Università degli Studi di Milano Bicocca

Scuola di Scienze

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di laurea in Informatica

Previsione della volatilità con Reti Neurali Ricorrenti per il trading di opzioni

Relatore: *Antonio Candelieri*

Co-relatore: *Silvio Maria Enrico Bencini*

Relazione della prova finale di:

Luca Poli

Matricola 852027

Anno Accademico 2021-2022

Indice

Introduzione

Il Machine Learning è adottato in un numero sempre crescente di domini applicativi; uno in cui ha riscontrato particolare successo è senza dubbio la finanza, e nello specifico i mercati finanziari. Qui è stato applicato in svariati modi, dalla gestione di portafoglio all'High Frequency trading, con l'utilizzo di modelli e tecniche sempre più innovative; specialmente negli ultimi decenni, in cui le risorse computazionali e i dati non sono più un grosso ostacolo. Le Reti Neurali, in particolare, hanno ricevuto un rinnovato interesse; grazie alla riduzione dei problemi sopracitati, è infatti possibile sfruttare la loro potenza e versatilità. Inoltre, si stanno sviluppando sempre più varianti; come le *Recurrent Neural Network* (*RNN*), specializzate per lavorare con sequenze di dati (come le serie temporali che compongono la maggior parte dei dati finanziari).

L'obbiettivo della tesi è dunque realizzare un sistema predittivo; basato su reti *Long Short Term Memory* (*LSTM*), una particolare classe di RNN, che sia in grado di anticipare la volatilità del prezzo di un'azione. Infine, si vuole fare profitto da tale previsione, con un sistema di trading a breve termine basato sulle opzioni. L'elaborato si pone come un'introduzione pratica; cioè prima affronterà i vasti concetti teorici, per poi proporre una possibile soluzione.

Il primo capitolo sarà diviso in tre sezioni: nella prima introdurrò il Machine Learning; nella seconda tratterò le Reti Neurali, passando da quelle più semplici a quelle più complesse, con particolare riguardo ai problemi e alle relative soluzioni, terminando poi con un approfondimento sulla versione Ricorrente; infine, nell'ultima parte affronterò il tema delle opzioni e della volatilità.

Il secondo capitolo è una descrizione generale del progetto: prima descriverò i dati coinvolti e le relative elaborazioni; poi definirò i modelli di Machine Learning adottati e confrontati; successivamente delineerò le strategie adottate dal sistema di trading; infine procederò con la presentazione dei risultati.

Capitolo 1: I concetti teorici

In questo capitolo si introducono i concetti teorici alla base del progetto, sia dal lato del ML e delle reti Neurali, che dal lato finanziario.

1.1 Brevi cenni al Machine Learning (ML)

Seguendo la definizione proposta in [1], definiamo il Machine Learning come una serie di tecniche e algoritmi che permettono la risoluzione automatizzata di problemi complessi, i quali sarebbero difficili da risolvere con il convenzionale approccio della programmazione. Tale sistema, nei suoi aspetti più generali, parte da una specifica che definisce il problema; cioè, in termini di software il compito finale del programma. Successivamente progetta e poi implementa l'insieme di regole che compone la soluzione specifica. Però questo approccio risulta molto difficile, nella pratica quasi impossibile, da applicare in situazioni dove la specifica è chiara, ma l'insieme di regole da definire no; qui entra in gioco il ML che permette al calcolatore di apprendere dalla specifica stessa il lungo, ed esaustivo, insieme di criteri che compongono la soluzione. Esso risolve il problema in modo indiretto: crea e aggiorna un modello sulla base dei dati di input, apprendendo regole e pattern comuni; e usa tale per produrre la soluzione al problema.

Un esempio è la rilevazione di caratteri testuali in immagini. Supponiamo di avere un dataset contenente delle immagini di esempio, ovvero con la relativa soluzione (in gergo detta *label*); l'informatico dovrebbe indagare tali immagini, comprenderne i fattori in comune e infine costruire una procedura generica che funzioni su nuove immagini. Per quanto la specifica sia chiara, il set generico di regole è complicato da stabilire. Dunque, applicare l'approccio tradizione risulterebbe complesso, perciò possiamo usare delle tecniche di ML.

Per comprendere a pieno il funzionamento del ML, è bene approfondire i problemi che risolve e i meccanismi dei suoi modelli. I problemi risolti, come proposto in [2], rientrano in diverse macrocategorie:

- *Classification*: assegnamento, ad ogni elemento, di una o più categorie di apparenza;
- *Clustering*: separazione di un grande insieme di dati in piccoli sottoinsiemi, di cui il numero e le caratteristiche non sono definite a priori;
- *Regression*: deduzione di un singolo valore reale, a partire da un insieme di informazioni. Si differenzia dalla classificazione, perché è presente la nozione di vicinanza tra due valori, che non è sempre presente tra due categorie;
- *Ranking*: ordinamento di elementi sulla base di alcuni criteri, per esempio l'ordinamento dei siti web più pertinenti ad una ricerca;
- *Dimensionality reduction*: trasformare la rappresentazione di un elemento, in una rappresentazione più piccola.

(Da notare che tale separazione è puramente formale e spesso le aree collassano in categorie più ampie o più ristrette)

Ogni modello di ML è caratterizzato da due categorie di variabili interne:

- Gli iperparametri, le cui quantità e ruoli variano a seconda del tipo; corrispondono ai settaggi del modello e perciò governano tutte le dinamiche interne dello stesso; vengono stabiliti a priori dal progettista.
- I parametri, al contrario, variano durante il tempo di vita del modello, a seconda degli iperparametri vengono definiti usati e modificati.

I modelli, prima di poter risolvere il problema, devono essere addestrati; ovvero devono apprendere i pattern e le regole, variando continuamente i loro parametri; al fine di risolvere il problema con una certa affidabilità. Tale stato è definito come fase di training, precede (alcune volte coincide) quella di funzionamento stessa, e può impiegare una grande quantità di dati, tempo e risorse computazionali. A seconda di come essa avviene, distinguiamo diversi tipi di modelli di apprendimento; come approfondito in [1], i più usati sono:

- *Supervised Learning*: i modelli di questa categoria vengono addestrati con dati etichettati (in gergo: *labelled*), ovvero di cui conosciamo il significato e il risultato finale. Essi sfruttano queste conoscenze e imparano per tentativi, in cui variano i loro parametri interni sulla base dello scarto tra il valore ottenuto e quello conosciuto. Solitamente affrontano problemi di classificazione e regressione.
- *Unsupervised Learning*: in questa categoria i modelli risolvono solitamente problemi di clustering, e vengono addestrati con dati non etichettati; dunque, l'elaboratore dovrà riconoscere autonomamente dai dati iniziali similitudini, caratteristiche comuni e correlazioni; al fine di classificare essi in diverse categorie.
- *Semi-supervised Learning*: è una via di mezzo tra le due precedenti, i modelli di questa categoria apprendono tramite dati parzialmente etichettati; utilizzano tecniche di clustering per riconoscere i sottogruppi, e classificano i tali sfruttando tecniche di Supervised learning con i dati etichettati.
- *Reinforcement Learning*: in questo caso non esiste una vera e propria fase di training e il modello non è a conoscenza del risultato finale (quindi i dati non sono etichettati), il modello si addestra, durante il funzionamento, sulla base di un punteggio (score) che viene attribuito ad ogni computazione; con l'obbiettivo di massimizzarlo. Dunque, con il passare del tempo il modello continuerà ad aggiornarsi e a adattarsi alle nuove situazioni, senza che sia necessario un nuovo addestramento.

1.2 Reti Neurali Artificiali

Le Reti Neurali Artificiali (ANN) sono un modello di ML che si inspira ai principi di funzionamento del cervello umano. Così come esso è composto da cellule detti neuroni interconnessi tra loro tramite le sinapsi; le Reti Neurali Artificiali sono composte da strati interconnessi di neuroni. La computazione inizia dai neuroni di input, si evolve di strato in strato, e viene propagata tramite i pesi (che sono i parametri del modello) fino a quelli di output.

Ogni neurone riceve dall'esterno o dai neuroni precedenti dei dati di input, e computa tramite una funzione, detta di Attivazione (o *Activation*), un dato di output. Dunque, la struttura a più neuroni che caratterizza la rete, ci permette di combinare diverse funzioni e costruire un sistema più "intelligente" rispetto ad altri modelli di ML. Tuttavia, esso, risulta più complesso da addestrare e perciò richiede un maggior numero di dati e potenza computazionale. Per la mancanza di queste risorse le ANN, nonostante nascano già negli anni '60 (nel '58 Rosenblatt introdusse la prima rete a singolo neurone: Il Perceptron) verranno studiate e utilizzate approfonditamente solo dai primi anni del 2000.

Le ANN sono, tipicamente, un modello di Supervised Learning; infatti, durante la fase di training i pesi vengono modificati sulla base di una funzione d'errore (detta *Loss Function*) che esprime la distanza tra i valori predetti e quelli forniti. Tipicamente viene usato per risolvere problemi di classificazione; ma, soprattutto per alcune varianti, si applica bene anche a problemi di predizione.

Per esempio, un problema risolvibile potrebbe essere la classificazione di un'immagine; i dati in input sono i pixel dell'immagine mentre l'output è la classe a cui l'immagine dovrebbe appartenere; durante il training la relativa coppia (input, output) di dati viene passata alla rete e i pesi interni dei collegamenti vengono modificati grazie alla computazione della funzione di errore, che esprime la precisione nella classificazione.

1.2.1 I principi base delle reti neurali

In questo paragrafo verranno introdotti i principi di funzionamento delle reti neurali artificiali, concentrandosi principalmente sui fondamenti delle architetture standard. Nello specifico si discuterà prima il la ANN più basica: il Perceptron e le sue componenti di funzionamento; per passare poi alle ANN a strati multipli, e indagando infine sul training e sul problema dell'overfitting.

1.2.1.1 Il singolo neurone: Il Perceptron

La rete neurale più semplice è composta da uno strato fittizio di neuroni di input e un singolo neurone, detto Perceptron, che effettua i calcoli e restituisce l'output.

L'insieme dei dati di training, definito come dataset (*D*), è suddiviso in sottoinsiemi di uguale dimensione. detti *batch* che includono più istante prese casualmente. Ad ogni ciclo, detto: *epoch* (epoca) la rete viene addestrata prendendo in input i batch in ordine casuale. Ogni istanza è la coppia di input e output in forma di $(\overline{X}, y) \in D$; dove $\overline{X} = [x_1, \ldots, x_d]$ contiene *d* variabili che rappresentano l'input della rete, y contiene il valore della variabile di output che è stato precedentemente osservato (e quindi reale).

Lo strato di input è fittizio, e perciò spesso non contato, perché non effettua calcoli, e si limita a propagare i valori d'ingresso nella rete. È composto da *d* nodi che propagano le variabili di input $\overline{X} = [x_1, \ldots, x_d]$ tramite i pesi $\overline{W} = [w_1, \ldots, w_d]$ al nodo di output. Esso effettua due calcoli: prima la somma pesata degli input $[\overline{W} * \overline{X}]$; e successivamente, applica al risultato la suddetta funzione di attivazione Φ. Il prodotto finale è:

$$\hat{y} = \Phi(\overline{W} * \overline{X}) = \Phi(\sum_{i=1}^{d} w_i x_i)$$

Durante la fase di training, ad ogni epoca, si usa l'errore della predizione $E(\overline{X}) = y - \hat{y}$ sotto forma di una Loss function *L*. Essa varia a seconda delle funzioni di attivazione, del tipo di problema e della situazione; viene utilizzata per esprimere al meglio la "distanza" tra il valore reale (y) e quello calcolato ($\hat{y}$). L'ottimizzazione, ovvero la ricerca del minimo, di tale funzione ci permette di ottenere i valori di aggiornamento usati per modificare i pesi $\overline{W}$; tali valori vengono regolati da un iperparametro detto *learning rate* (α), che li amplifica o li riduce. Per raggiungere tale punto di ottimo si usano algoritmi e approcci matematici che solitamente si servono del gradiente.

Possiamo considerare come esempio il Perceptron formulato da Rosenblatt nel 1958. In questo caso le istanze di training sono in forma $(\overline{X}, y)$ con $\overline{X} = [x_1, \dots, x_d]$ e $y \in \{-1,1\}$ che è una variabile binaria che indica l'appartenenza ad una classe; dunque, la funzione di attivazione è la funzione segno: sign(x) -> {+1, -1} a seconda del segno di x; mentre la loss function è di tipo quadratica:

$$Minimaze_{\overline{W}} L = \sum_{(\overline{X},y)\in D} (y - \hat{y})^2 = \sum_{(\overline{X},y)\in D} (y - sign\{\overline{W} * \overline{X}\})^2$$

In origine Rosenblatt non aveva a disposizione gli strumenti matematici per trovare l'ottimo e ottimizzare i pesi, dunque, usò dei sistemi implementati direttamente nell'hardware. Al contrario noi, possiamo usare un approccio definito come *Gradient Descend*, in cui si calcola il gradiente della loss function per poter aggiornare i pesi. In particolare, viene moltiplicato il valore del gradiente per il *learning rate*, e infine sottratto ai pesi. Tale metodo richiede però una funzione differenziabile; tuttavia, la funzione sign non rispetta tale requisito, e il calcolo esatto del gradiente non è possibile; perciò, come proposto in [3], si ricorre ad un'approssimazione della loss function detta *smooth* (levigata):

$$\nabla L_{smo} = \sum_{(\overline{X},y)\in D} (y - \hat{y})\overline{X}$$

Dopo ogni istanza di training $(\overline{X}, y)$ i pesi $\overline{W}$ vengono così aggiornati:

$$\overline{W} \leftarrow \overline{W} + \alpha(y - \hat{y})\overline{X}$$

(dove α è il learning rate)

1.2.1.2 Le funzioni di attivazione

Sono un'iperparametro della ANN, quindi vengono stabilite a priori, e ogni neurone ne può avere una diversa.

Ne esistono di diversi tipi, e la loro scelta è critica durante la progettazione della ANN. Varia a seconda della situazione e del tipo di problema; per

esempio, se dobbiamo predire un valore reale useremmo la funzione Identity, se dobbiamo effettuare una classificazione binaria (come nell'esempio introduttivo) sceglieremo una sigmoid, con più classi opteremo per una softmax. Tuttavia, la valutazione della funzione più adatta dipende anche dalla complessità della tale, soprattutto se poi verrà combinata con altre. Infatti, come approfondito in [3], alcune funzioni sono più semplici da derivare, e il calcolo del gradiente è più "lineare"; perciò, l'individuazione dell'ottimo della loss function è più rapido.

Alcune funzioni di attivazione sono:

- Identity: $\Phi(x) = x$
- Sign: $\Phi(x) = sign(x)$
- Sigmoid: $\Phi(x) = 1/1 + e^{-x}$
- Tanh: $\Phi(x) = (e^{2x} - 1)/(e^{2x} + 1)$
- ReLu: $\Phi(x) = \max\{x, 0\}$
- Hard Tanh: $\Phi(x) = \max\{\min\{x, 1\}, -1\}$

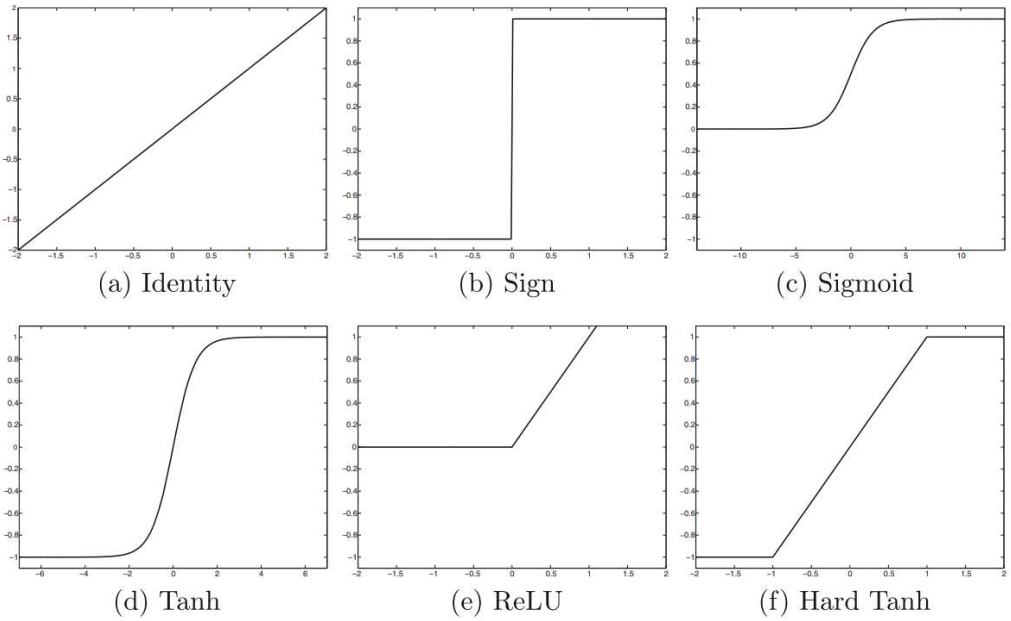


Figura 1: Alcune funzioni di attivazione[1]

[1] Neural Networks and Deep Learning: A Textbook, Charu C. Aggarwal, pag.13

Notiamo, sia dalle formule, che dai grafici in **Error! Reference source not found.** che sono molto diverse tra loro. Sigmoid e Tanh vengono usate per problemi di classificazione; Identity per problemi di predizione; infine, ReLu e Hard Tanh sono delle versioni simili a Sigmoid e Tanh, ma sono più semplici da derivare (viste le forme più "appuntite") e dunque più rapide da ottimizzare.

Nel caso del Perceptron ne abbiamo solo una da scegliere, tuttavia la reale potenza di apprendimento (ovvero "l'intelligenza" del modello), la si raggiunge combinandone molte fra loro. Tale argomento è affrontato successivamente.

1.2.1.3 Loss Function in dettaglio

Anch'essa è un iperparametro, è comune in tutta la rete e la scelta deriva dal tipo di problema e dall'obbiettivo dell'ANN; perché è necessario fare in modo che l'output della rete sia sensibile e sensato rispetto all'applicazione in questione.

Per esempio, risulterebbe irragionevole l'uso di una funzione come lo scarto quadratico medio per un problema di classificazione; questo perché in tale funzione è presente il concetto di vicinanza tra i valori che invece non è presente tra le etichette delle classi (esempio: ha senso dire che 3 è vicino a 4, non ha senso dire che cane è vicino a gatto).

Ne esistono diverse, e spesso derivano da altri modelli di ML, due per esempio sono:

- Mean Squared Error Loss (scarto quadratico medio):

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Con n = numero di istanze in D

Utilizzata per i problemi di regressione, è un valore positivo che indica lo scarto tra i valori predetti e quelli reali.

- Cross-Entropy Loss:

$$L = \sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Dove:

- *M*: è il numero di classi;
- o: è l'osservazione in questione
- y: è un indicatore binario (0,1) che indica se *c* è la corretta classificazione per o
- p: è la probabilità osservata che o sia della classe *c*
- La formula corrisponde ad una singola istanza

Utilizzata per i problemi di classificazione, più il valore risulta piccolo e più il modello sarà preciso.

Le formule mostrate corrispondono alla valutazione della Loss function nel caso del Perceptron; infatti, come analizzato da [3] , in caso di reti a più strati il calcolo è molto più complesso.

1.2.1.4 Multilayer Neural Network

Il Perceptron possiede un solo strato computazionale, quello di output, e uno d'ingresso, quello di input, al contrario le reti Multilayer (a più strati) includono tra i due sopracitati degli ulteriori strati interni, detti nascosti (*hidden*) perché non interagiscono con l'interfaccia esterna. Quest'ultimi svolgono le principali computazioni, inoltre possono essere di varie dimensioni e usare diverse funzioni di attivazione. La scelta del numero di layer, dei nodi in ognuno e delle relative funzioni possono determinare reti molto diverse; principalmente tali iperparametri influenzano "l'intelligenza" del modello, la difficoltà della fase di training e la reale efficacia di quest'ultima.

Inoltre, anche le connessioni fra strati e neuroni possono cambiare e danno origine ad architetture molto diverse, ne esistono di tutti i tipi, alcune tra le più importanti sono successivamente elencate.

- Fully connected network: particolare ANN dove ogni neurone è completamente connesso con tutti gli altri elementi della rete; non è molto usata nella pratica ed è citata solo dal punto di vista teorico.

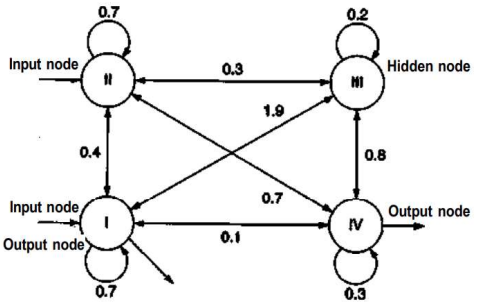


Figura 2[2]: Tipica rete Fully Connected

- Feed-Forward network: considerata come l'architettura convenzionale, ogni nodo di uno strato interno riceve l'input da tutti i nodi del livello precedente e propaga il risultato a tutti i nodi di quello successivo; a partire dal primo strato d'ingresso fino all'ultimo di output.

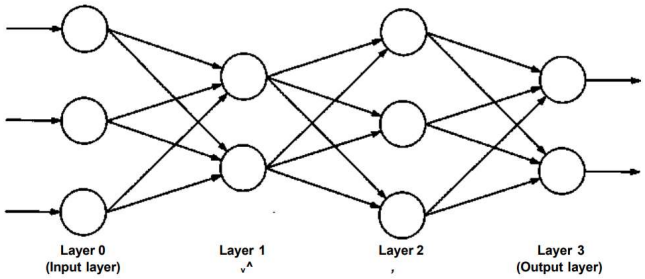


Figura 3[3]: Tipica rete Feed Forward

- Convolutional neural network: specifica architettura creata per lavorare con dati strutturati a griglia, che inoltre presentano correlazioni e dipendenze tra elementi spazialmente vicini. Un esempio è un'immagine; infatti, oltre ad avere due dimensioni di altezza e larghezza, possiamo aggiungerne una terza, il colore di ogni pixel, che è appunto spazialmente dipendente dalle altre 2. Per processare tali dati, lo strato di input di questa architettura presenta perciò una struttura a tre dimensioni.

[2] Elements of artificial neural networks, M. kishan, C.K. Mohan, S. Ranka, pag. 17
[3] Elements of artificial neural networks, M. kishan, C.K. Mohan, S. Ranka, pag. 20

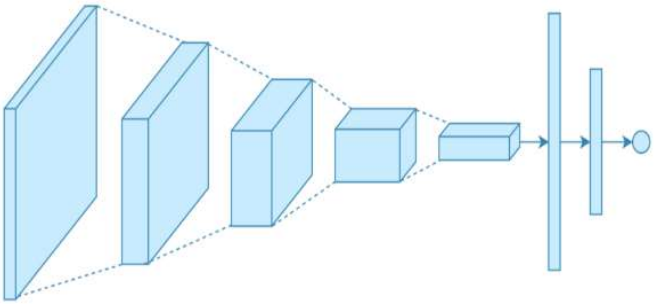


Figura 4: Tipica rete Convolutional

- Recurrent neural network: utilizzata per lavorare con sequenze di dati (come serie temporali e frasi), tramite l'uso di cicli nella rete, permette di apprendere le correlazioni tra gli elementi vicini in una sequenza.

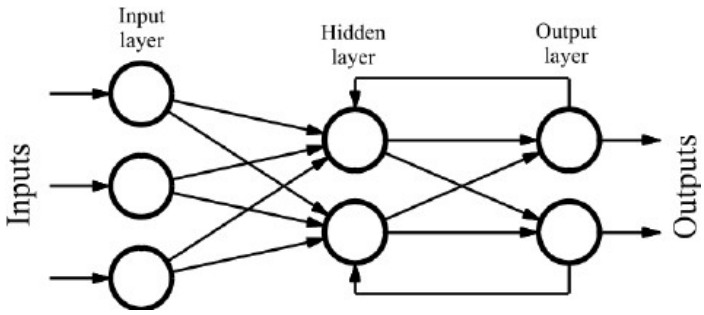


Figura 5[4]: Tipica rete Recurrent

Le architetture a singolo strato sono molto limitate, al contrario quelle multi-layer no. La reale potenza espressiva, a cui è dovuta la crescente popolarità delle ANN, risiede proprio nella possibilità di combinare, tramite layer diversi, più funzioni di attivazione non lineari; ciò porta ad un modello relativamente più semplice da addestrare e molto più espressivo, che permette di approssimare, a livello teorico, qualunque funzione. È necessario però l'uso di funzioni non lineari; perché è dimostrabile matematicamente (come fatto in [3] e [4]), che la combinazione di più funzioni lineari (come l'Identity) sia approssimabile all'uso di una sola, e quindi non porta alcun beneficio.

[4] Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Scheme-of-a-recurrent-neural-network_fig2_347917707

1.2.2 Feed-Forward Neural Network (FNN)

In questo paragrafo introdurrò le caratteristiche della rete Multilayer convenzionale, approfondendo gli aspetti positivi e indagando sulle complessità che tale architettura comporta rispetto al Perceptron. Analizzando anche nel dettaglio la fase di training.

1.2.2.1 L'architettura in dettaglio e i suoi vantaggi

Come già accennato precedentemente, l'architettura di tipo Feed-Forward rappresenta la struttura convenzionale di una ANN Multilayer. Il Perceptron può essere visto come la sua versione semplificata, le caratteristiche sono quindi in comune; tuttavia, la capacità di apprendere del Perceptron è ridotta e presenta le difficoltà della FNN in modo minimo.

La struttura della FNN è composta da 3 tipi di strati di neuroni:

- Strato di input: presente in una sola quantità, è l'unico in cui entrano i valori esterni; non svolge computazione, il numero di nodi componenti è pari alle feature in input alla rete;
- Strato di output: presente in una sola quantità, è l'unico in cui escono i valori all'esterno, il numero di nodi componenti è pari alle feature in output;
- Strato intermedio o hidden: posto in quantità variabile tra i due layer precedenti, il numero di nodi è anch'esso variabile.

Ogni nodo di un layer interno è interconnesso con tutti i nodi del layer precedente e del successivo, la computazione parte dal layer di input e termina in quello di output passando di strato in strato senza salti o cicli.

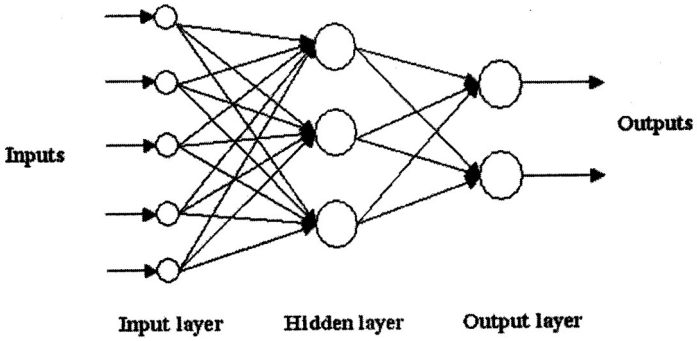


Figura 6: I layer di una FNN e l'ordine di computazione[5]

Tale FNN può essere vista come un grafo computazionale che combina le funzioni di attivazione tramite la composizione matematica; per esempio, preso uno strato *m*, con *k* nodi, che valuta la funzione $g(\cdot)$ e uno strato *m+1* che applica la funzione $f(\cdot)$; dunque, ogni nodo presente in *m+1* computa, con pesi diversi, la funzione: $f(g_1(\cdot), \dots\ g_k(\cdot))$. Se tale composizione non usa solo funzioni lineari, diventa uno strumento molto potente; perché permette di risolvere problemi non linearmente separabili; cioè, con una superficie decisionale complessa ed arbitraria (come approfondito in [5]). Si comporta quindi come approssimatore di funzione universale, cioè permette di calcolare, a livello teorico, qualsiasi funzione. Questo limite teorico è raggiungibile soltanto da una rete con sufficienti neuroni e addestrata perfettamente; tuttavia, più il numero di nodi in un layer aumenta più sono i dati richiesti affinché il training sia efficace.

1.2.2.2 La fase di Training

Nel caso del Perceptron tale fase era risolta con l'algoritmo del Gradient descend, che era semplice da calcolare visto l'uso di una sola funzione. Nel caso di una rete multilayer, le funzioni applicate negli ultimi strati sono una composizione molto complessa, che dipende anche dai pesi nei layer precedenti; dunque, il calcolo del gradiente risulta essere molto difficile. Per risolvere tale problema si ricorre all'algoritmo di *Backpropagation*, che calcola il gradiente andando a ritroso. Tuttavia, esso espone la rete ad un altro

[5] A brief review of feed-forward neural networks, M. H. Sazli, pag. 13

problema, detto *Vanishing and Exploding Gradient*, che rende il training instabile. Tali concetti sono approfonditi meglio nei sottoparagrafi successivi.

1.2.2.2.1 L'algoritmo di Backpropagation

L'invenzione della backpropagation è stato un vero punto di svolta nella storia delle ANN; infatti, la sua ideazione ha aiutato molto a renderle popolari negli ultimi vent'anni.

Tale algoritmo basa il suo funzionamento sulla regola della catena del calcolo differenziale; cioè, calcola il gradiente finale in termini di somme dei prodotti dei gradienti locali, calcolati su tutti i cammini che partono dal nodo in questione e terminano a quello di output. Questo calcolo, che prevede una serie esponenziale di percorsi, è reso possibile applicando i principi della programmazione dinamica.

Nello specifico prevede due fasi: Forward e Backward. La prima è la fase tradizionale del training supervisionato; i dati vengono forniti in input alla rete, e ciò risulta in una computazione a catena che produce il risultato finale; infine, esso è messo a confronto con il valore conosciuto e si deve calcolare l'errore. La seconda fase è responsabile di tale computazione, realizzata sfruttando la regola a catena del calcolo differenziale; ne segue una spiegazione semplificata.

Definiamo e_0 come la funzione di errore rilevata al generico nodo di output *o*. La regola a catena distribuisce il gradiente di e_0, su ogni nodo del layer precedente a cui è collegato *o*, a seconda del peso w_i del collegamento *i*-esimo; successivamente ogni nodo ridistribuisce, allo stesso modo, il gradiente ricevuto al layer precedente; e così via dicendo fino al raggiungimento del layer di input. Messa in altri termini: ogni nodo interno *h* riceve da ogni nodo di output *o*, un fattore di modifica dei pesi pari al gradiente di e_0, proporzionato ai pesi delle connessioni che collegano *h* con *o*.

Facendo un esempio, in cui abbiamo un nodo in un primo layer con un solo peso *w*, due nodi nel secondo layer e un nodo di output:

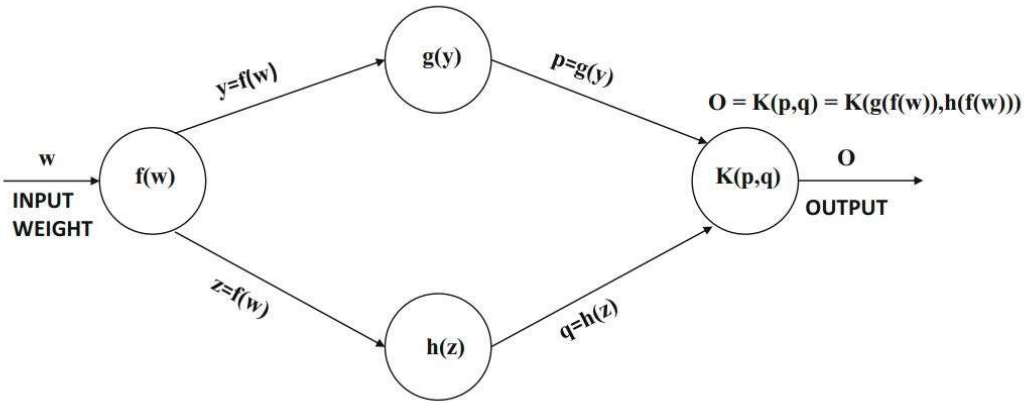


Figura 7[6]: Grafo di esempio

Dunque, nella fase backward, il nodo di output distribuisce, a seconda dei pesi, l'errore ai due nodi precedenti, i quali faranno la stessa cosa con il nodo del primo layer. La spiegazione esaustiva è approfonditile in [3]; in sintesi:

$$\frac{\partial o}{\partial w} = \frac{\partial o}{\partial p} * \frac{\partial p}{\partial w} + \frac{\partial o}{\partial q} * \frac{\partial q}{\partial w} =$$

$$\frac{\partial o}{\partial p} * \frac{\partial p}{\partial y} * \frac{\partial y}{\partial w} + \frac{\partial o}{\partial q} * \frac{\partial q}{\partial z} * \frac{\partial z}{\partial w} =$$

$$\frac{\partial K(p,q)}{\partial p} * g'(y) * f'(w) + \frac{\partial K(p,q)}{\partial q} * h'(z) * f'(w)$$

1.2.2.2.2 Vanishing and Exploding gradient

L'algoritmo di backpropagation, nonostante la sua efficacia, espone la rete ad alcuni problemi. Il primo di essi, seppur più trascurabile sono i minimi locali; infatti, quando la superfice della loss function è molto irregolare, c'è il rischio che l'algoritmo si blocchi in un minimo locale. Il secondo, e più importante problema, definito come Vanishing and Exploding gradient; può portare a instabilità durante il training e causare una vera e propria paralisi della rete.

[6] Neural Networks and Deep Learning: A Textbook, C. Aggarwal, pag. 22

Nello specifico, tale problema, è causato dalla complessità esponenziale dell'algoritmo; dove delle modifiche leggermente irregolari ai pesi dei primi layer, si propagano a valanga sul resto della rete; portando il gradiente ad assumere valori troppo grandi (Exploding) o troppo piccoli (Vanishing); e quindi l'algoritmo non raggiungerà la convergenza, perché gli incrementi dei pesi sono troppo grandi per essere precisi, o troppo piccoli per essere effettivi.

Tale problema è influenzato dal numero di layer; infatti, più essi aumentano, più la complessità del training è maggiore. Una strategia per ridurla consiste nell'usare alcune funzioni di attivazione rispetto ad altre, come: ReLu e Hard Tanh; perché presentano derivate più semplici da calcolare, e quindi rendono la composizione meno complessa. Un altro metodo si basa sull'utilizzo dinamico del learning rate.

1.2.2.2.3 Learning Rate

Il learning rate (α) è un iperparametro molto importante. Prende parte alla fase di training, in cui viene moltiplicato per il gradiente; al fine di ottenere il fattore finale con cui modificare i pesi. Dunque, variando da 0 a 1, si comporta come regolatore dell'apprendimento, cioè riduce o meno l'impatto che ogni epoca ha sulla rete.

Tale iperparametro (come approfondito in [6]) non è mai fisso ma, per un addestramento migliore e più consistente, cambia in modo dinamico. Esistono due approcci principali.

- *Decay based*: in cui il learning rate diminuisce di epoca in epoca, a seconda di un rateo regolatore *k*, che controlla il decadimento. Solitamente è realizzato in due alternative:
  - *Exponential decay*: $\alpha_t = \alpha_0 * e^{-k*t}$ (dove t è il numero di epoca corrente)
  - *Inverse decay*: $\alpha_t = \alpha_0/(1 + k * t)$ (dove t è il numero di epoca corrente)

- *Momentum based*: è una tecnica più complessa, che adegua il learning rate a seconda del momentum; cioè modificandolo sulla base della media degli ultimi movimenti. Tale effetto si ottiene in questo modo:

$$\overline{W} \Leftarrow \overline{W} + \overline{V}$$

$$\overline{V} \Leftarrow \beta * \overline{V} - \alpha * \frac{\partial L}{\partial \overline{W}}$$ (con β iperparametro regolatore del momentum)

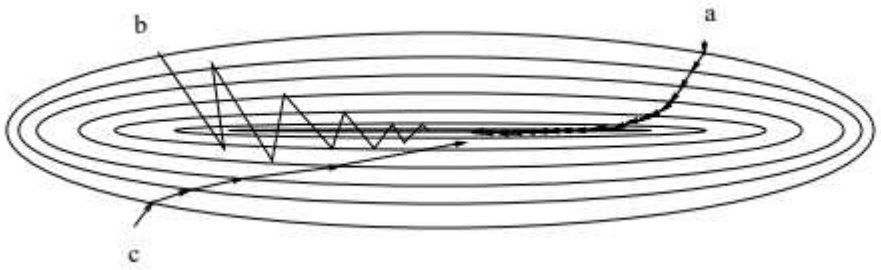


Figura 8[7]: Il gradient descend nello spazio dei parametri. a) con un learning rate basso. b) con un learning rate alto. C)con un learning rate alto con momentum

Altre strategie, per rendere la fase di training più consistente, si basano sull'uso di learning rate che varia per ogni parametro. Perché parametri con derivate parziali più complesse, tendono ad oscillare maggiormente, e necessitano di un learning rate più dinamico. Le più importanti, viste brevemente, sono:

- AdaGrad: enfatizza i movimenti consistenti, penalizzando le derivate parziali che tendo ad avere grosse fluttuazioni. Assegnando al valore A_i, aggregato di tutte le derivate parziali, dell'*i*-esimo parametro un aggiornamento così definito:

$$A_i \Leftarrow A_i + \left(\frac{\partial L}{\partial w_i}\right)^2 \quad \forall i$$

$$w_i \Leftarrow w_i - \frac{\alpha}{\sqrt{A_i}}\left(\frac{\partial L}{\partial w_i}\right) \quad \forall i$$

[7] An introduction to neural networks, B. Krose, P. van der Smagt, pag. 37

- RMSProp: al contrario di AdaGrad, dove l'aggregazione di A_i portava ad un rallentamento prematuro; al posto di sommare, si usa una media esponenziale regolata da un fattore di decadimento $\rho \in (0,1)$:

$$A_i \Leftarrow \rho A_i + (1-\rho)\left(\frac{\partial L}{\partial w_i}\right)^2 \quad \forall i$$

$$w_i \Leftarrow w_i - \frac{\alpha}{\sqrt{A_i}}\left(\frac{\partial L}{\partial w_i}\right) \quad \forall i$$

- AdaDelta: simile a RMSProp, ma elimina la necessità dell'iperparametro globale di learning rate α calcolandolo dai precedenti aggiornamenti:

$$w_i \Leftarrow w_i - \sqrt{\frac{\delta_i}{A_i}}\left(\frac{\partial L}{\partial w_i}\right) \quad \forall i$$

$$\delta_i \Leftarrow \rho\delta_i + (1-\rho)\left(\sqrt{\frac{\delta_i}{A_i}}\left(\frac{\partial L}{\partial w_i}\right)\right)^2 \quad \forall i$$

- Adam: corregge il problematico bias di inizializzazione di RMSProp, in cui i primi aggiornamenti influenzavano l'intera fase di learning. Utilizza quindi: una versione esponenzialmente levigata F_i del gradiente; un parametro ρ_f simile a ρ ma che incorpora il momentum; e il learning rate α è sostituito con α_t che dipende dall'indice di iterazione *t*:

$$A_i \Leftarrow \rho A_i + (1-\rho)\left(\frac{\partial L}{\partial w_i}\right)^2 \quad \forall i$$

$$F_i \Leftarrow \rho_f F_i + (1-\rho_f)\left(\frac{\partial L}{\partial w_i}\right) \quad \forall i$$

$$w_i \Leftarrow w_i - \frac{\alpha_t}{\sqrt{A_i}} F_i \quad \forall i$$

$$\alpha_t = \alpha\left(\frac{\sqrt{1-\rho^t}}{1-p_f^t}\right)$$

1.2.2.3 Overfitting

È un problema esistente in tutti i modelli di ML, ma è ben presente nelle ANN (e FNN) a causa del grande numero di parametri posseduti da una rete complessa. Rappresenta la perdita della capacità di generalizzazione del modello.

Per risolvere tale problema, dopo la fase di Training, seguono due fasi[8] di verifica: *validation* e *testing*; in cui il modello affronterà dei dati non ancora usati (detti rispettivamente *validation set* e *test set*). Nella prima, i risultati prodotti verranno usati per tenere traccia, e migliorare la capacità di generalizzazione della rete; nella seconda, invece verranno usati per avere una valutazione finale del modello.

1.2.2.3.1 Causa

Si verifica quando la performance sui dati di training è molto più alta, rispetto a quella ottenuta con dati mai visti dalla rete. Questo accade perché la FNN è diventata troppo specifica per quei dati; dunque, è come se li avesse memorizzati, piuttosto di imparare i pattern logici tra essi.

La causa generale di questo fenomeno è da imputarsi a tre fattori: pochi dati di training; oppure scarsa rappresentazione, da essi fornita, dell'insieme; rete troppo complessa. Esse sono correlate, perché una rete troppo complessa, ovvero con troppi parametri, necessità di più dati, e, se essi non sono rappresentativi dell'insieme generico, allora sono poco utili.

La scarsità dei dati è stato uno dei motivi principali, per cui le ANN non vennero usate fino agli anni 2000. Ad oggi, il problema dell'overfitting, è molto discusso e affrontato con diverse tecniche.

---

[8] *What are artificial neural networks?*, A. Krogh, pag 197

1.2.2.3.2 Soluzioni

La soluzione più semplice sarebbe quella di aumentare il numero di dati; tuttavia, quando essi sono limitati possiamo ricorrere a dei sistemi più o meno complessi. Di seguito ne sono introdotti alcuni.

- Tuning degli iperparametri: pratica molto comune, in cui si testa la rete variando gli iperparametri e misurando l'errore sul validation set; infine, si usa il set di iperparametri che rende la rete più performante.
- Regularization: si introduce un iperparametro di regolarizzazione che agisce come penalità; esso va a ridurre il valore assumibile dai parametri; e quindi, l'influenza singola che ognuno di essi ha sul risultato prodotto. Di fatto, sarà simile ad avere una NN con meno parametri, che risulta però essere più "intelligente".
- Early stopping: strategia molto comune, in cui, durante il training, si tiene traccia dell'errore anche sul validation set. Quando l'errore prodotto, sui due set di dati, diverge troppo, si procede ad interrompere prematuramente il processo.
- Parameter sharing: pratica adottabile in situazioni in cui è utile una correlazione fra due o più funzioni computate in più nodi della rete. In questi casi si tende ad usare specifiche architetture, come RNN e CNN, che sfruttano proprio questo principio.
- Trading Off Breadth for Depth: si diminuisce il numero di nodi per ogni livello, in favore di un maggior numero di livelli. In questo modo si riduce il numero di connessioni (e quindi di parametri) senza rendere il modello meno "intelligente"; anche se la fase di training diventa più difficile (a causa dei problemi di convergenza).
- Unsupervised pre-training: si precede al training stesso, con una fase che coinvolge ogni layer singolarmente. In essa si effettua il pre-training, cioè si inizializzano i pesi dei nodi, tramite un algoritmo non supervisionato.
- Ensemble Methods: famiglia di metodi comuni ad altri modelli di ML (seppur alcuni siano specifici per le ANN). Essi creano diversi modelli

base, tramite un principio che varia a seconda del metodo, e poi li combinano per ottenere quello ottimale.

1.2.3 Reti Neurali Ricorrenti (RNN)

Le RNN sono un'architettura non convenzionale di ANN multilayer; sono estremamente utili in certe applicazioni, ma aggiungono ulteriori problemi e complessità. Prima affronterò tali caratteristiche, per poi introdurre uno specifico tipo di RNN: LSTM, particolarmente efficiente nel risolvere certi problemi.

1.2.3.1 I principi base

Le FNN sono ideate per lavorare con dati multidimensionali, in cui ogni attributo è indipendente dagli altri; peccano dunque nel riconoscere e sfruttare le dipendenze tra essi. Al contrario, le RNN sono ideate per lavorare con dati che presentano dipendenze sequenziali, come serie temporali o testi; dove un elemento della sequenza è strettamente correlato con il precedente o il successivo. Esse, a differenza delle FNN, non solo riconoscono le dipendenze, ma anche l'ordine degli elementi.

Inoltre, la particolare struttura ricorsiva della RNN, permette l'uso di sequenze con lunghezza non definita. Ciò non sarebbe realizzabile con una FNN; perché, siccome ogni neurone è associato ad un elemento della sequenza, non sarebbe possibile avere una rete con un numero di neuroni indefinito.

Infine, le RNN sono definite *Turing Complete*, cioè sono in grado di simulare qualsiasi algoritmo, a patto di avere abbastanza dati e potenza computazionale. Tuttavia, questa caratteristica è poco utilizzata ed esplorata, se non con particolari varianti, perché le risorse richieste la rendono poco fattibile.

Tutti questi vantaggi derivano da una struttura molto più complessa, che sarà difficile da addestrare e più soggetta al problema del Vanishing and Exploding gradient.

In questo paragrafo introdurrò le caratteristiche fondamentali dell'architettura e le complessità da essa introdotte.

1.2.3.1.1 Architettura in dettaglio

L'architettura di una RNN è rappresentabile tramite una forma detta *Time Layering*; in cui la rete è composta da un numero di layer variabile, e dove ognuno corrisponde ad una posizione nella sequenza (detta *timestamp*). Ogni layer condivide lo stesso set di pesi, e ha un input, un output, e uno stato interno che interagisce con quello del layer successivo. Di fatto però la rete sarà composta da un unico layer ripetuto nel tempo, e la propagazione dello stato interno avviene tramite un self-loop (un ciclo ricorsivo).

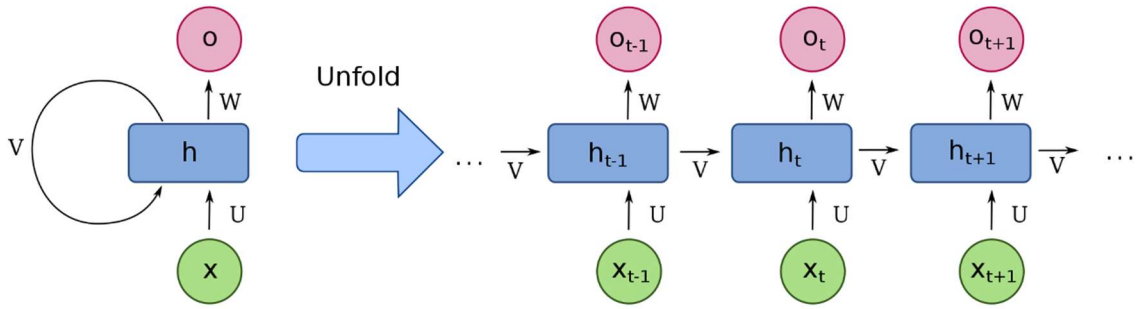


Figura 9: La rete ricorrente reale e la sua rappresentazione Time Layering

Quindi, ad ogni timestamp *t*:

- $\overline{h}_t = \Phi(U * \overline{x}_t + V * \overline{h}_{t-1})$
- $\overline{o}_t = W * \overline{h}_t$

Dove *x*, *h* e *o* sono i 3 layer (di input, interno e di output); e *U*, *V* e *W* le matrici dei pesi dei collegamenti che connettono i layer. Notiamo quindi che la forma Time Layering è come una FNN che rappresenta ciò che accade alla RNN in ogni timestamp.

La RNN proposta consiste nel caso più generico (*many to many*), in cui ad ogni timestamp la rete ha un nuovo input e un nuovo output. Infatti, in alcune situazioni tali componenti sono mancanti; cioè, la rete fornisce un solo output alla fine della sequenza, o prende un solo input all'inizio della sequenza, o tutti i casi intermedi.

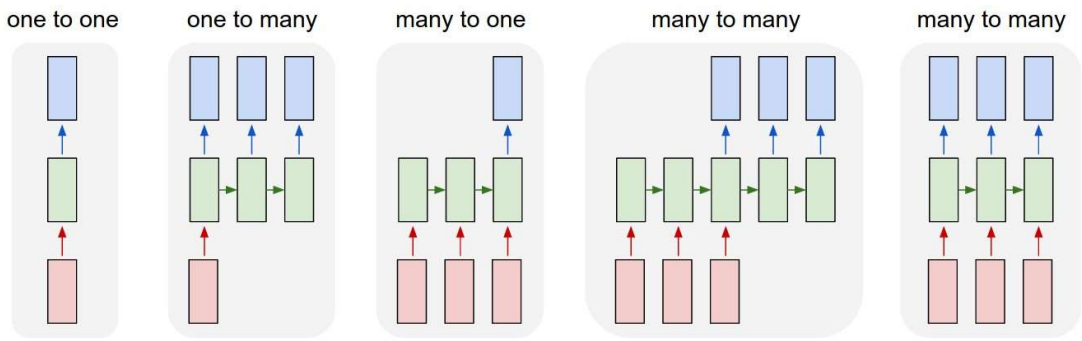


Figura 10: Diverse RNN con variabili strati di input e output

L'architettura fino ad ora analizzata è composta da un singolo layer ricorrente, è naturalmente possibile averne una con più layer ricorrenti di fila, aumentando così la potenza del modello e le relative complicazioni di training e generalizzazione.

Esistono inoltre diverse varianti, una molto importante è quella bidirezionale (*Bidirectional*), dove ogni nodo del layer ricorrente possiede due suoi stati interni; uno è quello tradizionale, calcolato dal timestamp precedente, l'altro invece è calcolato dal timestamp successivo.

1.2.3.1.2 Training

La fase di training per una RNN risulta più complicata perché non è possibile applicare direttamente la backpropagation, perché l'algoritmo assume che i pesi siano sempre distinti (invece sono condivisi nei vari timestamp); dunque si usa la variante *backpropagation through time* (BPTT), approfondito in [4].

Il funzionamento della BPTT è suddivisibile in tre fasi:

1. Simile alla fase forward, si fornisce in input alla rete la sequenza di dati in ordine temporale, calcolando l'errore ad ogni timestamp.

2. Simile alla fase backward, si calcola il gradiente dell'errore di ogni peso andando a ritroso; assumendo però che i parametri nei diversi layer temporali siano diversi. Per fare ciò introduciamo delle variabili temporali che dovrebbero essere diverse ma che di fatto sono uguali; per esempio, i pesi *W* diventano diverse variabili $W^{(t)}$ per ogni timestamp *t*.
3. Si sommano i gradienti delle variabili temporali dei parametri condivisi:

$$\frac{\partial L}{\partial U} = \sum_{t=1}^{T} \frac{\partial L}{\partial U^{(t)}}$$

$$\frac{\partial L}{\partial V} = \sum_{t=1}^{T} \frac{\partial L}{\partial V^{(t)}}$$

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L}{\partial W^{(t)}}$$

Questa architettura, all'allungarsi della sequenza, amplifica il problema del vanishing and exploding gradient, portando un'incredibile instabilità al training. Perché, oltre ad avere una rete più lunga e complessa; il BPTT, ad ogni timestamp, moltiplica il gradiente con lo stesso peso $w_t = w$; quindi se *w* è molto grande o piccolo causa l'exploding o il vanishing del gradiente.

Sono nate diverse soluzioni a questo problema: forte regolarizzazione dei parametri, che però porta a minore espressività; un buon punto di inizializzazione dei pesi; utilizzo di metodi di Gradient-descend che sfruttano derivate del secondo ordine, ma che richiedono tante risorse computazionali; tecnica del *gradient clipping*, utile in presenza di parametri condivisi, perché rende l'update ad ogni timestamp simile; utilizzo di alcune varianti, come *LSTM*.

1.2.3.2 LSTM

LSTM sta per *Long-Short Term Memory*, è una particolare architettura della cella di un layer ricorsivo di una RNN; esso ci permette di risolvere, in parte, il problema del Vanishing and Exploding Gradient; ma soprattutto di

aumentare la “memoria” del modello, incrementando l’abilità di apprendere lunghe dipendenze trai dati. La struttura di una cella LSTM, perciò, è molto più complessa rispetto ad una classica RNN; in cui ricordiamo essere presente una sola funzione di attivazione, usata per calcolare lo stato interno e l’output.

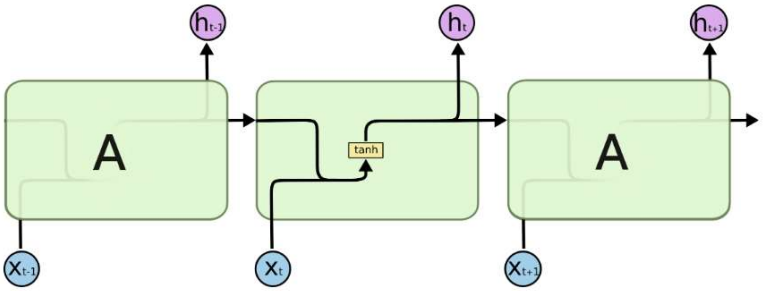


Figura 11[9]: Struttura cella standard RNN

Invece la cella LSTM, come analizzato approfonditamente in [7] e [8]; è composta da quattro funzioni di attivazione, strutturate per interagire in certi modi. L’idea principale si basa su due componenti fondamentali:

- Lo stato interno: strutturato come un nastro trasportatore, prende il proprio valore dal timestamp precedente, e viene modificato durante la computazione della cella.
- I gate: particolare operazione, composta da una sigmoid σ e una moltiplicazione a due fattori, usata per regolare l’informazione che passa in una linea; perché, l’output di una sigmoid è compreso tra 0 e 1. Per esempio, se *x* è il fattore regolatore, e *y* la linea da regolare: $y_t = y_{t-1} * \sigma(W_{xy} * x_{t-1})$.

[9] Understanding LSTM Networks, C. Olah

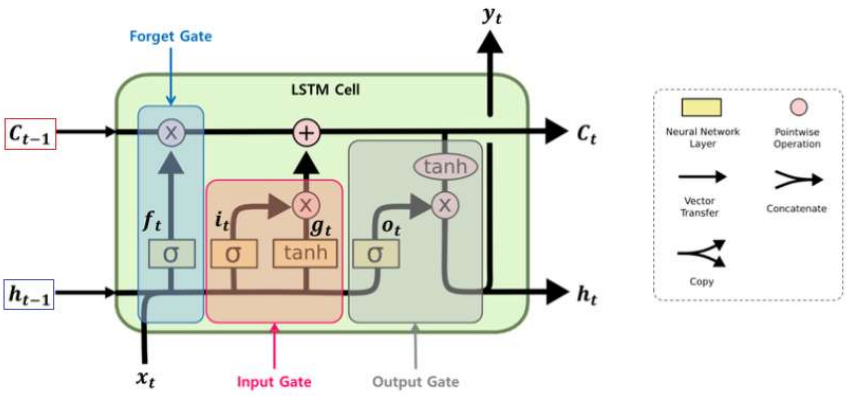


Figura 12: Struttura cella LSTM

Il suo funzionamento può essere diviso in tre fasi, a cui corrispondono i tre gate:

- Forget gate: stabilisce l'informazione da mantenere dello stato interno precedente c_{t-1}, a partire dall'output precedente h_{t-1} e dall'input x_t:

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

- Input gate: stabilisce l'informazione da aggiungere allo stato interno precedente c_{t-1}, a partire dall'output precedente h_{t-1} e dall'input x_t:

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

$$g_t = tanh(W_{hg} * h_{t-1} + W_{xg} * x_t)$$

$$c_t = f_t * c_{t-1} + i_t * g_t$$

- Output gate: stabilisce l'output y_t, h_t, a partire dall'output precedente h_{t-1} e dall'input x_t, e dallo stato interno c_t:

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

$$h_t = y_t = o_t * \tanh(c_t)$$

1.3 Opzioni

Le opzioni sono una security, ovvero un bene scambiabile, fungibile e con valore finanziario; più specificatamente sono dei derivati, cioè degli strumenti legati ad un sottostante su cui "operano". Esso, a sua volta, può essere una qualsiasi security, anche se tradizionalmente sono valute o equities (per esempio azioni); in questa tesi, se non diversamente esplicitato, si farà

riferimento sempre a opzioni su equities, anche se i principi fondamentali non cambiano con il sottostante.

I primi scambi di opzioni risalgono a Londra nel 1690; tuttavia l'uso massiccio di tale strumento si sviluppò solo negli anni 70', grazie all'invenzione del modello Black-Scholes-Merton, che ha reso più semplice stabilirne il prezzo corretto; e all'apertura del primo centro di scambi, il CBOE (Chicago Board Option Exchange).

1.3.1 I principi fondamentali

In questo paragrafo verranno introdotte le principali nozioni; come la definizione, l'uso, il modello di prezzo, come vengono combinate e i Greeks.

1.3.1.2 Cosa sono?

Un'opzione è un contratto stipulato, a fronte del pagamento di un premium, tra un compratore e un venditore (detto writer); esso fornisce al compratore il diritto, ma non l'obbligo, di comprare o vendere un determinato sottostante, ad un prezzo e in una certa data. Se forniscono il diritto di comprare sono dette Call, altrimenti Put. Tradizionalmente un'opzione controlla 100 azioni del sottostante.

L'opzione si dice esercitata quando viene fatto valere il diritto di acquisto o vendita; Il prezzo di esercizio viene detto *Strike*, mentre la data è detta *Expire* (o expiration), perché è il momento in cui il contratto scade. A seconda della differenza tra lo strike e il prezzo corrente del sottostante, l'opzione ha un valore pari a zero o positivo; a tal proposito si usano diversi termini:

- OTM (Out of The Money), quando il prezzo non ha raggiunto lo strike (in caso di call strike < prezzo, altrimenti strike > prezzo);
- ATM (At The Money), quando strike e prezzo sono simili;
- ITM (In The Money), quando lo strike ha raggiunto e superato il prezzo (in caso di call strike > prezzo, altrimenti strike < prezzo);

Per esempio: *A* stipula con *B* una call su un Apple (*AAPL*), con uno strike a 165$ che scade in 3 mesi, pagando un premium di 4$; in quel momento *AAPL* tradda a 150$, quindi l'opzione è OTM del 10% (cioè il prezzo deve salire del 10% per raggiungere lo strike). Se alla scadenza *AAPL* varrà meno di 165$, allora *A* non eserciterà l'opzione e perderà il premium; se varrà poco più di 165$ (tra i 165$ e i 168$), *A* eserciterà l'opzione comprando *AAPL* a 165$ mentre sul mercato vale di più, ma contando il premium sarà comunque in perdita; infine, se *AAPL* varrà molto più di 165$ (>169$) allora *A* avrà un profitto.

Chi acquista un'opzione (si dice che è *long*) ha dunque un profitto potenzialmente illimitato e una perdita limitata al premium; chi vende (si dice che è *short*) si trova invece in una situazione inversa. L'investitore, a seconda del punto di vista sul mercato, può decidere di:

- Comprare una call, se ha una visione molto rialzista (bullish);
- Vendere una put, se ha una visione neutrale o leggermente rialzista;
- Vendere una call, se ha una visione neutrale o leggermente ribassista;
- Comprare una put, se ha una visione molto ribassista (bearish)

Inoltre, esistono diversi tipi di opzioni che variano sulla base della loro modalità di esercizio, le principali sono:

- Europee: permettono l'esercizio solo alla data di scadenza;
- Americane: che possono essere esercitate anche prima della scadenza;
- Bermuda: possono essere esercitate solo in alcuni giorni specifici (oltre che all'expire)
- Asiatiche: il cui valore finale è dato dalla media del prezzo del sottostante in un determinato periodo;
- Barrier: possono essere esercitate solo se il prezzo del sottostante supera un prezzo barriera predeterminato;

Quelle più usate (ovunque) sono le Americane, ma per semplicità, si farà sempre riferimento a quelle Europee.

1.3.1.3 Il valore di un'opzione e il modello Black-Scholes-Merton

Il valore intrinseco di un'opzione è il massimo tra 0 e la differenza tra prezzo del sottostante e lo strike (nel caso di una put è strike meno sottostante), e corrisponde al valore che avrebbe l'opzione se venisse esercitata in quel momento. Il valore reale di un'opzione, non ancora scaduta, è però maggiore di quello intrinseco perché è anche presente il cosiddetto *Time Valu*e; il quale rappresenta il movimento che il sottostante può ancora compiere nel tempo rimante alla scadenza.

Le componenti che principalmente influiscono sul premium delle opzioni sono molte: lo strike, il prezzo del sottostante, la data di scadenza, il rateo d'interesse, la presenza di dividenti e la volatilità del sottostante. In particole quest'ultimo elemento è molto impattante (come la differenza tra strike e prezzo e la data), perché più è alta la volatilità (cioè la deviazione standard del prezzo) maggiori sono le oscillazioni del sottostante, e dunque i potenziali profitti delle posizioni long (e di conseguenza il premium che i compratori andranno a pagare).

Visti gli elementi coinvolti, il difficile compito di attribuire il prezzo corretto ha limitato molto l'uso di questi strumenti; fino all'invenzione del modello Black-Scholes-Merton, abbreviato come Black-Scholes (BS); che risolve, almeno in parte, questo problema. Esso, infatti permette di determinare un valore teorico che prenda in considerazione i parametri più impattanti.

Tale modello, assume che: il mercato sia randomico (cioè non prevedibile); non ci siano costi di transazione; il rateo d'interesse e la volatilità siano costanti e conosciute; i ritorni siano normalmente distribuiti; l'opzione sia di tipo Europea; non ci siano dividendi nel periodo di vita dell'opzione. E stabilisce che il prezzo di una call sia:

$$C = S * N(d_1) - N(d_2) * K * e^{-r*t}$$

$$d_1 = \frac{\ln\frac{S}{K} + \left(r + \frac{\sigma^2}{2} * t\right)}{\sigma * \sqrt{t}}$$

$$d_2 = d_1 - \sigma * \sqrt{t}$$

Dove:

- *C*: è il prezzo della call;
- S: è il prezzo del sottostante in quel momento;
- K: è lo strike;
- r: è il rateo d'interesse;
- t: è il tempo alla scadenza;
- N: una distribuzione di probabilità normale (media = 0, deviazione std. = 1);
- σ: è la varianza dei ritorni giornalieri del sottostante;
- (la formula per il prezzo di una put è simile)

Le assunzioni precedentemente viste lo rendono poco realistico; ma comunque utile per comprendere le componenti principali, e come la loro variazione influenzi il prezzo; ovviamente, con il passare del tempo, sono stati inventati nuovi modelli e varianti più "pratiche" (cioè con minori assunzioni).

1.3.1.5 Greeks

Il termine *Greeks* è associato ad una famiglia di parametri, specificatamente lettere greche, che descrivono l'esposizione al rischio di una posizione con opzioni. Tale esposizione può essere scomposta in diversi fattori, che ogni Greek descrive singolarmente.

Essi variano (per un'opzione che controlla 100 azioni) da -100 a 100, e indicano quanto la variazione di un determinato parametro influisca sulla variazione del prezzo o, in alcuni casi, di un altro parametro; e le loro definizioni derivano direttamente dal modello BS. Ne esistono diversi, i 4 principalmente più discussi, e utili nella tesi, sono:

- *Delta* (δ o Δ): Misura l'esposizione al prezzo. Cioè indica quanto la variazione di prezzo del sottostante influisca su quello dell'opzione. Viene anche usato come misura della probabilità che l'opzione scadi ITM. Per esempio, una con delta 100 si comporterà come una posizione long sul sottostante; quindi, un incremento del tale di 5% comporta ad un incremento del 5% anche sull'opzione.

$$\Delta(Call) = N(d_1) \qquad \Delta(Put) = 1 - N(d_1)$$

- *Gamma* (γ o Γ): Misura l'esposizione del delta al prezzo. Cioè indica quanto la variazione di prezzo del sottostante influisca sulla variazione del delta dell'opzione.

$$\Gamma = -\frac{N'(d_1)}{S\sqrt{T}\sigma}$$

- *Theta* (θ o Θ)*:* Misura il decadimento causato del tempo. Cioè indica come il passaggio del tempo (e quindi l'avvicinarsi all'expire) influisca sul prezzo dell'opzione.

$$\Theta(\text{Call}) = -\frac{S\sigma * N'(d_1)}{2\sqrt{T}} + r * K * e^{-rT} * N(d_2)$$

$$\Theta(\text{Put}) = -\frac{S\sigma * N'(d_1)}{2\sqrt{T}} + r * K * e^{-rT} * N(-d_2)$$

- *Vega* (v): Misura l'esposizione alla volatilità. Cioè indica come la variazione della volatilità del sottostante influisca sulla variazione di prezzo dell'opzione.

$$v = S\sqrt{T} * N'(d_1)$$

(Per posizioni short i Greeks avranno segno inverso)

1.3.1.6 Introduzione agli Spread

Gli Spread sono combinazioni complesse di più opzioni, con eventualmente anche il sottostante. Le opzioni che lo compongono posso variare per il tipo (call o put e long o short), lo strike o l'expiration, mantenendo invariato il sottostante. I Greeks sono particolarmente utili, perché essendo linearmente

combinabili, aiutano a tenere traccia nel complesso delle diverse esposizioni dello spread.

Essi sono molto utili perché permettono all'investitore di esporsi a dei fattori del mercato in dei modi, che non sarebbe possibile avere con il semplice sottostante. Per esempio, può annullare (ponendo a 0) un greek per esporsi agli altri; oppure, al contrario può isolarlo annullando gli altri.

Uno dei più semplici è il Bull Call Spread, che prevede l'acquisto di una call *C1* (ITM o OTM) e la vendita di una Call *C2* (OTM) con strike maggiore e stessa expiration; in questo modo l'investitore ha comunque un'esposizione Bullish al sottostante, causata da C1, ma è limitata superiormente da C2. Dunque, il massimo profitto è la differenza dei due strike (K2 – K1); mentre il costo della posizione, e dunque la massima perdita, è la differenza tra il premium pagato per C1 e quello ottenuto da C2 (C1 – C2). Uno spread del genere è l'ideale quando l'investitore si aspetta un movimento bullish con un'estensione limitata.

1.3.1.6 Usi generali

Le opzioni sono ampiamente coinvolte sia in strategie di trading a breve termine, sia in piani di investimento a lungo termine. Appunto perché il loro valore è influenzato da diversi fattori, e ciò le rende uno strumento molto versatile, ma allo stesso tempo molto rischioso.

I grandi investitori (fondi, istituzioni, banche ecc..) solitamente sfruttano le opzioni per tutelarsi dai rischi, tramite delle pratiche di *hedging*. Cioè comprano o vendono tali strumenti, rinunciando a potenziali profitti, al fine di scaricare su altri investitori dei rischi, come quello legato alla volatilità o a forti movimenti del prezzo. Per esempio (teoricamente) un fondo può comprare delle put OTM, come un'assicurazione contro forti ribassi.

Invece gli investitori più piccoli, o gli speculatori, sfruttano la naturale proprietà di leva finanziaria di questi strumenti; in modo da poter speculare maggiormente su determinati eventi. Infatti, uno speculatore può ottenere

un’esposizione long o short maggiore, allo stesso costo (rispetto all’azione); oppure può usare uno spread per esporsi ad uno specifico evento, come nel Bull Call Spread.

1.3.2 La Volatilità

La volatilità ha un ruolo fondamentale nel trading con opzioni; sia negli studi teorici, che nelle strategie adottate dai professionisti nei mercati; per questi motivi ha un ruolo centrale nella tesi e nel progetto. Verrà dunque meglio approfondita in questo paragrafo.

1.3.2.1 Cos’è

La volatilità, nella sua definizione più generica, è la statistica che misura la dispersione dei ritorni di una security in un periodo. Essa misura la “magnitudine” dei movimenti di una security; dunque, più è alta più ci si aspetta grossi movimenti a rialzo o a ribasso.

Una security viene considerata statisticamente più sicura, tanto quanto i suoi ritorni saranno simili alla media storica. Siccome la volatilità misura lo scarto tra il ritorno medio e quelli realizzati, allora è associata ad un indice di sicurezza della security.

Per esempio, prese due security A e B con lo stesso ritorno medio, se la volatilità di A è maggiore rispetto a quella di B; allora A sarà considerata meno sicura, perché la deviazione da quel rendimento è maggiore, e quindi il grado di rischio è più alto.

È un concetto ampio e largamente utilizzato in diversi settori della finanza; nell’ambito delle opzioni si parla di tre sue specializzazioni.

1.3.2.3 Realizzata (RV) e storica (HV)

La volatilità realizzata e storica sono due concetti simili (spesso sovrapposti), e si differenziano per il periodo di tempo considerato. Se esso è nel passato, e generalmente più lungo (un anno), si parla di volatilità storica; se invece è nel futuro, e generalmente più breve (un mese), si parla di volatilità realizzata

futura. Nella pratica vengono calcolate con strategie diverse, quella largamente più usata è la *Close to Close*; che definisce la volatilità come la radice della varianza dei ritorni logaritmici:

$$x_i = \ln\left(\frac{c_i + d_i}{c_{i-1}}\right)$$

(dove c_i è il prezzo di chiusura del giorno i e d_i il dividendo)

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \overline{x}\,)}{N}}$$

(dove N è la lunghezza del periodo e $\overline{x}$ è la media degli x_i)

Solitamente viene annualizzata: $\sigma(annualizzata) = \sigma * \sqrt{252}$

1.3.2.3 Volatilità Implicita (IV)

Rappresenta l'aspettativa del mercato in merito alla variazione di prezzo del sottante. Essa non è calcolabile direttamente, ma solo approssimabile dai modelli di prezzo (come il Black-Scholes), che devono ad essa la loro imprecisione.

Tale metrica è molto complessa e causata da diversi fattori; come: la domanda e offerta per quegli strumenti, la condizione macroeconomica, l'attesa di periodi più o meno volatili, il sentiment in generale, ecc. Inoltre, ogni opzione a seconda dello strike e della scadenza presenta un'IV diversa (maggiore ad una scadenza lontana, ed a uno strike OTM).

Quando si parla in termini generali dell'IV di un mercato (o azione) si fa riferimento ad un'aggregazione di IV delle opzioni più pertinenti; tale strumento viene detto *indice di volatilità IVX* e rappresenta l'IV generale di quel sottostante. Esso, essendo la stima di un valore atteso, non è quasi mai in linea con la corrispettiva RV, e dunque, esistono dei periodi dove le

opzioni per quello strumento sono relativamente sovra prezzate o sotto prezzate.

Capitolo 2: Progetto

In questo capitolo si descrive il progetto in tutti i suoi aspetti, concentrandosi sulle motivazioni dietro le scelte progettuali. Quindi nei successivi paragrafi si analizzeranno: i dati utilizzati; come essi sono stati processati; la rete neurale impiegata e le relative tecnologie; le strategie di trading con le opzioni; e infine i risultati con le relative conclusioni.

2.1 L'obbiettivo

Come introdotto nel paragrafo della volatilità, l'IV non coincide spesso con la volatilità futura realizzata in quel periodo. Il mercato commette degli errori in queste stime; spesso essi sono leciti e razionali, ma in altri casi sono frutto di valutazioni o impulsi irrazionali; andando perciò a violare l'ipotesi del mercato efficiente, con la conseguente opportunità di profitti (aggiustati al rischio) superiori al rendimento del mercato.

Esistono diverse strategie che sfruttano questo principio, e tradizionalmente si basano sul vendere o comprare volatilità, tramite degli spread, quando l'IV è molto superiore o inferiore alla media storica della RV. Tuttavia, tale sistema non è preciso, e si basa sull'idea comune che L'IV dovrebbe, nel lungo periodo, ripareggiarsi con l'HV media. Il principio alla base del progetto è di utilizzare il ML per avere una valutazione più precisa nel breve periodo.

La rete neurale, a partire da una serie di dati (tra cui IV e RV), deve classificare se la RV nei successivi 5 giorni (lavorativi) sarà significativamente maggiore o minore rispetto a quella attesa (ovvero l'IV stimata a 5 giorni). A seconda della valutazione, verranno aperte delle posizioni long o short sulla volatilità, tramite 4 tipi di spread diversi; nel caso in cui non ci sia l'inefficienza del mercato, non verrà aperta alcuna posizione.

2.2 Dataset utilizzato

Il modello per la sua previsione impiega diversi dati; prima di essere usati, essi vengono, scaricati, selezionati, processati, scalati e trasposti nel tempo.

In particolare, i dati prima di essere usati vengono elaborati in 6 fasi:

1. Download: dove vengono scaricati;
2. Preprocessing: in cui si effettua lo smoothing per poi renderli stazionari;
3. Feature selection: dove si riduce il numero di feature mantenendo solo le più utili;
4. PCA: in cui si usa la tecnica della PCA per rimuovere le correlazioni tra le feature;
5. Scaling: dove essi vengono scalati in un intervallo;
6. Lag e Split: in cui vengono trasposti e convertiti nei dataset finali;

2.2.1 Download

Vista la scarsa disponibilità di dati, soprattutto per quanto riguarda lo storico delle opzioni; sono stati utilizzati gli storici reali per l'azione Apple (AAPL) tra il 01/01/2011 e il 06/01/2022:

Il dataset utilizzato dal modello include 15 feature, più il target (ovvero la classificazione), esse sono:

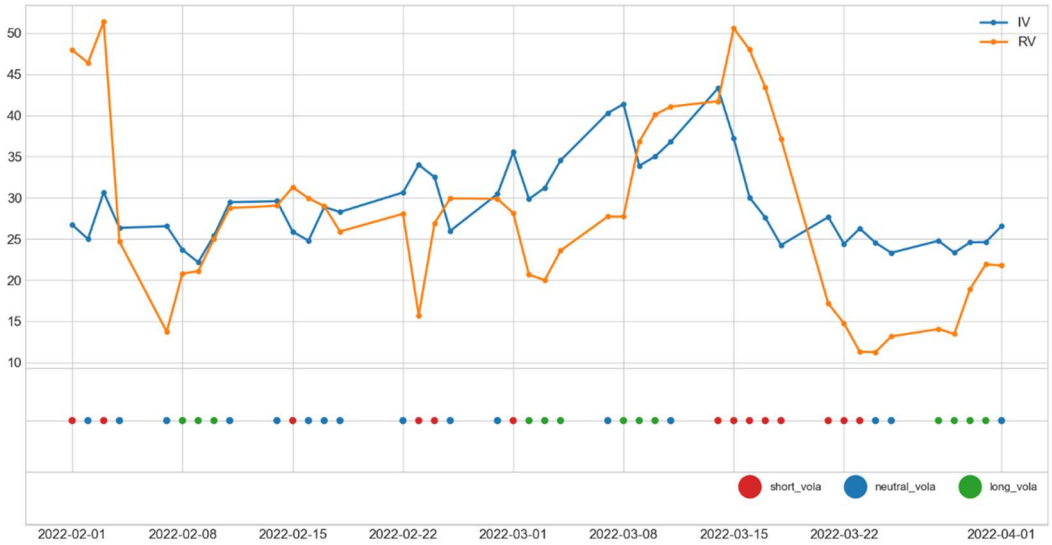
- La serie dei prezzi (*Stock_Price*): Serie di prezzi di chiusura, aggiustate per split e dividendi, scaricata da Yahoo Finance tramite l'apposita API;
- La serie della RV passata (*RV*): Calcolata tramite una finestra scorrevole a 5 giorni, dalla serie dei prezzi (per semplicità si escludono i dividendi);
- La serie della RV passata, a lungo termine (*RV_Long_Term*): Calcolata tramite una finestra scorrevole a 252 giorni (un anno lavorativo);
- La serie dell'IV (*IV*): calcolata a come indice di volatilità (IVX) a 5 giorni, la strategia adottata per il calcolo è spiegata meglio in seguito;
- La serie dei volumi (Volume): Serie del numero di azione scambiate nel mercato in un giorno, scaricata da Yahoo Finance tramite l'apposita API;

- La serie del rateo price to earnings (*PE*): rateo dei guadagni in relazione al prezzo, ottenuto dalla divisione tra il prezzo dell'azione e i guadagni per azione annualizzati, quest'ultimi sono scaricati dal sito di Alpha Advantage tramite l'apposita API;
- La serie d'interesse dell'azienda (Trend): l'interesse in termine di ricerche su Google dell'azienda relativa, scaricata dall'apposita API di GoogleTrends.
- La serie dei giorni mancati ai report (*DtE*): cioè i giorni lavorativi mancanti al successivo report trimestrale, le date sono scaricate dal sito di Alpha Advantage tramite l'apposita API;
- La serie dei giorni mancanti ai dividendi (*DtD*): cioè i giorni lavorativi mancanti al successivo dividendo, le date sono scaricate da Yahoo Finance, tramite l'apposita API;
- La serie di High Minus Low (HML): il primo dei tre fattori del modello fama french, rappresenta la differenza fra i ritorni tra azioni di aziende value e growth;
- La serie di Small Minus Large (SMB): il secondo dei tre fattori del modello fama french, rappresenta la differenza fra i ritorni tra azioni di aziende piccole e grandi;
- La serie del ritorni del mercato (MKR): i cambiamenti annualizzati in percentuale dell'indice di mercato di riferimento, in questo caso quello considerato è il Nasdaq, inoltre i valori sono sempre ottenuti dall'API di Yahoo Finance;
- La serie dell'IV del mercato (MK_IV): indice di volatilità del mercato di riferimento, per mancanza di dati si considera mercato l'S&P500 e dunque l'indice VIX, scaricata da Yahoo Finance tramite l'apposita API;
- La serie di ritorni dello strumento risk free (RFR): ritorni delle bond americane a 1 mese, ottenute dall'API della FRED (Federal Reserve Economic Data);
- La serie del rateo d'inflazione (INFL): rateo d'inflazione a 10 anni, ottenute dall'API della FRED (Federal Reserve Economic Data);

Questo dataset è divisibile logicamente in 4 parti: nella prima abbiamo informazioni (Stock_price, IV, RV, RV_Long_Term, Volume, Trend) che descrivono l'andamento del titolo in borsa; nella seconda (PE) abbiamo una variabile per descrivere lo "stato di salute" dell'azienda; nella terza (DtE, DtD) abbiamo dei parametri per stimare la distanza da possibili catalizzatori della volatilità (tende ad aumentare nei periodi precedenti a report e dividendi); infine abbiamo delle metriche per descrivere lo stato generale del mercato.

La serie temporale del target varia tra 3 label: "long", "neutral" e "short"; indicando appunto la valutazione prevista.

Figure 1: Serie delle volatilità e del target da classificare



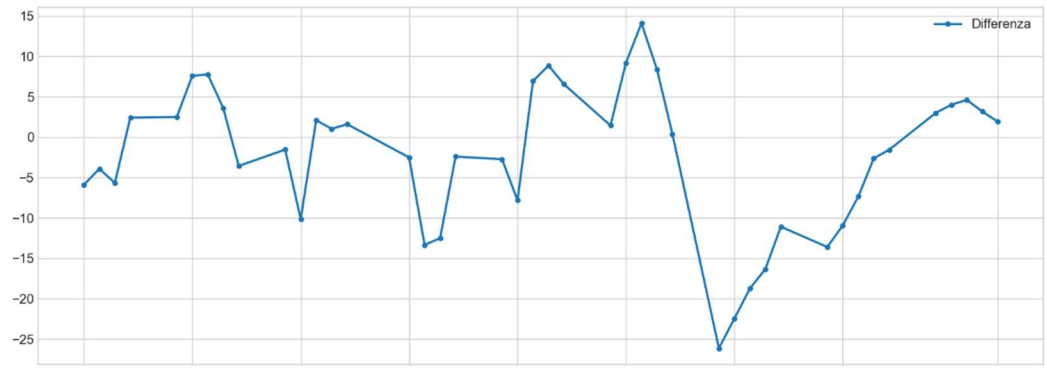


Figure 2: Differenza tra volatilità implicita e realizzata futura. Da cui si può notare la variazione del target con quella della differenza

2.2.1.1 Calcolo dell'indice di volatilità IVX

L'indice viene di volatilità, come già accennato, viene utilizzato per definire in generale l'IV del sottostante. Esso è necessario perché le opzioni di una stessa security con scadenze e strike diversi, presentano IV molto diversi, è dunque utile un indicatore che mostri l'insieme generale. Attualmente il calcolo di un IVX viene realizzato con altri strumenti derivati, come gli swap sulla varianza. Per mancanza di tali dati adotto due strategie diverse, che però non sono più di moda.

La prima strategia si rifà al sistema usato dal CBOE (Chicago Board Options Exchange) per il calcolo del VIX (IVX del S&P500 a 30g), i cui dettagli si possono trovare nel relativo White Paper [9]; essa però, essendo nativa per un IVX a 30 giorni (cioè 21 lavorativi), risulta poco precisa su intervalli più brevi (5g).

La seconda metodologia si rifà a quelle adottate dai Data Provider Quantcha, Nasdaq Data Link e IVolatility , descritte in [10] e [11]. Esse differiscono su alcuni punti, ma concordano su un approccio simile. Cioè: per il calcolo di ogni giorno, si scelgono le due scadenze più vicine alla data da stimare; successivamente, per ognuna si selezionano le opzioni ATM e quelle relativamente vicine, di cui si calcola la media pesata per la distanza; poi, si calcola la media aritmetica tra il valore delle call e quello delle put; infine, si effettua l'interpolazione tra le due date, al fine di ottenere l'approssimazione per quella intermedia.

Nel progetto adotto tale sistema, scegliendo l'interpolazione lineare tra le due date (quella quadratica anche se più adeguata, non è fattibile per mancanza di dati), e selezionando gli strike ATM e a $\pm 2\%, 4\%, 6\%$; di cui si ottiene la media pesata. Essa viene prima calcolata in modo lineare, ma successivamente, aumentando il realismo dell'indicatore uso una media pesata esponenzialmente.

Per esempio, al mercoledì 06/04/2022 si vuole stimare l'IV a 5g (13/04/2022):

- Si selezionano le due date più vicine alla scadenza (08/04/2022 e 15/04/2022).
- In entrambe le date si selezionano le call e le put con strike ATM e a $\pm 2\%, 4\%, 6\%$.
- Si effettua la media esponenziale tra le opzioni scelte, separando call e put.
- Si calcola la media aritmetica tra call e put, ottenendo L'IV stimata per l'08/04 e il 15/04.
- Si effettua l'interpolazione lineare tra le due date, ottenendo l'IV stimata per il 13/04.

2.2.2 Preprocessing

In questa fase vengono convolti le feature non stazionarie: Stock_Price, Volume, PE, Trend, RFR, INFL). Esse verranno prima levigate (tramite l'operazione di smoothing) per poi diventare stazionarie.

La fase di smoothing avviene tramite il filtro Savitzky-Golay, applicando una levigazione leggera (impostato con la finestra a 11 giorni e 3 di ordine polinomiale); in modo da ridurre il rumore nei dati e rendere più semplice la previsione.

La stazionarietà viene ottenuta calcolando la differenza in percentuale di un valore con il suo precedente; per esempio, la generica feature *x* è così ottenuta: $x_t = \frac{x_t - x_{t-1}}{x_{t-1}}$. Tale caratteristica, oltre a rimuovere alcuni problemi legati allo scaling, è essenziale in alcuni modelli di ML; nel caso di LSTM, non è indispensabile ma ne migliora le performance.

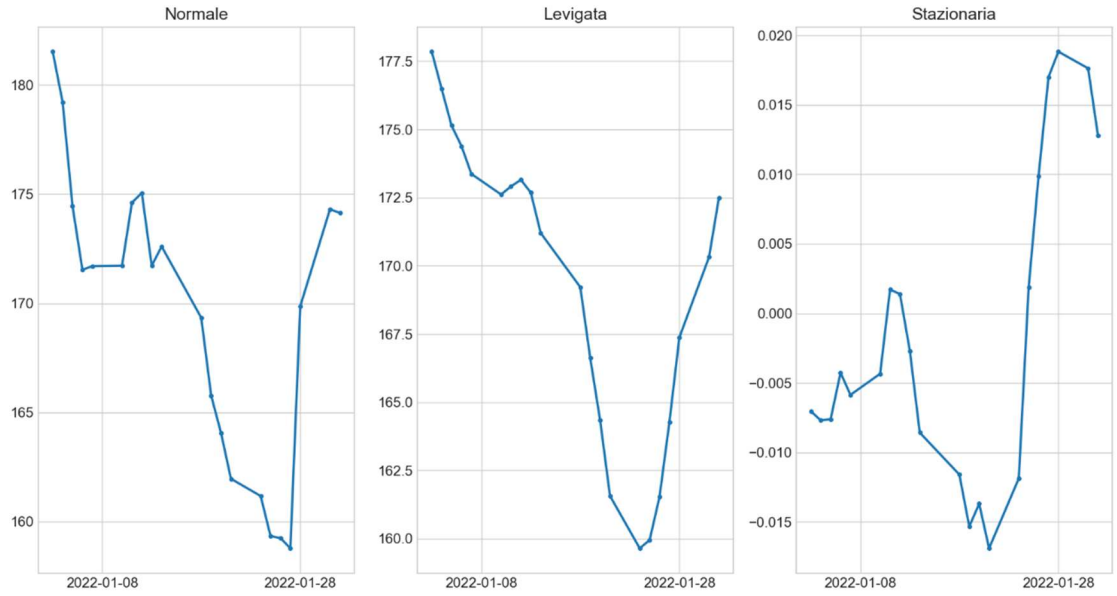


Figure 3: Serie dei prezzi a confronto nei due passaggi di Preprocessing

2.2.3 Feature Selection

Vista la numerosa disponibilità di feature, il modello potrebbe beneficiare dalla riduzione alle variabili più utili; in modo da ridurre il rumore portato da caratteristiche poco correlate con il target, e migliorando quindi il training e le previsioni.

Per realizzare tale fase si utilizza l'algoritmo definito come *Recursive Feature Elimination* (*RFE*). Esso associa ad ogni feature un peso, che ne determina l'importanza; ciò grazie ad un modello esterno, che durante l'addestramento incrementerà i pesi associati alle feature più pertinenti. Successivamente RFE rimuoverà la feature con il peso minore e procederà, in modo ricorsivo, fino al raggiungimento della quantità desiderata.

2.2.4 PCA

Questa fase sfrutta il principio, meglio approfondito in [12]; secondo cui, la rete neurale apprende in modo più efficace da poche serie non correlate rispetto a molte correlate. Infatti, se due variabili sono altamente correlate, varieranno in modo simile portando alla rete la stessa informazione.

Prima i dati verranno scalati, tramite un semplice MinMaxScaler; e successivamente, si sfrutta la tecnica della PCA (Principal Component Analysis), la quale elimina le correlazioni delle feature di input; producendo delle serie non correlate. Essa è implementata dalla funzione omonima, della libreria *scikit-learn*, i cui dettagli si possono trovare in [13].

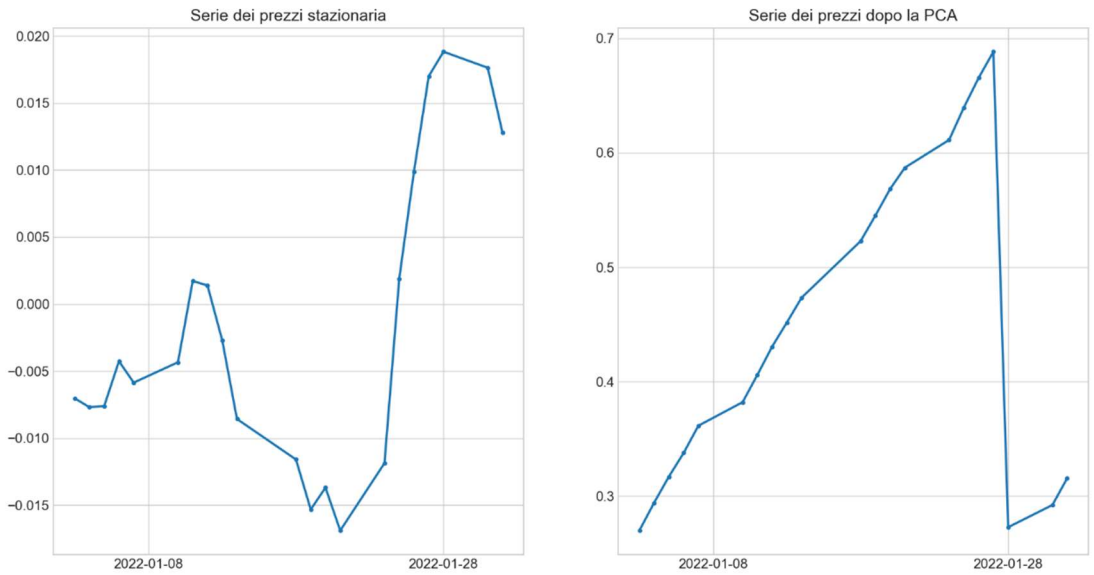


Figure 4: Gli effetti della PCA su una feature

2.2.5 Scaling

In questa fase le feature vengono scalate in intervalli compatibili con le funzioni di attivazione della rete (-1, +1). Mentre al target si applica l'*hot encoding*; cioè si trasformano le 3 etichette, prima in valori numerici e poi in 3 variabili distinte, eliminando così le relazioni di vicinanza tra i numeri. Per esempio, la label "neutral" sarà trasformata nei valori [0, 1, 0], eliminando le relazioni di vicinanza con le alternative long [1, 0, 0] e short [0, 0, 1].

Le feature vengono scalate secondo la stessa strategia, scelta tra una serie di approcci diversi. Tale decisione è trattata come iperparametro del modello e perciò dipende dalle performance ottenute sul validation Test. Ogni approccio è implementato tramite il paradigma OOP: eredita da una classe madre astratta l'interfaccia comune; e ne implementa le logiche di *fit*, *trasform* e *inverse_transform*. Essi sono i metodi canonici di un oggetto

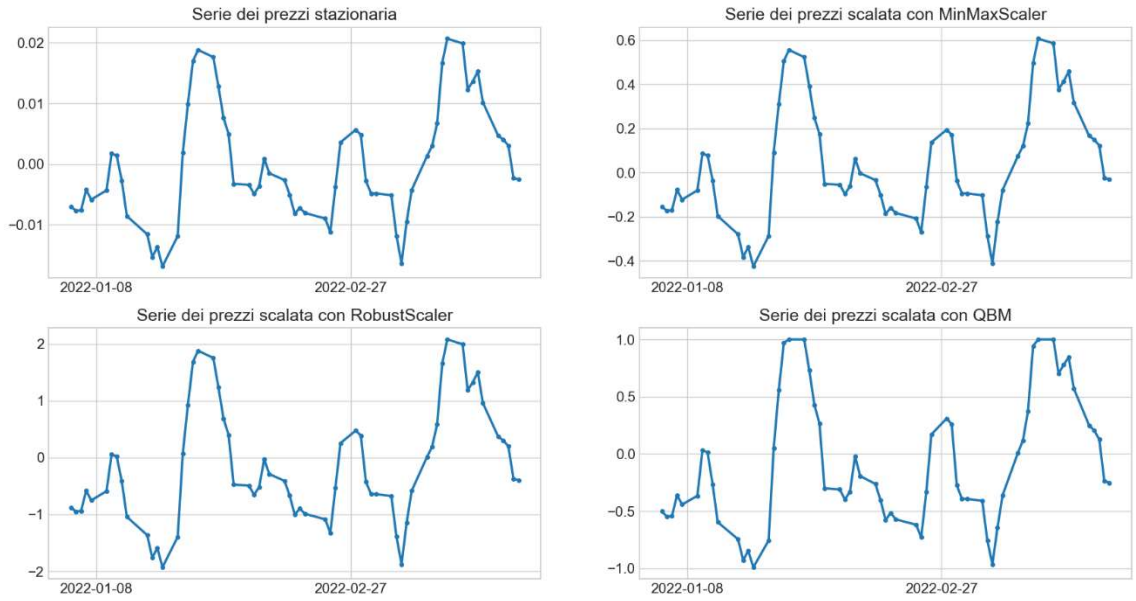
scaler, e gestiscono rispettivamente l'addestramento dei parametri interni, la trasformazione dei dati, e l'inversione della trasformazione. Ogni strategia addestra i propri parametri sul dataset di train ed applica la trasformazione su quello intero. Gli approcci implementati sono:

- Scaler *M* (MinMaxScaler): scala ogni dato nel range (-1, 1):

$$X_{std} = \frac{X - \min(X)}{\max(x) - \min(X)}$$

$X_{scaled} = X_{std} * (\max - min) + min$ (c*on max e min estremi dell'intervallo scelto)*

- Scaler *R* (RobustScaler): scala i dati con statistiche robuste agli outliers, cioè in accordo con gli intervalli dei quantili
- Scaler *RM*: combina i due scaler, applicando prima il RobustScaler e successivamente il MinMaxScaler per avere i valori finali nel range (-1, 1)
- Scaler *BM (Bound-MinMax)*: applica una trasformazione agli outliers, fissa due soglie (bound) ai quantili Q1 e Q3, e riduce ai tali i valori che li superano, successivamente applica il MinMaxScaler;
- Scaler *QBM* (*QuantileTransformer-Bound-MinMax*): prima applica il QuantileTrasformer, con cui trasforma ogni dato in una distribuzione normale, rispettando gli outliers, successivamente applica la metodologia BM.



2.2.6 Lag e Split

Una rete LSTM, come approfondito nel capitolo 1.2.3.1, utilizza dati disposti a sequenze e cioè in 3 dimensioni; è necessaria perciò un'ulteriore fase, in cui trasformare le serie temporali, in serie di sequenze di valori temporalmente trasposti.

Ogni dato sarà quindi composto da una serie di sequenze; cioè per ogni giorno una feature è formata dal valore in quel giorno e da quelli precedenti. Invece la dimensione del target rimane invariata, visto che la rete dovrà produrre un solo valore e non una sequenza (è del tipo Many-to-one).

Infine, il dataset viene diviso, mantenendo l'ordinamento, nelle 3 parti canoniche:

- Dataset di train (0-90%): sono i dati che il programma ha a disposizione; vengono usati per l'addestramento della rete LSTM e dei precedenti strumenti (scaler, RFE, PCA ecc.…).
- Dataset di validation(90-95%): essi vengo usati durante la fase di tuning degli iperparametri; cioè si testa il modello con più configurazioni scegliendo quello che fornisce le performance migliori.

- Dataset di test (95-100%): essi non vengono visti dal modello e sono usati per misurare la precisione finale del modello.

2.3 Definizione del Modello

Per la classificazione sono stati usati più modelli simili, che hanno gli stessi componenti e variano per la loro disposizione e per alcuni iperparametri. Verranno dunque presentati gli elementi e i punti in comune, per poi descrivere i modelli impiegati.

2.3.1 Layer e funzioni di attivazione

In ogni modello vengono adottati 4 tipi di Layer secondo una struttura comune: uno di input; uno di output; due interni ripetuti più volte a coppie con lo stesso ordine. Essi sono:

- *Input Layer*: strato di input, non svolge computazioni e si limita a far entrare le feature nella rete.
- *Gaussian Noise Layer*: non svolge computazioni e quindi non ha pesi da addestrare; aggiunge rumore ai dati per ridurre l'overfitting, è spiegato in dettaglio successivamente.
- *LSTM Layer*: layer contenente celle LSTM; prende in input una sequenza e restituisce un valore o una sequenza; e seguiranno altri layer LSTM, dovrà restituire una sequenza, altrimenti un valore.
- *Dense Layer di output*: layer di neuroni semplici; effettua la classificazione finale, grazie all'uso della funzione di attivazione *softmax*, è sempre composto da 3 neuroni (perché le possibili classi sono 3); come per i layer LSTM è preceduto da un Gaussian Noise Layer.

I primi due layer non svolgono computazioni, e quindi non usano funzioni di attivazione; LSTM invece, impiega le funzioni predefinite: Tanh per il risultato in uscita dalla cella; sigmoid per le attivazioni ricorsive. Mentre il layer di output usa la funzione softmax, necessaria per la classificazione.

L'approccio standard per modellare un problema di classificazione consiste nel prevedere la probabilità che un elemento appartenga ad una classe; in questo modo si risolve la classificazione come se fosse la previsione di un valore reale. Esistono diverse funzioni candidate a tale compito (come la max, argmax e la sigmoid), tuttavia esse sono utilizzabili solo in problemi ad una classe binaria. In questo caso, avendo tre classi, si usa la sigmoid perché restituisce un array di probabilità di appartenenza, la cui somma totale è sempre 1 (cioè il 100% dell'insieme di probabilità).

2.3.2 Loss function e metriche

La funzione di loss usata è la *Crossentropy,* il cui uso è descritto approfonditamente in [14]. In sostanza è la misura della differenza tra due distribuzioni di probabilità. In particolare, si usa la variante *Categorical* che è specifica per i problemi di classificazione a multi-label. Dunque, il modello viene addestrato con l'obbiettivo di minimizzare tale funzione.

Cross-Entropy Loss:

$$L = \sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

Dove:

- *M*: è il numero di classi;
- o: è l'osservazione in questione
- y: è un indicatore binario (0,1) che indica se *c* è la corretta classificazione per o
- p: è la probabilità osservata che o sia della classe *c*
- La formula corrisponde ad una singola istanza; il totale è dato dalla media

La metrica usata è l'*Accuracy*, nella declinazione CategoricalAccuracy. È una funzione molto semplice che calcola la frequenza in cui le classi predette dalla rete coincidono con quelle reali.

2.3.3 Strategie di generalizzazione

Per rendere i modelli più generali ho adottato 4 strategie diverse: il dropout; l'introduzione del rumore con i Gaussian Noise Layer; l'Early Stopping e il Pretraining. Esse agiscono nella fase di Training della rete, con l'obbiettivo di rendere più generalista il modello senza ridurre la sua complessità.

Il dropout è una strategia di regolarizzazione che fa parte della famiglia degli Ensemble Methods. In questa strategia, applicata ad un layer con un parametro ρ; vengono addestrate più reti in parallelo, in cui alcune connessioni in output verranno annullate (dropped out) con una probabilità ρ, mantenendo infine quella ottimale. Introducendo tale effetto randomico la rete non può fare troppo affidamento su ogni nodo, ed è costretta quindi ad essere più generica.

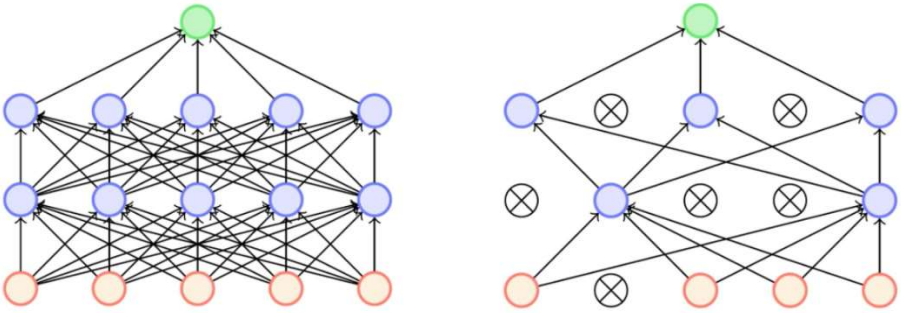


Figure 5: Esempio di Dropout. A sinistra la rete normale, a destra una dopo il dropout

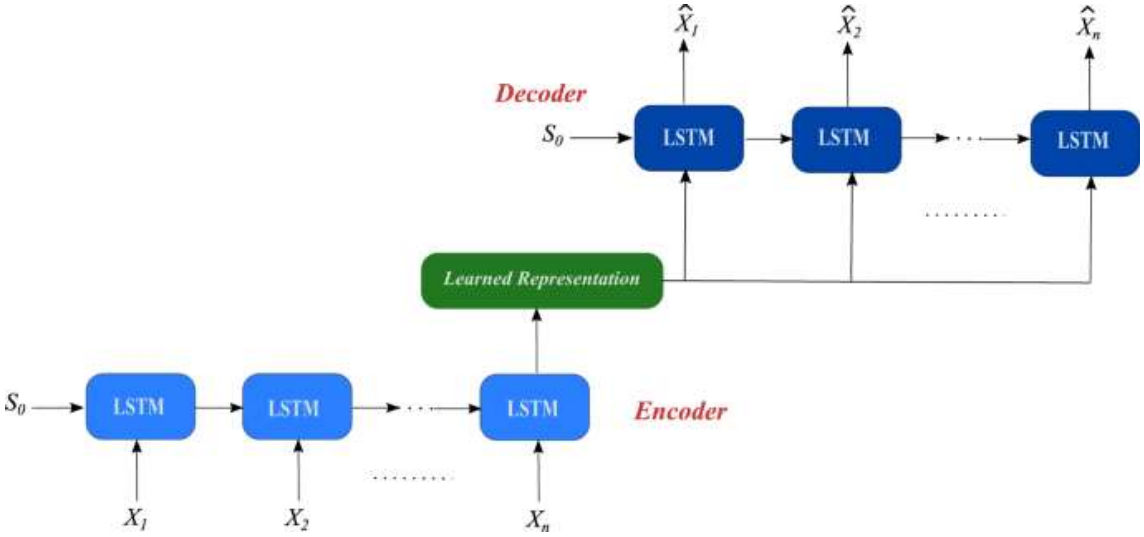
I Gaussian Noise Layer sono degli strati che aggiungono ai dati in input del "rumore" per poi passarli al layer successivo. Esso è preso da una distribuzione gaussiana con varianza definita come iperparametro del layer. In questo modo si riduce l'overfitting perché la rete non può "memorizzare" la componente randomica passata.

L'Early Stopping è una tecnica che si basa sull'interruzione anticipata del processo di training, quando la funzione di loss (o una metrica scelta) calcolata sul dataset di validation, non subisce miglioramenti dopo un numero di epoche prestabilito. In questo caso. si sfrutta uno strumento della libreria keras. per ripristinare la rete dall'ultimo punto di miglioramento.

2.3.3.1 Pretraining in dettaglio

Il Pretraining è una fase, in questo caso non supervisionata, che anticipa l'addestramento e viene utilizzata per inizializzare i pesi dei neuroni; in modo da evitare aree di minimi locali, e rendendo quindi il modello più generico. L'idea generale è di addestrare un modello non supervisionato con le sole feature d'input, per poi trasferire i pesi nel modello finale.

L'implementazione di tale pratica sfrutta un LSTM *Autoencoder*, cioè una rete composta da più strati di LSTM; le cui celle prima diminuiscono in quantità, fino a raggiungere un nucleo unico, per poi invertire il processo fino al ritorno del layer originale. Per esempio, un autoencoder a 2 strati con 4 feature avrà: 4 nodi nel primo layer; 2 nodi nel secondo layer; 1 nodo come nucleo; 2 nodi nel terzo layer; 4 nodi nel quarto layer. Durante il suo addestramento il nucleo e i layer centrali "memorizzano", con sempre meno pesi, una rappresentazione ridotta degli stessi dati; e ciò rende più efficiente il futuro addestramento della rete.



La procedura implementata segue il seguente flusso logico:

1. Vengono copiati, dal modello iniziale, solo i layer LSTM che restituiscono una sequenza, da inizializzare.
2. Si costruisce l'autoencoder; cioè:
   2.1. Si aggiunge il nucleo (composto da una sola cella LSTM).
   2.2. Si copia la struttura iniziale con una simmetria invertita (a specchio).
   2.3. Si aggiunge infine un layer LSTM con numero di celle pari alle feature, in questo modo le dimensioni di input e output coincidono.
3. Si addestra l'autoencoder in modo non supervisionato con i dati del dataset di train.
4. Si prelevano i pesi computati dai layer originali, per inizializzare i layer del modello.

2.3.4 Modelli adottati

Sono stati usati 3 modelli; essi differiscono per il numero di layer, nodi, grado di dropout e utilizzo di Gaussian Noise. Tali modelli sono messi a confronto tra loro, e con sé stessi variando gli iperparametri. Ogni modello è quindi addestrato con sequenze di lunghezza pari a 20 e 60; per 300 e 500 epoche; con l'uso di ogni scaler implementato (M, R, RM, BM e QBM); con e senza l'uso della feature selection.

Ogni modello segue dei parametri comuni:

- in caso di feature selection il numero di unità nel primo strato viene dimezzato;
- la batch size è di 32;
- la finestra di smoothing è di 11;
- ogni layer LSTM, che è seguito da un altro LSTM, restituisce una sequenza;
- la PCA è sempre impiegata;
- le epoche di Early Stopping sono pari ad un terzo di quelle di training;
- le epoche di Pretraining sono pari ad un quarto di quelle di training;
- l layer di input e output sono identici per ogni modello;
- Ogni layer (a eccezione di quello di input) è preceduto da un layer Gaussian Noise.

Nello specifico gli strati interni sono:

- Il primo modello ha i seguenti layer interni:

Layer	**Unità**	**Dropout**	**Recurrent dropout**	**Gaussian Noise**
1° layer LSTM	30	0.2	0.2	0.2
2° layer LSTM	8	0	0	0

- Il secondo modello ha i seguenti layer interni:

Layer	Unità	Dropout	Recurrent dropout	Gaussian Noise
1° layer LSTM	15	0.2	0.1	0.2
2° layer LSTM	8	0.1	0.1	0.1
3° layer LSTM	5	0	0	0

- Il terzo modello ha i seguenti layer interni:

Layer	Unità	Dropout	Recurrent dropout	Gaussian Noise
1° layer LSTM	30	0.2	0.1	0.2
2° layer LSTM	8	0.2	0.2	0.2
3° layer LSTM	5	0	0	0

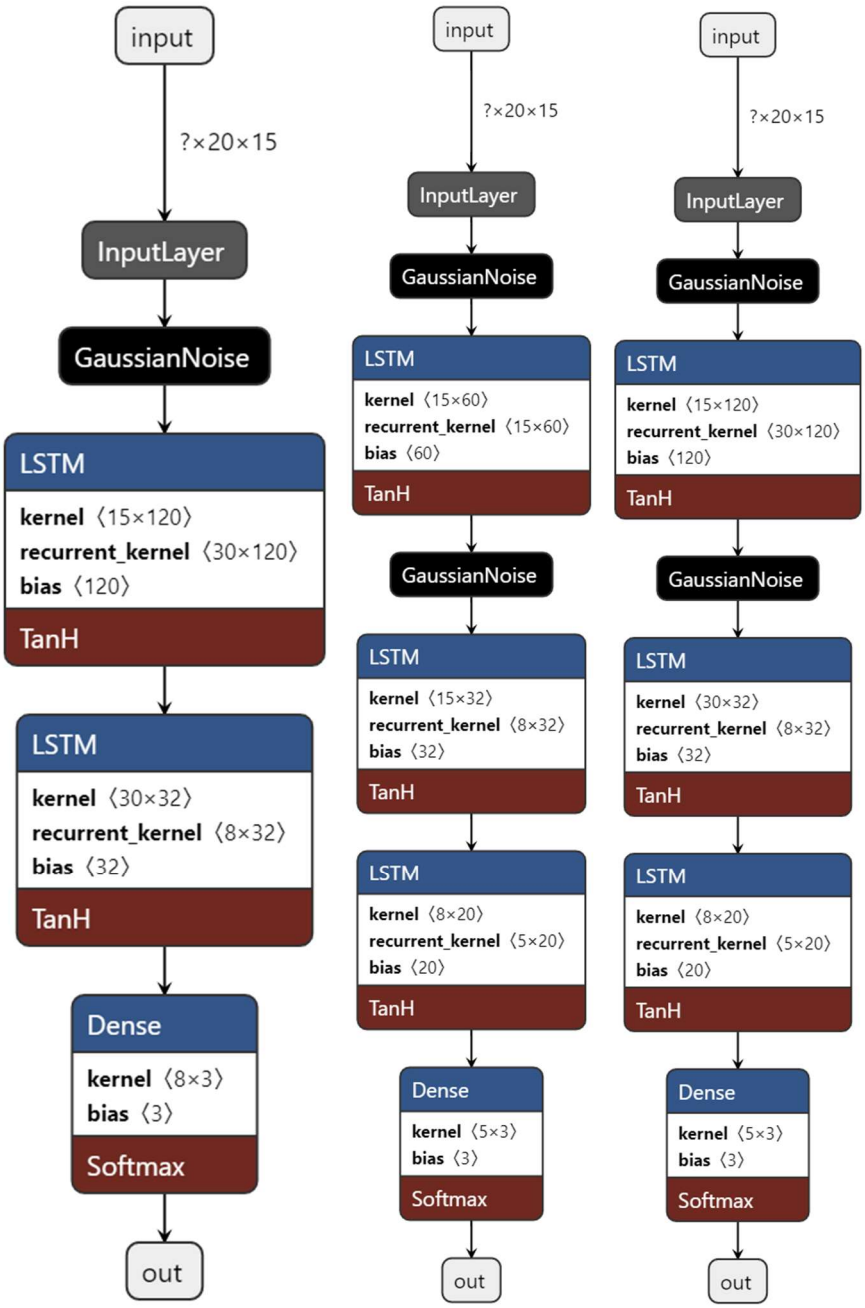


Figure 6: Rispettivamente il primo, il secondo e il terzo modello

2.4 Strategia di Trading

In questo paragrafo seguirà la spiegazione del sistema di trading, approfondendo la metodologia impiegata e gli spread adottati, approfondendo il loro mantenimento durante il tempo di vita.

2.4.1 Funzionamento generale

Come precedentemente accennato, l'obbiettivo della rete è quello di fornire un ordine di acquisto, vendita o nullo alla sistema di trading; a seconda dell'esito verrà presa la conseguente posizione, con l'ausilio di 4 spread che verranno messi a confronto.

Gli spread verranno aperti il venerdì a chiusura (vista la disponibilità dei soli dati di chiusura), con scadenza di una settimana e mantenuti fino alla tale. Essi sono di 4 tipi, e sono tutti progettati per l'esposizione alla volatilità implicita. Perciò hanno:

- delta neutro;
- vega positivo se lo spread è long alla volatilità, altrimenti negativo;
- theta negativo se lo spread è short alla volatilità, altrimenti positivo.

Un ordine è nullo, se la differenza tra IV e RV non supera una soglia, pari ad un percentile scelto e calcolato sui 125 giorni precedenti.

2.4.2 Gli spread adottati

Possiamo suddividere gli spread in 2 categorie binarie; a seconda delle opzioni, anche dette *Leg,* che li compongono: quelli semplici, composti da 2 opzioni; e quelli avanzati, composti da 4 opzioni. Essi sono:

- *Straddle*: composto da una call e una put, alla stessa scadenza e stesso strike ATM, ha un debito o credito d'ingresso generalmente alto; se viene:
    - Comprato: tramite l'acquisto delle due Leg, si ottiene un'esposizione long sulla volatilità con un debito d'ingresso, ha rischio limitato e profitto illimitato;

- Venduto: tramite l'acquisto delle due Leg, si ottiene un'esposizione short sulla volatilità con un credito d'ingresso, ha rischio illimitato e profitto limitato;

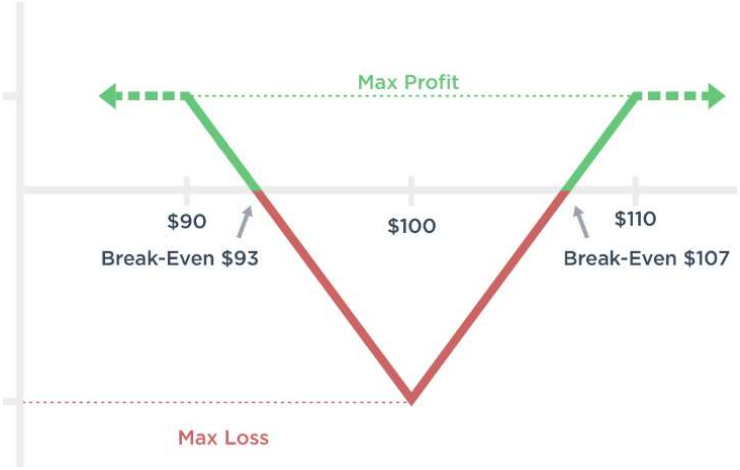


Figure 7: Long straddle

- *Strangle*: composto da una call e una put, alla stessa scadenza e a due strike OTM simmetrici, ha un debito o credito d'ingresso minore della straddle; se viene:
  - Comprato: tramite l'acquisto delle due Leg, si ottiene un'esposizione long sulla volatilità con un debito d'ingresso, ha rischio limitato e profitto illimitato;
  - Venduto: tramite la vendita delle due Leg, si ottiene un'esposizione short sulla volatilità con un credito d'ingresso, ha rischio illimitato e profitto limitato;

Figure 8: Long strangle

- *Butterfly Call*: composto da quattro call, alla stessa scadenza; due vendute ATM, due comprate a due strike simmetrici uno ITM e l'altro OTM; se viene:
    - Comprato: si ottiene un'esposizione short sulla volatilità, con un debito d'ingresso, ha rischi e profitti limitati;
    - Venduto: tramite l'inversione delle posizioni precedentemente descritte, si ottiene un'esposizione long sulla volatilità, con un credito d'ingresso, ha rischi e profitti limitati;

Figure 9: Long Butterfly

- *Iron Condor*: composto da due coppie di una call e una put con strike simmetrici e alla stessa scadenza; la prima ha strike OTM e viene venduta, la seconda ha strike OTM più distante e viene acquistata; se lo si:
    - Compra: si ottiene un'esposizione short sulla volatilità, con un credito d'ingresso, ha rischi e profitti limitati;
    - Vende: tramite l'inversione delle posizioni precedentemente descritte, si ottiene un'esposizione long sulla volatilità, con un debito d'ingresso, ha rischi e profitti limitati;

Figure 10: Long Iron Condor

Quindi i primi due spread hanno dei costi di commisione minori (perché ci sono meno opzioni da scambiare), però non hanno delle perdite potenzialmente illimitate. Invece spread come la long strangle e short iron condor hanno costi d'ingresso minori ma sono necessari movimenti maggiori per andare in profitto; al contrario short strangle e long iron condor, necessitano di movimenti minori ma portano meno profitti.

L'implementazione viene fatta tramite l'usilio di una classe *SpreadLeg*, che a seconda della posizione e dell'opzione (call o put) preleva dal dataset i dati corretti; e di una classe *Spread* che definisce l'interfaccia e i metodi comuni degli spread (come il plot e la funzione di hedging), Infine ogni spread eredita la classe generica, e implementa i metodi astratti con cui si calcolano gli strike e le leg.

2.4.3 Delta hedging

Questi spread, seppur comprati con delta neutro, assumono un delta positivo o negativo più il prezzo continua a spostarsi dal valore iniziale in cui esso è stato comprato. Tale proprietà è descritta dal gamma dello spread, che in questi casi non è neutro. Ciò comporta che il valore dello spread subirà influenze dovute alla direzione di movimento.

Per risolvere questo problema si può chiudere la posizione (in profitto o in perdita) prematuramente, ma ciò andrebbe contro l'obbiettivo del progetto. La soluzione adottata è quella di fare l'hedging della posizione.

L'hedging comporta comprare o vendere il sottostante, pari al delta assunta della posizione in quel momento; quando esso supera una certa soglia (in questo caso 30). Nello specifico si compra quando il delta è negativo e si vende quando è positivo, al fine di ripareggiare l'esposizione al delta. Ciò rende di nuovo lo spread neutro alla direzione di movimento, portando quindi ad una riduzione del rischio, con la conseguenza di minor profitti.

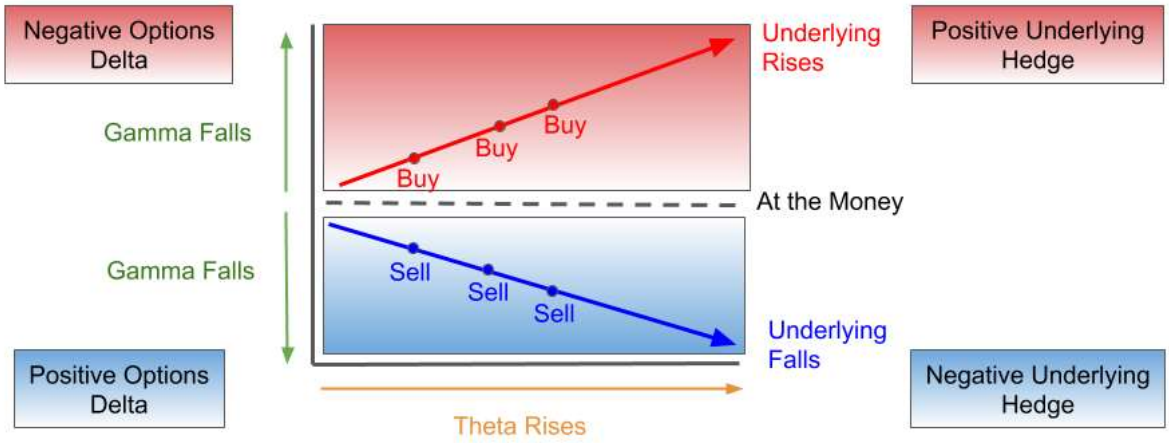


Figure 11: Short straddle con delta hedging

2.4.4 Alcuni esempi

Figure 12: Esempio di Short Straddle calcolata. In questo caso l'IV prevista non viene realizzata, e quindi lo spread va in profitto. Inoltre, l'hedging non viene usato perché il delta non supera la soglia minima.

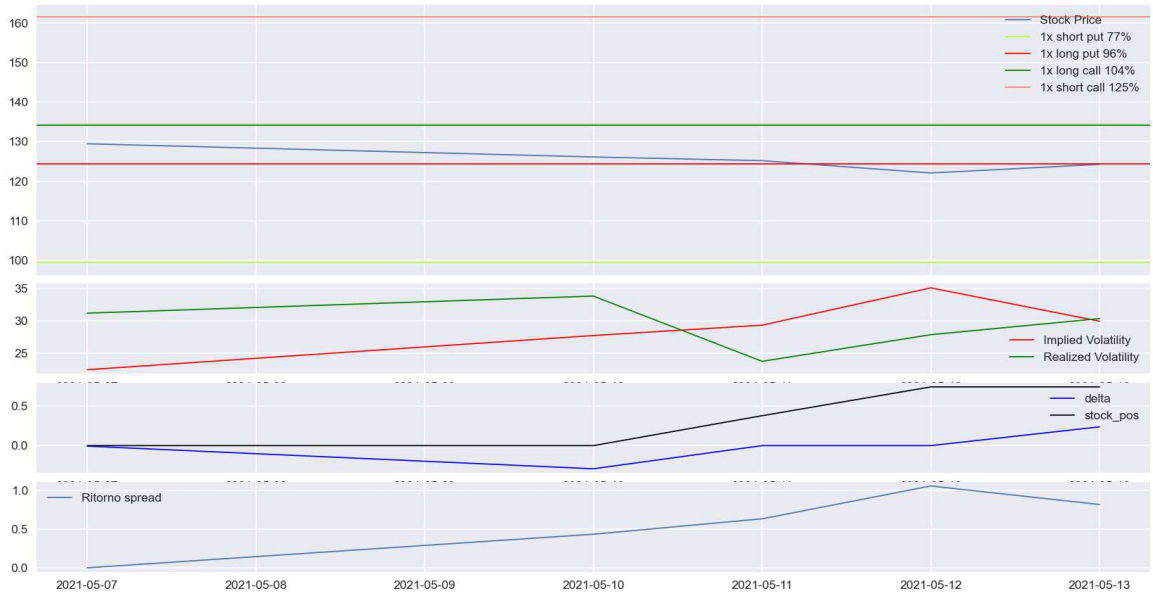


Figure 13: Esempio di Short Iron Condor calcolato. In questo caso la volatilità implicita viene superata da quella realizzata, con conseguente profitto. Tuttavia, il delta hedging è superfluo e riduce il profitto ottenuto dallo spread.

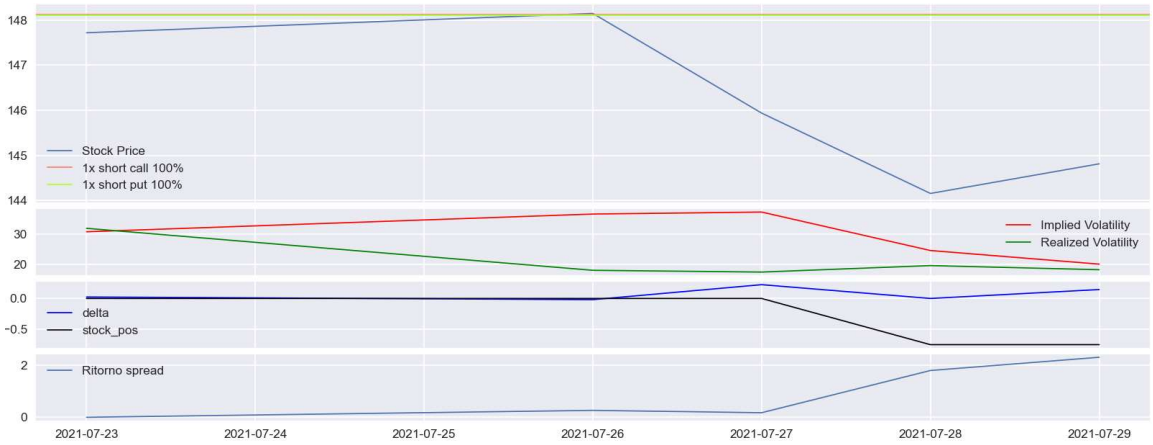


Figure 14: Esempio di Short Straddle calcolata. In questo caso la volatilità implicita non viene realizzata, ma visto il movimento di prezzo sviluppato, tale spread sarebbe comunque in perdita. Grazie all'hedging il delta rimane neutrale e lo spread finisce in profitto.

2.4.5 Selezione degli strike degli spread

Gli strike, come già definito precedentemente, sono composti da una o due coppie di opzioni che la stessa distanza dal prezzo corrente; la quale deve essere stabilita a priori ed è influenzata dalla frequenza di hedging e dal percentile selezionato. Per effettuare tale scelta, si selezionano prima dei

valori candidati; poi si utilizza l'ultimo 20% del dataset di train, per eleggere i valori migliori.

2.5 I risultati

In questo paragrafo verranno mostrati i risultati ottenuti dai processi di selezione e dai test. Visualizzando dunque i modelli e gli spread ottimali, terminando poi con il risultato finale.

2.5.1 Risultati dei modelli

I migliori risultati sono stati ottenuti generalmente con:

- 500 epoche di addestramento;
- l'uso della feature selection;
- sequenze di lunghezza pari a 20

Il resto degli iperparametri è stato testato approfonditamente, con i seguenti risultati:

Indice di Modello	scaler	precisione trainSet	precisione validSet
1	RM	0.708	0.653
3	RM	0.712	0.617
2	BM	0.648	0.617
1	RM	0.659	0.617
1	QBM	0.672	0.614
3	BM	0.707	0.61
3	QBM	0.665	0.606
1	RM	0.707	0.603
1	M	0.657	0.603
2	RM	0.653	0.588

Figure 15: I risultati dei test degli iperparametri

2.5.2 Risultati degli spread

Ogni risultato è il ritorno in percentuale composto della strategia. Cioè, prima si calcola il ritorno semplice di ogni posizione e poi si calcola loro composizione. Tuttavia, tali valori sono utili solo a scopi confrontativi; perché

non considerano le commissioni e il capitale richiesto a margine per le operazioni short.

Ritorno semplice di uno spread:

$$r = \frac{P - V}{|P|} + r_s$$

Dove:

- *P*: è premium dello spread (se è un credito è positivo, altrimenti è negativo)
- *V:* è il prezzo finale dello spread
- r_s: è il ritorno composto della stock per la quota mantenuta (causata dall'hedging)

Il ritorno totale è la composizione dei ritorni semplici lungo tutto il periodo di trading.

Verranno confrontati il metodo di calcolo per l'IV, la frequenza di hedging e gli offset degli strike; al fine di stabilire i parametri migliori.

2.5.2.1 Straddle:

IV media	Frequenza di hedging	Ritorno
esponenziale	0.2	11305.3374
lineare	0.2	10870.4783
lineare	0.3	8000.4443
esponenziale	0.3	7864.4198

Figure 16: I migliori ritorni per la Straddle

2.5.2.2 Strangle:

IV media	Frequenza di hedging	Offset Long Volatilità	Offset Short Volatilità	Ritorno
lineare	0.3	4	4	5309952.4936
lineare	0.2	2	4	4218263.196
esponenziale	0.2	2	4	3724726.285
esponenziale	0.3	4	4	3116941.7007
lineare	0.2	4	4	1735763.3391
esponenziale	0.2	4	4	1279257.3179
lineare	0.3	2	4	824595.2517
esponenziale	0.3	2	4	617621.5925

Figure 17: I migliori ritorni per la Strangle

2.5.2.3 Butterfly:

IV media	Frequenza di hedging	Offset (copertura) Long Volatilità	Offset (copertura) Short Volatilità	Ritorno
esponenziale	0.2	8	6	3322.0731
lineare	0.2	8	6	2944.9868
esponenziale	0.2	8	8	2486.6051
lineare	0.2	8	8	2204.3237
esponenziale	0.3	10	6	2059.8842
esponenziale	0.3	8	6	2037.6358
lineare	0.3	10	6	2037.4611
lineare	0.3	12	6	1963.1141
esponenziale	0.3	12	6	1951.3294
lineare	0.3	8	6	1944.2632

Figure 18: I migliori ritorni per il Butterfly

2.5.2.1 Iron Condor:

IV media	Frequenza di hedging	Offset Long	Offset Short	Offset (copertura) Long	Offset (copertura) Short	Ritorno
lineare	0.2	2	2	12	16	760818.585
esponenziale	0.2	2	2	12	16	753210.3892
lineare	0.2	2	2	16	16	649901.8709
lineare	0.2	2	2	20	16	628679.7926
esponenziale	0.2	2	2	16	16	610907.6987
esponenziale	0.2	2	2	20	16	575870.606
lineare	0.2	2	2	12	8	274002.9074
esponenziale	0.2	2	2	12	8	271262.8683
lineare	0.2	2	2	16	8	234057.0205
lineare	0.2	2	2	20	8	226414.0359

Figure 19: I migliori ritorni per l'Iron Condor

2.5.3 Risultato finale

Il sistema finale utilizzerà il modello dalle migliori performance e i settaggi ottimali per gli spread. Ottenendo:

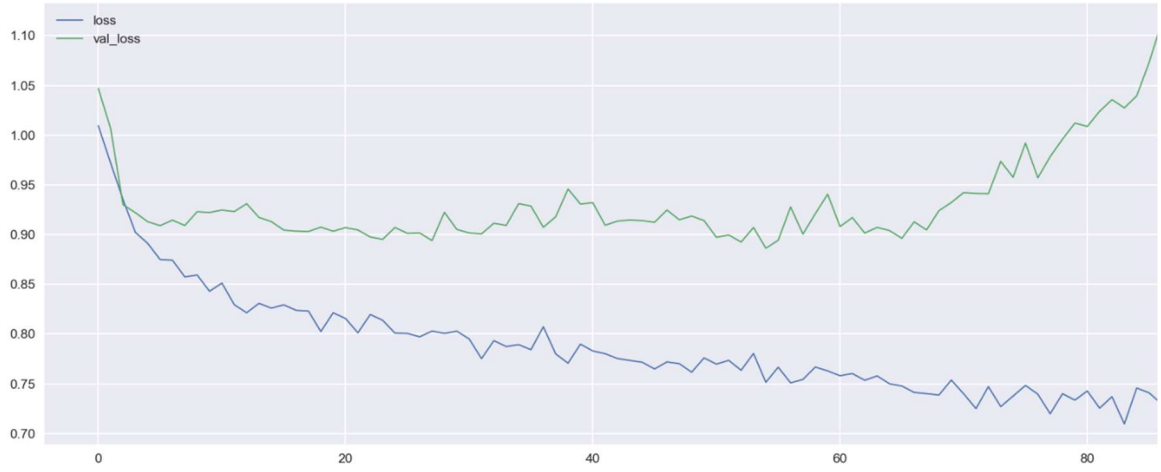


Figure 20: History di training. Come si può osservare dalla loss di validation (in verde), dopo una primissima fase di miglioramento la rete smette di apprendere i pattern.

Figure 21: Previsione sul TrainSet. (Precisione: 71%)

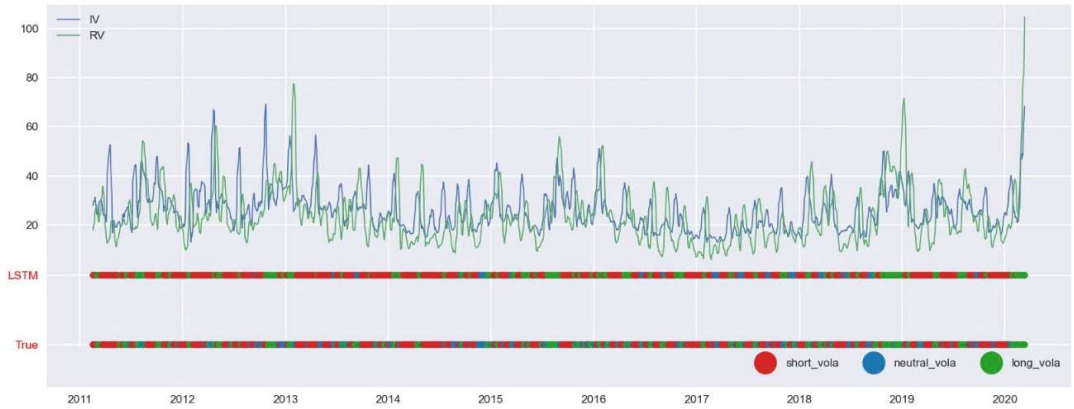


Figure 22: Previsione sul ValidationSet. (Precisione: 65%)

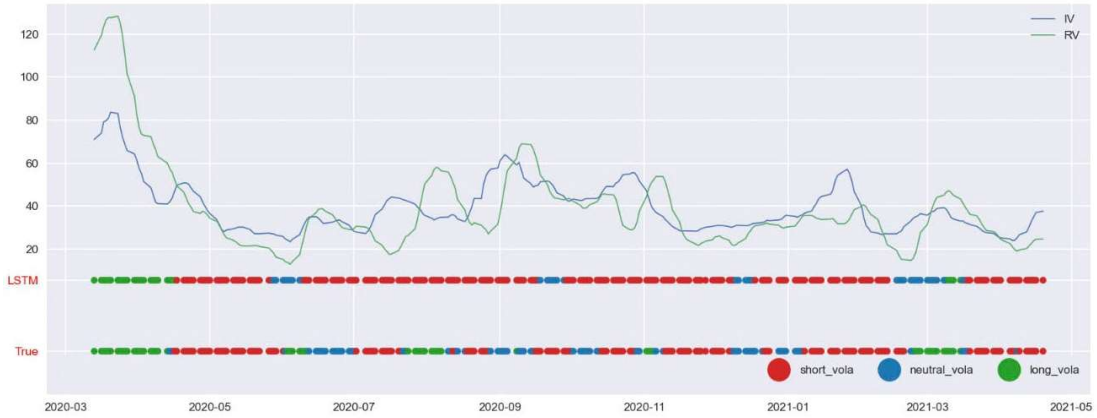


Figure 23: Previsione sul TestSet.(Precisione del 61%) I risultati della previsione sono messi a confronto con i valori reali e con il Naive Forecast

Le performance del sistema sono messe a confronto con un algoritmo non intelligente, il così detto *Naive Forecast*. Esso viene realizzato senza il ML; dunque, il nostro modello diventa inutile se non riesce a batterlo. In questo caso il Naive Forecast è realizzato con la strategia base; quella in cui si compra o vende volatilità, se la differenza tra IV e RV supera una determinata soglia storica (solitamente la media).

Come si può notare, il modello non riesce a superare il Naive Forecast. Questo significa che la Rete Neurale fallisce nell'imparare i pattern dei dati, e seppur le cause possano essere molteplici, una tra le più probabilità è legata alla teoria del *Random Walk*.

Infine, applicando le previsioni ottenute, il sistema realizza dei profitti per le prime settimane; ma appena si raggiungono le previsioni errate, il sistema realizza del perdite portando il ritorno totale in negativo.

2.5.3.1 Random Walk Hypotesis

È un'ipotesi molto diffusa e centro di ricerche e dibattiti; anche se è tipicamente considerata vera, almeno dal punto di vista teorico.

Tale teoria, approfondita e testata in [15], afferma che non è possibile prevedere i cambiamenti di prezzo futuri, da quelli passati; perché, ogni informazione viene assimilata sul mercato in modo randomico, con conseguenti deviazioni casuali. Altrimenti, se le deviazioni fossero sistematiche, sarebbe possibile realizzare degli arbitraggi; che però, nel lungo termine, finirebbero per annullarsi. Dunque, la serie dei prezzi è associabile con un movimento randomico e perciò non prevedibile con il ML.

I soggetti del progetto erano la volatilità implicita e realizzata; esse da definizione, sono la deviazione standard delle variazioni di prezzo (una attesa, l'altra ottenuta). Questo legame potrebbe significare che la teoria Random Walk si estende indirettamente anche alla volatilità, almeno nel breve termine. Uno sviluppo futuro, molto interessante, poterebbe occuparsi di indagare e testare l'esistenza di questo legame.

Conclusioni

Il Machine Learning, e in particolare le reti neurali, sono uno strumento molto potente. In questo elaborato si è presa una visione sommaria delle loro caratteristiche; da cui si possono osservare sia i punti di deboli che i punti di forza, ma soprattutto le potenzialità di questi strumenti.

La finanza è solo uno dei possibili domini applicativi per le ANN, e come evidenziato in [16], il numero di paper in merito e quindi il relativo interesse sono in continua crescita; tuttavia, la maggior parte di essi sono orientati alla previsione dei prezzi (di stock e indici) e delle tendenze. Questo paper approfondisce un tema non ancora troppo coinvolto: quello della volatilità, e propone una soluzione pratica. Inoltre, pone un particolare interesse sull'uso dei dati reali e non simulati; che solitamente semplificano le previsioni, rendendole irrealistiche.

Come evidenziato nel paragrafo dei risultati, nonostante i diversi mezzi impiegati, la soluzione proposta non riesce a superare con consistenza il Naive forecast. Futuri sviluppi possono indagare il fallimento di questo tentativo; dimostrando l'eventuale legame della volatilità con la teoria del Random Walk, e come questo renda inapplicabile l'approccio usato. Altri sviluppi possono invece proporre nuove soluzioni: isolando una delle due volatilità, limitandosi ad un'unica previsione; oppure applicando altre tecnologie, come quelle dell'apprendimento per rinforzo.

Bibliografia:

[1] G. Rebala, A. Ravi e S. Churiwala, An Introduction to Machine Learning, Springer Publishing Company, 2019.

[2] M. Mohri, A. Rostamizadeh e A. Talwalkar, Foundations of Machine Learning 2, Cambridge (MA): MIT Press, 2018.

[3] C. Aggarwal, Neural Networks and Deep Learning: A Textbook (1st. ed.), Springer Publishing Company, Incorporated, 2018.

[4] B. Krose e P. Van der Smagt, An introduction to neural networks, 2011.

[5] A. F. Murray, Applications of neural networks, Boston: Kluwer Academic Publishers, 1995.

[6] K. Mehrotra, C. K. Mohan e S. Ranka, Elements of artificial neural networks, MIT press, 1997.

[7] Y. Yong, S. Xiaosheng e Z. Jianxun, «A review of recurrent neural networks LSTM cells and network architectures,» in *Neural Computation*, 2019, pp. 1235-1270.

[8] C. Olah, «Understanding LSTM Networks,» 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs.

[9] CBOE, «VIX White Paper,» [Online]. Available: https://cdn.cboe.com/resources/vix/vixwhite.pdf.

[10] N. D. L. Quantcha, «US Equity Historical & Option Implied Volatilities,» [Online]. Available: https://data.nasdaq.com/data/VOL-us-equity-historical-option-implied-volatilities/documentation?anchor=knowledge-base.

[11] IVolatility, «IVolatility Education - Implied Volatility Index (IV Index),» [Online]. Available: https://www.ivolatility.com/help/5.html.

[12] B. Boukhatem, S. Kenai, A. T. Hamou, D. Ziou e M. Ghrici, «Predicting concrete properties using neural networks(NN) with principal component analysis(PCA) technique,» Techno-Press, P. O. Box 33 Yusong Taejon

305-600 Korea, 2012.

[13] scikit-learn, «Principal component analysis (PCA),» [Online]. Available: https://scikit-learn.org/stable/modules/decomposition.html#pca.

[14] G. E. Nasr, E. Badr e C. Joun, «Cross entropy error function in neural networks: Forecasting gasoline demand,» in *FLAIRS conference*, 2002, pp. 381-384.

[15] V. Horne, J. C. Parker e G. C. Parker, «The Random-Walk Theory: An Empirical Test,» in *Financial Analysts Journal, vol. 23*, 1967, pp. 87-92.

[16] O. B. Sezer, M. U. Gudelek e A. M. Ozbayoglu, Financial time series forecasting with deep learning: A systematic literature review: 2005–2019, 2020.

[17] A. Krogh, «What are artificial neural networks?,» Nature Biotechnology, 2008, pp. 195-197.

[18] D. Svozil, V. Kvasnicka e J. Pospichal, «Introduction to multi-layer feed-forward neural networks,» in *Chemometrics and Intelligent Laboratory Systems*, Elsevier Science, 1997, pp. 43-62.

[19] H. M. Sazli, A brief review of feed-forward neural networks, Communications Faculty of Sciences University of Ankara Series A2-A3, Physical Sciences and Engineering 50, 2006.

[20] D. Erhan, A. Courville, Y. Bengio e P. Vincent, «Why Does Unsupervised Pre-training Help Deep Learning? Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics,» in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Teh, Yee Whye and Titterington, Mike; PMLR, 2010, pp. 201-208.

[21] F. Wang e D. M. J. Tax, Survey on the attention based RNN model and its applications in computer vision, ArXiv abs/1601.06823, 2016.

[22] W. C. Merrill, W. Gail, Y. Goldberg, R. Schwartz, A. N. Smith e E. Yahav, A Formal Hierarchy of RNN Architectures, ACL, 2020.

[23] M. Schuster e K. Paliwal, «Bidirectional recurrent neural network,» in

Transactions on Signal Processing, IEEE, 1997, pp. 2673-2681.

[24] C. Bennett, Trading Volatility: Trading Volatility, Correlation, Term Structure and Skew, 2014.

[25] L. Downey, «Essential Options Trading Guide.,» 2022. [Online]. Available: https://www.investopedia.com/.