UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

"RETI NEURALI RICORRENTI PER LA PREDIZIONE DEI PREZZI DI MERCATO"

Relatore: Prof. Testolin Alberto

Laureando: Trabucco Francesco

ANNO ACCADEMICO 2021– 2022

Data di laurea 16/10/2022

Avendo concluso questo capitolo universitario, voglio ringraziare l'organizzazione e tutti i professori sempre preparati e disponibili, in particolar modo il professore Testolin che mi ha seguito in questa ricerca con una rara professionalità.
Ringrazio anche tutte le persone che mi hanno supportato e mi sono state d'aiuto. In primis i miei compagni universitari Giorgia e Davide, che da studenti modelli mi hanno sempre motivato non facendomi mai perdere la strada. Ringrazio la mia famiglia per avermi sempre spronato a continuare, nonostante i momenti di difficoltà. Grazie ad Alice, compagna di studio e di vita che ha saputo donarmi la serenità e la tranquillità di affrontare il mio percorso al meglio.

ABSTRACT

L'obiettivo di questa tesi di progetto è la previsione dei valori futuri dei prezzi di chiusura dei mercati. Per visualizzare i risultati dei prezzi di chiusura a 1 giorno, 5 giorni e 15 giorni vengono utilizzati tre diversi algoritmi di reti neurali ricorrenti, una RNN pienamente connessa, una LSTM e una GRU.
Le reti neurali ricorrenti (RNN) sono una classe di modelli di deep learning per l'elaborazione di dati sequenziali. Nelle applicazioni pratiche, invece, le reti neurali long short term memory (LSTM), che appartengono a RNN gated, possono apprendere dipendenze a lungo termine più facilmente rispetto alle semplici architetture ricorrenti. Nel campo le reti LSTM si sono rivelate molto efficaci per la previsione di serie temporali e sono state applicate con successo a molti campi.
Per la costruzione del dataset sono state scelte tre serie temporali in base alla loro diversità sull'andamento dei prezzi. Questa scelta è stata fatta per valutare il comportamento delle reti neurali in funzione dell'instabilità della linea dei prezzi.
In particolare sono stati selezionati: Cisto System (CSCO), Petrolio grezzo (CROI), Amazon (AMZN).
I risultati sperimentali mostrano che i modelli LSTM e GRU sono più precisi di modelli RNN semplici ma si dimostrano comunque inadatti alla previsione futura, in quanto la rete impara a seguire l'andamento dei prezzi e non il cambio di trend, poiché i pattern di continuazione sono di gran lunga maggiori di quelli di cambio.
Gli strumenti utilizzati, soprattutto l'utilizzo delle LSTM, sono tutt'oggi allo stato dell'arte e i risultati ottenuti ci si avvicinano molto.

The goal of this project thesis is the forecast of future market closing prices. Three different recurrent neural network algorithms, a fully connected RNN, an LSTM, and a GRU, are used to display the 1-day, 5-day and 15-day closing price results.
Recurrent neural networks (RNNs) are a class of deep learning models for processing sequential data. In practical applications, however, long-term memory (LSTM) neural networks, which belong to gated RNNs, can learn long-term dependencies more easily than simple recurring architectures. LSTM networks have proved to be very effective for predicting time series and have been successfully applied to many fields.
For the construction of the dataset, three graphs were chosen based on their diversity on the price trend. This choice was made to evaluate the behavior of neural networks as a function of the instability of the price line.
In particular, the following were selected: Cisto System (CSCO), Crude oil (CROI), Amazon (AMZN).
The experimental results show that the LSTM and GRU models are more accurate than simple RNN models but they prove to be unsuitable for future forecasting, as the network learns to follow the price trend and not the trend change, since the continuation patterns they are far greater than exchange rates.
The tools used, especially the use of LSTMs, are still state of the art and the results obtained are very close.

INDICE

1– INTRODUZIONE

Negli ultimi anni, algoritmi intelligenti sono stati progettati e analizzati in molti domini come il tracciamento dell'assetto e il controllo di veicoli spaziali [1], [2], il controllo dei sistemi di conversione dell'energia eolica [3], [4] e l'analisi e la progettazione di convertitori step-down [5].

Con il rapido progresso dell'intelligenza artificiale, sempre più ricercatori utilizzano tecniche di apprendimento automatico per le previsioni del mercato azionario. Support Vector Regression (SVR), ad esempio, è un modello di apprendimento automatico in grado di identificare l'iperpiano in uno spazio ad alta dimensione. SVR è stato utilizzato per prevedere il prezzo di apertura dei mercati nel giorno successivo, sulla base dei dati storici delle serie temporali [6].

In alcuni studi, è stato dimostrato che le reti neurali ricorrenti (RNN) hanno una migliore precisione rispetto ai modelli SVR nella previsione dei prezzi delle azioni [7]. Le reti neurali ricorrenti (RNN) sono una classe di modelli di deep learning per l'elaborazione di dati sequenziali.
Nelle applicazioni pratiche, le reti neurali long short term memory (LSTM), che appartengono a RNN gated, possono apprendere dipendenze a lungo termine più facilmente rispetto alle semplici architetture ricorrenti. Le reti LSTM si sono rivelate molto efficaci per la previsione di serie temporali e sono state applicate con successo a molti campi, tra cui il riconoscimento vocale e la traduzione automatica [8].
Recentemente, LSTM ha attratto molti studiosi nell'utilizzo per prevedere i prezzi delle azioni. Nel 2016 è stato applicato il paragraph vector per ottenere rappresentazioni distribuite di articoli di giornale ed è stato proposto un modello LSTM che tiene conto delle informazioni testuali per affrontare l'influenza delle serie temporali [9]. Nel 2018, il modello Conv1D-LSTM, una combinazione di rete neurale convoluzionale (CNN) e modello LSTM, è stato proposto per prevedere i prezzi delle azioni [10]. Questo modello può assorbire i punti di forza di due reti: la CNN può eseguire efficacemente l'estrazione di funzionalità e LSTM può gestire bene i dati sequenziali. Nel 2020, S.L. Lin e H.W. Huang hanno proposto un metodo che combina il deep learning con la scomposizione in modalità empirica per prevedere accuratamente le tendenze finanziarie dai dati finanziari [11], [12]. Dopo che il modello del trasformatore è stato proposto per la modellazione in sequenza [13] in ulteriori studi, sono stati proposti nuovi approcci basati sul trasformatore per affrontare il compito di previsione del movimento delle scorte [14].
La previsione dei prezzi delle azioni è un argomento di ricerca molto importante e difficile per i ricercatori del settore finanziario e accademico. Come è noto, il prezzo di un'azione è influenzato da molti fattori tra cui profili aziendali, prospettive del settore, politiche governative, comportamento degli investitori, notizie, social media, ecc. Si comprende che i prezzi delle azioni sono influenzati da molti fattori incontrollati e potrebbe essere necessario studiare il rapporto annuale sulle azioni, le prestazioni dell'azienda e così via per ottenere una previsione significativa per un'azione.
Questa ricerca mira ad utilizzare tre diversi algoritmi di reti neurali ricorrenti, una RNN pienamente connessa, una LSTM e una GRU, per prevedere i valori futuri dei prezzi di chiusura a 1 giorno, 5 giorni e 15 giorni.

2– STUDIO SULLE RETI

2.1 RECURRENT NEURAL NETWORK

Un algoritmo di apprendimento automatico è un algoritmo in grado di apprendere dai dati. Ma cosa intendiamo per apprendimento? Mitchell (1997) [15] fornisce una definizione: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E”.
L’apprendimento automatico consente di affrontare compiti troppo difficili per risolvere programmi scritti e progettati da esseri umani [16]. Da un punto di vista scientifico e filosofico, l’apprendimento automatico è interessante perché comprenderlo implica elaborare la comprensione dei principi che stanno alla base dell’intelligenza. In questa definizione della parola “task”, il processo di apprendimento stesso non è il compito. L’apprendimento è il nostro mezzo per raggiungere la capacità di svolgere il compito. Ad esempio, se si vuole che un robot sia in grado di camminare, allora il compito è camminare.
Si può programmare il robot in modo che impari a camminare, oppure è possibile tentare di scrivere direttamente un programma che specifica come camminare manualmente. Le attività di machine learning, solitamente, sono descritte in termini di come il sistema di apprendimento automatico dovrebbe elaborare un esempio [16]. Un esempio è una raccolta di funzionalità che sono state misurate quantitativamente da qualche oggetto o evento che si vuole venga elaborato dal sistema di apprendimento automatico.
Per valutare le capacità di un algoritmo di apprendimento automatico, è necesario progettare una misura quantitativa delle sue prestazioni. Solitamente questa prestazione è una misura specifica per il task T svolto dal sistema. Per task come la classificazione, la classificazione con input mancanti e la trascrizione, spesso si misura l’accuratezza del modello. La precisione è solo la proporzione di esempi per i quali il modello produce l’output corretto. Si possono anche ottenere informazioni equivalenti misurando il tasso di errore, la proporzione di esempi per i quali il modello produce un output errato. Spesso ci si riferisce al tasso di errore come alla perdita prevista di 0-1. La perdita 0-1 su un particolare esempio è 0 se è classificato correttamente e 1 se non lo è. Per attività come la stima della densità, non ha senso misurare l’accuratezza, il tasso di errore o qualsiasi altro tipo di perdita 0-1. Bisogna, invece, utilizzare una metrica di performance diversa che dia al modello un punteggio a valore continuo per ogni esempio. L’approccio più comune è riportare la probabilità logaritmica media che il modello assegna ad alcuni esempi [16]. Di solito è d’interesse quanto bene l’algoritmo di apprendimento automatico esegue dati che non ha mai visto prima, poiché questo determina quanto bene funzionerà quando distribuito nella realtà. Pertanto si valutano queste misure delle prestazioni utilizzando un insieme di dati separato dai dati utilizzati per addestrare il sistema di apprendimento automatico. Gli algoritmi di apprendimento automatico possono essere ampiamente classificati come non supervisionati o supervisionati dal tipo di esperienza che possono avere durante il processo di apprendimento. La maggior parte degli algoritmi di apprendimento può essere intesa come la possibilità di sperimentare un intero set di dati. Un dataset è una raccolta di molti esempi, a volte chiamato data points. Uno dei dataset più antichi studiati da statistici e ricercatori di machine learning è il dataset Iris (Fisher, 1936) [16]. È una raccolta di misurazioni di parti diverse di 150 piante di iris.

Ogni singola pianta corrisponde ad un esempio. Le caratteristiche all'interno di ogni esempio sono le misure di ciascuna parte della pianta: la lunghezza del sepalo, la larghezza del sepalo, la lunghezza del petalo e la larghezza del petalo. Il set di dati registra anche a quali specie apparteneva ciascuna pianta. Nel set di dati sono rappresentate tre specie diverse. Gli algoritmi di apprendimento senza supervisione sperimentano un dataset contenente molte caratteristiche, quindi apprendono proprietà utili della struttura di questo set di dati. Nel contesto del deep learning, di solito si vuole imparare l'intera distribuzione di probabilità che ha generato un set di dati, sia esplicitamente, come nella stima della densità, sia implicitamente, per compiti come la sintesi [16]. Alcuni altri algoritmi di apprendimento non supervisionato svolgono altri ruoli, come il clustering, che consiste nel dividere il set di dati in cluster di esempi simili. Gli algoritmi di apprendimento supervisionato sperimentano un set di dati contenente funzionalità, ma ogni esempio è anche associato ad un'etichetta o ad un target. Ad esempio, l'Iris dataset è annotato con le specie di ciascuna pianta di iris. Un algoritmo di apprendimento supervisionato può studiare il set di dati dell'iride e imparare a classificare le piante di iris in tre specie diverse in base alle loro misurazioni.
In poche parole, l'apprendimento non supervisionato implica l'osservazione di diversi esempi di un vettore casuale e il tentativo di apprendere implicitamente o esplicitamente la distribuzione di probabilità ***p(x)***, oppure alcune proprietà interessanti di tale distribuzione; mentre l'apprendimento supervisionato implica l'osservazione di diversi esempi di un vettore casuale ***x*** e un valore associato o vettoriale, quindi l'apprendimento della previsione da ***x***, solitamente stimando ***p(y | x)***.

2.1 A Cos'è una RNN

Una rete neurale ricorrente (RNN) è un tipo di rete neurale artificiale che utilizza dati sequenziali o dati di serie temporali. Questi algoritmi di deep learning sono comunemente usati per problemi ordinali o temporali, come la traduzione linguistica, l'elaborazione del linguaggio naturale (nlp), il riconoscimento vocale e i sottotitoli di immagini. Essi sono incorporati in applicazioni popolari come Siri, ricerca vocale e Google Translate [17].
Come le reti neurali feedforward e convoluzionali (CNN), le reti neurali ricorrenti utilizzano i dati di addestramento per apprendere. Si distinguono per la loro "memoria" in quanto prendono informazioni da input precedenti per influenzare l'input e l'output correnti.
Mentre le reti neurali profonde tradizionali presuppongono che input e output siano indipendenti l'uno dall'altro, l'output delle reti neurali ricorrenti dipende dagli elementi precedenti all'interno della sequenza. Sebbene anche gli eventi futuri sarebbero utili per determinare l'output di una determinata sequenza, le reti neurali ricorrenti unidirezionali non possono tenere conto di questi eventi nelle loro previsioni [17].
La rete neurale ricorrente della *Figura 1* e dell'equazione *(formula 1)* è universale, ovvero qualsiasi funzione calcolabile da una macchina di Turing può essere calcolata da una tale rete ricorrente di dimensione finita [18].
L'output può essere letto dalla RNN dopo un numero di passi temporali che è asintoticamente lineare nel numero di passi temporali utilizzati dalla macchina di Turing e asintoticamente lineare nella lunghezza dell'input (Siegelmann e Sontag, 1991; Siegelmann, 1995; Siegelmann e Sontag , 1995; Hyotyniemi, 1996). Le funzioni calcolabili da una macchina di Turing sono discrete, quindi questi risultati riguardano l'esatta implementazione della funzione, non le approssimazioni. La RNN, quando usata come macchina di Turing, prende una sequenza binaria come input, e le sue uscite devono essere discretizzate per fornire un binario produzione.

È possibile calcolare tutte le funzioni in questa impostazione usando un singolo RNN specifico di dimensione finita (Siegelmann e Sontag usarono 886 unità).
L'"input" della macchina di Turing è una specifica della funzione da calcolare, quindi la stessa rete che simula questa macchina è sufficiente per tutti i problemi.
Assumendo la tangente iperbolica come funzione di attivazione delle unità nascoste e assumendo che l'output sia discreto, come se la rete fosse usata per prevedere parole o caratteri, possiamo applicare l'operazione softmax come fosse di post elaborazione per ottenere un vettore $\hat{y}$ di probabilità normalizzate sull'output. La propagazione in avanti inizia con una specificazione dello stato h_0 iniziale [18].

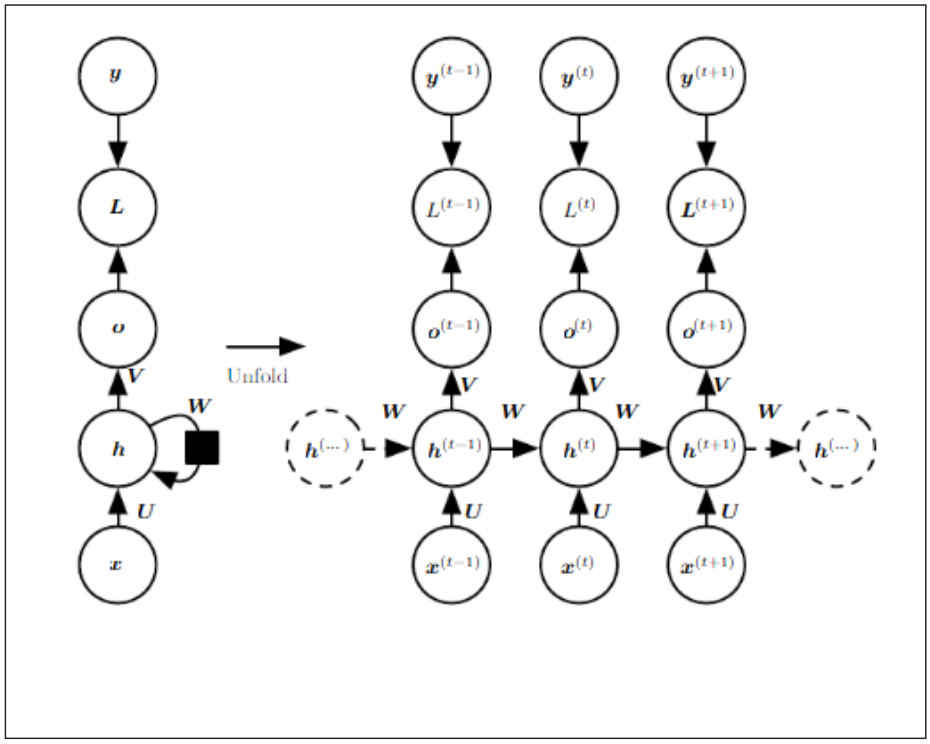

Figura 1: Il grafico computazionale per calcolare la "training loss" di una rete ricorrente che mappa una sequenza di input di valori **x** *su una sequenza corrispondente di valori di output* **o**.

Una perdita **L** *misura quanto distante è ciascun* **o** *dal target di addestramento corrispondente* **y**. *Quando si utilizza la funzione "softmax" come output, si assume che* **o** *sia la probabilità logaritmica non normalizzata. La perdita* **L** *calcola internamente* $\hat{\mathbf{y}}$*=softmax(***o***) e la confronta con il target* **y**.

La RNN ha connessioni "input-to-hidden" parametrizzate da una matrice di peso **U**, *connessioni ricorrenti "hidden-to-hidden" parametrizzate da una matrice di peso* **W** *e connessioni "hidden-to-output" parametrizzate da una matrice di peso* **V**. *La formula 1 definisce la propagazione in avanti in questo modello. A sinistra: La RNN e la sua perdita tracciata con connessioni ricorrenti. A destra: lo stesso visto come grafo computazionale "time-unfolded", in cui ogni nodo è ora associato a un'istanza di tempo particolare.*

Quindi, per ogni passo temporale da $t = 1$ a $t = \tau$, applichiamo le seguenti equazioni di aggiornamento [18]:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)} \quad \textit{(formula 1)}$$

$$h^{(t)} = tanh\left(a^{(t)}\right) \quad \textit{(formula 2)}$$

$$o^{(t)} = c + Vh^{(t)} \quad \textit{(formula 3)}$$

$$\hat{y}^{(t)} = softmax\left(o^{(t)}\right) \quad \textit{(formula 4)}$$

dove i parametri sono i vettori di bias b e c insieme alle matrici di peso U, V e W, rispettivamente, per le connessioni da input a nascosto, da nascosto a output e da nascosto a nascosto. Questo è un esempio di una rete ricorrente che associa una sequenza di input a una sequenza di output della stessa lunghezza. La perdita totale per una data sequenza di valori x accoppiati con una sequenza di valori y sarebbe quindi solo la somma delle perdite su tutti i passaggi temporali. Ad esempio, se L_t è la log-likelihood negativa di y_t dato $x_1, \ldots, x_t$, allora [18]

$$L\left(\left\{x^{(1)},\ldots,x^{(\tau)}\right\},\left\{y^{(1)},\ldots,y^{(\tau)}\right\}\right) \quad \textit{(formula 5)}$$

$$= \sum_t L^{(t)} \quad \textit{(formula 6)}$$

$$= -\sum_t log\left(p_{model}\left(y^{(t)}|\left\{x^{(1)},\ldots,x^{(t)}\right\}\right)\right) \quad \textit{(formula 7)}$$

dove $p\text{model}\ (y_t \mid \{x_1, \ldots, x_t\})$ è dato leggendo la voce per y_t dal vettore di output del modello $\hat{y}_t$. Calcolare il gradiente di questa loss-function rispetto ai parametri è un'operazione costosa. Il calcolo del gradiente implica l'esecuzione di un passaggio di propagazione in avanti che si sposta da sinistra a destra attraverso la nostra illustrazione del grafico svolto nella *Figura 2*, seguito da un passaggio di propagazione all'indietro che si sposta da destra a sinistra attraverso il grafico. Il tempo di esecuzione è O_τ e non può essere ridotto mediante la parallelizzazione perché il grafico di propagazione in avanti è intrinsecamente sequenziale; ogni passo temporale può essere calcolato solo dopo il precedente. Gli stati calcolati nel passaggio in avanti devono essere archiviati fino a quando non vengono riutilizzati durante il passaggio all'indietro, quindi anche il costo della memoria è O_τ. La rete con ricorrenza tra unità nascoste è quindi molto potente ma anche costosa da addestrare.

2.1 B Tipi di RNN

Le reti feedforward mappano un input su un output e, sebbene siano state visualizzate le reti neurali ricorrenti in questa modalità nei diagrammi sopra, in realtà non hanno questo vincolo. Invece, i loro input e output possono variare in lunghezza e diversi tipi di RNN vengono utilizzati per diversi casi d'uso, come la generazione di musica, la classificazione dei sentimenti e la traduzione automatica.
Diversi tipi di RNN sono generalmente espressi utilizzando i seguenti diagrammi [17]:

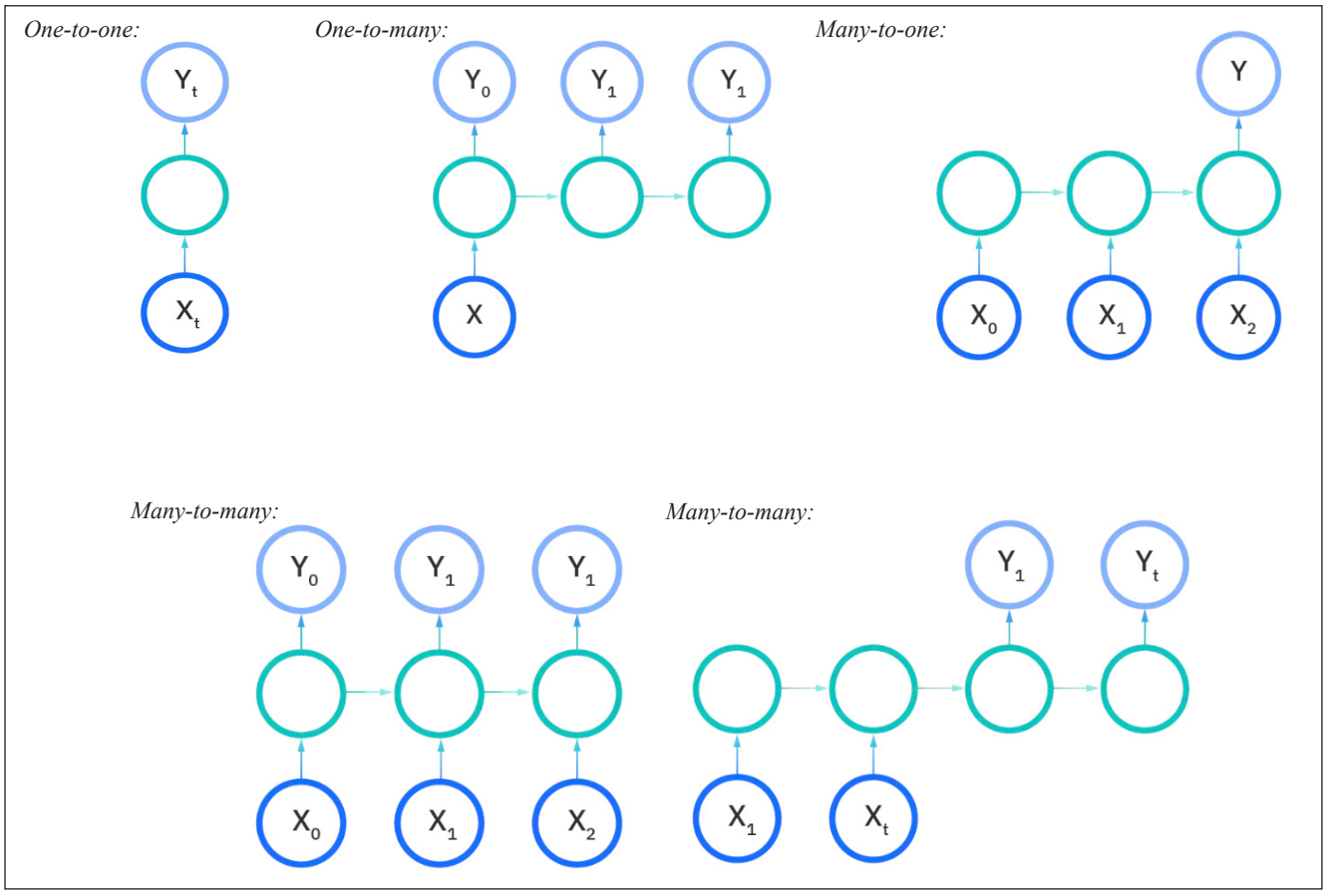

Figura 2: Varie tipologie di RNN.
One-to-one: ad ogni input corrisponde un output.
One-to-many: ad ogni input corrispondono più output.
Many-to-one: a più input corrisponde un solo output.
Many-to-many: a più input corrispondono diversi output.

2.1 C Funzioni di attivazione

Una funzione di attivazione determina se un neurone deve essere attivato. Le funzioni non lineari in genere convertono l'output di un dato neurone in un valore compreso tra 0 e 1 o -1 e 1. Alcune delle funzioni più comunemente utilizzate sono definite come segue [17]:

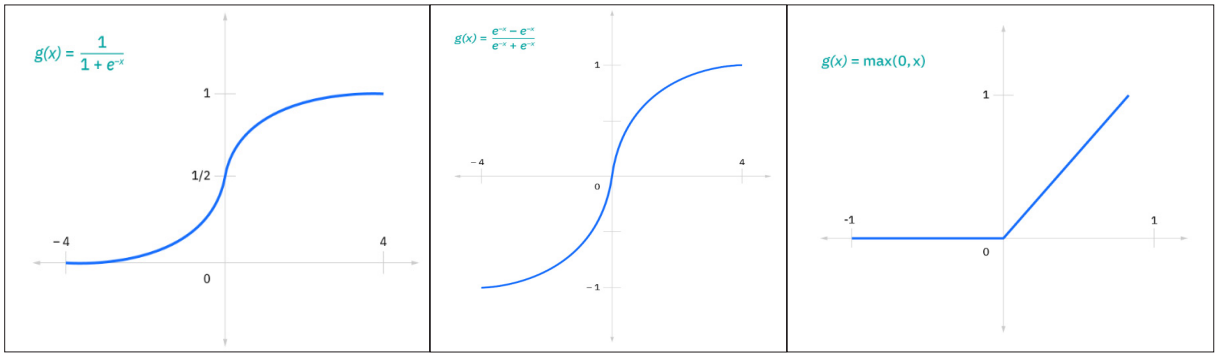

Figura 3:
Sigmoid: rappresentata con la formula $g(x) = \frac{1}{(1+e^{(-x)})}$ *(formula 8)*

Tangente iperbolica: rappresentata con la formula $g(x) = \frac{(e^{(-x)} - e^{(-x)})}{(e^{(-x)} + e^{(-x)})}$ *(formula 9)*

Relu: rappresentata con la formula $g(x) = max(0, x)$ *(formula 10)*

2.1 D Svantaggi delle RNN

Tra gli ostacoli che affrontano le RNN vi sono [19]:

• Exploding gradients:
si verificano quando l'algoritmo assegna un'importanza troppo alta ai pesi. Ciò si verifica quando i pesi vengono inizializzati a valori troppo grandi, oppure quando il tasso di apprendimento è mal calibrato, quindi la dinamica di retro-propagazione continua a moltiplicare tra di loro valori maggiori di 1. Questo problema può essere facilmente risolto troncando o schiacciando i gradienti.

• Vanishing gradients:
si verificano quando i valori di un gradiente sono troppo piccoli e il modello smette di apprendere e di conseguenza impiega troppo tempo. Questo è stato un grosso problema negli anni '90 ed è molto più difficile da risolvere rispetto agli exploding gradients. Sono riusciti a risolverlo, attraverso il concetto di LSTM, Sepp Hochreiter e Juergen Schmidhuber.

• Memoria a breve termine:
le RNN soffrono di memoria a breve termine. Se una sequenza è piuttosto lunga, avranno difficoltà a trasportare le informazioni dai passaggi temporali precedenti a quelli successivi. Quindi, se si cerca di elaborare un paragrafo di testo per fare previsioni, potrebbero tralasciare informazioni importanti dall'inizio.

2.2 LONG SHORT TERM MEMORY

Le Long Short Term Memory, di solito chiamate semplicemente "LSTM", sono un tipo speciale di RNN, in grado di apprendere dipendenze a lungo termine. Sono state introdotte da Hochreiter & Schmidhuber (1997) e sono state perfezionate e rese popolari da molti studiosi durante gli anni. Funzionano su un'ampia varietà di problemi e attualmente sono ampiamente utilizzate. Le LSTM sono esplicitamente progettate per evitare il problema della dipendenza a lungo termine, in quanto sono in grado di ricordare le informazioni per lunghi periodi di tempo [20].
Tutte le reti neurali ricorrenti hanno la forma di una catena di moduli ripetuti di rete neurale. Nelle RNN standard, questo modulo ripetuto avrà una struttura molto semplice, come un singolo strato tanh. Anche le LSTM hanno questa struttura simile a una catena, ma il modulo ripetuto ha una struttura diversa. Invece di avere un singolo *gate* nella cella, ce ne sono quattro, che interagiscono in modo specifico.

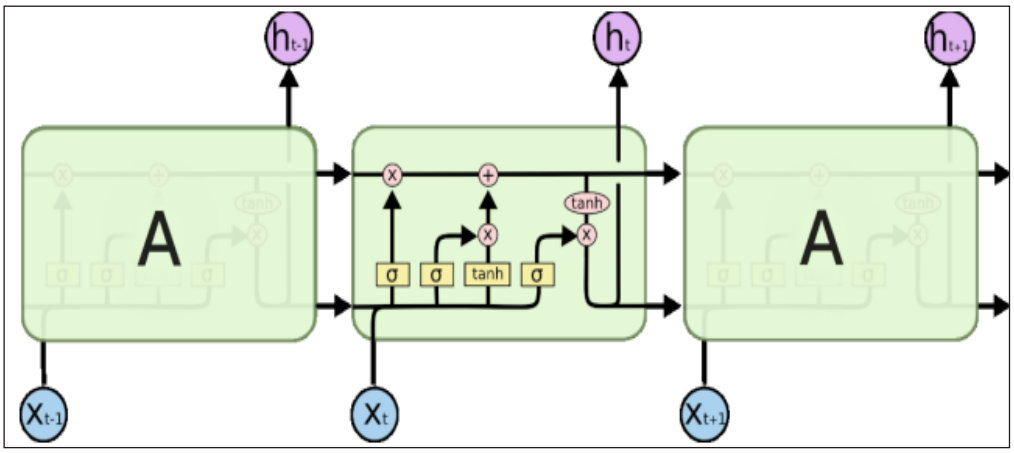

Figura 4: La ripetizione dei moduli in una LSTM contenente quattro gate interattivi.

Esaminando in maniera più dettagliata i quattro *gate* che compongono la LSTM (*Figura 5*) abbiamo [20], [21]:

- Forget gate: questo *gate* decide quali informazioni devono essere gettate o conservate. Le informazioni dallo stato nascosto precedente e le informazioni dall'input corrente vengono passate attraverso la funzione sigmoide. I valori sono compresi tra 0 e 1. Più vicino a 0 significa dimenticare e più vicino a 1 significa mantenere.

- Input gate: Per aggiornare lo stato della cella, abbiamo il *gate* di input.
Innanzitutto, passiamo lo stato nascosto precedente e l'input corrente in una funzione sigmoidea. Questo decide quali valori verranno aggiornati trasformandoli in modo che siano compresi tra 0 e 1. 0 significa non importante e 1 significa importante.
L'output del sigmoide deciderà quali informazioni è importante mantenere dall'output del tanh (change gate).

- Change gate: vengono passati lo stato nascosto e l'input corrente nella funzione tanh per ridurre i valori compresi tra -1 e 1 per aiutare a regolare la rete.

- Output gate: la porta d'uscita decide quale dovrebbe essere il prossimo stato nascosto, contenente informazioni sugli input precedenti. Lo stato nascosto viene utilizzato anche per le previsioni.

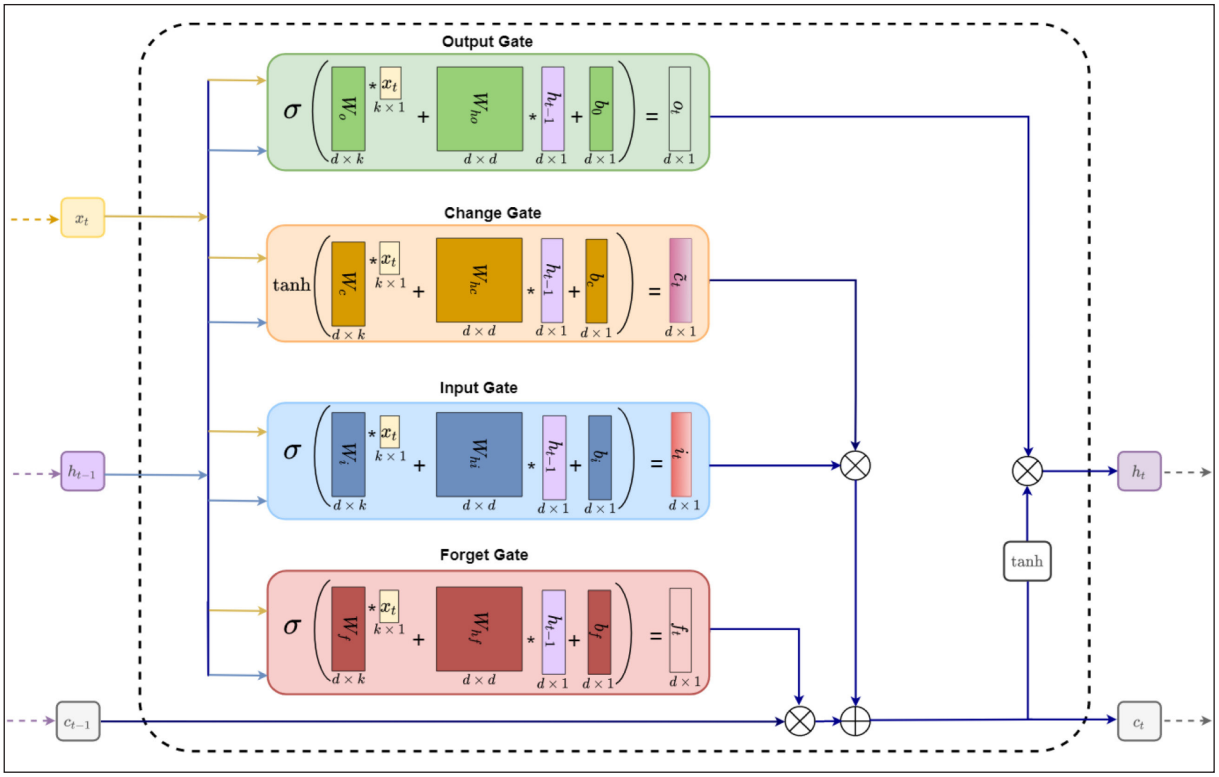

Figura 5: Dettaglio dei quattro gate della LSTM.

Per una data sequenza di input $\{x_1, x_{2,\dots}, x_n\}, x_t \in \mathbb{R}^{kx1}$ è la sequenza di input all'istante t.

La cella di memoria aggiorna le informazioni utilizzando i tre *gate*: input gate i_t, forget gate f_t e change gate $\hat{c}_t$. Lo stato nascosto h_t viene aggiornato utilizzando il *gate* di uscita o_t e la cella di memoria c_t. All'istante t, i rispettivi *gate* e strati calcolano le seguenti funzioni [20]:

$$i_t = \sigma\left(W_i x_t + W_{hi} h_{t-1} + b_i\right) \quad \textit{(formula 11)}$$

$$f_t = \sigma\left(W_f x_t + W_{hf} h_{t-1} + b_f\right) \quad \textit{(formula 12)}$$

$$o_t = \sigma\left(W_o x_t + W_{ho} h_{t-1} + b_o\right) \quad \textit{(formula 13)}$$

$$\hat{c}_t = tanh\left(W_c x_t + W_{hc} h_{t-1} + b_c\right) \quad \textit{(formula 14)}$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \hat{c}_t \quad \textit{(formula 15)}$$

$$h_t = o_t \otimes tanh\left(c_t\right) \quad \textit{(formula 16)}$$

dove σ e **tanh** rappresentano rispettivamente le funzioni sigmoide e tangente iperbolica, l'operatore $\otimes$ è il prodotto vettoriale, $W \in \mathbb{R}^{dxk}$, $W_h \in \mathbb{R}^{dxd}$ sono matrici di peso e $W \in \mathbb{R}^{dx1}$ sono vettori di bias. Inoltre, n, k, d sono rispettivamente la lunghezza della sequenza, il numero di caratteristiche e la dimensione nascosta (Greff et al., 2017, Lei et al., 2019, Qiu et al., 2020).

Riassumendo, la cella LSTM accetta 3 diverse informazioni: la sequenza di input corrente x_t, la memoria a breve termine dalla cella precedente h_{t-1} e la memoria a lungo termine dallo stato della cella precedente in quel momento c_{t-1}. La forget gate prende le informazioni da x_t e h_{t-1} e produce l'output tra 0 e 1 attraverso lo strato sigmoideo e quindi identifica quali informazioni scartare dallo stato della cella precedente c_{t-1}. Quando il valore è 1, memorizza tutte le informazioni nella cella, mentre con un valore 0 dimentica tutte le informazioni dallo stato della cella precedente. Allo stesso modo, il *gate* di input identifica quali informazioni devono essere aggiornate dal *gate* di modifica. Il *gate* di uscita decide quali informazioni devono essere prese come uscita dallo stato attuale della cella.
Le LSTM risolvono il problema del vanishing gradient e della memoria breve termine che hanno le RNN.

2.3 GATED RECURRENT UNIT

Le GRU (Gated Recurrent Unit) possono essere considerate come una variazione della LSTM perché entrambe sono progettate in modo simile e, in alcuni casi, producono risultati ugualmente eccellenti. Inoltre, come succede per le LSTM, risolvono i problemi del vanishing gradient e della memoria a breve termine [22].
Per risolvere il problema del vanishing gradient di una RNN standard, GRU utilizza i cosiddetti update gate e reset gate. Fondamentalmente, questi sono due vettori che decidono quali informazioni devono essere passate all'output. La loro caratteristica è che possono essere addestrati a conservare le informazioni di un lungo lasso di tempo, senza rimuovere informazioni che sono irrilevanti per la previsione.

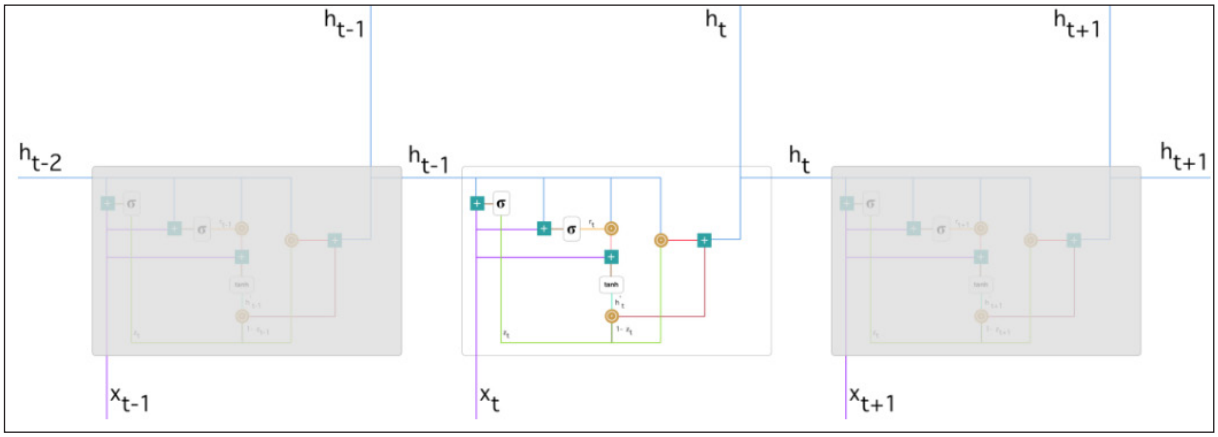

Figura 6: GRU.

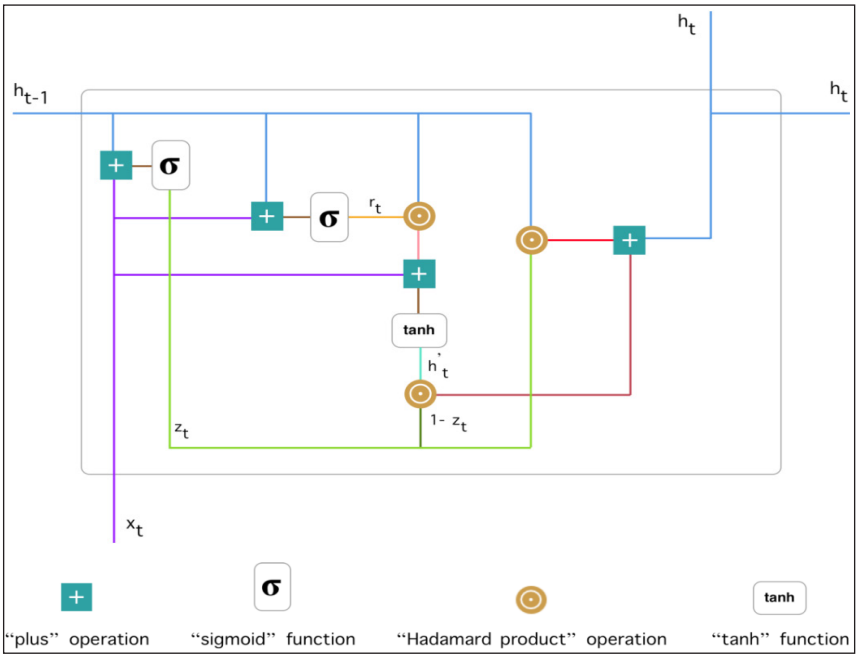

Figura 7: Dettaglio della GRU.

Esaminando in maniera più dettagliata la GRU (*Figura 7*) [22]:

- Update gate: agisce in modo simile ai forget gate e input gate di una LSTM. Decide quali informazioni buttare via e quali nuove informazioni aggiungere. Il modello può decidere di copiare tutte le informazioni dal passato ed eliminare il rischio di avere un vanishing gradient.

$$z_t = \sigma\left(W_z x_t + U_z x_{t-1}\right) \qquad \textit{(formula 17)}$$

Quando $\boldsymbol{x_t}$ è collegato all'unità di rete, viene moltiplicato per il proprio peso $\boldsymbol{W_z}$. Lo stesso vale per $\boldsymbol{h_{t-1}}$ che contiene le informazioni per le unità t-1 precedenti e viene moltiplicato per il proprio peso $\boldsymbol{U_z}$. Entrambi i risultati vengono sommati e viene applicata una funzione di attivazione sigmoidea per comprimere il risultato tra 0 e 1.

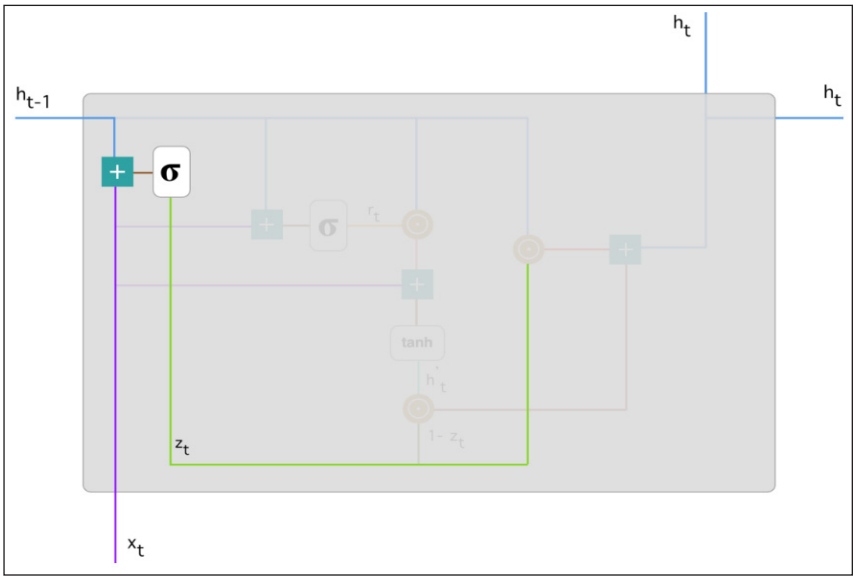

Figura 8: Update gate.

- Reset gate: viene utilizzata dal modello per decidere quante informazioni passate dimenticare.

$$r_t = \sigma\left(W_r x_t + U_r x_{t-1}\right) \qquad \textit{(formula 18)}$$

Questa formula è la stessa di quella per la update gate. La differenza sta nei pesi e nell'utilizzo del *gate*.

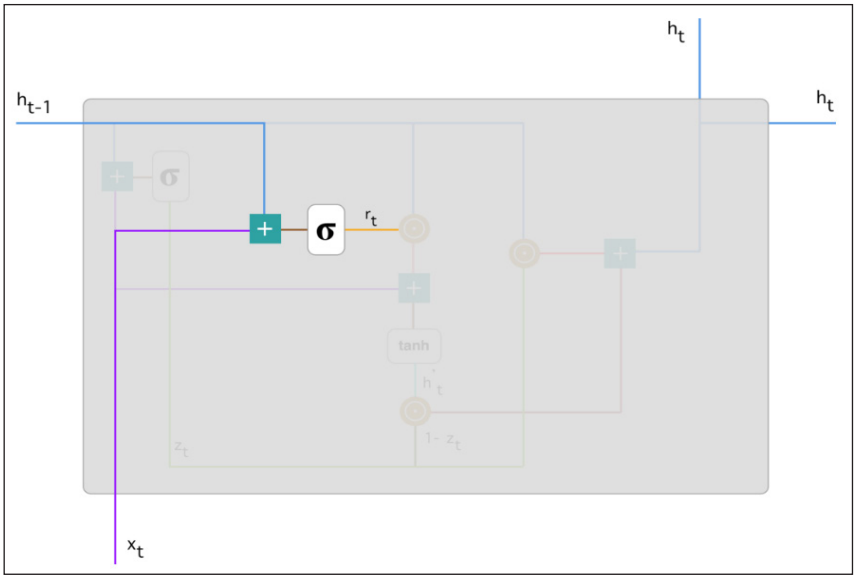

Figura 9: Reset gate.

Vengono collegate $\boldsymbol{h_{t-1}}$ (linea blu) e $\boldsymbol{x_t}$ (linea viola) e moltiplicate per i pesi corrispondenti. Vengono sommati i risultati ed applicata la funzione sigmoide.

- Contenuto della memoria attuale: decide come verrà influenzato l'output finale. Iniziando con l'utilizzo del reset gate, viene introdotto un nuovo contenuto di memoria che utilizzerà il reset gate per memorizzare le informazioni rilevanti del passato:

$$h'_t = tanh\left(Wx_t + r_t \otimes Ut_{t-1}\right) \qquad \textit{(formula 19)}$$

1 - Viene moltiplicato l'input $\boldsymbol{x_t}$ per un peso $\boldsymbol{W}$ e $\boldsymbol{h_{t-1}}$ per un peso $\boldsymbol{U}$.
2 - Si calcola il prodotto vettoriale tra il reset gate $\boldsymbol{r_t}$ e $\boldsymbol{Uh_{t-1}}$. Ciò determinerà cosa rimuovere dai passaggi temporali precedenti. Se, ad esempio, la rete deve determinare un'opinione su un libro da una recensione ed il testo inizia con "Questo libro fantasy illustra..." e dopo un paio di paragrafi termina con "Il libro non mi è piaciuto molto perché penso che argomenti troppo i dettagli", per determinare il livello di soddisfazione generale del libro serve solo l'ultima parte della recensione. In tal caso, man mano che la rete neurale si avvicina alla fine del testo, imparerà ad assegnare al vettore $\boldsymbol{r_t}$ un valore vicino a 0, cancellando il passato e concentrandosi solo sulle ultime frasi.
3 - Viene sommato il risultato dei punti 1 e 2.
4 - Si applica la funzione tanh al punto 3.

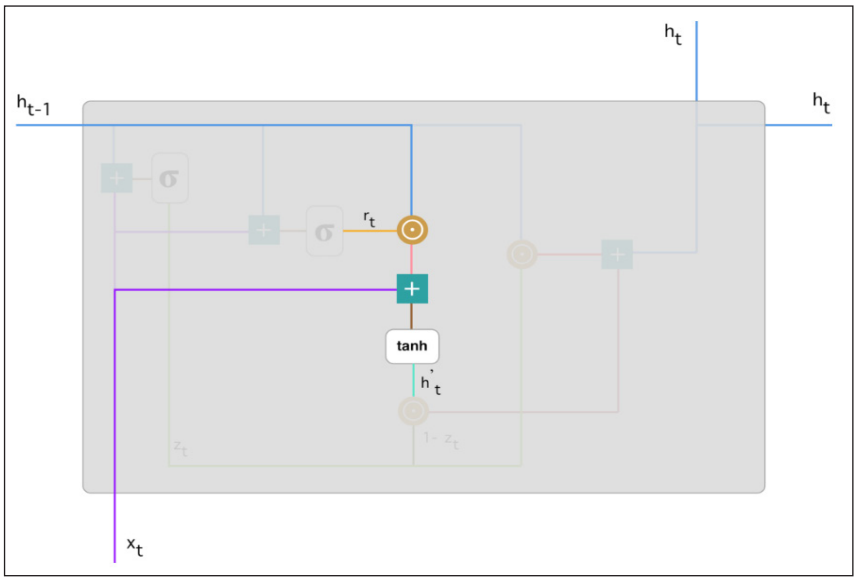

Figura 10: Current memory.

Viene fatta una moltiplicazione per elemento di h_{t-1} (linea blu) e r_t (linea arancione) e poi viene sommato il risultato (linea rosa) con l'input x_t (linea viola). Infine, tanh viene utilizzato per produrre h'_t (linea verde).

- Memoria finale: la rete deve calcolare il vettore h_t che contiene le informazioni per l'unità corrente e trasmetterle alla rete. Per fare ciò è necessario l'update gate. Si determina cosa tenere dal contenuto della memoria corrente h'_t e dai passaggi precedenti h_{t-1}:

$$h_t = z_t \otimes h_{t-1} + (1 - z_t) \otimes h'_t \qquad \textit{(formula 20)}$$

1 - Viene applicato il prodotto vettoriale tra l'update gate z_t e h_{t-1}.
2 - Si applica il prodotto vettoriale tra $(1 - z_t)$ e h'_t.
3 - Vengono sommati i risultati dei passaggi 1 e 2.
Riprendendo l'esempio della recensione del libro: le informazioni più rilevanti sono posizionate all'inizio del testo. Il modello può imparare a impostare il vettore z_t vicino a 1 e mantenere la maggior parte delle informazioni precedenti. Poiché z_t sarà vicino a 1 in questo momento, $(1 - z_t)$ sarà vicino a 0, questo vuol dire che ignorerà gran parte del contenuto corrente (in questo caso l'ultima parte della recensione che spiega la trama del libro) che è irrilevante per la nostra predizione.

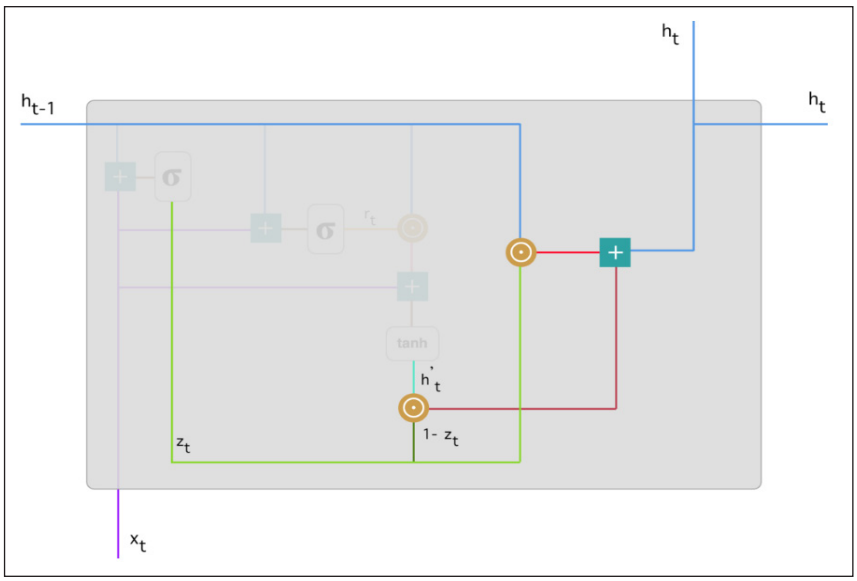

Figura 11: Final memory.

3– METODOLOGIA E DATI

L'obiettivo della ricerca è quello di valutare, per tipi diversi di serie temporali, come si comportano le tre reti prese in valutazione. Nello specifico, si studia quanto precisamente è possibile "prevedere" i prezzi che avranno le azioni domani, tra cinque giorni e tra quindici giorni.

3.1 SCELTA DEI DATI

Per la costruzione del dataset sono state scelte tre serie di dati, recuperate dalla piattaforma Kaggle, in base alla loro diversità sull'andamento dei prezzi. Questa scelta è stata fatta per valutare il comportamento delle reti neurali in funzione dell'instabilità della linea dei prezzi.

In particolare sono stati selezionati:

• Cisto System (CSCO), con un andamento di crescita costante, quasi lineare, ma caratterizzato da un picco in fase di training;

• Petrolio grezzo (CROI), con un andamento vario, caratterizzato da crescite o ribassi più o meno importanti per tutto il grafico;

• Amazon (AMZN), in cui il training viene eseguito su un andamento quasi piatto per poi esplodere in una forte crescita;

Il pattern del dataset è composto da una situazione giornaliera su prezzi e volumi d'acquisto. Più precisamente le features sono:

1. Il prezzo minimo registrato durante la giornata

2. Il prezzo di apertura della giornata

3. Il volume di acquisto/vendita giornaliero

4. Il prezzo massimo registrato durante la giornata

5. Il prezzo di chiusura della giornata

Il dataset riguarda uno storico di circa 20 anni, quindi intorno a 5000 righe, che verranno divise tra training, validation e test sets.

3.2 PREPARAZIONE DEL DATASET

Si possono distinguere le varie fasi di elaborazione per l'adattamento dei dati alle reti:
- Trasformazione da csv in array: vengono estratte le varie rige del csv e copiate all'interno di un numpy.array. Le righe sono composte dalle 5 features discusse appena sopra.
- Normalizzazione del dataset: le reti neurali ricorrenti che vengono utilizzate sono molto sensibili alle variazioni sui nodi, soprattutto le LSTM e le GRU, per cui la normalizzazione è un passaggio obbligatiorio per ottenere dei buoni risultati.
È stata scelta una normalizzazione tra 0 e 1 in quanto dà più stabilità alle reti ricorrenti.
- Suddivisione del dataset per il train e test set: vengono utilizzati un 80% dei dati per la fase di training e il rimanente 20% per la fase di test. Questi dati vengono presi sequenzialmente, per natura delle RNN, quindi il test è basato sui prezzi degli ultimi anni.
È stata decisa questa suddivisione anche se non sempre è corretta. Se negli ultimi anni ci fosse stato un "distribution shift" [23] sarebbe un compito impossibile per la rete apprendere statistiche sulla distribuzione. Il problema si può affrontare suddividendo i dati in modo casuale e introducendo tecniche più avanzate di monitoraggio degli shift.
- Creazione del training set: per avere uno storico tra i dati è stato creato, per ogni riga, un array contenente i precedenti 160 valori sui prezzi, associando come label il prezzo di chiusura registrato n giorni nel futuro, che per i test eseguiti saranno 1, 5 o 15 giorni. Quindi sono stati presi i prezzi giornalieri degli ultimi 7 mesi per capire come chiuderà il prezzo tra, ad esempio, 15 giorni.

3.3 PREPARAZIONE DELLA RETE

Per le reti neurali sono state scelte delle configurazioni in risposta ai test eseguiti per rispettare un'ottimale taratura. Il numero di nodi utilizzato per ogni layer viene calcolato tramite una funzione che ci aiuta a prevenire l'overfitting durante l'apprendimento:

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))} \qquad \textit{(formula 21)}$$

N_i = numero dei neuroni di input
N_0 = numero dei neuroni di output
N_s = numero degli elementi nel training data set
α = fattore di scala arbitrario, solitamente tra 2-10

Per le RNN, essendo reti interamente connesse, è stato utilizzato un singolo hidden layer. Utilizzarne di più comportava una rete troppo grande, difficile da addestrare, che overfittava dopo poche epoch.
Il layer iniziale e quello hidden sono seguiti da uno dropout per stabilizzarne i pesi.

Layer (type)	Output Shape	Param #
simple_rnn_2 (SimpleRNN)	(None, 160, 169)	29575
dropout_2 (Dropout)	(None, 160, 169)	0
simple_rnn_3 (SimpleRNN)	(None, 169)	57291
dropout_3 (Dropout)	(None, 169)	0
dense_1 (Dense)	(None, 1)	170

Total params: 87,036
Trainable params: 87,036
Non-trainable params: 0

Figura 12: Configurazione RNN

Per le LSTM e per le GRU ne sono stati invece utilizzati tre, come prima, ognuno dei quali seguito da un layer droput per la stabilizzazione dei pesi.

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 160, 84)	30240
dropout (Dropout)	(None, 160, 84)	0
lstm_1 (LSTM)	(None, 160, 84)	56784
dropout_1 (Dropout)	(None, 160, 84)	0
lstm_2 (LSTM)	(None, 160, 84)	56784
dropout_2 (Dropout)	(None, 160, 84)	0
lstm_3 (LSTM)	(None, 84)	56784
dropout_3 (Dropout)	(None, 84)	0
dense (Dense)	(None, 1)	85

Total params: 200,677
Trainable params: 200,677
Non-trainable params: 0

Figura 13: Configurazione LSTM

Layer (type)	Output Shape	Param #
gru_4 (GRU)	(None, 160, 84)	22932
dropout_4 (Dropout)	(None, 160, 84)	0
gru_5 (GRU)	(None, 160, 84)	42840
dropout_5 (Dropout)	(None, 160, 84)	0
gru_6 (GRU)	(None, 160, 84)	42840
dropout_6 (Dropout)	(None, 160, 84)	0
gru_7 (GRU)	(None, 84)	42840
dropout_7 (Dropout)	(None, 84)	0
dense_1 (Dense)	(None, 1)	85

Total params: 151,537
Trainable params: 151,537
Non-trainable params: 0

Figura 14: Configurazione GRU

Il livello di output è composto da uno strato dense e come funzione di attivazione una RELU.

3.4 CALCOLO DELL'ERRORE

Dopo aver addestrato le reti sul training set e averle utilizzate sul test set viene calcolato l'errore quadratico medio su entrambi i dataset.

$$RMSE_{f_o} = \left[\frac{\sum_{i=1}^{N} \left(z_{f_i} - z_{o_i} \right)^2}{N} \right]^{\frac{1}{2}} \qquad \textit{(formula 22)}$$

Oltre al RMSE, per stimare l'errore ottenuto vengono discussi anche i grafici dell'errore (mean squared error) del training set e del validation set, fondamentali per capire se la rete ha imparato e quanto ha imparato, e i grafici dei prezzi di chiusura reali e predetti.

4– RISULTATI E DISCUSSIONE

4.1 CSCO

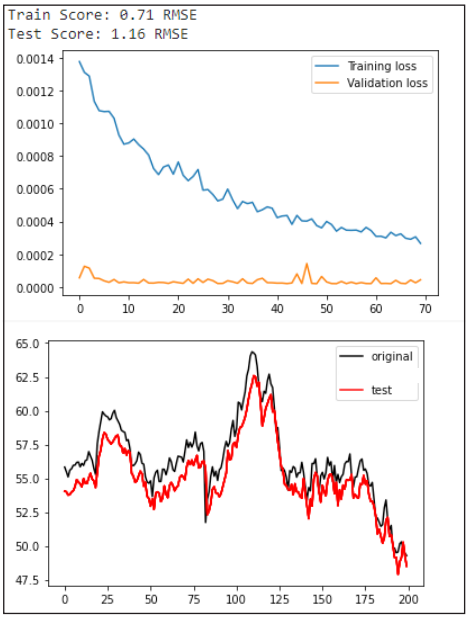

Figura 15: predizione di 1 giorno di CSCO con RNN.

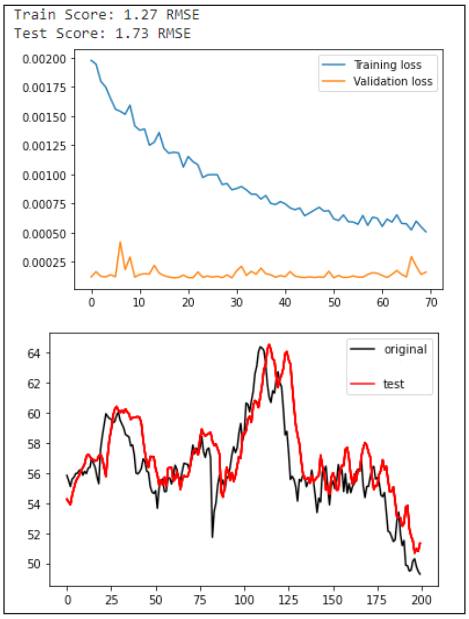

Figura 16: predizione di 5 giorni di CSCO con RNN.

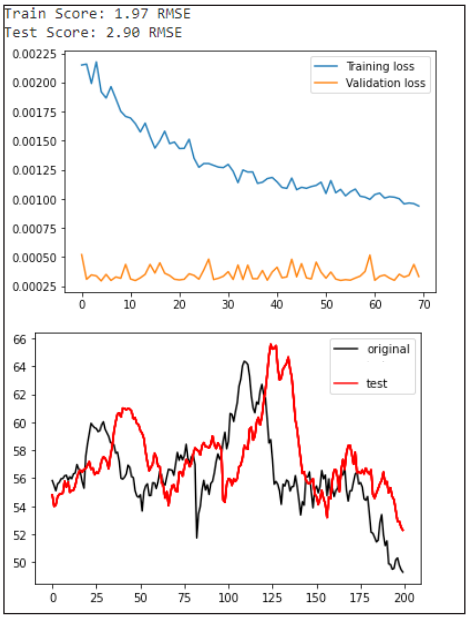

Figura 17: predizione di 15 giorni di CSCO con RNN.

Viene analizzato il comportamento della RNN sulla serie temporale dei prezzi di Cisco Systems su una previsione di 1, 5, 15 giorni.
Dalla forma degli errori si può notare una curva del training loss che diminuisce in modo quasi lineare, mentre il validation loss rimane quasi costante per tutto il tempo.
Si comprende quindi che la rete, già dall'inizio, sta andando verso l'overfit.
Dalla comparazione tra la serie temporale ottenuta dal test e la serie originale si può vedere, invece, che i prezzi si avvicinano molto a quelli reali, ma con un ritardo in giorni dello stesso periodo scelto per la previsione.

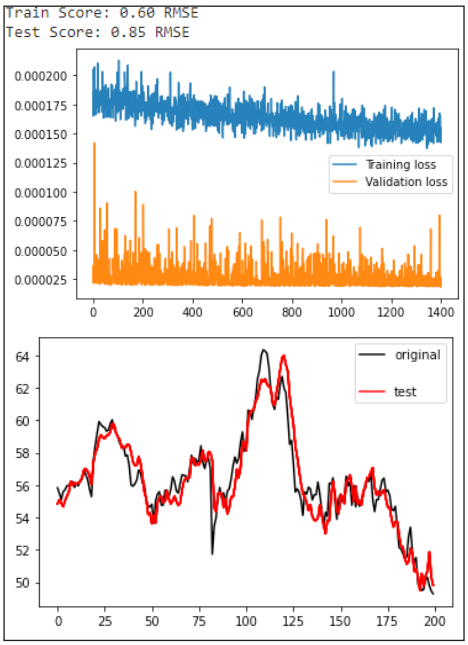

Figura 18: predizione di 1 giorno di CSCO con LSTM.

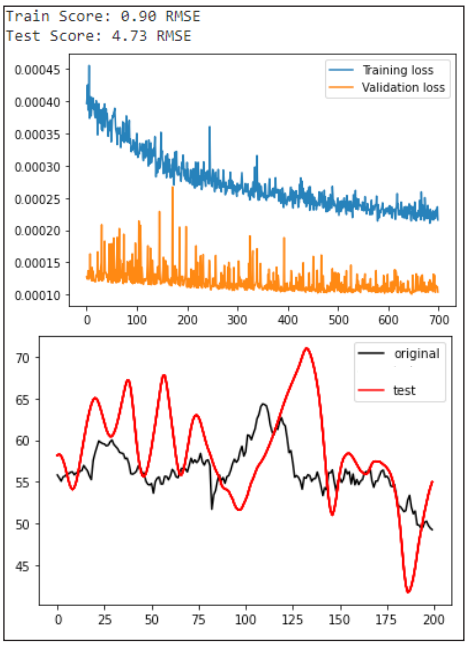

Figura 19: predizione di 5 giorni di CSCO con LSTM.

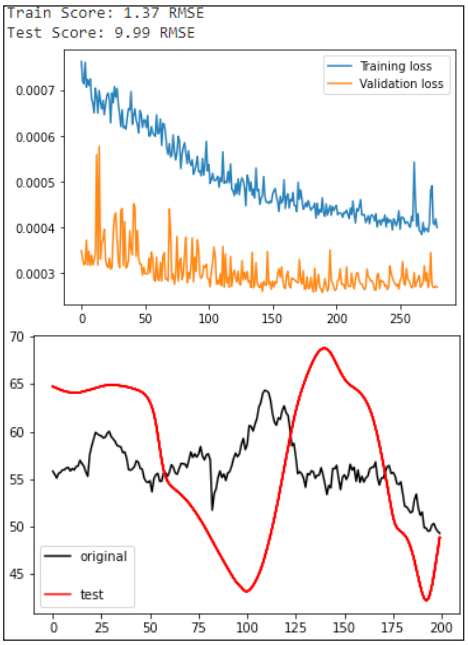

Figura 20: predizione di 15 giorni di CSCO con LSTM.

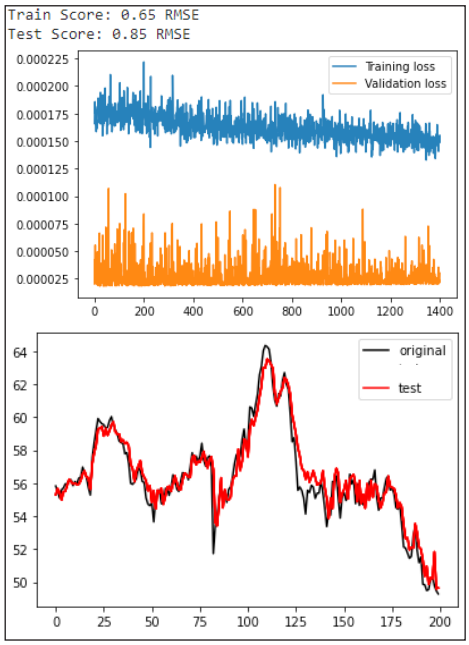

Figura 21: predizione di 1 giorno di CSCO con GRU.

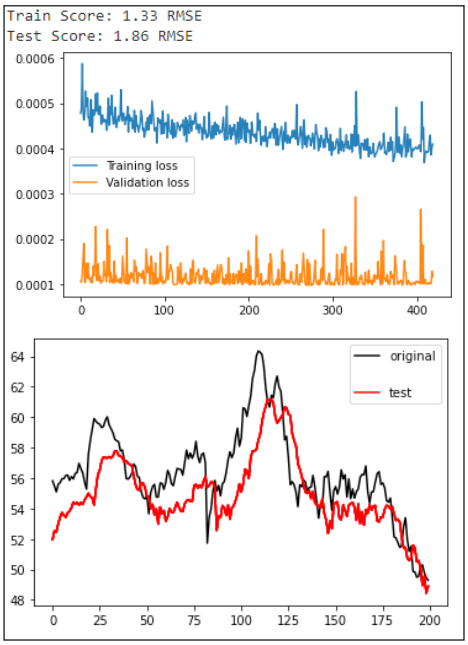

Figura 22: predizione di 5 giorni di CSCO con GRU.

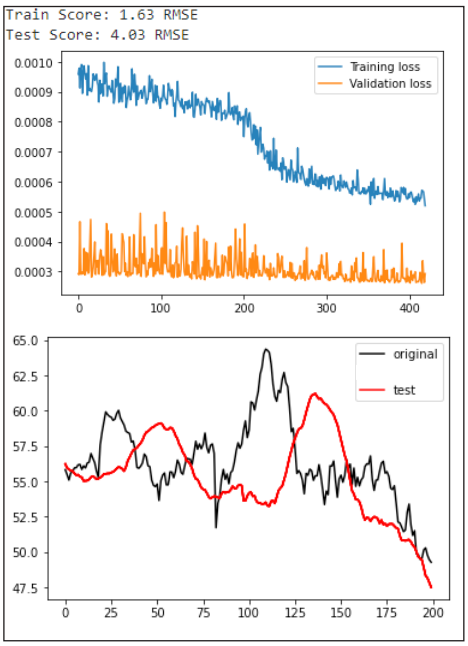

Figura 23: predizione di 15 giorni di CSCO con GRU.

Con LSTM e GRU si riescono ad avere risultati più precisi e quindi un avvicinamento ai prezzi migliore. Rimangono però le stesse considerazioni fatte per le RNN.

4.2 CROI

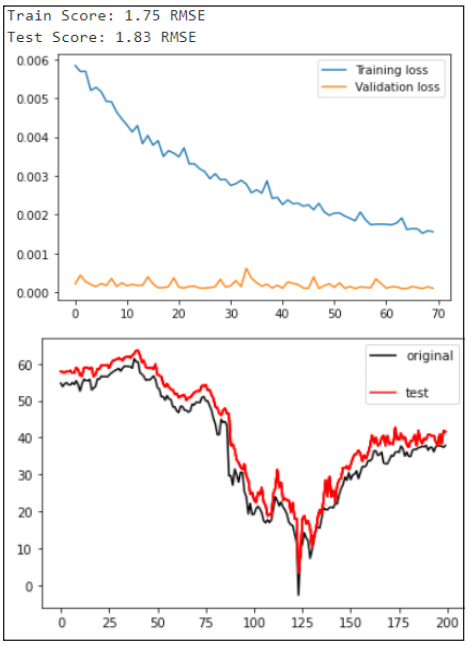

Figura 24: predizione di 1 giorno del CROI con RNN.

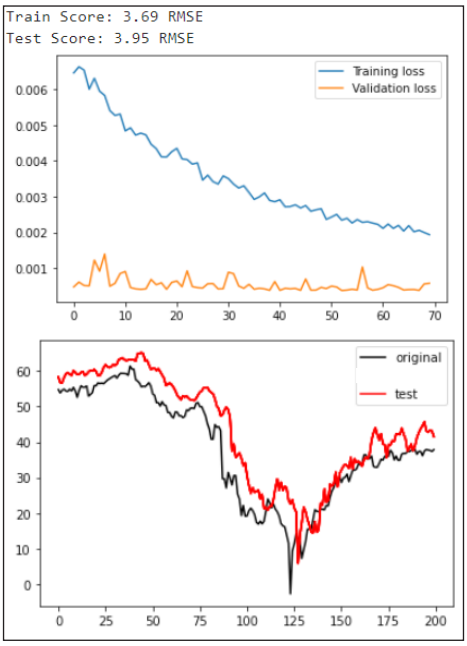

Figura 25: predizione di 5 giorni del CROI con RNN.

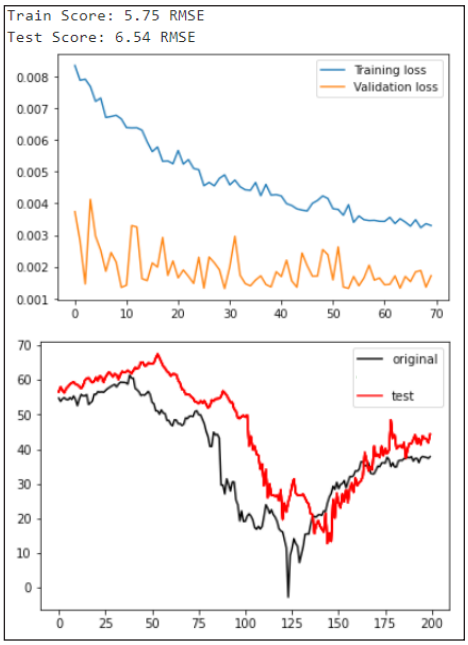

Figura 26: predizione di 15 giorni del CROI con RNN.

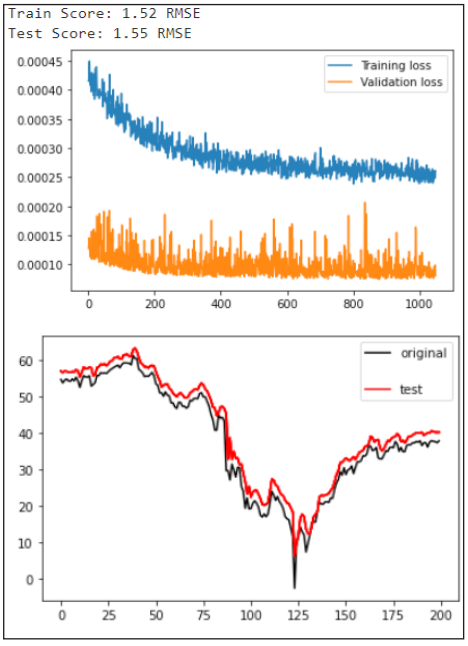

Figura 27: predizione di 1 giorno del CROI con LSTM.

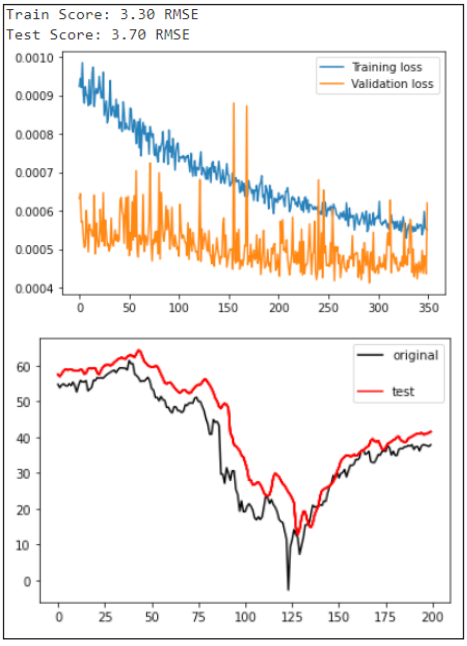

Figura 28: predizione di 5 giorni del CROI con LSTM.

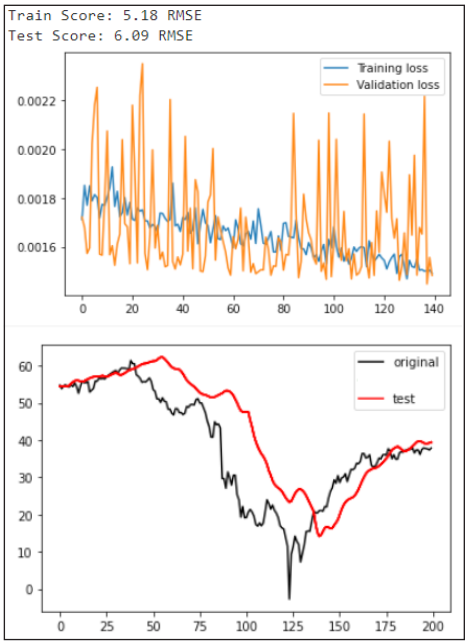

Figura 29: predizione di 15 giorni del CROI con LSTM.

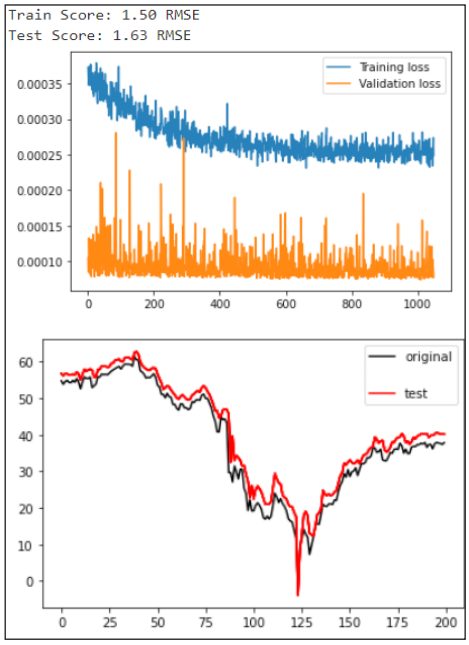

Figura 30: predizione di 1 giorno del CROI con GRU.

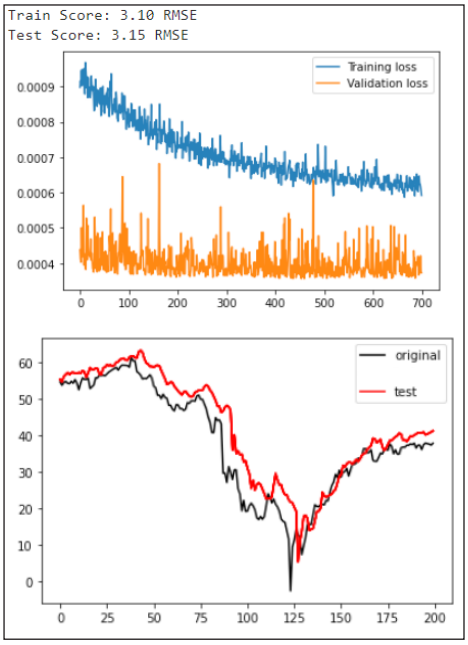

Figura 31: predizione di 5 giorni del CROI con GRU.

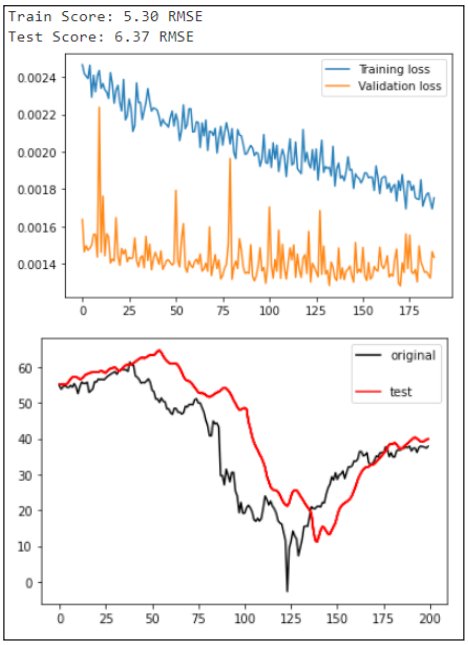

Figura 32: predizione di 15 giorni del CROI con GRU.

Con il CROI abbiamo una situazione simile al CSCO.

4.3 AMZN

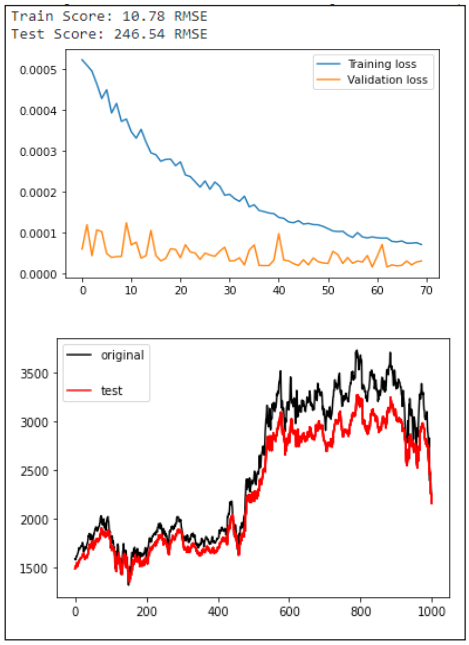

Figura 33: predizione di 1 giorno di AMZN con RNN.

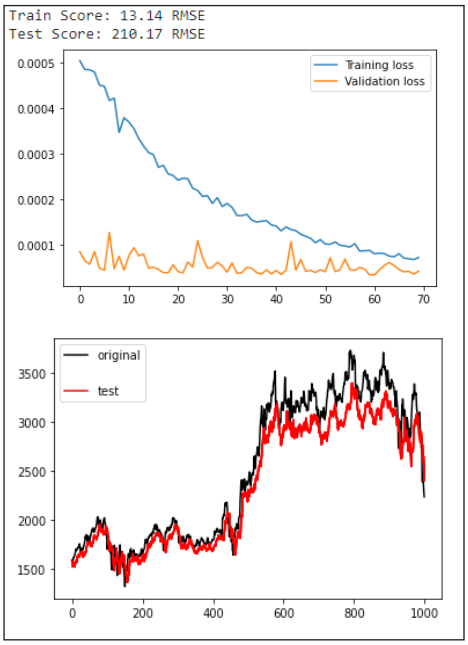

Figura 34: predizione di 5 giorni di AMZN con RNN.

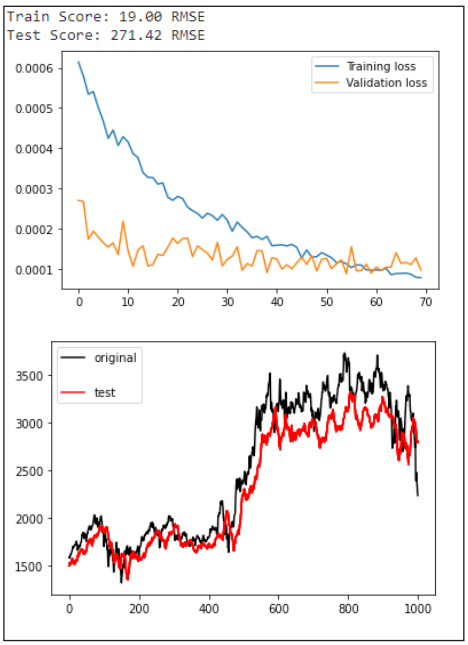

Figura 35: predizione di 15 giorni di AMZN con RNN.

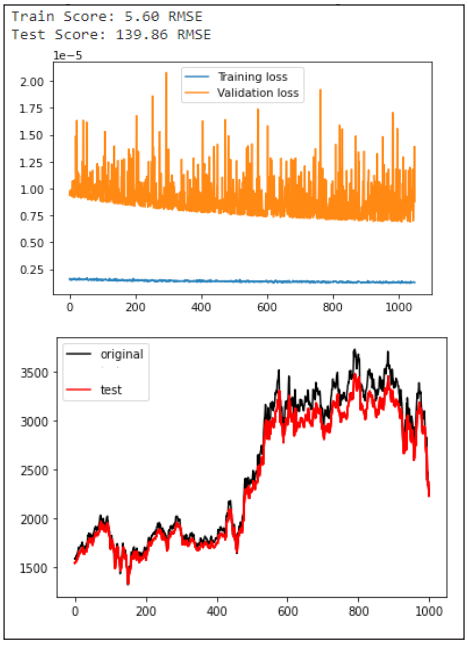

Figura 36: predizione di 1 giorno di AMZN con LSTM.

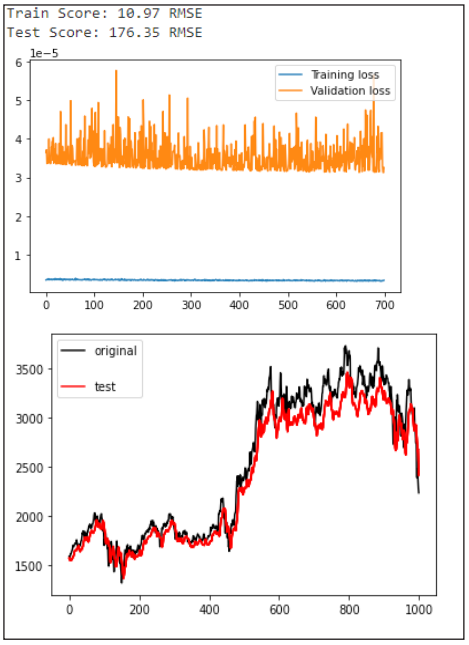

Figura 37: predizione di 5 giorni di AMZN con LSTM.

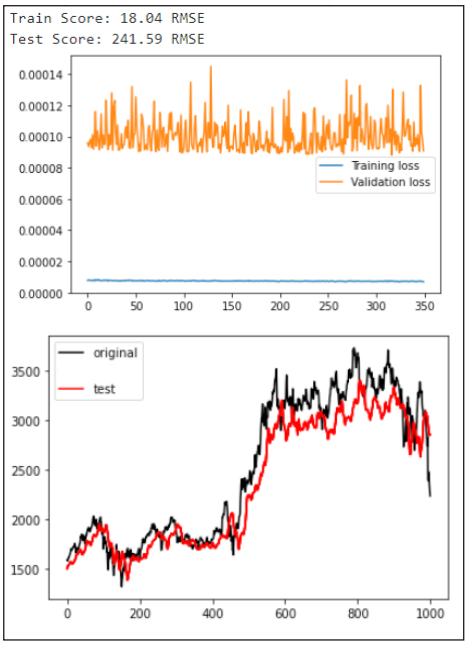

Figura 38: predizione di 15 giorni di AMZN con LSTM.

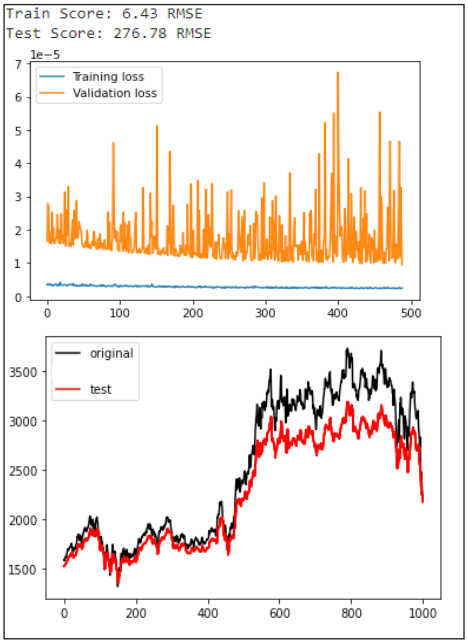

Figura 39: predizione di 1 giorno di AMZN con GRU.

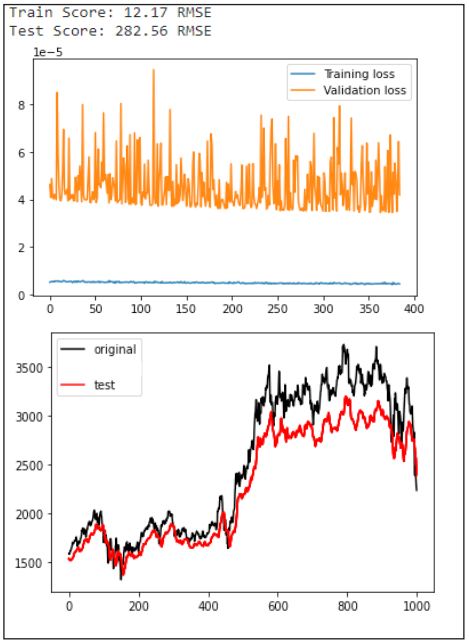

Figura 40: predizione di 5 giorni di AMZN con GRU.

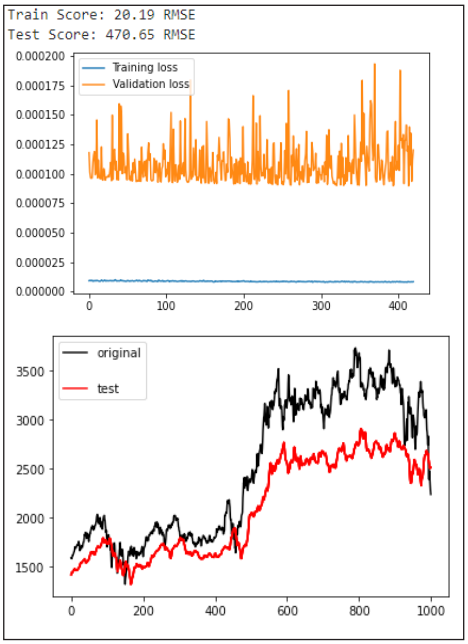

Figura 41: predizione di 15 giorni di AMZN con GRU.

Con AMZN si nota invece che l'addestramento della rete non riesce a stare al passo con il forte rialzo dei prezzi in fase di test, questo porta ad avere un grosso errore.

5– CONCLUSIONI

La previsione del prezzo delle azioni e, più in generale, la previsione per i vari mercati è un'area di grande interesse per i trader azionari, gli investitori individuali e i gestori di portafoglio. Tuttavia, una previsione precisa e coerente del prezzo delle azioni è un compito difficile da effettuare a causa dell'influenza che questi percepiscono da eventi esterni casuali, come i dati fondamentali del mercato, i dati macroeconomici, gli indicatori tecnici e altri. Questo studio si concentra sullo sviluppo di modelli basati su reti neurali profonde ricorrenti. Si sono considerate le reti neaurali ricorrenti semplici, long short term memory e gated recurrent unit per prevedere il prezzo di chiusura di indici come Cisco System, il prezzo del petrolio grezzo e Amazon, sulla base dei prezzi e dei volumi di acquisto.
I risultati sperimentali mostrano che i modelli LSTM e GRU sono più precisi di modelli RNN semplici ma si dimostrano comunque inadatti alla previsione futura, in quanto la rete impara a seguire l'andamento dei prezzi e non il cambio di trend, poiché i pattern di continuazione sono di gran lunga maggiori di quelli di cambio.
Per poter prevedere in modo accurato il futuro di un prezzo bisognerebbe conoscere in anticipo diversi dati come le trimestrali delle aziende o gli investimenti di tutte le persone nel mondo in un determinato mercato. Sarebbe necessario, addirittura, presagire l'uscita di una notizia o di un twitter da parte di qualche persona importante. Ad esempio, un caso rilevante è avvenuto con il bitcoin dopo l'uscita del twitt di Elon Musk, in cui dichiarava il suo personale appoggio alla moneta virtuale facendone crescere il valore da 30 mila a 60 mila dollari. Sempre lui, durante il suo negoziato per l'acquisto di Twitter, affermando che la trattativa non sarebbe continuata, ha fatto perdere il 25% del valore di mercato a Twitter in un giorno.

6- BIBLIOGRAFIA

[1] - Q. Chen, M. Tao, X. He, and L. Tao, “Fuzzy adaptive nonsingular fixed-time attitude tracking control of quadrotor UAVs,” IEEE Transactions on Aerospace and Electronic Systems, vol. 57, no. 5, pp. 2864–2877, 2021.

[2] - Q. Chen, Y. Ye, Z. Hu, J. Na, and S. Wang, “Finite-time approximation-free attitude control of quadrotors: theory and experiments,” IEEE Transactions on Aerospace and Electronic Systems, vol. 57, no. 3, pp. 1780–1792.

[3] - C. Wei, Z. Zhang, W. Qiao, and L. Qu, “Reinforcement-learning-based intelligent maximum power point tracking control for wind energy conversion systems,” IEEE Transactions on Industrial Electronics, vol. 62, no. 10, pp. 6360–6370, 2015.

[4] - C. Wei, Z. Zhang, W. Qiao, and L. Qu, “An adaptive network-based reinforcement learning method for MPPT control of PMSG wind energy conversion systems,” IEEE Transactions on Power Electronics, vol. 31, no. 11, pp. 7837–7848, 2016.

[5] - C. Wei, Y. Zhao, Y. Zheng, L. Xie, and K. Smedley, “Analysis and design of a non-isolated high step-down converter with coupled inductor and ZVS operation,” IEEE Transactions on Industrial Electronics, p. 1, 2021.

[6] - Y. Xia, Y. Liu, and Z. Chen, “Support Vector Regression for prediction of stock trend,” in Proceedings of the 2013 6th International Conference on Information Management, Innovation Management and Industrial Engineering, vol. 123-126, IEEE, Piscataway, NJ, USA, 2013.

[7] - Y. Liu, Z. Qin, P. Li, and T. Wan, “Stock volatility prediction using recurrent neural networks with sentiment analysis,” in Advances in Artificial Intelligence: From Theory to Practice, S. Benferhat, K. Tabia, and M. Ali, Eds., Springer, Berlin, Germany, 2017.

[8] - I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, MIT Press, Cambridge, UK, 2016.

[9] - R. Akita, A. Yoshihara, and T. Matsubara, “Deep learning for stock prediction using numerical and textual information,” in Proceedings of the 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS), IEEE, Okayama, Japan, June 2016.

[10] - S. Jain, R. Gupta, and A. Moghe, “Stock price prediction on daily stock data using deep neural networks,” in Proceedings of the 2018 International Conference on Advanced Computation and Telecommunication (ICACAT), Bhopal, India, December 2018.

[11] - T. B. Shahi, A. Shrestha, A. Neupane, W. Guo, and S. Price, “Stock price forecasting with deep learning: a comparative study,” Mathematics, vol. 8, no. 9, p. 1441, 2020.

[12] - S.-L. Lin and H.-W. Huang, “Improving deep learning for forecasting accuracy in financial data,” Discrete Dynamics in Nature and Society, vol. 2020, pp. 1–12, 2020.

[13] - A. Vaswani, “Attention is all you need,” in Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 2017.

[14] - Q. Ding, S. Wu, H. Sun, J. Guo, and J. Guo, “Hierarchical multi-scale Gaussian transformer for stock movement prediction,” in Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence Special Track on AI in FinTech, pp. 4640–4646, Yokohama, Japan, January 2020.

[15] - S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] - Ian Goodfellow, Yoshua Bengio, Aaron Courville, “Deep Learning, An MIT Press Book”, Chapter 1, pp. 96-103.
Available: https://www.deeplearnignbook.org/contents/intro.html

[17] - “Recurrent Neural Networks”, IBM Cloud Education, 2020, Settembre 14.
Available: https://www.ibm.com/cloud/learn/recurrent-neural-networks

[18] - Ian Goodfellow, Yoshua Bengio, Aaron Courville, “Deep Learning, An MIT Press Book”, Chapter 10, pp. 372-375.
Available: https://www.deeplearnignbook.org/contents/rnn.html

[19] - Simeon Kostadinov, “How Recurrent Neural Networks work”, 2017, Dicembre 2.
Available: https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7

[20] - Christopher Olah, “Understanding LSTM Networks”, 2015, Agosto 27.
Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[21] - Michael Phi, “Illustrated Guide to LSTM’s and GRU’s: A step by step explanation”, 24 settembre, 2018.
Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[22] - Simeon Kostadinov, “Understanding GRU Networks”, 2017, Dicembre 16.
Available: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

[23] - A. Zhang, Z. C. Lipton, Mu Li, A. J. Smola, “Dive into Deep Learning” Preview Version, Chapter 4.7.
Available: https://d2l.ai/chapter_linear-classification/environment-and-distribution-shift.html