short-term memory, 1997. http://www.bioinf.jku.at/publications/older/2604.pdf.

[15] G. C. J. D. Tomas Mikolov, Kai Chen. Efficient estimation of word representations in in vector space. https://arxiv.org/pdf/1301.3781.pdf.

[16] Y. H. H. L. T. Z. Weijun Wang, Ying Li. A method for identifying the mood states of social network users based on cyber psychometrics, 2017. www.mdpi.com/1999-5903/9/2/22/pdf.

[17] P. V. C. J. Yoshua Bengio, RÃľjean Ducharme. A neural probabilistic language model. http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf.